Laborator 3 – PPD

Analiza cerintelor

Scrieti un program bazat pe MPI care face suma a 2 numere. Numerele vor fi reprezentate cu ajutorul unor tablouri de cifre (numere intregi fara semn – byte), in care cifra cea mai nesemnificativa este pe prima pozitie.

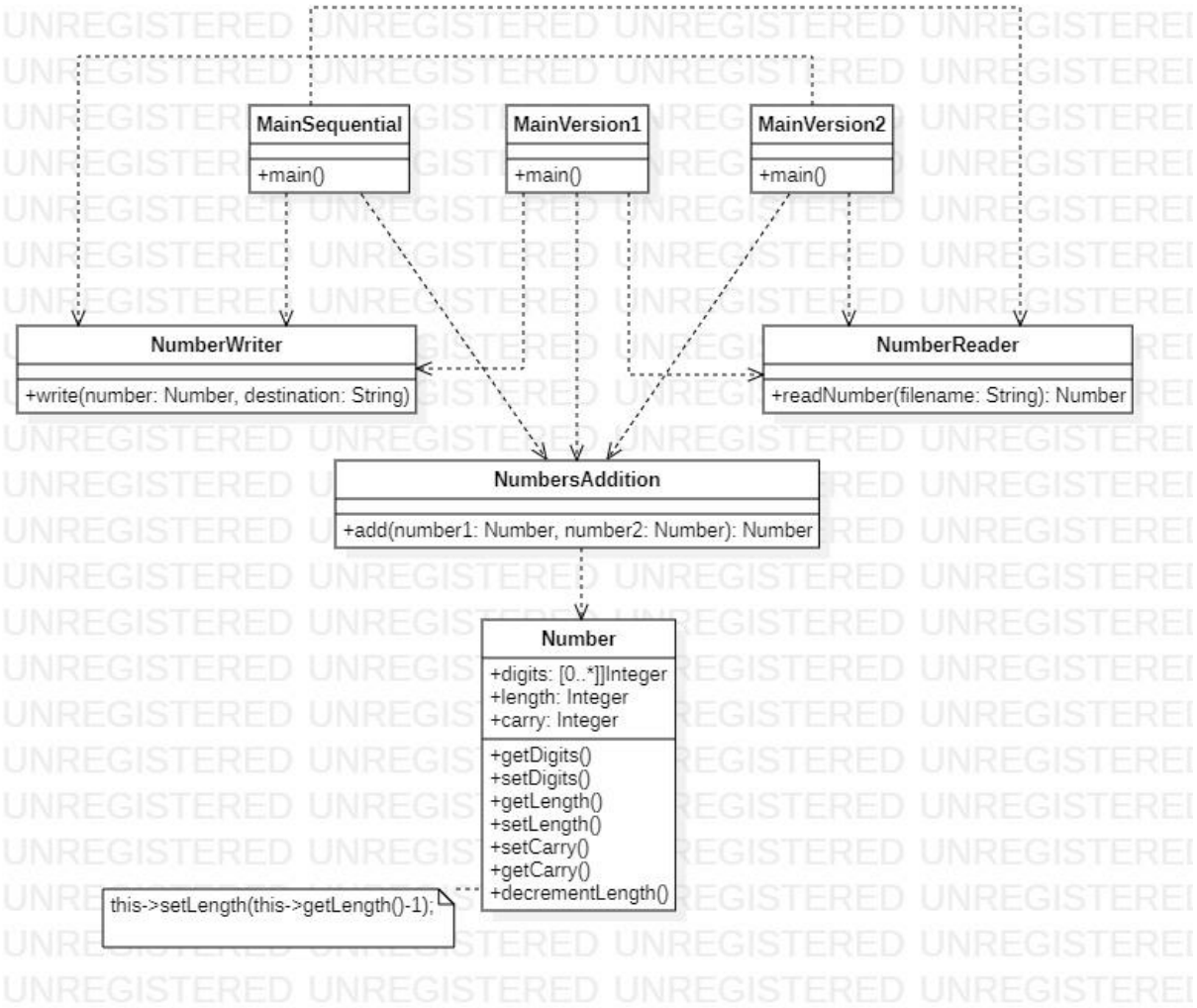
Numerele vor fi citite din fisierele "Numar1.txt", cu N1 cifre si "Numar2.txt", cu N2 cifre. Fisierul va avea pe prima linia numarul de cifra, iar pe urmatoarea linie, numarul efectiv.

Implementati 3 variante:

- Secventiala
- Folosind MPI cu MPI_Send si MPI_Receive
- Folosing MPI cu MPI_Scatter si MPI_Gather

Proiectare

Diagrama ce prezinta clasele folosite in implementare, impreuna cu metodele lor si relatiile dintre ele.



Clasa `Number` ofera o abstractizare a notiunii de numar mare, avand ca si atribute un tablou unidimensional de cifre si numarul de cifre (in campul `length`). Atributul `carry` indica faptul daca in urma ultimei operatii de adunare, in care instanta curenta este si rezultat, a crescut numarul de cifre. Mai precis,

$carry = 1 \leftrightarrow length = \max\{N_1, N_2\} + 1$.

Clasa `NumbersAddition` ofera o implementare a algoritmului de adunare a doua numere. Algoritmul ia amandoua numerele, cifra cu cifra si contruieste intr-un alt numar rezultatul, tinand cont si de transporturile (carry-urile) de care e nevoie. Algoritmul seteaza campul `carry` corespunzator rezultatului la final.

Clasele NumberWriter si NumberReader sunt responsabile cu scrierea, respectiv citirea, din fisier a diferitelor numere, necesare de-a lungul executiei programului.

Detalii de implementare

Toate cele 3 variante implementate citesc mai intai cele 2 numere din fisierele corespunzatoare si sunt construite instante ale clasei `Number`. Fiecare tablou de cifre va fi inversat, in sensul ca cifra cea mai putin semnificativa este pe prima pozitie.

La finalul executiei programului, fiecare varianta scrie in fisier, folosind o instanta a clasei `NumberWriter`, numarul rezultat. Metoda `write` se asigura ca in fisier nu vor fi scrise cifre 0, care se afla la inceputul numarului.

Daca varianta secventiala doar apeleaza apoi algoritmul de `add`, celelalte 2 variante completeaza, in procesul master (cel cu rang 0), numerele cu cifre 0, astfel inca tau dimensiunea egala si lungimea lor e divizibila cu numarul de procese care ruleaza.

Apoi, in varianta 1, procesul 0 imparte in mod echitabil bucatile din cele 2 numere pe care fiecare proces trebuie sa le adune, iar la final, reuneste rezultatele. Celelalte procese aduna cifrele si trimit transportul (carry-ul) procesului urmator (cu exceptia celui de rang 1, care considera carry-ul egal cu 0) si isi actualizeaza rezultatul. Procesul de cel mai inalt rang ii va trimite procesului de rang 0 toate cifrele rezultat, inclusiv daca a avut loc un transport final in care a crescut numarul de cifre.

In varianta 2, pe de alta parte, folosind `MPI_Scatter`, fiecare proces (inclusiv cel de rang 0) primeste o bucata din cele 2 numere si este responsabil de adunarea lor. Trimiterea carry-ului este similara cu varianta 1. Apoi, cu `MPI_Gather`, sunt reunite rezultatele, iar procesul 0 le scrie in fisier.

Cazuri de testare

Numar1	**Numar2**	**Rezultat asteptat**
123456789	123456789	246913578
999	123	1122
1000	600	1600
45819	0	45819
99999	99999	199998

Rezultate

Tip numar	**Numar procese**		**Timp executie (ms)**
Numar1 = Numar2 = “123456789”	Secvential		1.8081900000000002
	Varianta 1	4	49.437990000000006
		8	34.34774
	Varianta 2	4	29.933429999999998
		8	33.20545

N1=1000 N2=1000 (random digits)	Secvential		6.32582
	Varianta 1	4	33.72329
		8	39.981989999999996
	Varianta 2	4	30.868990000000004
		8	45.85866
N1=100 N2=100000 (random digits)	Secvential		353.927
	Varianta 1	4	371.2676
		8	411.18549999999993
	Varianta 2	4	430.8652
		8	457.4543

Nota: fiecare test a fost rulat de 10 ori si pentru evaluarea timpului de executie s-a considerat media aritmetica a acestor 10 rulari. Fiecare test a fost rulat pe o masina cu Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 MHz 2 cores 4 Logical Processors cu sistem de operare Windows 10 versiunea 21H1.

