Laborator 4 – PPD

Analiza cerintelor

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.

Rezultatul obtinut se scrie intr-un fisier rezultat.

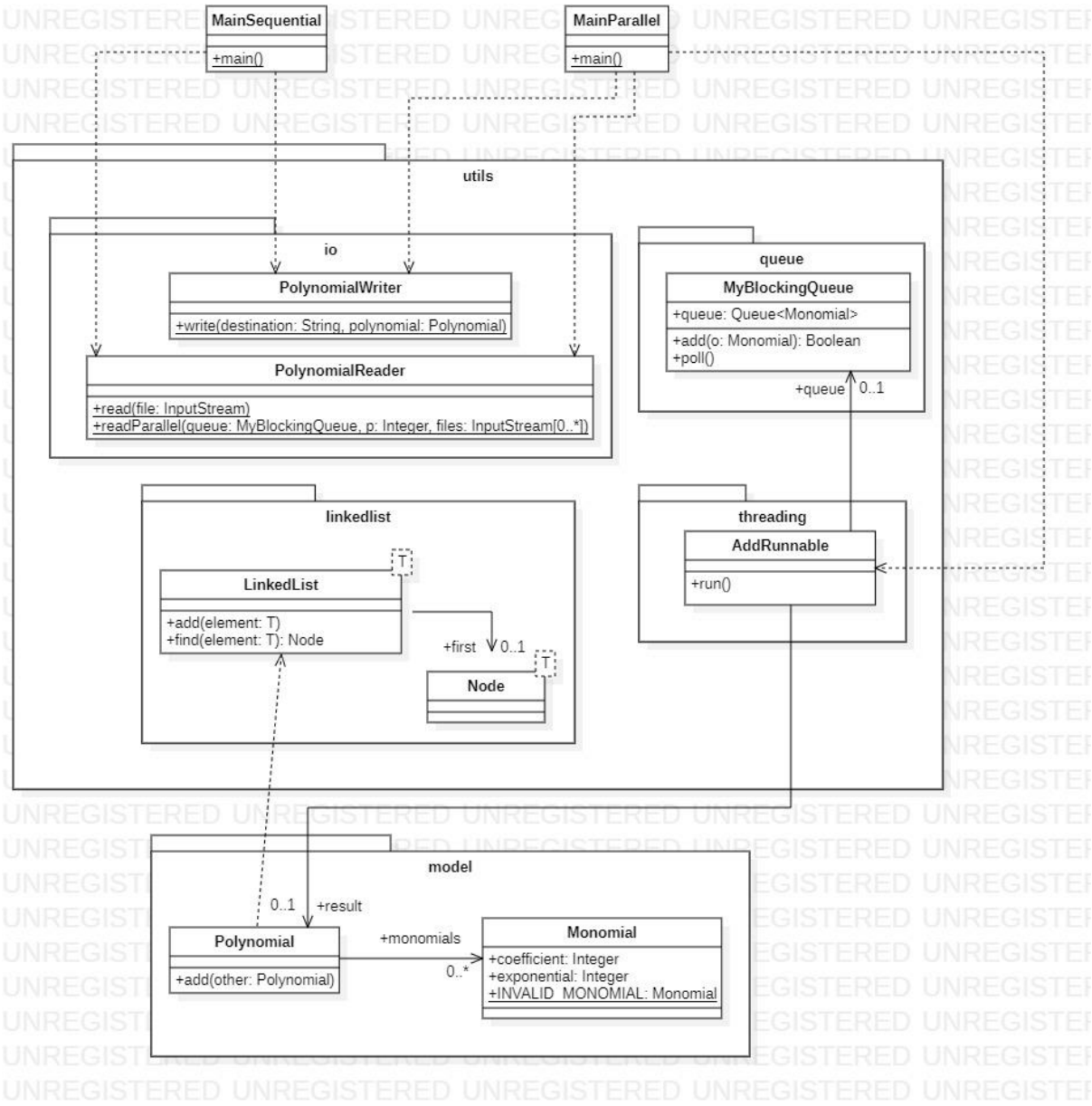
Preconditie: Fisierele nu contin monoame cu coeficient egal cu 0.

