Laborator 1 - Nichita Utiu 233

Cerinta

Se vor generiza operatiile implementate la lab-ul 1 astefel incat sa poata primi matrici cu elemente de orice tip. De asemenea, operatia de adunare trebuie sa poata primi orice operator binar care sa fie aplicat pe elemente. **Codul va fi tradus si in C++**

Se vor face benchmark-uri pentru testarea performantei operatiilor pe matrici aleatoare de dimensiuni mari(1000 x 1000). Se vor testa pentru matrici de numere intregi si pentru numere complexe. Pentru fiecare se vor masura performantele cu umatorii 2 opertori:

- cel de inmultier
- $a \odot b = 1 (1 / a + 1/ b)$

Proiectare

Proiectul e impartit in urmatoarele clase:

- **MatrixAdder** - Clasa care contine metoda statica de adunare a doua matrici
  - `T[][] add(T[][] first, T[][] second, int numThreads, BinaryOperator<T,T,T>)` - metoda statica care primeste doua matrici si returneaza matricea rezultata aplicarii operatorului specificat pe fiecare perche corespunaztoare de elemente din cele 2 matrici. Operatia va rula pe numarul specificat de thread-uri.
- **MatrixMultiplier** - Clasa care contine metoda statica de inmultire a doua matrici
  - `T[][] multiply(T[][] first, T[][] second, int numThreads)` - metoda statica care primeste 2 matrici si returneaza rezultatul inmultirii matriciale. Operatia va rula pe numarul specificat de thread-uri.
- **Main** - clasa main care creaaza matricile random, cheama metodele din `MatrixAdder` si `MatrixMultiplier` pe ele si cronometreaza executia lor.

Performanta

Sistem: Antergos Linux(Arch Linux) 64bit - Intel® Core™ i7-5500U CPU @ 2.40GHz × 4

Java - Intreg

Inmultire

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	86.130263
1000 x 1000	2	69.434133
1000 x 1000	4	63.750555
1000 x 1000	6	62.822963
1000 x 1000	8	59.597876

$\odot$

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	86.158578
1000 x 1000	2	71.65631
1000 x 1000	4	91.52792
1000 x 1000	6	69.762152
1000 x 1000	8	63.534165

Java - Complex

Inmultire

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	84.452587
1000 x 1000	2	330.193763
1000 x 1000	4	135.053536
1000 x 1000	6	166.33981
1000 x 1000	8	69.988748

$\odot$

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	137.180066
1000 x 1000	2	207.063715
1000 x 1000	4	238.237953
1000 x 1000	6	213.37712
1000 x 1000	8	229.179165

C++ - Intreg

Inmultire

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	71.6773
1000 x 1000	2	59.268
1000 x 1000	4	46.4045
1000 x 1000	6	47.7051
1000 x 1000	8	47.1014

$\odot$

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	81.6074
1000 x 1000	2	65.8035
1000 x 1000	4	50.443
1000 x 1000	6	55.2248

dimensiune	# thread-uri	timp(ms)
1000 x 1000	8	52.2598

C++ - Complex

Inmultire

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	143.459
1000 x 1000	2	123.359
1000 x 1000	4	112.065
1000 x 1000	6	110.693
1000 x 1000	8	107.725

$\odot$

dimensiune	# thread-uri	timp(ms)
1000 x 1000	1	219.871
1000 x 1000	2	167.261
1000 x 1000	4	140.096
1000 x 1000	6	138.59
1000 x 1000	8	141.115

Comparatie

Comparatie dupa tipul de element

Indiferent de limbaj sau de operator, operatiile pe intregi sunt mai rapide decat cele pe numere complexe. Cel mai probabil fiindca numerele complexe opereaza prin intermediul apelurilor de functii in ambele implementari care adauga overhead.

Comparatie dupa operator

Operatiile de inmultire sunt mai rapide decat cele cu operatorul $\odot$. Acest lucru se datoreaza probabil din cauza faptului ca o succesiune de inmultiri este probabil optimizata la nivel de compilator, pe cand cele mai complexe nu.

Comparatie dupa limbaj

Pentru aceleasi operatii si tipuri de date C++ este net mai rapid decat Java.