Analiza performanței pentru diferite paralelizări

Algoritmul Cocktail Sort

Filișanu Mihai-Alexandru

Grupa 341 C3

Am implementat 5 modalitati diferite de paralelizare a algoritmului de sortare Cocktail. Pentru varianta seriala am folosit codul sursa de la: Cocktail Sort, pe care l-am modificat astfel încât limbajul sa fie C.

Algoritmul se rezuma la urmatorii pasi:

1. Forward pass: se itereaza de la stanga la dreapta si, la fel ca la Bubble sort, se inverseaza elementele prin simpla lor comparare

2. Backward pass: se itereaza de la dreata la stanga si se inverseaza elementele incepand cu primul nesortat.

Pasii se repeta pana la obtinerea array-ului sortat.

Complexitatea algoritmului este O(n) pentru best case scenario si O(n^2) in cazul average.

Timpul de rulare pentru varianta seriala este de 22s.

Resursele folosite pentru rulare: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz; 16 GB RAM DDR4.

Ca date de test, am folosit un array de 100000 de elemente pe care am aplicat algoritmul de shuffle.

Interpretarea rezultatelor poate fi făcută luând în considerare timpul total de execuție obținut pentru fiecare variantă de paralelizare. În cazurile pe care le-am testat, o valoare mai mică a timpului de execuție indică o performanță mai bună, mai ales dat fiind faptul că rezultatul a fost întotdeauna satisfăcător:

1. **Serial (./cocktailsort-serial):**
    - Timpul de execuție este de 22.951655 secunde.
    - Acesta este timpul de bază pentru algoritmul Cocktail Sort fără paralelizare. Celelalte implementări ar trebui să îmbunătățească acest timp.
2. **OpenMP (./cocktailsort-openmp):**
    - Timpul de execuție este de 2.666267 secunde.
    - Comparativ cu versiunea serială, implementarea OpenMP reduce semnificativ timpul de execuție. Totuși, performanța depinde de caracteristicile procesorului folosit. Este posibil ca pentru un processor cu 16 core-uri rezultatele să fie mult mai bune.
3. **MPI (mpiexec -n 4 ./cocktailsort-mpi):**
    - Timpul de execuție este de 1.734400 secunde.
    - Implementarea MPI folosește un model de calcul distribuit, iar timpul mai mic de execuție indică beneficiile aduse de paralelizarea distribuită pe mai multe noduri.
4. **Pthreads (./cocktailsort-pthreads):**
    - Timpul de execuție este de 1.785456 secunde.
    - Implementarea cu Pthreads utilizează fire de execuție (thread-uri) pentru a îmbunătăți performanța. Acest timp de execuție arată cât de bine se scalabilizează programul pe mai multe fire de execuție.
5. **MPI + OpenMP (mpiexec -n 4 ./cocktailsort-mpi-openmp):**
    - Timpul de execuție este de 0.881302 secunde.

- Această implementare combină MPI și OpenMP pentru a beneficia atât de paralelizarea distribuită, cât și de paralelizarea în cadrul fiecărui nod. Timpul redus arată eficiența combinației acestor paradigme.

6. **MPI + Pthreads (mpiexec -n 4 ./cocktailsort-mpi-pthreads):**
    - Timpul de execuție este de 0.392504 secunde.
    - Acesta este cel mai mic timp de execuție dintre toate variantele. Combinarea MPI și Pthreads oferă beneficiile atât ale paralelizării distribuite, cât și ale celei bazate pe fire de execuție.

În concluzie, implementările cu MPI și combinațiile MPI+OpenMP și MPI+Pthreads par să ofere cele mai bune performanțe, iar alegerea între ele poate depinde de resursele hardware disponibile. Pentru mine, varianta MPI+Pthreads este cea mai bună alegere.