**Documentatie Lab3 PPD**

**Cerinta:**

**Text

Description automatically generated**

**Text, application

Description automatically generated**

**Proiectare**

Chart, box and whisker chart

Description automatically generated

**Implementare:**

Varianta1:

* Impart datele in niste chunck-uri inegale(folosind „restul”), umplu cu 0 numarul mai mic
* Trimit aceste chunck-uri la procese cu MPI\_SEND (si numerele si lungimea chunck-ului)
* Fiecare proces != 1 asteapta cf procesului cu rank-1 fata de al sau
* Apoi fiecare proces care primeste chunck-ul si cf-ul face suma si trimite cf-ul sau la procesul rank+1, dupa care trimite la procesul 0 chunck-ul de date calculat
* In procesul 0 se primeste cu MPI\_RECV chunck-urile si se concateneaza, dupa care se scrie in fisier

Varianta2:

* Impart datele in niste chunck-uri egale(folosind „restul”), umplu cu 0 numarul mai mic(ca sa le fac egale mai adaug zerouri)
* Dau MPI\_SCATTER la cei 2 vectori care contin numerele
* Calculul datelori si cf urmeaza aceeasi logica ca la varianta 1
* Dupa calcularea datelor le „adun” cu MPI\_GATHER in procesul 0, unde le voi afisa

Varianta3:

* Citirea e la fel din fisier si alocarea chunck-urilor ca in varianta 1
* Procesul 0 trimite la fiecare proces cu MPI\_ISEND si cu MPI\_SEND dimensiunea chunck-ului
* Celelalte procese primesc lungimea cu MPI\_RECV si cu MPI\_IRECV datele din chunck
* La procesul 1 astept sa primesc prin MPI\_IRECV datele calculez vectorul rezultat si trimit la procesul rank+1 prin MPI\_ISEND cf
* Restul proceselor primesc cu MPI\_IRECV si cf si vectorii de date; verific daca s-a primit prima oara cf, caz in care calculez vectorul rezultat folosind cf-ul primit; daca nu am primit inca cf, verific daca am primit vectori, caz in care fac calculul fara cf, dupa care astept sa primesc cf si il adaug
* Dupa ce calculeaza vectorul rezultat fiecare proces trimite la procesul 0 cu MPI\_ISEND acest vector
* Procesul 0 primeste apoi fiecare vector prin MPI\_IRECV de la fiecare proces, trecand in vectorul finalul chunck-ul primit cand este primit, iar daca nu e primit asteapta sa il primeasca.
* Apoi scrie in fisier vectorul final

**C++:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip** | **Varianta** | **Nr.** | **Timp Executie(milisecunde)** |
| **n1=n2**  **n = 18**  **m=18** | **1** | **4** | **17.5** |
| **8** | **18.1** |
| **2** | **4** | **17.9** |
| **8** | **18.3** |
| **3** | **4** | **19.3** |
| **8** | **18.7** |
| **N=1000**  **M=1000** | **1** | **4** | **32.1** |
| **8** | **31.7** |
| **2** | **4** | **25.9** |
| **8** | **33.2** |
| **3** | **4** | **29.8** |
| **8** | **32.6** |
| **N=100**  **M=100000** | **1** | **4** | **1112.3** |
| **8** | **1205.7** |
| **2** | **4** | **687.1** |
| **8** | **771.8** |
| **3** | **4** | **1045.6** |
| **8** | **1195.2** |

**Observatii:**

* **Varianta 2 este cea mai rapida de obicei.**
* **In general cu 4 procese merge mai rapid decat cu 8 procese, diferenta devenind destul de mare cu cat cantitatea de date creste**
* **Se intampla totusi, uneori ca 8 procese sa fie mai rapide ca 4**