Constrangere: Sincronizare la nivel de lista

Postconditie: Fisierul rezultat nu contine monoame cu coeficient egal cu 0

Proiectare

Diagrama ce prezinta clasele folosite in implementare, impreuna cu metodele lor si relatiile dintre ele.



Detalii de implementare

Tema de laborator a fost realizata in limbajul de programare Java. Toate sincronizarile au fost realizate cu ajutorul cuvantului cheie `synchronized`, plasat in metodele cu cod critic. De asemenea, toti parametrii (numar thread-uri, fisiere cu polinoame) sunt date ca si argumente in linia de comanda.

Clasele din pachetul `model` reprezinta o abstractizare a conceptului de monom si de polinom.

Un polinom contine mai multe monoame, memorate intr-un `LinkedList` din pachetul `utils.linkedlist`, ce reprezinta o implementare proprie a unei liste simplu inlantuie ordonata crescator. Lista este sincronizata in intregime, nu la nivel de nod. Monoamele sunt inserate in aceasta lista crescator dupa exponentul lor.

`MyBlockingQueue` reprezinta o implementare proprie a conceptului de `BlockingQueue`, regasit in libraria standard din Java, unde operatia de scoatere a unui element dintr-o coada este blocanta, pana coada are un element care poate fi scos. `MyBlockingQueue` suporta doar operatiile necesare pentru aceasta tema de laborator, respectiv `add` si `poll`, ce respecta specificatia regasita in interfata `Queue` din libraria standard. Pentru a refolosi cod, `MyBlockingQueue` se foloseste de o instanta de `Queue` pentru a stoca elementele si implementeaza partea de sincronizare in metodele ei.

Atat varianta secventiala, cat si cea paralela, se folosesc de toate clasele din pachetul `utils.io`, responsabile cu citirea si scrierea polinoamelor din fisiere.

Facand o analogie cu problema producatorului-consumatorului, metoda readParallel din `PolynomialReader` reprezinta producatorul, in timp ce metoda run din `utils.threading.AddParallel`, consumatorul. Producatorul citeste cate un monom din fisierele date si il adauga in coada de tip `MyBlockingQueue`, ce va fi preluat de catre un thread ce executa metoda run, acesta primind aceeasi instanta de `MyBlockingQueue`. Dupa ce il citeste, il adauga la un polinom rezultat, initializat cu polinomul null, care poate fi accesat de toate thread-urile, prin metoda `add` din clasa `Polynom`.

Dupa ce producatorul termina de citit in intregime fisierele date, el va trimite fiecarui thread un monom invalid, cu exponentul egal cu -1. Thread-urile vor detecta fiecare aparitia acestui monom cand scot un element din coada si isi vor termina executia.

Cazuri de testare

1)

$f[X] = 1 + 2X + 3X^2 + X^7$

$g[X] = 4 + 2X - 9X^7$

$h[X] = 90 + 2X^9$

Rezultat asteptat: $95 + 4X + 3X^2 - 8X^7 + 2X^9$

2)

$f[X] = 55X + 47X^3 + 58X^4 + 84X^5$

$g[X] = 8X + 46X^2 + 32X^3 + 75X^5$

$h[X] = 21X + 61X^2 + 46X^3 + 57X^4$

$i[X] = 72X - 125X^3 + 90X^4$

Rezultat asteptat: $156X + 107X^2 + 205X^4 + 159X^5$

Rezultate

Tip polinoame	**Numar thread-uri**	**Timp executie (ms)**
10 polinoame, fiecare cu gradul maxim 1000 si cu maxim 100 monoame *(test1)*	secvential	127
	4	132
	6	131
	8	130
5 polinoame, fiecare cu gradul maxim 10000 si cu maxim 500 monoame *(test2)*	secvential	182
	4	184
	6	187
	8	198

