Programare paralela si distribuita – Laborator 3

Munteanu Bianca-Stefania

Grupa 235/1

**Analiza cerințelor**

* **Tema**: Scrieti un program bazat pe MPI care face suma a 2 numere mari. Un ‘numar mare’ este un numar cu mai mult de 10 cifre
* **Reprezentare:** tablou de cifre (numere intregi fara semn - byte) in care cifra cea mai nesemnificativa este pe prima pozitie.
* **Datele de intrare**
* Cele 2 numere mari se citesc din fisierele “Numar1.txt” (un numar cu N\_1 cifre) si “Numar2.txt” (un numar cu N\_2 cifre).
* Fiecare din aceste fisiere contine la inceput un numar (N) care reprezinta numarul de cifre si apoi cifrele numarului respectiv.
* **Implementare:** C++ ( cel putin C++11 )
* **Testare:** masurati timpul de executie pentru

1. Numar1=Numar2=”123456789123456789” ; p=1,2,3,4,8,16
2. N1=N2=1000 (random digits); p=1,2,4,8,16
3. N=100, N2=100000 (random digits); p=1,2,4,8,16

Fisierul trebuie creat anterior prin adaugare de numere generate aleator. Toate rularile trebuie executate cu acelasi fisier.

* **Constrangeri:**

1. N\_1, N\_2 > 10

* **Postconditie:** Fisierul “Numar3.txt” va contine suma celor 2 numere mari.

**Proiectare**

**Varianta 0** – implementare secventiala C++11.

p - procese MPI

**Varianta 1** – considera rezolvarea problemei prin executia urmatoarelor etape:

1) id\_proces\_curent=1

2) procesul 0 repeta urmatoarele actiuni pana cand se citesc toate cifrele numerelor

a. citeste cate N/p cifre din cele 2 fisiere

b. le trimite procesului “id\_proces\_curent”

c. incrementeaza “id\_proces\_curent”

3) procesele fac suma cifrelor primite si calculeaza “report” (carry) corespunzator;

4) fiecare proces (cu exceptia ultimului) trimit “reportul” la procesul urmator care il foloseste pentru actualizarea rezultatului (procesul id=1 nu primeste carry - il considera egal 0)

5) rezultatul final se obtine in procesul 0. care scrie rezultatul in fisierul “Numar3.txt”

Posibilitati:

a) procesele primesc carry inainte de a primi cifrele pe care trebuie sa le adune

b) procesele primesc cifrele pe care trebuie sa le adune si apoi carry de la precedent

alegeti pentru implementare varianta care este mai buna!

**Apeluri MPI**

* **In Procesul 0**

// trimite cifrele catre celelalte procese

MPI\_Send(numar1 + i, cifre\_per\_proces[id\_proces\_curent], MPI\_INT, id\_proces\_curent, 20, MPI\_COMM\_WORLD);

MPI\_Send(numar2 + i, cifre\_per\_proces[id\_proces\_curent], MPI\_INT, id\_proces\_curent, 30, MPI\_COMM\_WORLD);

// primeste sumele partiale de la procese si le reuneste intr-un vectorul rezultat "suma"

MPI\_Status out;

for (int i = 1; i < nr\_procese; i++) {

int poz = 0;

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

poz += cifre\_per\_proces[j];

}

MPI\_Recv(suma + poz, cifre\_per\_proces[i], MPI\_INT, i, 40, MPI\_COMM\_WORLD, &out);

}

}

* **Procesul ‘rank’ (rank != 0)**

// primeste cifrele de la procesul 0

MPI\_Status out;

MPI\_Recv(subnumar1, cat, MPI\_INT, 0, 20, MPI\_COMM\_WORLD, &out);

MPI\_Recv(subnumar2, cat, MPI\_INT, 0, 30, MPI\_COMM\_WORLD, &out);

// primeste carry de la procesul precedent

MPI\_Recv(&received\_carry, 1, MPI\_INT, rank - 1, 50, MPI\_COMM\_WORLD, &out);

// trimite carry la procesul urmator, daca nu am ajuns la ultimul proces

if (rank != nr\_procese - 1)

MPI\_Send(&carry, 1, MPI\_INT, rank + 1, 50, MPI\_COMM\_WORLD);

// trimite suma partiala la procesul 0

MPI\_Send(suma\_partiala, cat, MPI\_INT, 0, 40, MPI\_COMM\_WORLD);

