



Софийски университет „Св. Климент Охридски”

Факултет по математика и информатика

Умножение на матрици

Проект по Разпределени софтуерни архитектури

Изготвил:

Валентин Змийчаров, фак. № 61481, Софтуерно инженерство III курс

Съдържание

1. Описание на задачата	2
2. Описание на решението и реализация	3
CalculationThread.java	3
Helpers.java	3
RSA_Matrices.java	3
3. Получени резултати и графики	4

1. Описание на задачата

Зад. 10 (Умножение на матрици)

Разглеждаме матриците A с размерност (m, n) и B с размерност (n, k) . Матрицата $C = A \cdot B$, 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()** или класа **java.util.Random**; (Тоест матриците може да имат float или double елементи)

(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;

(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 запуска** графичния интерфейс. Причината за това е, Вашата програма да може да се тества отдалечено.

(о) Задачата може да се реши и с помощта на RMI (**java.rmi**). За целта трябва да се помисли за разпределения достъп до общия ресурс в случая матриците A , B и C .

Уточнения:

(о) В условието на задачата се говори за разделянето на работата на две или повече нишки. Работата върху съответната задача на една нишка ще служи за еталон, по който да измерваме евентуално ускорение ($T1$). Тоест в кода реализиращ решенията на задачите 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 работа с „p“ нишки - Tp , изчисляваме Sp . Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже Ep .

(о) Командните аргументи (параметри) на терминална (конзолна) Java програма, получаваме във масива `String args[]` на `main()` метода, на стартовия 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/>).

2. Описание на решението и реализация

Програмата е разработена на езика Java и използва паралелни изчисления за по-бързо умножение на матрици. Тя решава проблема по следния начин:

Дефинирани са 3 класа:

CalculationThread.java

Реализация на нишка. Подават ѝ се параметри (startRow, endRow, startCol, endCol) – от кой ред/колона до кой ред/колона трябва да извърши пресмятанията съответната нишка. Те се извършват върху глобално дефинирана матрица.

Helpers.java

Този клас дефинира следните помощни методи:

- createOptions – Дефинира опциите m, n, k, t, q, които ще бъдат прочетени от входа. m, n и k са размерите на 2-те матрици, t е максималният брой допустими нишки, а q обозначава дали програмата ще работи в „quiet“ режим или не.
- assignInputValues – Валидира входа и ако е коректен, присвоява стойностите на вече дефинираните опции на локални променливи, които по-късно използвам за пресмятането на умножението. Методът е от тип boolean и изходът от него е дали входните параметри са валидни.
- showMessageIfNotQuiet – Изписва на конзолата подадено като параметър съобщение ако програмата не работи в режим “quiet”
- calculateOptimalThreadsCount – Този метод изчислява оптималния брой нишки, които да бъдат използвани и по колко пресмятания ще извършва всяка от тях. Първоначално като максимум дефинирам стойността на въведената като вход опция „t“. След това намирам колко са ядрата на компютъра, на който се изпълнява програмата и ограничавам максимума до тази бройка, защото разпределянето на повече нишки от ядрата е ненужно и неефективно. Най-накрая на база на размерността на матриците, изчислявам оптималния брой нишки, които ще се използват. Например ако имаме да изчисляваме 9 елемента и можем да използваме 4 нишки, ще ги разпределим по следния начин: 3x2x2x2. Същото време можем да получим ако използваме 3 нишки (3x3x3) и това е оптималното решение.
- generateMatrix – По подадени размери на матрица, генерира такава с произволни числа между 0 и 2. Ограничението съм наложил, за да мога да прецизирам резултатите. Ако генерирам числа в по-голям диапазон, времето за пресмятане ще бъде много зависимо от техните стойности.
- printMatrix – Приема матрица и я принтира на конзолата. Ако някое от измеренията е по-голямо от 10, се изписват „...“. Това е с цел по-добре изглеждащ изход.

RSA_Matrices.java

Това е основният клас за програмата. Той използва вече дефинираните помощни методи и първо дефинира опциите. След това прочита входа и ако той е некоректен изписва съобщение и прекратява изпълнението на програмата. Намира ядрата на текущия компютър и изчислява колко нишки ще използва и всяка от тях по колко клетки от новата матрица ще пресмята. Генерирам random матрици A и B съответно с размерности (m,n) и (n,k). След това празна матрица C с размери (m,k). Дефинирам масив от нишки и запазвам времето на започване на програмата в наносекунди. На база на броя

нишки и действията, които всяка от тях ще върши, стартирам тяхното изпълнение, като с 2 цикъла определям коя нишка от къде до къде ще пресмята стойностите в новата матрица. Стартирам всички нишки и ги “join”-вам. След тяхното приключване взимам текущото време в наносекунди и изкарвам колко секунди е отнело изпълнението на програмата и колко нишки са използвани. Ако програмата не е в “quiet” режим, тя изкарва подходящи съобщения през цялото време. Програмата използва **Apache Commons CLI** за прочитане на входа.

