**Laborator 3**

**Cerința:**

**MPI**

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

* 1. 2) procesul 0 repeta urmatoarele actiuni pana cand se citesc toate cifrele numerelor a. citeste cate N/p cifre din cele 2 fisiere
  2. b. le trimite procesului “id\_proces\_curent”
  3. c. incrementeaza “id\_proces\_curent”
  4. 3) procesele fac suma cifrelor primite si calculeaza “report” (carry) corespunzator;
  5. 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)
  6. 5) rezultatul final se obtine in procesul 0. care scrie rezultatul in fisierul “Numar3.txt”

Posibilitati:

1. a) procesele primesc carry inainte de a primi cifrele pe care trebuie sa le adune
2. 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. 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. 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. 3) procesele fac suma cifrelor primite si calculeaza “report” (carry) corespunzator
  4. 4) fiecare process (cu exceptia ultimului) trimit “reportul” la procesul urmator care il foloseste pentru actualizarea rezultatului
  5. 5) rezultatul final se obtine in procesul 0 (MPI\_Gather)
  6. 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

* 1. 2) procesul 0 repeta urmatoarele actiuni pana cand se citesc toate cifrele numerelor a. citeste cate N/p cifre din cele 2 fisiere
  2. b. le trimite procesului “id\_proces\_curent”
  3. 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:**

Se citesc din fișierele de input cele două numere, care sunt gata scrise în ordine inversă.

Numerele vor fi stocate in vectori, de tipul vector<int> (V0), int[] (V1) și int\* (V2).

O cifră se calculează în felul următor: *(a[i] + b[i] + carry) % 10*, unde carry este inițial egal cu *0*, iar apoi egal cu *(a[i] + b[i] + carry) / 10*.

După calculul tuturor cifrelor, se vor scrie elementele vectorului rezultat în fișier, în ordine inversă (deoarece au fost citite în ordine inversă).

**Pentru Varianta 1:**

* Procesul cu rank 0 calcula pentru fiecare din celelalte p – 1 procese începutul și sfârșitul bucății de interval care va fi procesată de ele. De asemenea, va citi cifră cu cifră din fișier și va trimite proceselor aceste cifre. Dacă bucata ce trebuie procesată e mai mare decât câte numere mai sunt în fișier, se va trimite cifra 0.
* Celelalte procese vor primi de la procesul cu rank 0 începutul și sfârșitul, și numerele ce trebuiesc procesate. De asemenea, procesele vor trimite la următoarele și vor primi de la precedentele carry-ul calculat de ele. În cazul procesului cu rank 1, acesta va considera carry ca fiind 0 (nu are de la ce proces să primească), iar procesul cu ultimul rank va trebui să trimită un 1 în plus, în cazul în care carry-ul său este 1 (a apărut o cifră în plus la rezultat).
* Procesul cu rank 0 va primi fiecare bucată din rezultat procesată de celelalte procese și va adăuga cifra extra (dacă este cazul).

**Pentru Varianta 2:**

* Procesul cu rank 0 va citi numerele din fișiere cât se poate, umplând restul vectorului cu 0-uri în cazul în care e nevoie (vectorii au resultSize elemente, unde resultSize este CMMMC între max(n1, n2) și p; n1 – lungimea primului număr, n2 – lungimea celui de-al doilea număr, p – numărul de procese).
* Apoi, se vor calcula vectorii *displacements* și *offsets,* care vor reține pozițiile de start și numărul de numere ce trebuie procesate de fiecare proces.
* Se aplică Scatter pe vectorii cu numere.
* Se calculează cifrele, și se trimite/primește carry-ul în manieră asemanătoare cu cea prezentată la V1.
* Se aplică carry-ul primit.
* Se aplică Gather pe vectorii rezultat.
* Se verifică dacă mai e nevoie de încă o cifră, și se adaugă dacă da.

**Timpii de execuție:**

* **Varianta 0**: 0.1981
* **Varianta 1:**

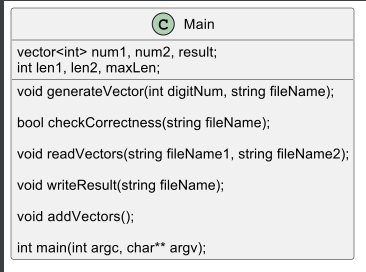
|  |  |  |
| --- | --- | --- |
| Dimensiuni numere | Nr threads | Timp executie |
| N1=100000 N2=100000 | 3 | 104.5618 |
|  | 5 | 357.7612 |
|  | 7 | 300.7367 |
|  | 9 | 196.5297 |
|  | 17 | 134.951 |
| N1=18 N2=18 | 3 | 0.64368 |
|  | 5 | 1.09444 |
|  | 7 | 1.52473 |
|  | 9 | 2.03503 |
|  | 17 | 4.00807 |
| N1=1000 N2=1000 | 3 | 1.59252 |
|  | 5 | 1.82064 |
|  | 7 | 2.15344 |
|  | 9 | 2.56377 |
|  | 17 | 4.71132 |
| N1=100 N2=100000 | 3 | 104.989 |
|  | 5 | 109.6658 |
|  | 7 | 224.0502 |
|  | 9 | 179.7249 |
|  | 17 | 145.4939 |

* **Varianta 2:**

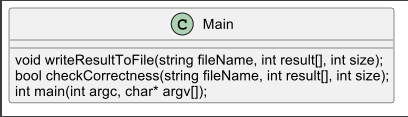
|  |  |  |
| --- | --- | --- |
| Dimensiuni | Nr threads | Timp executie |
| N1=100000 N2=100000 | 2 | 1.44412 |
|  | 4 | 1.77134 |
|  | 6 | 2.28499 |
|  | 8 | 5.94742 |
|  | 16 | 8.69052 |
| N1=18 N2=18 | 2 | 0.35066 |
|  | 4 | 0.74569 |
|  | 6 | 1.03612 |
|  | 8 | 1.3304 |
|  | 16 | 2.781 |
| N1=1000 N2=1000 | 2 | 0.34274 |
|  | 4 | 0.76004 |
|  | 6 | 1.01728 |
|  | 8 | 1.29901 |
|  | 16 | 2.7869 |
| N1=100 N2=100000 | 2 | 1.0857 |
|  | 4 | 1.62237 |
|  | 6 | 2.3298 |
|  | 8 | 2.78221 |
|  | 16 | 5.94919 |

**Diagrame:**

* Varianta 0:



* Varianta 1:



* Varianta 2:

