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

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:

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

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

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

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

(o) 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:

(o) 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.

(o) 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:

(o) 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.
(o) 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

Ще представя резултатите за два входа : **„-m 1024 -n 512 -k 2048"** и **„-m 1600 -n 800 -k 2048"** и сравнение между двете:

3.1 Тестване с входни параметри „-m 1024 -n 512 -k 2048";

За тестване на програмата бе ползван сървър с 6 процесора по 4 ядра на всеки. Тествано бе времето за умножение на матрици с размерности (1024/512), (512/2048), използвайки 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 и 12, 20 и 24 нишки.

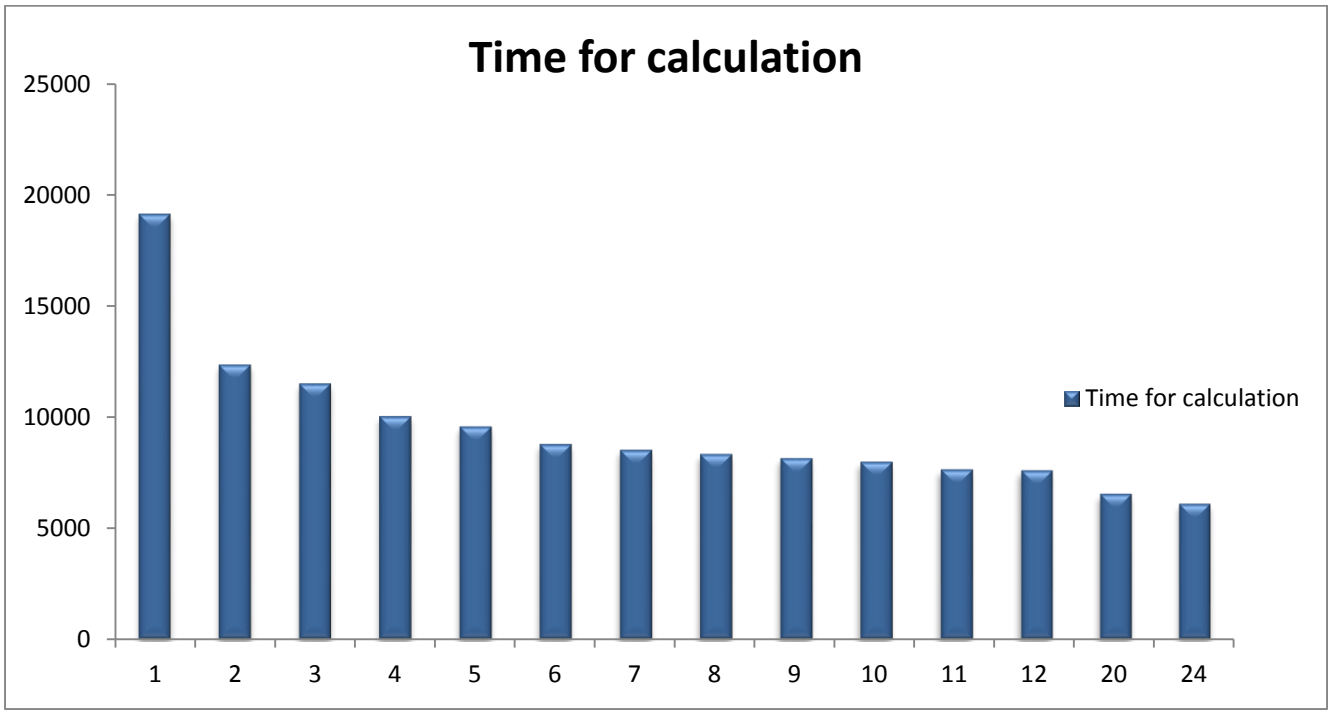
По абсцисата на графиката са представени броят нишки за изчисление, а по ординатата – времето в милисекунди за изчисление.

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.

