Mihai Cristina Florentina
CR3.2B

PROIECT

Cerintele/tema proiectului:

- ***Topicul***: Sortarea unui vector : Quick Sort.
- ***Limbajul de programare utilizat:*** C++.
- ***Sistemele si/sau framework-urile folosite:*** MS-MPI.

Informatii despre masinia pe care am rulat codul:

- Nume dispozitiv: LAPTOP-80KCL35B
- Procesor: 13th Gen Intel(R) Core(TM) i7-13620H 2.40 GHz
- RAM instalat: 16.0 GB
- Tip sistem: Sistem de operare pe 64 de biți, procesor de tip x64
- Ediție: Windows 11 Home
- Versiune: 24H2
- Versiune SO: 26100.3476

VARIANTA SECVENTIALA

Rezultate experimentale:

→ Rulare 1 → 10 000 elmente generate random ↓

```
Timpul de rulare: 1 milisecunde
```

→ Rulare 2 → 50 000 elmente generate random ↓

```
Timpul de rulare: 7 milisecunde
```

→ Rulare 3 → 100 000 elmente generate random ↓

```
Timpul de rulare: 16 milisecunde
```

→ Rulare 4 → 500 000 elmente generate random ↓

```
Timpul de rulare: 106 milisecunde
```

→ Rulare 5 → 1 000 000 elmente generate random ↓

```
Timpul de rulare: 246 milisecunde
```

→ Rulare 6 → 5 000 000 elmente generate random ↓

```
Timpul de rulare: 2648 milisecunde
```

→ Rulare 7 → 10 000 000 elmente generate random ↓

```
Timpul de rulare: 8773 milisecunde
```

→ Rulare 8 → 25 000 000 elmente generate random ↓

```
Timpul de rulare: 33233 milisecunde
```

→ Rulare 9 → 50 000 000 elmente generate random ↓ (3 minute)

```
Timpul de rulare: 184173 milisecunde
```

VARIANTA I PARALELA

Scopul programului

Să sorteze eficient un șir de numere întregi citit dintr-un fișier folosind procesarea paralelă distribuită pe mai multe procese MPI.

Etapele principale:

- Citirea datelor (doar în procesul cu rank 0):

- Fișierul de intrare este citit, iar numerele sunt salvate într-un vector.
- Dacă numărul de elemente nu este divizibil cu numărul de procese, vectorul este completat cu INT_MAX pentru a asigura împărțirea uniformă.

➢ **Distribuirea datelor:**

- Dimensiunea totală este transmisă tuturor proceselor prin MPI_Bcast.
- Datele sunt împărțite în blocuri egale și trimise fiecărui proces cu MPI_Scatter.

➢ **Sortarea locală:**

- Fiecare proces sortează porțiunea proprie de date folosind QuickSort.

➢ **Colectarea și sortarea finală:**

- Datele sortate local sunt reunite în procesul 0 cu MPI_Gather.
- Procesul 0 aplică din nou QuickSort pentru a asigura ordonarea globală (deoarece Gather doar concatenează segmentele sortate).

➢ **Scrierea rezultatului și măsurarea timpului:**

- Se salvează fișierul sortat și se afișează durata execuției în milisecunde.

Rezultate experimentale:

→ Rulare 1 → 10 000 elmente generate random ↓

→ Rulare 2 → 50 000 elmente generate random ↓

→ Rulare 3 → 100 000 elmente generate random ↓

Mihai Cristina Florentina
CR3.2B

```
Timpul total : 1462 milisecunde
```

→ Rulare 4 → 500 000 elmente generate random ↓

```
Timpul total : 24429 milisecunde
```

→ Rulare 5 → 1 000 000 elmente generate random ↓

```
Timpul total : 124499 milisecunde
```