Documentatie laboratorul 3

Nita Dan-Lucian 239

1) Cerinta 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 0: secvential

Varianta1: 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
  - citeste cate N/p cifre din cele 2 fisiere
  - le trimite procesului "id_proces_curent"
  - 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"
- a) procesele primesc carry inainte de a primi cifrele pe care trebuie sa le adune

Varianta2: – 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”

Implementare:

0) Pentru varianta secventiala, pur și simplu am citit numerele din fișier în doi vectori și le-am adunat.

1) Pentru varianta1 paralela, am citit câte n/p numere din fișier și le-am trimis procesului nr_curent folosind **MPI_Send**, incrementandu-l apoi. nr_curent începe de la 1, astfel ca suma propriu-zisă se face pe p-1 procesoare. La finalul citirii, dacă în ultimul batch nu am mai citit n/p numere, am trimis vectorul așa cum era spre ultimul procesor. Fiecare procesor ce calculează suma așteaptă restul de la procesorul precedent și vectorii de la procesorul 0, folosind **MPI_Recv** apoi calculează suma și trimite restul la următorul procesor. Procesorul 1 nu așteaptă după niciun rest, iar procesorul p-1 trimite restul la procesorul 0. Procesul 0 face apoi receive la vectorii rezultat și la restul de la ultimul procesor, folosind aceeași funcție de receive, apoi scrie rezultatul în fișier.

2) Pentru varianta2 paralela, procesorul 0 citește cele doua numere din fișiere. Calculam un total_size ce este inițial maximul dintre lungimea celor doua numere. Dacă un număr este mai mic, completam cu 0 pana ce lungimea acestuia este total_size. Calculam apoi câte zero-uri trebuie sa adăugăm celor doua numere, pentru ca total_size sa se împartă exact la p. Adaugam zero-uri celor doi vectori pana la noua valoare calculata. Calculam batch_size imparting total_size la p, is trimitem aceste doua valori tuturor procesoarelor folosind MPI_Send în procesorul 0 și MPI_Recv în celelalte procesoare. Din procesorul 0, împărțim cei doi vectori celorlalte procesoare folosind MPI_Scatter. Fiecare procesor așteaptă rest, calculează suma, și trimite suma următorului procesor. La final, facem MPI_Gather pentru a reconstrui vectorul rezultat și scriem rezultatul în fișier.

Pentru toate cele trei implementare, pentru a nu adauga complexitate în plus, majoritatea calculelor s-au făcut în funcția main. Pentru implementarea secventiala, ne-am folosit de o funcție pentru scrierea numerelor aleatorii în fișier.

```
void generate_random_numbers(string file, int size){
    /*
     * generate :size: random numbers and write them into :file:
     */
    srand(time(0));
    fout.open( s: file);
    if (!fout.is_open()){
        throw exception();
    }
    for (int i=0; i<size; i++){
        fout << rand() % 10 << " ";
    }
}
```

Timpi de executie (microsecunde)

Processor: Apple M2

Exemplu trivial 2 thread-uri		
Rulare numarul	Tip rulare	Timp
1	secvential	0
1	paralel1	6
1	Paralel2	4
2	secvential	0
2	paralel1	7
2	Paralel2	4
3	secvential	0
3	paralel1	15
3	Paralel2	2
4	secvential	0
4	paralel1	7
4	Paralel2	0
5	secvential	0

5	paralel1	6
5	Paralel2	2
6	secvential	0
6	paralel1	6
6	Paralel2	5
7	secvential	0
7	paralel1	6
7	Paralel2	2
8	secvential	0
8	paralel1	6
8	Paralel2	6
9	secvential	0
9	paralel1	6
9	Paralel2	5
10	secvential	0
10	paralel1	7
10	Paralel2	5
Exemplu mediu 2 thread-uri		
1	secvential	16
1	paralel1	206
1	Paralel2	25
2	secvential	22
2	paralel1	208
2	Paralel2	55
3	secvential	18
3	paralel1	223
3	Paralel2	28
4	secvential	12
4	paralel1	225
4	Paralel2	26
5	secvential	14
5	paralel1	221