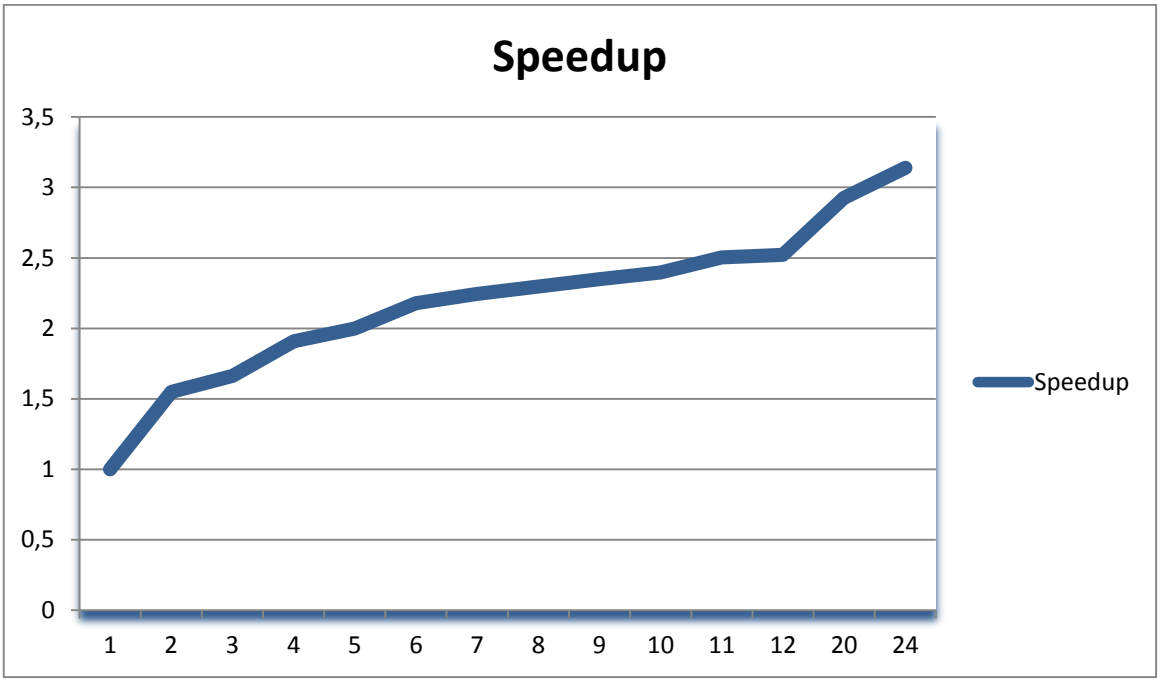
Графика на времето за изчисление



Както се вижда серийното решение, т.е. на една нишка, умножава матрицата за 19128 милисекунди. При 2 нишки – за 12349 милисекунди и т.н. се вижда ускорение на пресмятането до 24 нишки.

