Documentatie Tema 3

# Cerinta

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

‘numar mare’ = numar cu mai mult de 10 cifre

## Consideratii generale:

Reprezentare unui numar = tablou de cifre (numere intregi fara semn - byte) in care cifra cea mai nesemnificativa este pe pozitia 0.

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

## Implementari MPI cu p procese:

### 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
   1. citeste cate N/(p-1) cifre din cele 2 fisiere
   2. le trimite procesului “id\_proces\_curent”
   3. 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:

1. procesele primesc carry inainte de a primi cifrele pe care trebuie sa le adune
2. procesele primesc cifrele pe care trebuie sa le adune si apoi carry de la precedent

Alegeti pentru implementare varianta care este mai buna!

Optimizare -? Adunarea cifrelor inainte de a astepta carry. Este posibil?

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

1. procesul 0 citeste cele 2 numere si le stocheaza in 2 tablouri:
   1. 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 (se foloseste MPI\_Gather)
6. procesul 0 scrie rezultatul in fisierul “Numar3.txt”

# Modul de rezolvare

## Varianta 0

**void citesteDinFisier(string numeFisier, int\* numar, int& n)**  
Citește un număr mare dintr-un fișier text și îl stochează într-un vector numar. Prima valoare din fișier reprezintă numărul de cifre ale numărului (stocată în n), iar următoarele valori sunt cifrele propriu-zise.

**void scrieInFisier(string numeFisier, int\* numar, int n)**  
Scrie un număr mare stocat într-un vector numar într-un fișier text. Prima linie a fișierului conține numărul de cifre n, urmată de cifrele numărului separate prin spații.

**void genereazaNumarMare(int\* numar, int n)**  
Generează un număr mare aleatoriu de n cifre și îl stochează într-un vector numar. Fiecare cifră generată este între 0 și 9.

**void inverseazaSir(int\* sir, int n)**  
Inversează ordinea elementelor într-un vector sir de dimensiune n. Această funcție este utilă pentru operațiile care necesită manipularea numărului în ordine inversă.

**void egaleazaLungimeNumere(int\* numar1, int& n1, int n2)**  
Extinde un număr stocat într-un vector numar1 cu zerouri la început pentru a-i egala lungimea cu un al doilea număr de dimensiune n2. Această funcție ajustează lungimea pentru a facilita adunarea numerelor de lungimi diferite.

**void sumaSecventiala(int\* numar1, int& n1, int\* numar2, int& n2, int\* numar3, int& n3)**  
Calculează suma a două numere mari stocate în vectorii numar1 și numar2. Rezultatul este stocat în numar3, iar n3 reține numărul de cifre din rezultat. Funția gestionează operația de adunare, inclusiv carry, și ajustează lungimile numerelor dacă e necesar.

## Varianta 1

1. **Initializare si setari MPI**

* Alocă memorie pentru vectorii de cifre numar1, numar2 și numar3.
* Inițializează MPI, obține numărul de procese (nrprocs) și identificatorul (id) fiecărui proces.

1. **Generarea numerelor si distribuirea segmentelor (doar pt procesul 0)**

* Procesul 0 generează două numere mari, le salvează în fișiere și apoi le citește pentru a le folosi în adunare.
* Se ajustează lungimile numerelor pentru a fi egale și multiplu de nrprocs - 1 prin completarea cu zerouri.
* Distribuie dimensiunea segmentului (chunckSize) și părți din numere către celelalte procese.

1. **Colectarea rezultatelor de la procese (doar procesul 0)**

* Procesul 0 așteaptă și primește rezultatele parțiale de la fiecare proces și le combină pentru a obține rezultatul final.
* Calculează și afișează durata execuției și scrie rezultatul final în fișier după eliminarea zerourilor de început.

1. **Procesarea segmentelor (pentru procesele diferite de 0)**

* Procesele care nu sunt 0 primesc dimensiunea segmentului și segmentele aferente din numar1 și numar2 de la procesul 0.
* Vectorii locali auxNumar1, auxNumar2, și auxNumar3 sunt inițializați pentru a stoca aceste segmente.

1. **Calculul sumei segmentelor cu transport**

* Fiecare proces calculează suma cifrelor corespunzătoare din segmentele sale și reține transportul pentru a-l trimite procesului precedent, dacă este cazul.

1. **Trimiterea si receptionarea transportului intre procese**

* Procesele trimit transportul către procesul anterior, dacă nu sunt primul proces.
* Procesul 1 gestionează transportul final, dacă există, și ajustează segmentul său pentru a include acest transport.
* Fiecare proces, cu excepția ultimului, primește transportul de la procesul următor și îl adaugă la segmentul său.

1. **Trimiterea rezultatelor segmentelor catre procesul 0**

* Fiecare proces non-0 trimite dimensiunea segmentului rezultat și conținutul lui auxNumar3 către procesul 0 pentru a fi combinate.

1. **Incheierea executiei MPI**

* MPI\_Finalize finalizează execuția MPI, eliberând resursele folosite.

