Laborator 5

Adunare n polinoame. Sincronizare la nivel de lista

* Cerinta

Se considera n polinoame reprezentate prin lista de monoame (reprezentare: lista inlantuita ordonata dupa exponentii monoamele). Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri). Polinoamele se citesc din fisiere – cate un fisier pentru fiecare polinom: - un fisier contine informatii de tip (coeficient, exponent) pentru fiecare monom al unui polinom.

Dintre cele p threaduri p1 sunt cititoare si p2 sunt de tip ‘worker’ (preiau din coada si adauga in lista rezultat) –p=p1+p2

Sincronizarile se vor face folosind variabile de tip Lock asociate cu fiecare nod.oDaca rezulta un nod cu coeficient nul acest nod se va sterge din lista!

* Preconditie, Postconditie

Fisierele de intrare si cel de iesire nu contin monoame cu coeficient egal cu 0.

* Structura

Exista o entitate principal numita Polinom pentru care se retine o lista inlantuita de obiecte de tip Element (retine un numar intreg pentru coeficient si un numar intreg pentru monom).

Pentru fiecare tip de executie (secventiala sau paralela) exista o clasa care contine un obiect de tip Polinom ce va fi rezultatul adunarii si un numar intreg (n) ce reprezinta numarul de polinoame. In clasa Paralel avem in plus un numar intreg (p1) ce reprezinta numarul de threaduri de citire, numar intreg (p2) ce reprezinta numarul de threaduri worker si o lista de obiecte de tip Polinom.

* Detalii implementare

La executia secventiala se citeste fiecare polinom din fisier si se creeaza un obiect de tip Polinom. Pentru fiecare element al polinomului se verifica daca coeficientul current exista deja in polinomul rezultatul sau nu. Daca exista, atunci se adauga valoarea monomului la ce exista deja in rezultat, altfel se adauga o noua intrare in lista inlantuita. Inainte de a ascris in fisier se sorteaza polinomul in functie de valorile coeficientilor.

La executia paralela exista mai multe threaduri care citesc toate polinoamele din fisiere si le adauga in lista retinuta care are rolul de coada. Astfel, in momentul in care un thread vrea sa faca calculele de adunare prima data verifica daca mai sunt elemente in coada; daca nu mai sunt atunci i se opreste executia. Altfel, se extrage primul polinom din coada si se face adunarea la polinomul rezultat, exact la fel ca in cazul executiei secventiale.

Pentru realizarea calculelor se va folosi lock fie pe elementele listei cand se face adunarea.

**Java**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip** | **Nr. threads** | **Impartire threads** | **Timp exec.** |
| 10 polinoame  Grad maxim: 1000  Max monoame: 50 | secvential | secvential | 51.90302 |
| 4 | p1 = 2, p2 = 2 | 11.61007 |
| p1 = 3, p2 = 1 | 10.56257 |
| 6 | p1 = 2, p2 = 4 | 12.71698 |
| p1 = 3, p2 = 3 | 11.39623 |
| 8 | p1 = 2, p2 = 6 | 13.05509 |
| p1 = 3, p2 = 5 | 12.28785 |
| 5 polinoame  Grad maxim: 10000  Max monoame: 100 | secvential | secvential | 51.03044 |
| 4 | p1 = 2, p2 = 2 | 10.63404 |
| p1 = 3, p2 = 1 | 9.92205 |
| 6 | p1 = 2, p2 = 4 | 11.61892 |
| p1 = 3, p2 = 3 | 11.47843 |
| 8 | p1 = 2, p2 = 6 | 11.83829 |
| p1 = 3, p2 = 5 | 11.8624 |

# Analiza rezultatelor

Secvential vs. Paralel

* + Pentru ambele cazuri, variantele paralele ruleaza mai rapid decat variantele secventiale.

Tip 1 vs. Tip 2

* + Diferenta nu este observabila, insa varianta 2 ruleaza cu aproximativ o milisecunda mai repede decat varianta 1.

4 threaduri vs. 6 threaduri vs. 8 threaduri

* + Pentru fiecare dintre variante, se obtin timpi de executie mai mici la rularea programului cu 4 threaduri, acestia crescand direct proportional cu numarul de threaduri.

Laborator 4 vs. Laborator 5

* + Datorita faptului ca seturile de date sunt diferite (setul de date din laboratorul 4 contine mai multe monoame), rezultatele sunt mai bune in laboratorul 5
  + In cazul rularii laboratorului 5 pe setul de date de la laboratorul 4, timpii de executie cresc, programul din laboratorul 4 ruland mai repede