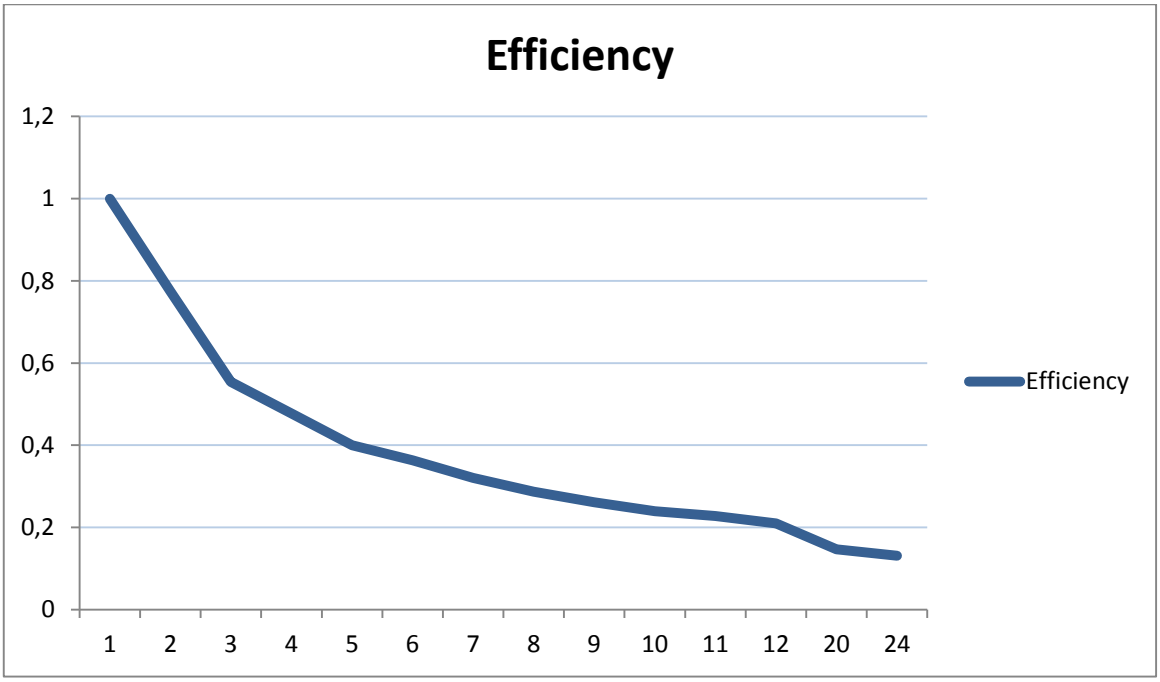
На следващата графика е представено ускорението при използване на 2 и повече нишки. Коефициента на ускорение получаваме като разделим времето за изпълнение от 1 нишка на времето за изпълнение на n нишки.