3. Получени резултати и графики

За тестване на програмата бе ползван сървър с 16 ядра. Тествано бе времето за умножение на матрици с размерности (1600/800), (800/2048), използвайки 1, 2, 3, 4, 6, 8, 10, 12 и 16 нишки.

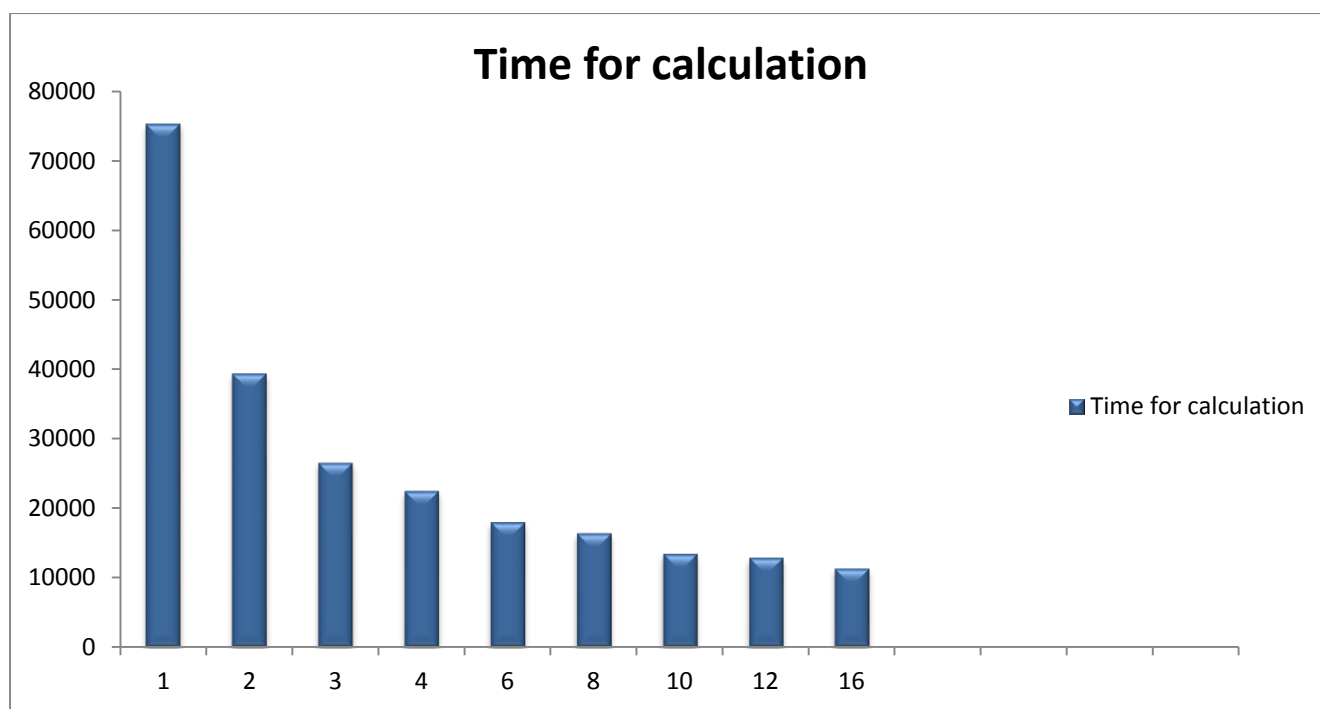
По абсцисата на графиката са представени броят нишки за изчисление, а по ординатата – времето в милисекунди за изчисление.

Формулите за изчисляването на Ускорението S и ефективността E са:

