**Merge sort**

Temă realizată de Popescu David, Grupa 342 A3

**Structura proiectului**

* Generate\_array.cpp –generează in mod aleatoriu array-ul care va fi folosit ulterior pentru sortare paralelă
* Merge\_openmp.cpp – sortează in ordine crescătoare array-ul generat, folosind OpenMP , iar rezultatele pentru 1,2,4,8 procese sunt salvate in fișierul time\_openmp.txt
* merge\_mpi.cpp - sortează în ordine crescătoare un array folosind MPI. Implementarea MPI permite paralelizarea sortării pe mai multe procese. Rezultatele pentru 1, 2, 4, 8 procese MPI sunt salvate în fișierul time\_mpi.txt.
* merge\_pthreads.cpp - sortează în ordine crescătoare un array folosind pthreads . Rezultatele pentru 1, 2, 4, 8 fire de execuție pthreads sunt salvate în fișierul time\_pthreads.txt.
* merge\_hybrid.cpp - Acest fișier combină abordările OpenMP și MPI pentru a realiza sortarea în ordine crescătoare. Se folosesc ambele tehnologii pentru o abordare hibridă. Rezultatele pentru 1, 2, 4, 8 procese MPI și 1, 2, 4, 8 fire de execuție OpenMP sunt salvate în fișierul time\_hybrid.txt.
* plot.py - utilizează biblioteca matplotlib pentru a crea trei grafice separate: unul pentru timpul de execuție, unul pentru speed-up și unul pentru eficiență. Datele sunt citite din fișierele corespunzătoare implementărilor MPI, OpenMP, pthreads și hibrid.

Pentru a rula programul, am făcut un Makefile, prin intermediul căruia putem compila, rula si șterge foarte ușor prin intermediul următoarelor comenzi:

-make all generează executabilul

-make run\_{algorithm} rulează executabilul și execută sortarea cu metoda aleasă

-make clean șterge executabilele

**Descrierea temei**

Fiecare implementare (MPI, OpenMP, pthreads, hibrid) sortează un array în ordine crescătoare.

Timpurile de execuție pentru fiecare configurație (1, 2, 4, 8 fire de execuție sau procese) sunt salvate în fișiere separate (time\_mpi.txt, time\_openmp.txt, time\_pthreads.txt, time\_hybrid.txt).

Scriptul plot.py calculează speed-up-ul și eficiența pentru fiecare implementare.

Acesta generează trei grafice:

* + Timpul de execuție în funcție de numărul de fire de execuție sau procese.
  + Speed-up-ul în funcție de numărul de fire de execuție sau procese.
  + Eficiența în funcție de numărul de fire de execuție sau procese.

**Merge Sort OpenMP:**

Algoritmul merge sort în implementarea OpenMP își optimizează timpul de execuție prin împărțirea eficientă a sarcinilor pe fire pe mai multe proces. Prin utilizarea directivei #pragma omp task, fiecare buclă importantă a algoritmului a fost paralelizata, astfel incat am obtinut rularea in paralel a mai multe procese.

**Merge Sort MPI:**

În implementarea MPI, fiecare proces devine un executor independent, contribuind la sortarea array-ului. Comunicarea eficientă între aceste procese eficientizează rularea. MPI\_Gather și MPI\_Bcast asigură coordonarea necesară.

**Merge Sort Pthreads:**

Implementarea cu Pthreads folosește fire de execuție pentru a accelera procesul de sortare. Fiecare fir de execuție se ocupă independent de o parte a array-ului, aducând o abordare multitasking la sortare.

**Merge Sort Hibrid:**

Abordarea hibridă OpenMP-MPI aduce o sinergie între cele două tehnologii, capitalizând beneficiile fiecăreia. MPI se ocupă de distribuția array-ului între procese, iar OpenMP adaugă o paralelizare la nivel de fire de execuție în cadrul fiecărui proces.