
# Laborator 05

## Exerciții

1. Pornind de la bubbleSort implementați **Odd-even Transposition Sort paralel**.
2. Verificați corectitudinea făcând **verificare inițială (sanity check)**, **stres test cu scriptul** și **măsurători scalabilitate**. În readme se va scrie “Exercițiul 2\n” urmat de rezultatele sanity check (N=10), urmat de rezultatele stress test (pozitiv sau negativ), urmat de timpul de execuție pentru rularea cu 1 thread, 2 thread-uri și 4 thread-uri.
3. Implementați **mergeSort paralel**, doar pentru **N** putere a lui 2 .
4. Idem 2.

**Exercițiile de la 1 la 4** sunt **obligatorii**. Conceptele explorate sunt esențiale pentru obținerea notei **minime** de promovare.

**Vă recomandăm, pentru a crește șansele de a obține o notă cât mai mare să explorați și următoarele exerciții:**

5. Implementați secvențial iar apoi paralelizați **Sheer Sort** . Prescursor acestui exercițiu recomandăm implementarea sortării unei linii și a unei coloane.
6. Idem 2.

**CITIȚI MAI JOS DOAR DACĂ VĂ BLOCAȚI. NU CITIȚI TOT O DATĂ, CITIȚI O PARTE ȘI MAI ÎNCERCAȚI IMPLEMENTARE FĂRĂ RESTUL HINT-urilor.**

**Hints oets:**

Înainte de a paraleliza implementați Odd Even Transposition Sort secvențial și asigurați-vă că merge.

Nu folosiți while(sorted) deoarece va fi greu de paralelizat (trebuie făcută variabila sorted globală, trebuie sincronizat între thread-uri, și apoi trebuie asigurat că toate thread-urile se opresc și nu doar o parte). În schimb puteți schimba acel while cu un for de N pași.

În interior trebuie să rupem for-ul existent în unul care face toate interschimbările pentru perechile de elemente pentru care cel din stânga este pe poziție pară. Apoi un for care face toate interschimbările de elemente pentru care cel din stânga este pe poziție impară.

Acest două for-uri interioare trebuie paralelizate. Ca și până acum, vom folosi start și end dar trebuie să facem corecții la start. Start trebuie să fie un multiplu de 2, sau pentru cazul cu impare un multiplu de 2 la care adăugăm 1.

Nu uitați barierele.


**Hints merge sort:**

Este important să vă uitați pe slide-uri pe schema mergeSort și să înțelegeți care for se potrivește cărui părți din algoritm. Care este for-ul de log2(N) pași pe înălțimea arborelui? Care este for-ul pe lățimea sa? Pe care îl putem paraleliza?

Fie **w** determinat din nivelul la care ne aflăm pe arbore. El reprezintă câte elemente sunt în fiecare din cei doi vectori dați ca input operației de **merge**. Bazat pe Tid (identificatorul thread-ului), w, N (=32 aici) și P (=4 aici) putem să calculăm următoarele valori pentru Start și End:

| | Tid=0 | | Tid=1 | | Tid=2 | | Tid=3 | |
|---|---|---|---|---|---|---|---|---|
| | Start | End | Start | End | Start | End | Start | End |
| w=16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 |
| w=8 | 0 | 0 | 0 | 16 | 16 | 16 | 16 | 32 |
| w=4 | 0 | 8 | 8 | 16 | 16 | 24 | 24 | 32 |
| w=2 | 0 | 8 | 8 | 16 | 16 | 24 | 24 | 32 |
| w=1 | 0 | 8 | 8 | 16 | 16 | 24 | 24 | 32 |

Zona marcată cu portocaliu reprezintă cazurile când thread-urile nu au nimic de făcut, o iterație de la 0 la 0 sau de la 16 la 16. Asta ne și dorim.

Dacă analizați numerele vă puteți da seama ca ele reprezintă valorile pe care le obțineam în mod normal pentru Start și End (ultimul rând) rotunjite la 2*w.

Reminder: Putem să rotunjim împărțind la 2*w, apelând floor și înmulțind cu 2*w.

Atenție la interschimbarea pointerilor de vectori, v cu vNew. Dorim să păstrăm interschimbarea, dar în programul secvențial se face o singură dată per iterație. Trebuie să ne asigurăm că și în cel multithread se face la fel. E posibil ca într-o implementare naivă să fie P interschimbări și să apară conflicte write-write sau read-write pentru acești pointeri.