Зад. 10 (Умножение на матрици)

Разглеждаме матриците A с размерност (m, n) и B с размерност (n, k). Матрицата C = A.B, равна на произведението на A и B ще има размерност (m, k). Да се напише програма която пресмята матрицата C. Работата на програмата по умножението на матриците да се раздели по подходящ начин на две или повече нишки (задачи).

Изискванията към програмата са следните:

(о) Размерността на матриците се задава от подходящо избрани командни параметри – например **„-m 1024 -n 512 -k 2048“**;

(о) Елементите на матриците генерираме произволно с помощта на **Math.random()** (класа **java.util.Random)** или **java.util.concurrent.ThreadLocalRandom;** (Тоест матриците може да имат float или double елементи)

Разликата между двата начина - **Math.random()** е достъпен във всички версии на Java и ужасно бавен. Не е проектиран за работа в много-нишкова (multi-threaded) среда; В Java 7 и 8, разполагаме с **ThreadLocalRandom**, който е специално проектиран за работа в рамките на отделен thread (респективно multi-threaded среда);

Ще бъде много интересно да реализирате програма, използваща и двата начина и съответно получим два вида резултати от работата на програмата;

(о) Команден параметър указващ входен текстов файл, съдържащ матриците, които ще умножаваме – например **„-i m3x-data.in“**. Който и да е от параметрите **„-m“ „-n“ „-k“** и параметърът **„-i“** са взаимно-изключващи се; Ако все пак бъдат зададени и четирите (или някой от **„-m“ „-n“ „-k“**, заедно с **„-i“**), решението как да реагира програмата е Ваше. (Идеята? Или генерираме произволни матрици или използваме отнапред даден вход).

Форматът на файла **m3x-data.in** е следният:

=== цитат ===
m n k
a11 a12 a13 … a1n
a21 a22 a23 … a2n
…
am1 am2 am3 … amn
b11 b12 b13 … b1k
b21 b22 b23 … b2k
…
bn1 bn2 bn3 … bnk
=== цитат ===

Тоест:

1вият ред съдържа числата **m**, **n** и **k**, разделени с интервал;

На оставащите **m**+**n** реда във файла са разположени редовете на матриците A и B. Първо тези на A, след което тези на B. Елементите на всеки ред от матрицата са разделени със интервали (един или повече).

(о) Команден параметър указващ изходен файл, съдържащ резултата от пресмятането – например **„-o m3x-data.out“**. Форматът на изходният файл е следният:

=== цитат ===
m k
c11 c12 c13 … c1k
c21 c22 c23 … c2k
…
cm1 cm2 cm3 … cmk
=== цитат ===

При липса на този команден параметър **не се** записва във файл резултата от умножението на

матриците;

(о) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането елементите на C – например **“–t 1”** или **“–tasks 3”**;

(о) Извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление;

Примери за подходящи съобщения:

„**Thread-<num> started.**“,

„**Thread-<num> stopped.**“,

„**Thread-<num> execution time was (millis): <num>**“,

„**Threads used in current run: <num>**“,

„**Total execution time for current run (millis): <num>**“ и т.н.;

(о) Да се осигури възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето отделено за изчисление на резултантната матрица, отново чрез подходящо избран друг команден параметър – например **“-q”;**

ЗАБЕЛЕЖКА:

(о) При желание за направата на подходящ графичен потребителски интерфейс (**GUI**) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра **„-q“** се добавя изискването **да не пуска** графичният интерфейс. Причината за това е, че Вашата програма трябва да позволява отдалечено тестване, а то ще се извършва в **terminal**.

Уточнения (hints) към задачата:

(о) В условието на задачата се говори за разделянето на работата на две или повече нишки. Работата върху съответната задача, в случаят в който е зададен **„-t 1“** (т.е. цялата задача се решава от една нишка) ще служи за еталон, по който да измерваме евентуално ускорение (т.е. това е **T1**). В кода реализиращ решението на задачата трябва да се предвиди и тази възможност – задачата да бъде решавана от единствена нишка (процес); Пускайки програмата да работи върху задачата с помощта на единствена нишка, ще считаме че използваме серийното решение на задачата; Измервайки времето за работа на програмата при използването на **„p“** нишки – намираме **Tp** и съответно можем да изчислим **Sp**. Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже **Ep**.

Като обобщение - данните събрани при тестването на програмата Ви, трябва да отразяват **Tp**, **Sp** и **Ep**. Желателно е освен табличен вид, да добавите и графичен вид на **Tp**, **Sp**, **Ep**, в три отделни графики.

(о) Не се очаква от Вас да реализирате библиотека, осигуряваща математически операции със комплексни числа. Подходяща за тази цел е например **Apache Commons Math3 (http://commons.apache.org/proper/commons-math/userguide/complex.html).** При изчисленията, свързани с генерирането на множеството на Манделброт (задачите за фрактали), определено ще имате нужда от нея.

(о) Не се очаква от вас да реализирате библиотека, осигуряваща математически операции със голяма точност. Подходяща за тази цел библиотека е например **Apfloat (http://www.apfloat.org)**. Ако програмата Ви има нужда от работа с големи числа, можете да използвате нея.

Разбира се **BigInteger** и **BigDecimal** класовете в **java.math** са също възможно решение – въпрос на избор и вкус.

Преди да направите избора, проверете дали избраната библиотека не използва също нишки – това може да доведе до неочаквани и доста интересни резултати; ;)

(о) Не се очаква от Вас да търсите (пишете) библиотека за генериране на **.png** изображения. Java има прекрасна за нашите цели вградена библиотека, която може да се ползва. Примерен проект, показващ генерирането на чернобялата и цветната версия на фрактала на Манделброт /множество на

Манделброт за формула (2)/, цитирани в задачите за фрактали, е качена на **http://rmi.yaht.net/docs/example.projects/ - pfg.zip**.

(о) Командните аргументи (параметри) на терминална Java програма, получаваме във масива **String args[]** на **main()** метода, намиращ се в стартовият клас. За „разбирането“ им (анализирането им) може да ползвате и външни библиотеки писани специално за тази цел . Един добър пример за това е: **Apache Commons CLI (http://commons.apache.org/cli/)**.