**Varianta 2** – considera rezolvarea problemei prin executia urmatoarelor etape:

1) procesul 0 citeste cele 2 numere si le stocheaza in 2 tablouri:

a. daca un numar are mai putine cifre se completeaza cu cifre nesemnificative

2) cifrele celor 2 numere se distribuire proceselor folosind MPI\_Scatter (daca nu este valabila conditia p|N,

unde N=max{N\_1,N\_2}, N\_1 nr de cifre ale primului numar, N\_2 nr de cifre ale celui de-al doilea, atunci

se mareste N corespunzator si se completeaza cu 0-uri)

3) procesele fac suma cifrelor primite si calculeaza “report” (carry) corespunzator

4) fiecare process (cu exceptia ultimului) trimit “reportul” la procesul urmator care il foloseste pentru

actualizarea rezultatului

5) rezultatul final se obtine in procesul 0 (MPI\_Gather)

6) procesul 0 scrie rezultatul in fisierul “Numar3.txt”

* **Apeluri MPI**

// cifrele celor 2 numere se distribuire proceselor folosind MPI\_Scatterv

MPI\_Scatterv(numar1, cifre\_per\_proces, displs, MPI\_INT, subnumar1, cifre\_per\_proces[rank], MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Scatterv(numar2, cifre\_per\_proces, displs, MPI\_INT, subnumar2, cifre\_per\_proces[rank], MPI\_INT, 0, MPI\_COMM\_WORLD);

// primeste carry-ul de la procesul precedent

MPI\_Recv(&received\_carry, 1, MPI\_INT, rank - 1, 0, MPI\_COMM\_WORLD, &out);

// trimite carry la procesul urmator, daca nu am ajuns la ultimul proces

if (rank != nr\_procese - 1)

MPI\_Send(&carry, 1, MPI\_INT, rank + 1, 0, MPI\_COMM\_WORLD);

// rezultatul final

MPI\_Gatherv(suma\_partiala, cifre\_per\_proces[rank], MPI\_INT, suma, cifre\_per\_proces, displs, MPI\_INT, 0, MPI\_COMM\_WORLD);

**Varianta 3**– considera rezolvarea problemei prin executia urmatoarelor etape:

1) id\_proces\_curent=1

2) procesul 0 repeta urmatoarele actiuni pana cand se citesc toate cifrele numerelor

a. citeste cate N/p cifre din cele 2 fisiere

b. le trimite procesului “id\_proces\_curent”

c. incrementeaza “id\_proces\_curent”

3) un process cu id<>0 primeste setul de cifre de la procesul 0 si face adunarea intr-un vector rezultat si

actualizeaza “reportul”(carry”) pe care il trimite la procesul urmator

(atentie un proces cu id (id<>1, id<>0) primeste informatie de la procesul 0 si de la procesul (id-1) dar

ordinea intre cele 2 nu este sigura … se cere sa se foloseasca MPI\_Irecv )

4) rezultatul final se obtine in procesul 0 prin agregarea rezultatelor folosind transmitere asincrona

5) procesul 0 scrie rezultatul in fisierul “Numar3.txt”

**Apeluri MPI**

* **In Procesul 0**

MPI\_Isend(numar1 + i, cifre\_per\_proces[id\_proces\_curent], MPI\_INT, id\_proces\_curent, 20, MPI\_COMM\_WORLD, &req1);

MPI\_Wait(&req1, MPI\_STATUSES\_IGNORE);

MPI\_Isend(numar2 + i, cifre\_per\_proces[id\_proces\_curent], MPI\_INT, id\_proces\_curent, 30, MPI\_COMM\_WORLD, &req2);

MPI\_Wait(&req2, MPI\_STATUSES\_IGNORE);

MPI\_Irecv(suma + poz, cifre\_per\_proces[i], MPI\_INT, i, 40, MPI\_COMM\_WORLD, &resultReq[i]);}

MPI\_Wait(&resultReq[i], MPI\_STATUS\_IGNORE);

* **Procesul ‘rank’ (rank != 0)**

MPI\_Irecv(subnumar1, cat, MPI\_INT, 0, 20, MPI\_COMM\_WORLD, &req1);

MPI\_Wait(&req1, MPI\_STATUSES\_IGNORE);