Графика на ускорението



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

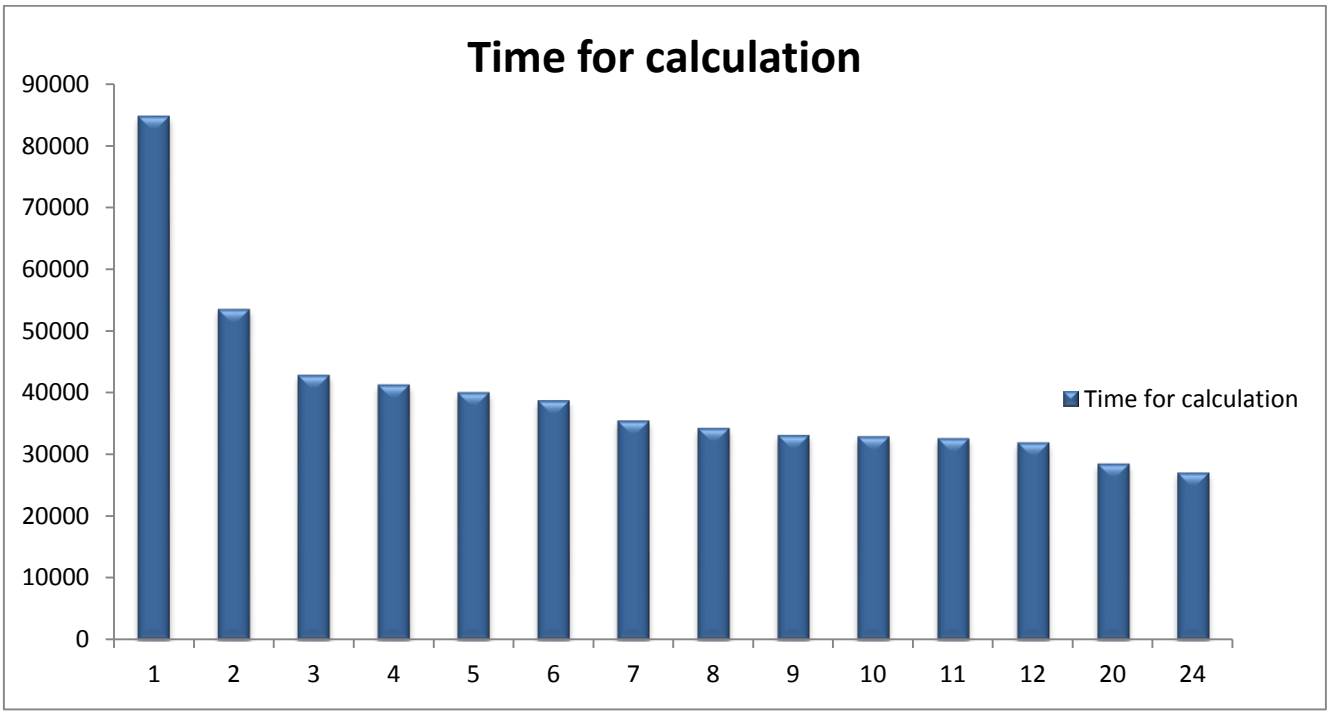
Графика на ефективността



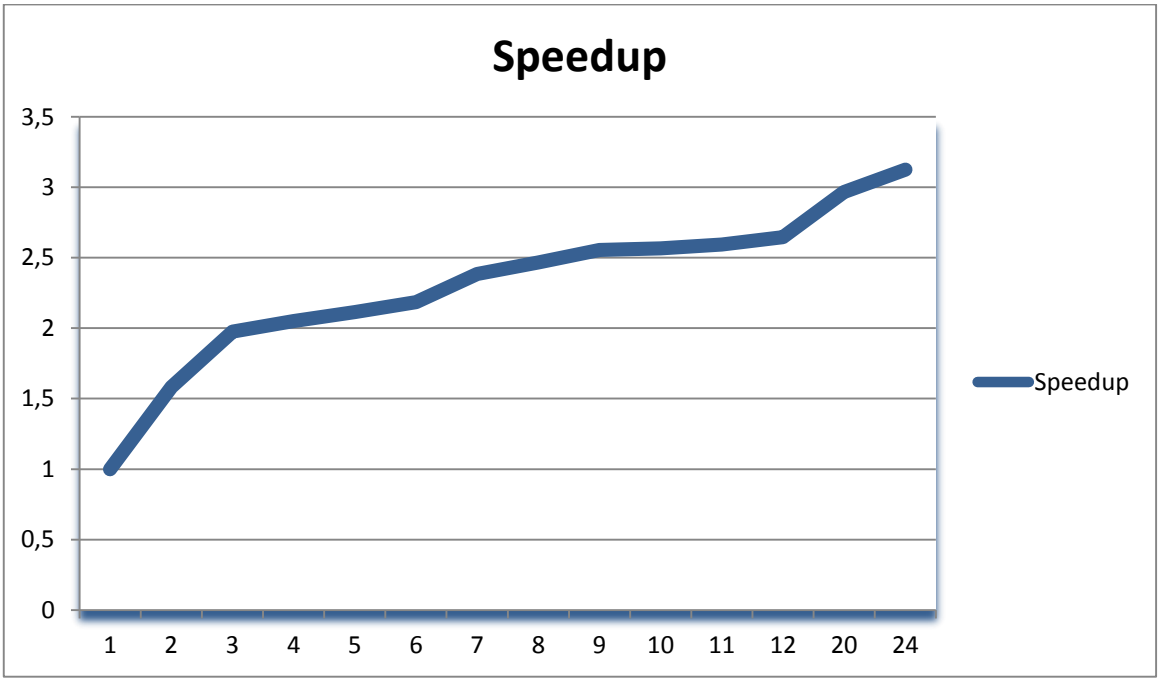
3.2 Тестване с входни параметри „-m 1600 -n 800 -k 2048“;

Следват резултатите, получени при умножение на матрици с размерности (1800/600), (600/2048):

