**Documentație Tema 3 Baciulescu Raul, gr. 236**

**Enunțul problemei:**

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 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”

**Detalii de implementare**

Am ales sa implementez varianta 2 de rezolvare a problemei. Mi-am declarat tot ce aveam nevoie in functia main,

apoi am folosit functiile MPI pentru a incepe lucrul cu procese (MPI\_Iinit, MPI\_Comm\_size, MPI\_Comm\_rank). In procesul de rank 0 am citit din fisier cele 2 numere mari sau am creat cei 2 vectori de numere mari folosind valori random. Am calculat un un nmax, care reprezinta dimensiunea maxima a celor 2 vectori, care mai are cateva pozitii aditionale in caz ca dimensiunea nu se imparte exact la numarul de procese. Dupa ce am calculat nmax, am completat vectorii cu 0-uri. Vectorii au fost cititi de la n pana la 0, pentru ca, calculul sa fie mai usor. De asemenea din procesul cu rank 0 am trimis nmax-ul la celelalte procese, pentru ca doar procesul 0 stie cat este nmax.

Dupa ce au primit toate procesele valoarea lui nmax, am inceput sa calculez vectorii de start si offset, necesari pentru functia MPI\_Scatterv. Acesti vectori(starts si offsets) sunt responsabili de pozitiile din vectorii initiali care trebuie procesate de fiecare proces. Vectorii aux\_a, aux\_b vor avea in ei pe anumite pozitii valorile din a si b, (de la starts[my\_rank] pana la starts[my\_rank] + offsets[my\_rank].

Dupa fiecare calcul se tine minte un carry care se aduna la urmatorul calcul.

Dupa fiecare for pe care procesul il are de procesat, trimite la procesul cu rank, my\_rank + 1, carry-ul corespunzator. Daca suntem la ultimul proces si primim carry, punem pe ultima pozitie din vector inca un 1.

Implementarea fara Scatterv este similara.

**Teste:**

* Functioneaza toate cazurile de testare

Varianta cu send si recv, fara MPI\_Scatterv

N1 = n2 = 18

A = b = 123456789123456789

Timp de executie:

p = 2 => 2.906

P = 4 => 1.54

P = 8 => 3.92

P = 16 => 7.52

N1 = n2 = 1000, vectorii au valori random

P = 2 => 0.6752

P = 4 => 1.8359

P = 8 => 3.0267

P = 16 => 7.8678

N1 = 100, N2 = 10000, vectorii au valori random

P = 2 => 0.9235

P = 4 => 2.2148

P = 8 => 4.5652

P = 16 => 7.1588

Varianta cu MPI\_Scatterv

N1 = n2 = 18

A = b = 123456789123456789

Timp de executie:

P = 2 => 0.6865

P = 4 => 1.8345

P = 8 => 4.5167

P = 16 => 7.2426

N1 = n2 = 1000, vectorii au valori random

P = 2 => 1.2625

P = 4 => 1.96

P = 8 => 3.6626

P = 16 => 8.0171

N1 = 100, N2 = 10000, vectorii au valori random

P = 2 => 27.1815

P = 4 => 27.2618

P = 8 => 29.8357

P = 16 => 33.7274

* In urma testelor se observa ca e mai rapid programul cand rulam pe 4 procese in ambele abordari.
* In urma testarii mele, varianta in care trimit doar send si recv este mai rapida decat cea cu Scatterv si Gatherv