$$S(n) = T_1/T_n \text{ и } E(n) = S(n)/n,$$

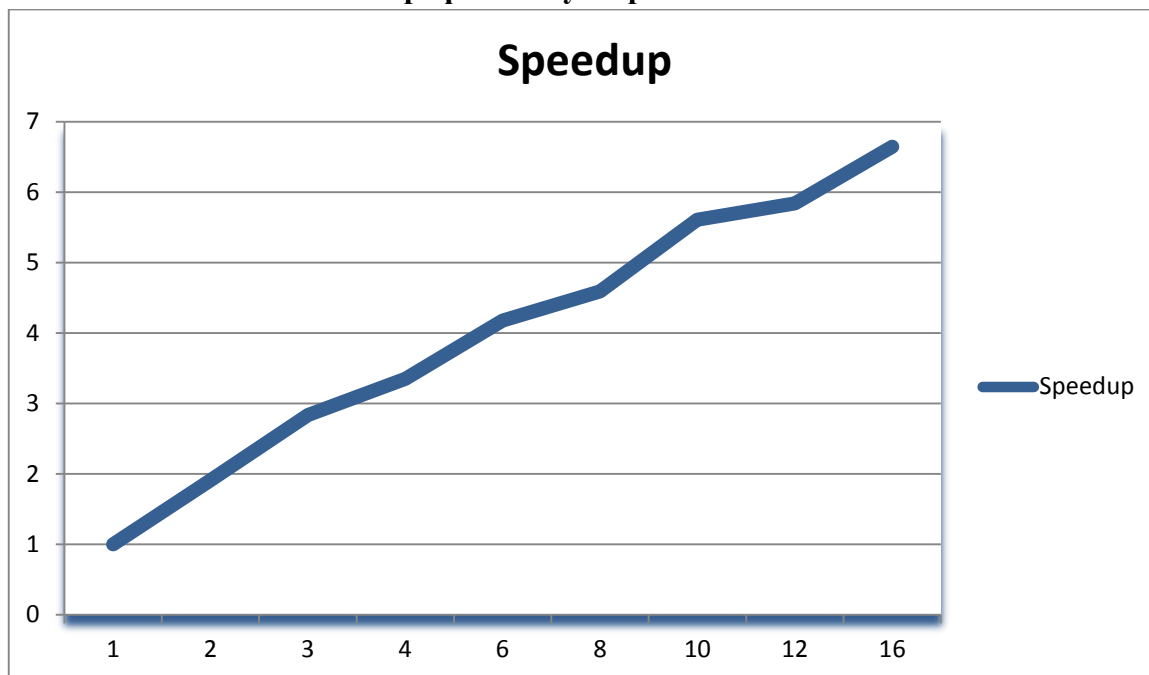
където n е броя нишки, а T_n е времето необходимо за завършване на работата на алгоритъм с n на брой нишки.

Графика на времето за изчисление



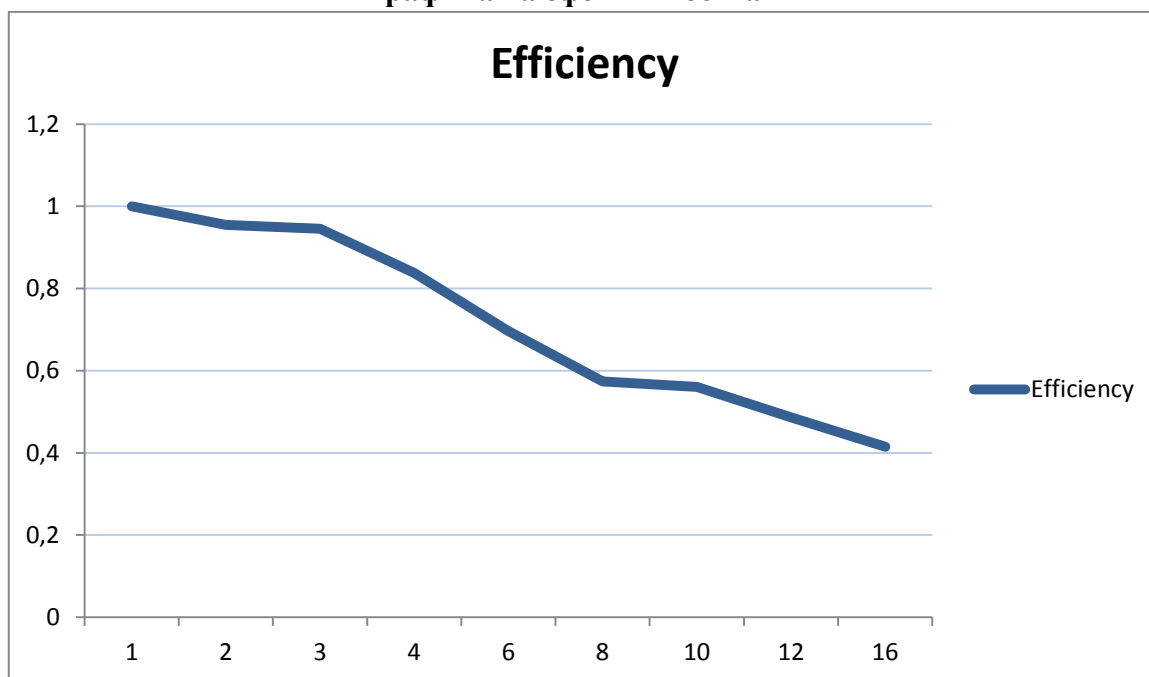
На следващата графика е представено ускорението при използване на 2 и повече нишки. Коефициента на ускорение получаваме като разделим времето за изпълнение от 1 нишка на времето за изпълнение на n нишки.

Графика на ускорението



Тук е представена ефективността при изчисленията, която се пресмята като разделим ускорението, получено при различен брой нишки, разделено на броя нишки.

Графика на ефективността



-m 1600 -n 800 -k 2048

Threads	Time	Acceleration	Efficiency
1	75335	1	1
2	39442	1,91002	0,95501
3	26571	2,835234	0,945078
4	22483	3,350754	0,837688
6	18031	4,178082	0,696347
8	16413	4,589959	0,573745
10	13432	5,608621	0,560862
12	12901	5,83947	0,486622
16	11340	6,643298	0,415206