5	Paralel2	29
6	secvential	20
6	paralel1	212
6	Paralel2	27
7	secvential	19
7	paralel1	211
7	Paralel2	26
8	secvential	22
8	paralel1	208
8	Paralel2	29
9	secvential	22
9	paralel1	206
9	Paralel2	39
10	secvential	21
10	paralel1	208
10	Paralel2	24
Exemplu mare 2 thread-uri		
1	secvential	948
1	paralel1	1220
1	Paralel2	1950
2	secvential	1409
2	paralel1	1263
2	Paralel2	2411
3	secvential	1090
3	paralel1	1217
3	Paralel2	1650
4	secvential	1237
4	paralel1	1240
4	Paralel2	1643
5	secvential	1393
5	paralel1	1250
5	Paralel2	1640

6	secvential	1046
6	paralel1	1350
6	Paralel2	1700
7	secvential	1412
7	paralel1	1216
7	Paralel2	1450
8	secvential	1251
8	paralel1	1372
8	Paralel2	1600
9	secvential	1245
9	paralel1	1350
9	Paralel2	1600
10	secvential	1344
10	paralel1	1240
10	Paralel2	1400
Exemplu mediu 4 thread-uri		
1	Send/Recieve	208
1	Scatter/Gather	36
2	Send/Recieve	222
2	Scatter/Gather	33
3	Send/Recieve	207
3	Scatter/Gather	34
4	Send/Recieve	207
4	Scatter/Gather	51
5	Send/Recieve	295
5	Scatter/Gather	64
6	Send/Recieve	212
6	Scatter/Gather	35
7	Send/Recieve	212
7	Scatter/Gather	85
8	Send/Recieve	213
8	Scatter/Gather	82

9	Send/Recieve	206
9	Scatter/Gather	38
10	Send/Recieve	207
10	Scatter/Gather	67
Exemplu mare 4 thread-uri		
1	Send/Recieve	976
1	Scatter/Gather	650
2	Send/Recieve	875
2	Scatter/Gather	390
3	Send/Recieve	896
3	Scatter/Gather	1388
4	Send/Recieve	945
4	Scatter/Gather	1540
5	Send/Recieve	1143
5	Scatter/Gather	1431
6	Send/Recieve	1241
6	Scatter/Gather	1630
7	Send/Recieve	1668
7	Scatter/Gather	710
8	Send/Recieve	980
8	Scatter/Gather	415
9	Send/Recieve	1214
9	Scatter/Gather	750
10	Send/Recieve	913
10	Scatter/Gather	739
Exemplu mediu 6 thread-uri		
1	Send/Recieve	-
1	Scatter/Gather	-
2	Send/Recieve	-
2	Scatter/Gather	-
3	Send/Recieve	-
3	Scatter/Gather	-

4	Send/Recieve	-
4	Scatter/Gather	-
5	Send/Recieve	-
5	Scatter/Gather	-
6	Send/Recieve	-
6	Scatter/Gather	-
7	Send/Recieve	-
7	Scatter/Gather	-
8	Send/Recieve	-
8	Scatter/Gather	-
9	Send/Recieve	-
9	Scatter/Gather	-
10	Send/Recieve	-
10	Scatter/Gather	-
-		
1	Send/Recieve	-
1	Scatter/Gather	-
2	Send/Recieve	-
2	Scatter/Gather	-
3	Send/Recieve	-
3	Scatter/Gather	-
4	Send/Recieve	-
4	Scatter/Gather	-
5	Send/Recieve	-
5	Scatter/Gather	-
6	Send/Recieve	-
6	Scatter/Gather	-
7	Send/Recieve	-
7	Scatter/Gather	-
8	Send/Recieve	-
8	Scatter/Gather	-
9	Send/Recieve	-

9	Scatter/Gather	-
10	Send/Recieve	-
10	Scatter/Gather	-