## Varianta 2

**void adaugare\_zero(int\* a, int& na, int nrprocs)**  
Această funcție completează vectorul a cu zerouri la început, astfel încât lungimea lui na să fie un multiplu de nrprocs. Acest lucru este necesar pentru a permite distribuirea egală a cifrelor între procesele MPI.

* Inversează vectorul a pentru a adăuga zerourile la sfârșit.
* Calculează câte zerouri trebuie adăugate (k).
* Adaugă k zerouri la finalul vectorului și actualizează lungimea na.
* Inversează din nou vectorul pentru a reveni la ordinea corectă.

**void sterge\_zero(int\* sir, int& n)**  
Această funcție elimină zerourile suplimentare de la începutul vectorului sir după finalizarea calculului, pentru a obține rezultatul corect al adunării.

* Inversează vectorul pentru a muta zerourile la sfârșit.
* Înlătură zerourile de la sfârșit, actualizând lungimea n.
* Inversează vectorul înapoi la ordinea inițială.

1. **Declaratii initiale si setari MPI**

* Inițializează vectorii pentru cele două numere mari și pentru rezultat.
* Obține dimensiunea segmentului (chunkSize) pentru fiecare proces.
* MPI\_Init inițializează execuția MPI, iar MPI\_Comm\_size și MPI\_Comm\_rank determină numărul total de procese și identificatorul procesului curent.

1. **Generarea si citirea numerelor mari (executat doar de procesul 0)**

* Procesul 0 generează două numere mari și le scrie în fișiere pentru a le folosi în adunare.
* Se citește fiecare număr din fișier și se ajustează lungimea lor pentru a fi egale.
* Completează fiecare număr cu zerouri, dacă este necesar, astfel încât lungimea lor să fie un multiplu de nrprocs (numărul de procese MPI).
* Calculează chunkSize (numărul de cifre procesate de fiecare proces) și trimite această informație fiecărui proces.

1. **Primirea dimensiunii segmentului pentru procesele diferite de 0**

* Procesele care nu sunt 0 (adică nu sunt principale) primesc dimensiunea segmentului (chunkSize) de la procesul 0.

1. **Distribuirea partilor de numere pentru fiecare process**

* Vectorii locali auxNumar1, auxNumar2 și auxNumar3 sunt creați pentru a stoca segmentele locale ale fiecărui proces.
* MPI\_Scatter împarte vectorii numar1 și numar2 în segmente și distribuie fiecare segment către procese. Fiecare proces primește chunckSize elemente.

1. **Calculul partial al sumei in fiecare proces**

* Fiecare proces își calculează partea proprie din suma celor două numere, folosind transportul la fiecare pas, și stochează rezultatul în auxNumar3.

1. **Trimiterea si receptionarea transportului intre procese**

* Fiecare proces (în afară de procesul 0) trimite transportul către procesul anterior.
* Procesele (în afară de ultimul) primesc transportul de la procesul următor și îl adaugă la segmentul local, continuând până când transportul devine 0.

1. **Colectarea rezultatelor de la procese**

* MPI\_Gather colectează segmentele rezultate auxNumar3 de la fiecare proces și le combină în vectorul complet numar3 pe procesul principal (0).

1. **Finalizarea rezultatului pe procesul 0**

* Procesul 0 inversează vectorul numar3 pentru a aranja corect cifrele rezultatului.
* Dacă transportul final este diferit de 0, acesta este adăugat la rezultat.
* Durata execuției este calculată și afișată.
* Funcția sterge\_zero elimină zerourile de început, iar rezultatul final este scris în fișierul test3Rezultat.txt.

1. **Finalizarea executiei MPI**

* MPI\_Finalize încheie execuția MPI și eliberează resursele folosite de MPI.

# Teste

# Varianta 0

|  |  |
| --- | --- |
| Test | Timp de executie (microsecunde) |
| Numar 1 = “123456789123456789” = Numar2 | 1143 |
| N\_1=1000 si N\_2=1000 (random digits) | 14231 |
| N\_1= 100 si N\_2=100000 (random digits) | 204439 |
| N\_1=N\_2=16 Numar1 = “9999444499999999”  Numar2=”9999555599999999” | 10397 |

# Varianta 1

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Timp de executie  (microsecunde) | | |
| 4 | 8 | 16 |
| Numar 1 = “123456789123456789” = Numar2 | 337718 | 18297266 | 36628995 |
| N\_1=1000 si N\_2=1000 (random digits) | 203109 | 15115001 | 33086683 |
| N\_1= 100 si N\_2=100000 (random digits) | 1007309 | 10145935 | 29725076 |
| N\_1=N\_2=16 Numar1 = “9999444499999999”  Numar2=”9999555599999999” | 1522138 | x | x |

# Varianta 2

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Timp de executie  (microsecunde) | | |
| 4 | 8 | 16 |
| Numar 1 = “123456789123456789” = Numar2 | 780302 | 12770015 | 41406667 |
| N\_1=1000 si N\_2=1000 (random digits) | 444031 | 17143754 | 28609561 |
| N\_1= 100 si N\_2=100000 (random digits) | 773151 | 18847756 | 30969758 |
| N\_1=N\_2=16 Numar1 = “9999444499999999”  Numar2=”9999555599999999” | 1211658 | x | x |