Laborator 5 PPD – Salsigan Razvan-Dan 236/2

Data predarii: 7.12.2021

# Cerinta

Se considera n polinoame reprezentate prin lista de monoame (reprezentare: lista inlantuitaordonata 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 alunui polinom, (conditie: fisierelenu contin monoame cu coeficient egal cu 0)

Metoda:

1.Se creeaza o lista inlantuita -L corespunzatoare unui polinom nul.

2. Dintre cele p threaduri p1 sunt cititoare si p2 sunt de tip ‘worker’ (preiau din coada si adauga in lista rezultat) –p=p1+p2.(conditie –pentru structura de tip coada NUse admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

3.Celelalte threaduri (p2) preiau cate un monom din coada si il aduna la polinomulreprezentat in lista L.Se continua operatiile 1.,.2., 3. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.

4.Rezultatul obtinut in lista L se scrie intr-un fisier rezultat(conditie: fisierul nu contine monoame cu coefficient egal cu 0)

Am folosit implementari de la 0 pentru coada si lista inlantuita.

Pentru sincronizarea la nivel de nod, la adaugare este nevoie ca threadul current sa tina blocate cate doua noduri (precedent si current), astfel incat un singur thread sa poata adauga un nod intre aceleasi doua noduri vecine.

Am folosit sentinele pentru lista inlantuita, pe prima pozitie avand un monom cu grad -1, iar pe ultima unul cu grad 9999

Sincronizarea la nivel de nod este mai granulara si ar trebui sa dea rezultate mai bune, deoarece un thread blocheaza doar cate 2 noduri adiacente, in loc sa blocheze toata lista

public void insert(Monom data){  
 Node newNode = new Node(data);  
  
 Node current = head;  
 Node previous = null;  
  
 current.lock.lock();  
 while(data.getExp() >= current.getMonom().getExp()){  
 if(previous!=null)  
 previous.lock.unlock();  
 current.lock.unlock();  
  
 previous = current;  
 current = current.getNext();  
  
 previous.lock.lock();  
 current.lock.lock();  
 }  
  
 if(previous!=null && previous.getMonom().getExp() == data.getExp()){  
 previous.getMonom().add(data);  
 }  
 else if(previous!=null)  
 previous.setNext(newNode);  
 newNode.setNext(current);  
  
 if(previous!=null)  
 previous.lock.unlock();  
 current.lock.unlock();  
}

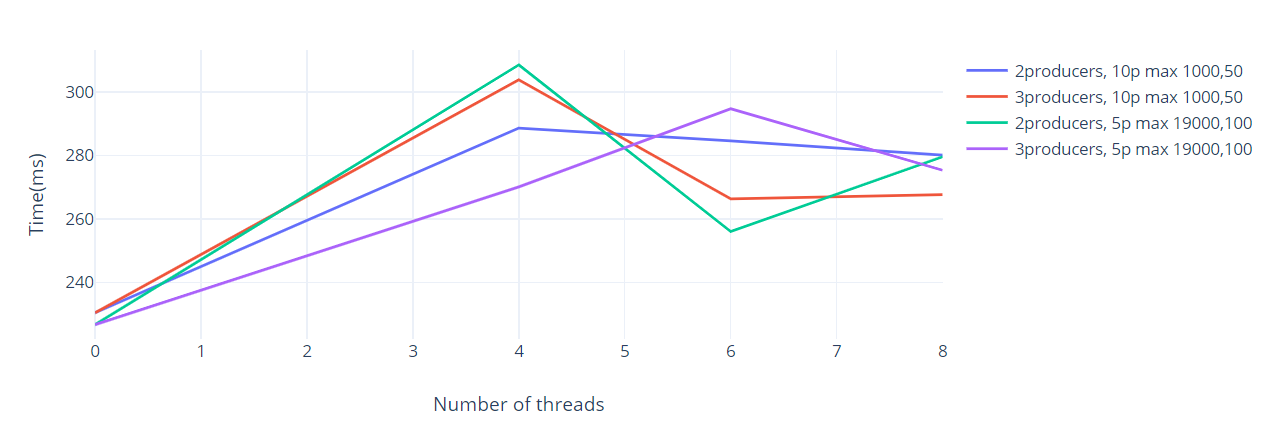
# Analiza timpilor de executie

Testele au fost efectuate pe un procesor i5-9300H (4C, 8T) , in Windows ; implementare Java

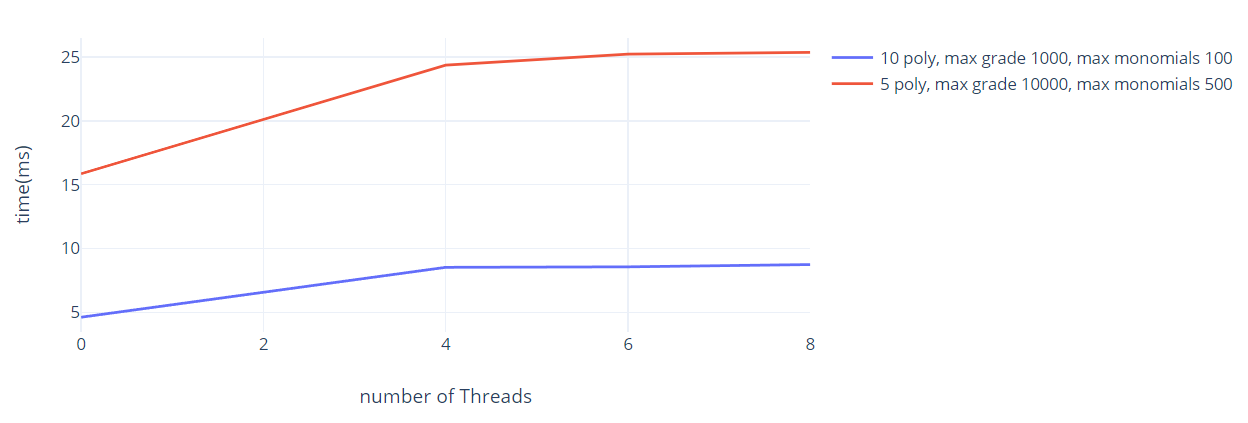
Timpul afisat este in ms

|  |  |  |  |
| --- | --- | --- | --- |
| Detalii polinoame | Distributie producer/consumer | Nr threads | Timp executie |
| 10 polinoame  Grad maxim 1000  Maxim 50 monoame | 2p | Secvential | 230.4216 |
| 4 | 288.6582 |
| 6 | 284.6274 |
| 8 | 280.1365 |
| 3p | 4 | 303.882 |
| 6 | 266.3343 |
| 8 | 267.7091 |
| 5 polinoame  Grad maxim 10000  Maxim 100 monoame | 2p | Secvential | 226.704 |
| 4 | 308.5988 |
| 6 | 256.0754 |
| 8 | 279.6178 |
| 3p | 4 | 270.1415 |
| 6 | 294.7909 |
| 8 | 275.3843 |

#### Grafice



Lab4:



Concluzie

Cele mai bune rezultate s-au inregistrat pentru 8 threaduri, cu 3 producers, in ambele variante

Comparativ cu lab4, timpii de executie au fost mai mari, dar probabil pentru ca my laptop was feeling lucky cand am facut testele pentru lab4