Laborator 3

*Documentație*

*Enunț*

Scrieți un program bazat pe MPI care face suma a 2 numere mari (cu mai mult de 10 cifre). Cele 2 numere mari se citesc din fișierele “Număr1.txt” (un număr cu *noDigits1*) si “Număr2.txt” (un numar cu *noDigits2* cifre).

Fiecare din aceste fișiere conține la început un număr (N) care reprezintă numărul de cifre și apoi cifrele numărului respectiv.

*Abordare*

Se folosesc câte 3 funcții diferite, *secvențial()*, *varianta1()* și *varianta2()* pentru implementare. Toate primesc ca parametri *argc, argv, file1, file2, file3* și *fileCheck*, file1 și file2 conțin numerele de intrare, file3 este fișierul output, iar fileCheck este fișierul de verificare (generat de *secvențial()* ).

Secvențial

Cifrele numerelor sunt citite în doi vectori *n1*, respectiv *n2*. Un al treilea vector *n3* este inițializat cu dimensiunea maximă dintre *n1* și *n2*. Se parcurg concomitent cifrele celor doi vectori *n1* și *n2*, iar suma lor este memorată în *n3*. La final, cifrele din *n3* sunt salvate într-un fișier care este verificat cu fișierul de test *fileCheck*.

Varianta 1

Procesul 0 citește câte *lenMaxx*  = *maxxDigits* / *numprocs* din cele două fișiere, unde *maxxDigits* este maximul dintre *noDigits1* și *noDigits2*, iar *numprocs* este numărul de procese date, și le trimite prin *MPI\_Send()* celorlalte procese. Dacă *noDigits1 != noDigits2*, fișierul cu mai puține cifre va fi completat în vectorul său cu valori de 0 pentru o împărțire echitabilă a cifrelor între procese. Toate procesele primesc cifrele de la procesul 0, precum și un un *carry* (în afară de procesul 1), care reprezintă trecerea peste ordin, apoi calculează suma dinter vectorii proprii și trimit procesului următor *carry­*-ul obținut. Apoi, trimit procesului 0 vectorul sumă *n3* calculat.

Procesul 0 primește prin *MPI\_Recv()* întreaga componență a *n3*, o scrie în fișier și verifică egalitatea cu *fileCheck*.

Varianta 2

Procesul 0 citește întreaga componență a fișierelor *file1* și *file2*, păstrând regula completării cu valori de 0. Le împarte tuturor proceselor prin *MPI\_Scatter()*. După ce fiecare proces primește cifrele și *carry*-ul de la procesul anterior (cu excepția procesului 0), calculează suma cerută. *N3* este apoi ”strâns” în procesul 0 prin funcția *MPI\_Gather()*, proces care scrie cifrele în fișierul *file3* și verifică corectitudinea datelor obținute.

|  |  |  |
| --- | --- | --- |
| **Tip metodă** | **Nr procese** | **Timp executie** |
| secvențial | - | 1.9119 |
| Varianta 1 |  |  |
| 1 | 12.4792 |
| 2 | 14.4101 |
| 4 | 21.0311 |
| 8 | 40.0824 |
| 16 | 82.6644 |
| Varianta 2 |  |  |
| 1 | 12.2818 |
| 2 | 13.8224 |
| 4 | 23.3463 |
| 8 | 36.5981 |
| 16 | 80.3548 |

La final se afișează în consolă timpul de execuție măsurat anterior. Timpii obținuți în funcție de *no­\_threads* sunt:

|  |  |  |
| --- | --- | --- |
| **Tip metodă** | **Nr procese** | **Timp executie** |
| secvențial | - | 10.0644 |
| Varianta 1 |  |  |
| 1 | 16.3826 |
| 2 | 21.7771 |
| 4 | 34.7831 |
| 8 | 62.0099 |
| 16 | 184.537 |
| Varianta 2 |  |  |
| 1 | 23.0512 |
| 2 | 29.9767 |
| 4 | 36.6687 |
| 8 | 71.0544 |
| 16 | 171.353 |

*Fișierul 1 (noDigits1 = noDigits2 = 18) Fișierul 2 (noDigits1 = noDigits2 = 1000)*

|  |  |  |
| --- | --- | --- |
| **Tip metodă** | **Nr procese** | **Timp executie** |
| secvențial | - | 629.621 |
| Varianta 1 |  |  |
| 1 | 589.936 |
| 2 | 588.348 |
| 4 | 608.729 |
| 8 | 633.05 |
| 16 | 774.615 |
| Varianta 2 |  |  |
| 1 | 605.906 |
| 2 | 592.387 |
| 4 | 610.23 |
| 8 | 6628.50 |
| 16 | 768.253 |

*Fișierul 3 (noDigits1 = 100, noDigits2 = 100000)*

*Observații*

Pentru valori mici ale *noDigits1* și *noDigits2* se observă timpi mai buni obținuți în cadrul rulării secvențiale. Pentru valori mari (*noDigits2 = 100000*), timpii obținuți folosind procese pot fi mai mici.

În general, folosirea variantei 2, cu *MPI\_Scatter* / *MPI\_Gather*, deși mai ușoară de folosit, generează timpi ușor mai slabi decât cei obținuți în varianta 1, cu *MPI\_Send* și *MPI\_Recv*.

În cazul folosirii proceselor, se poate observa că folosind un număr relativ mic de procese (până la 4) generează timpi mai buni decât utilizarea unui număr exagerat de procese.