MPI\_Irecv(subnumar2, cat, MPI\_INT, 0, 30, MPI\_COMM\_WORLD, &req2);

MPI\_Wait(&req2, MPI\_STATUSES\_IGNORE);

MPI\_Irecv(&received\_carry, 1, MPI\_INT, rank - 1, 50, MPI\_COMM\_WORLD, &req3);

MPI\_Wait(&req3, &status1);

MPI\_Isend(&carry, 1, MPI\_INT, rank + 1, 50, MPI\_COMM\_WORLD, &req3);

MPI\_Isend(suma\_partiala, cat, MPI\_INT, 0, 40, MPI\_COMM\_WORLD, &req4);

**Rezultate Varianta 1**

|  |  |  |
| --- | --- | --- |
| Dimensiune numere | **Nr procese** | **Timp executie** (ms) |
|  |  |  |
| N1=N2=18 | secvential | 1,27765 |
| 2 | 2,57787 |
| 4 | 4,52593 |
| 8 | 7,95343 |
| 16 | 11,91264 |
|  |  |  |
| N1=N2=1000 | secvential | 6,4395 |
| 2 | 7,54623 |
| 4 | 9,30137 |
| 8 | 12,99766 |
| 16 | 17,62065 |
|  |  |  |
| N1=100  N2=100000 | secvential | 457,3646 |
| 2 | 456,6487 |
| 4 | 448,3059 |
| 8 | 451,1886 |
| 16 | 461,4132 |

**Rezultate Varianta 2**

|  |  |  |
| --- | --- | --- |
| Dimensiune numere | **Nr procese** | **Timp executie** (ms) |
|  |  |  |
| N1=N2=18 | secvential | 1.40069 |
| 2 | 2.47608 |
| 4 | 3,83717 |
| 8 | 7,06826 |
| 16 | 13,33978 |
|  |  |  |
| N1=N2=1000 | secvential | 6,18461 |
| 2 | 7,5844 |
| 4 | 8,77815 |
| 8 | 11,33759 |
| 16 | 16,81273 |
|  |  |  |
| N1=100  N2=100000 | secvential | 444,3847 |
| 2 | 450,0332 |
| 4 | 441,0425 |
| 8 | 448,0527 |
| 16 | 461,2735 |

**Rezultate Varianta 3**

|  |  |  |
| --- | --- | --- |
| Dimensiune numere | **Nr procese** | **Timp executie** (ms) |
|  |  |  |
| N1=N2=18 | secvential | 1,92801 |
| 2 | 3,46932 |
| 4 | 4,85235 |
| 8 | 8,43964 |
| 16 | 12,93037 |
|  |  |  |
| N1=N2=1000 | secvential | 6,88278 |
| 2 | 8,41106 |
| 4 | 9,84281 |
| 8 | 13,07712 |
| 16 | 18,84885 |
|  |  |  |
| N1=100  N2=100000 | secvential | 469,5387 |
| 2 | 458,1161 |
| 4 | 471,1689 |
| 8 | 467.5006 |
| 16 | 469,9168 |

**Analiza rezultatelor**

Observatii:

* Rezultatele ambelor variante sunt asemanatoare, insa varianta 1 (cu send si recv) este putin mai eficienta decat varianta 2 (cu scatter si gather) in majoritatea situatiilor evaluate.
* Nu se observa o imbunatatire a timpului de executie atunci cand crestem numarului de procese care efectueaza calculele, varianta secventiala ramanand cu timpul de executie cel mai mic, in majoritatea situatiilor. Exceptie face cazul in care N1=100, N2=100000 cand observam o imbunatatire la cresterea numarului de procese.
* In cadrul variantei 1 ni s-a cerut sa alegem varianta cea mai buna dintre:
* a) procesele primesc carry inainte de a primi cifrele pe care trebuie sa le adune
* b) procesele primesc cifrele pe care trebuie sa le adune si apoi carry de la precedent

Am ales sa folosesc varianta b) si sa adun carry-ul la final. Astfel, procesele pot efectua adunarile paralelizat si nu sunt nevoite sa astepte pana cand celelalte procese isi incheie executia pentru a trimite carry-ul, asa cum s-ar fi intamplat in varianta a)

In continuare, voi reprezenta grafic rezultatele obtinute pentru fiecare varianta:

**Grafice – Varianta 1 vs Varianta 2**