Documentatie Laborator 3

**Analiza cerintelor:**

Scrieti un program bazat pe MPI care face suma a 2 numere mari.

‘numar mare’ = 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.

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++11.

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!

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”

- Varianta3– optionala pentru 4 puncte suplimentare! (transmitere asincrona care va produce performanta mai buna) (=> nota 14 pentru laboratorul L3)

Se 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”

Teste pentru fiecare varianta:

1) Numar 1 = “123456789123456789” = Numar2

2) N\_1=1000 si N\_2=1000 (random digits)

3) N\_1=100 si N\_2=100000 (random digits

**Proiectare:**

**C++:**

|  |
| --- |
| +main(int, char\*): int |
| +metoda1\_paralel():void |
| +metoda2\_paralel():void |
| +secvential():void |
| +generateFile(string,int):void |
| +checkEquality():bool |
| +writeInFile(int\*, int, string):void |
| +stringToNumber(string&, int[]):void |
| +addZerosToNumber(string&, int[], int):void |

**main(int, char\*):void** – functia principala a programului.

**metoda1\_paralel():void** – functie de calculare paralela a sumei celor 2 numere mari, cu ajutorul MPI, varianta 1.

**metoda2\_paralel():void** – functie de calculare paralela a sumei celor 2 numere mari, cu ajutorul MPI, varianta 2.

**secvential():void** – functie de calculare secventiala a sumei celor 2 numere mari.

**generateFile(string,int):void** – functie pentru generarea unui fisier ce contine numeral de cifre ale unui numar mare si numarul mare respective.

**checkEquality():Boolean** - functie pentru verificarea egalitatii fisierelor: secvential si parallel metoda 1 sau parallel metoda 2.

**writeInFile(int\*, int, string):void** – functie pentru scrierea in fisier a numarului rezultat in urma adunarii celor 2 numere mari.

**stringToNumber(string&, int[]):void** – functie pentru transformarea numarului mare citit din fisier, din string in array de int-uri.

**addZerosToNumber(string&, int[], int):void** – functie care adauga zerouri nesemnificative la un numar.

**Detalii de implementare:**

Am folosit 2 array-uri pentru cele 2 numere mari si le-am inversat, pentru a aduna mai usor. Adunam pe rand pozitiile primului numar cu pozitiile celui de-al doilea numar si se mai aduna un carry, care poate fi 1 sau 0, in functie daca suma valorilor de la indecsii precedenti depaseau valoarea 10 sau nu.

**Cazuri de testare:**

|  |  |  |
| --- | --- | --- |
| Numere | Type | Timp |
| number = 123456789123456789 | secvential | 77.6 |
| Varianta 1 | 112.6 |
| Varianta 2 | 118 |
| Numar1 – cifre  = Numar2 –cifre =1000 | secvential | 80.1 |
| Varianta 1 | 101.7 |
| Varianta 2 | 119.4 |
| Numar1 – cifre = 100  Numar2 –cifre = 100000 | secvential | 160.7 |
| Varianta 1 | 144.2 |
| Varianta 2 | 148.8 |

**Analiza rezultatelor:**

Putem observa faptul ca executia secventiala este mai rapida decat cea parallel.

De asemenea Varianta 1 este putin mai rapida decat Varianta 2.