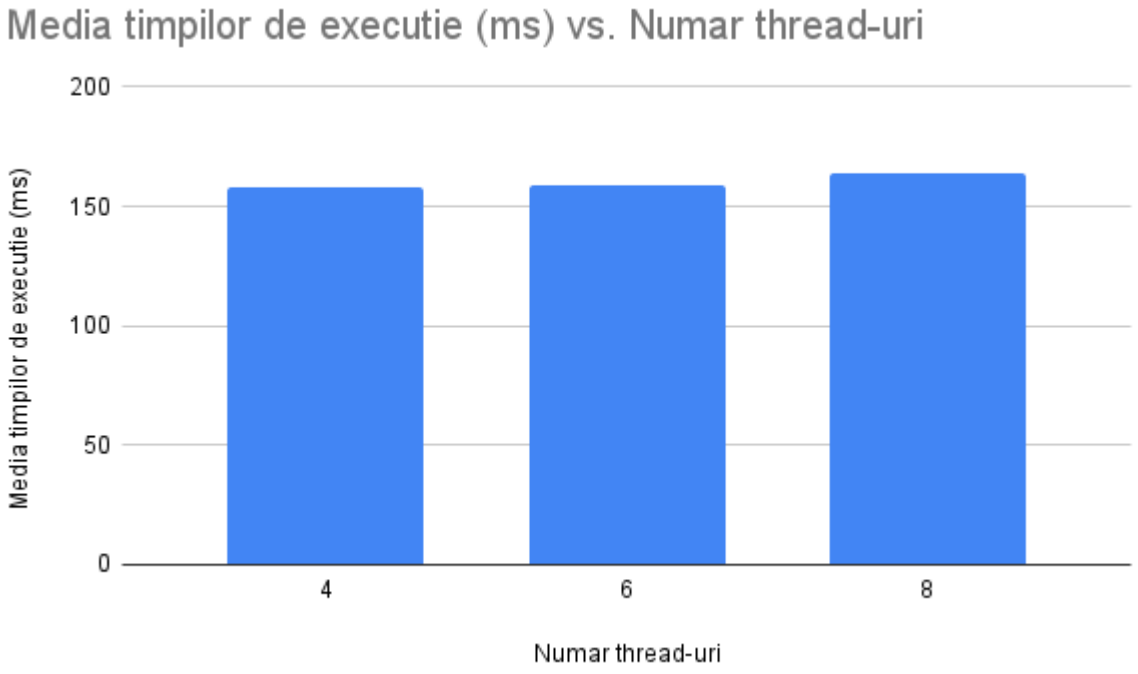
Nota: fiecare test a fost rulat de 10 ori si pentru evaluarea timpului de executie s-a considerat media aritmetica a acestor 10 rulari. Fiecare test a fost rulat pe o masina cu Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 MHz 2 cores 4 Logical Processors cu sistem de operare Windows 10 versiunea 21H1.

Analiza rezultatelor

Se poate observa ca varianta secventiala este per total cea mai performanta, desi s-au obtinut timpi foarte asemenatori. Un posibil motiv este faptul ca in varianta paralela sunt folosite multe sincronizari, ce adauga un overhead destul de mare, obtinandu-se conceptual o varianta similara cu cea secventiala, dar mai putin performanta; spre exemplu, fiind sincronizata intreaga lista, un singur thread va putea sa o acceseze.

Dintre variantele paralele, se observa ca cele in care se folosesc 4 thread-uri sunt cele mai performante. O posibila explicatie este faptul ca masina pe care au fost rulate testele are 4 procesoare logice, iar cu cat numarul de thread-uri creste, cu atat costul crearii si intretinerii lor devine mai costisitor.

Numar thread-uri	Media timpilor de executie (ms)
4	158
6	159
8	164


Media timpilor de executie (ms) vs. Numar thread-uri
200
150
100
50
0
Media timpilor de executie (ms)
4
6
8
Numar thread-uri