Algoritmi paraleli si distribuiti
Merge Sort-Proiect

Nedelcu Robert-Ionut CR3.2B

Introducere

Acest proiect va face o comparatie intre datele de iesire ale algoritmul de sortare Merge Sort varianta secventiala vs varianta paralela in limbajul de programare C# pe mai multe seturi de date de diferite dimensiuni si se va crea o tabela in care va rezulta timpul de executie pentru fiecare set de date ale fiecarei variante.

Merge Sort - Varianta Secventiala

Merge Sort este un algoritm de sortare eficient, bazat pe metoda "Divide et Impera". Acesta împarte lista inițială în bucăți mai mici, sortează fiecare bucată și apoi le combină pentru a obține lista finală sortată. Implementarea secvențială a algoritmului Merge Sort se desfășoară într-un singur fir de executie și constă în următorii pași:

1. **Divizare**: Lista inițială este divizată în mod recursiv în bucăți mai mici până când fiecare bucată conține un singur element.
2. **Sortare**: Bucățile individuale sunt sortate în mod separat.
3. **Combinație**: Bucățile sortate sunt combinate repetat pentru a obține liste sortate mai mari, până când lista inițială este reconstituită complet și sortată.

Implementarea secvențială a Merge Sort are o complexitate medie de caz de O(n log n), unde n reprezintă numărul de elemente din lista inițială.

Pro

Simplu de implementat: Varianta secvențială a algoritmului Merge Sort este relativ simplă și ușor de implementat, deoarece nu implică coordonarea și sincronizarea între procese sau fire de execuție paralele.

Utilizare eficientă a resurselor: Varianta secvențială poate fi mai eficientă în utilizarea resurselor sistemului, cum ar fi memoria și puterea de procesare, deoarece nu implică crearea și gestionarea mai multor procese sau fire de execuție în paralel.

Rezistență la probleme de concurență: În implementarea secvențială, nu există probleme de concurență, cum ar fi conflicte de date sau condiții de cursă, care pot apărea în varianta paralelă.

Contra

Performanță limitată pe seturi de date mari: Varianta secvențială poate avea o performanță mai slabă pe seturi de date mari, deoarece procesarea este realizată într-un singur fir de execuție, fără a putea beneficia de paralelismul pentru a accelera sortarea.

Timp de execuție mai lung: Din cauza lipsei de paralelism, varianta secvențială poate avea un timp de execuție mai lung în comparație cu varianta paralelă pe seturi de date semnificative.

Subutilizarea resurselor disponibile: Dacă sistemul dispune de mai multe nuclee sau fire de execuție, varianta secvențială nu va profita de potențialul maxim al resurselor disponibile și nu va utiliza eficient puterea de procesare paralelă a sistemului.

Merge Sort - Varianta Paralela

Merge Sort în varianta paralelă utilizează mai multe fire de execuție (sau procese) pentru a accelera procesul de sortare. Algoritmul poate fi paralelizat în două etape principale:

4. **Divizare paralelă**: Lista inițială este împărțită în bucăți mai mici într-un mod paralel. Fiecare fir de execuție (sau proces) primește o bucată separată de sortat.
5. **Combinație paralelă**: După ce fiecare bucată este sortată individual, bucățile sunt combinate folosind mai multe fire de execuție pentru a obține lista finală sortată.

Varianta paralelă a Merge Sort poate oferi o performanță mai bună decât varianta secvențială, mai ales pentru liste de dimensiuni mari. Cu toate acestea, implementarea paralelă introduce o complexitate suplimentară, deoarece implică sincronizarea și coordonarea între firele de execuție (sau procese).

Pro

Potențialul de scalabilitate: Varianta paralelă poate oferi o performanță mai bună și o scalabilitate mai mare în cazul seturilor de date foarte mari, în care avantajele paralelizării devin evidente.

Adaptabilitatea: Varianta paralelă poate fi adaptată mai ușor la sistemele distribuite și clusterelor de calcul, permițând procesarea eficientă a volumelor mari de date distribuite în rețea.

Contra

Overhead-ul de comunicare: Implementarea paralelă implică costuri suplimentare asociate comunicării și sincronizării între procese sau fire de execuție. Pentru seturi de date relativ mici, aceste costuri suplimentare pot depăși beneficiile obținute prin paralelizare, ceea ce duce la un timp de execuție mai mare în comparație cu varianta secvențială.

Dimensiunea setului de date: Pentru seturi de date de dimensiuni moderate sau mici, avantajele paralelizării pot fi limitate. În astfel de cazuri, costurile asociate cu paralelizarea pot depăși beneficiile obținute, iar varianta secvențială este mai eficientă.

Informatii configuratie masina

OS: Windows 10 Pro 64 Bit

CPU: AMD Ryzen 7 4800H 2.90GHz

RAM: 16 GB

GPU: NVIDIA RTX 3050 Laptop 4GB

Disk: INTEL SSD 512GB M2

Rezultate experimentale

Merge Sort - Varianta Secventiala

```
Logs:
File    Data Set        Duration (ms)
1       100     0
2       1000    4
3       10000   33
4       100000  1791
5       1000000 308103
```

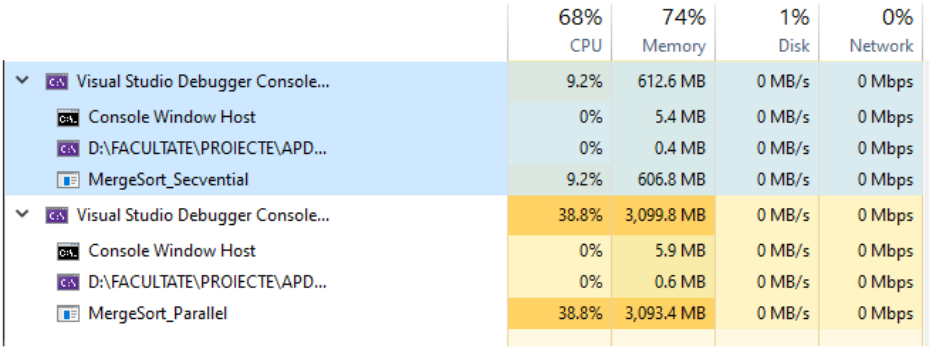
```
1           100       0
2           1000      0
3           10000     51
4           999999    546311
```

Merge Sort - Varianta Paralela

```
Logs:
File    Data Set        Duration (ms)
1       100     29
2       1000    37
3       10000   356
4       100000  5535
5       1000000 926893
```

```
1           100       34
2           1000      49
3           10000     456
4           999999    841385
```

Consum resurse



	68% CPU	74% Memory	1% Disk	0% Network
Visual Studio Debugger Console...	9.2%	612.6 MB	0 MB/s	0 Mbps
Console Window Host	0%	5.4 MB	0 MB/s	0 Mbps
D:\FACULTATE\PROIECTE\APD...	0%	0.4 MB	0 MB/s	0 Mbps
MergeSort_Secvential	9.2%	606.8 MB	0 MB/s	0 Mbps
Visual Studio Debugger Console...	38.8%	3,099.8 MB	0 MB/s	0 Mbps
Console Window Host	0%	5.9 MB	0 MB/s	0 Mbps
D:\FACULTATE\PROIECTE\APD...	0%	0.6 MB	0 MB/s	0 Mbps
MergeSort_Parallel	38.8%	3,093.4 MB	0 MB/s	0 Mbps

Concluzie

Pentru seturi de date relativ mici, costurile adiționale pot depăși beneficiile obținute din paralelizare, ceea ce poate duce la un timp de execuție mai mare în varianta paralelă în comparație cu varianta secvențială.