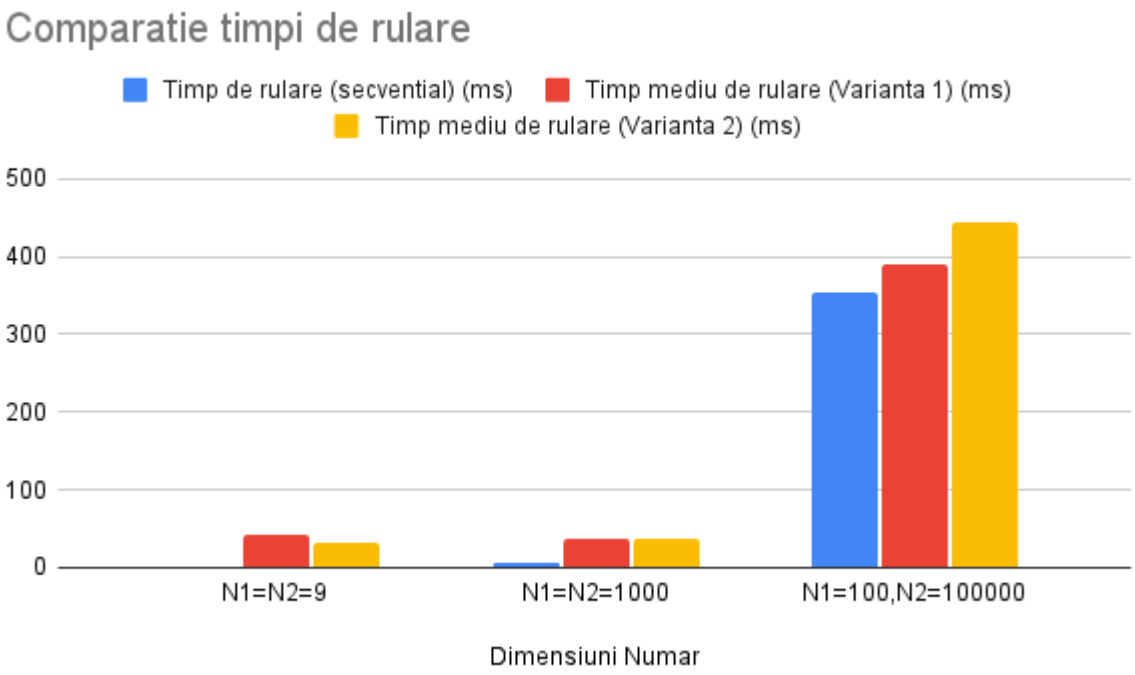
Analiza rezultatelor

Deoarece masina pe care au fost rulate testele are 2 core-uri (respective, 4 procesoare logice), cei mai buni timpi de rulare se obtin daca sunt folosite 4 procese (sau chiar 2), nu 8. Acest lucru este motivat de faptul ca devin mult mai costisitoare crearea si intretinerea proceselor, cu cat numarul lor devine mai mare.

Se observa faptul ca varianta secventiala se comporta mult mai bine pentru numere de dimensiune mica (ex. primul caz de testare). Pentru restul cazurilor de testare, timpii obtinuti sunt comparabili.

De asemenea, se observa ca in general, variantele care folosesc `MPI_Scatter` si `MPI_Gather` sunt mai performante decat cele care folosesc direct `MPI_Send` si `MPI_Receive`. Un posibil motiv este faptul ca sunt mai puternic optimizate aceste functii, combinand in cadrul aceluiasi apel de functie atat functionalitatea lui `MPI_Send`, cat si cea a lui `MPI_Receive`.

Dimensiuni Numar	Timp de rulare (secvential)	Timp mediu de rulare (Varianta 1)	Timp mediu de rulare (Varianta 2)
N1=N2=9	1.80819	41.892865	31.56944
N1=N2=1000	6.32582	36.85264	38.363825
N1=100,N2=100000	353.927	391.22655	444.15975


Comparatie timpi de rulare
Timp de rulare (secvential) (ms)
Timp mediu de rulare (Varianta 1) (ms)
Timp mediu de rulare (Varianta 2) (ms)
500
400
300
200
100
0
N1=N2=9
N1=N2=1000
N1=100,N2=100000
Dimensiuni Numar