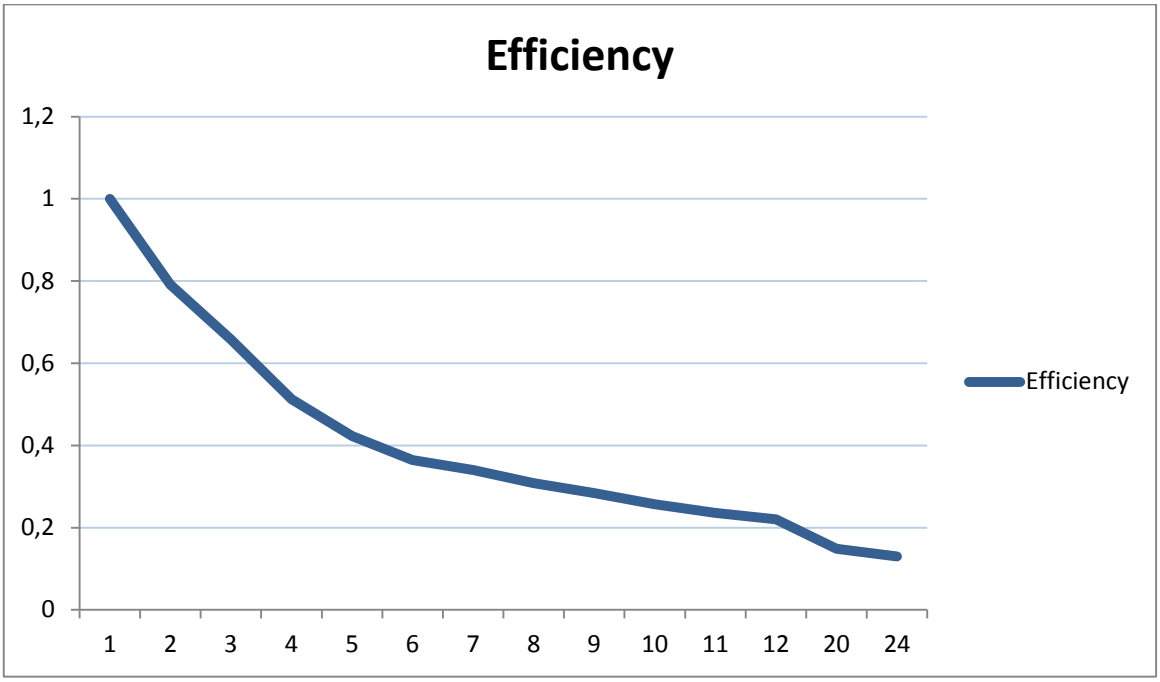
Графика на времето за изчисление



Графика на ускорението

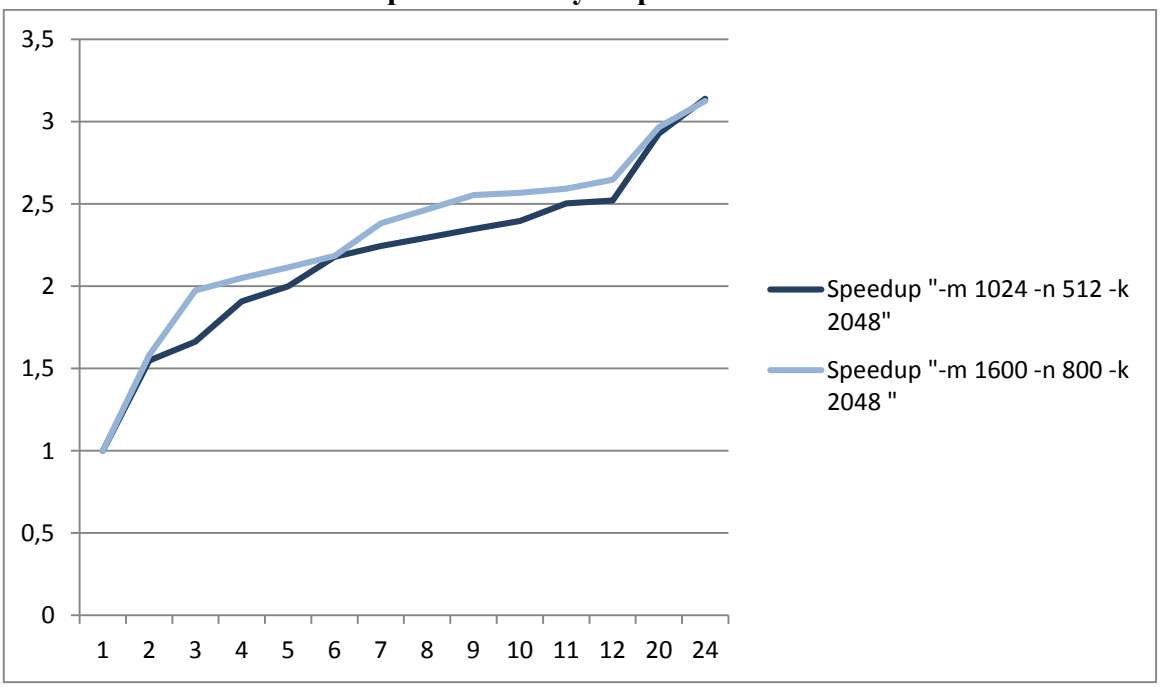


Графика на ефективността

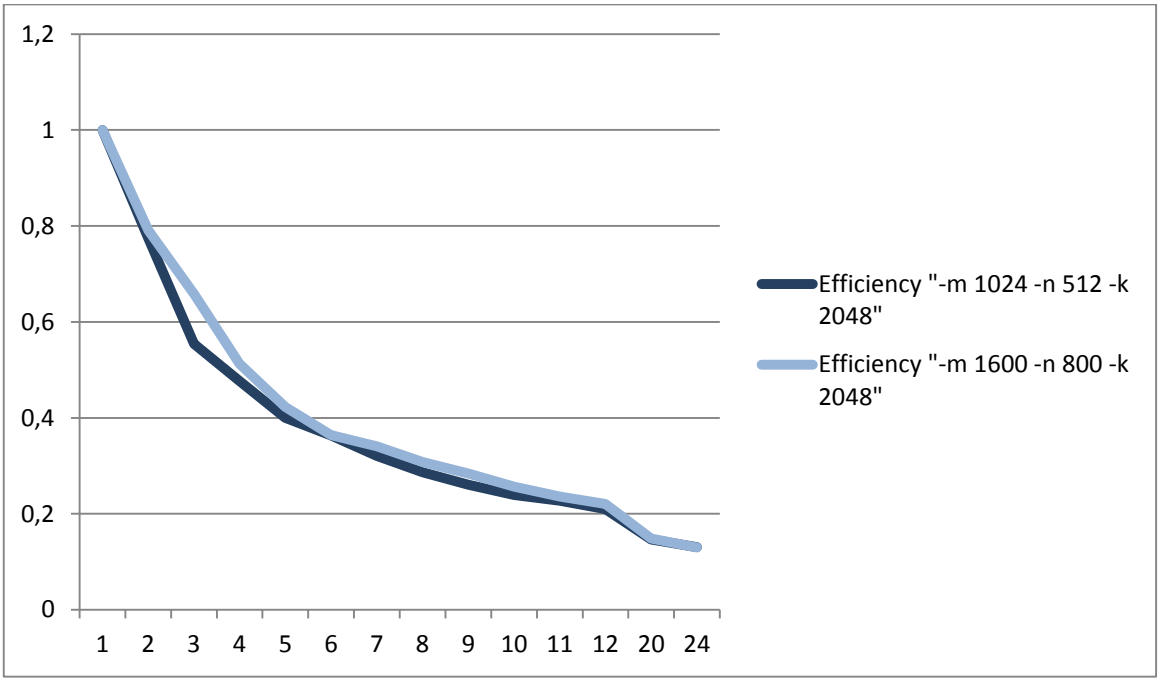


3.3 Сравнение на получените резултати

Сравнение на ускорението



Сравнение на ефективността



Резултати от тестовете, представени в таблица като всеки ред е съответно: брой нишки (Threads), време за пресмятане в милисекунди (Time), ускорение (Acceleration) и ефективност (Efficiency).

-m 1024 -n 512 -k 2048

Threads	Time	Acceleration	Efficiency
1	19128	1	1
2	12349	1,548951332	0,774475666
3	11500	1,663304348	0,554434783
4	10029	1,90726892	0,47681723
5	9568	1,99916388	0,399832776
6	8776	2,179580675	0,363263446
7	8522	2,244543534	0,320649076
8	8331	2,296002881	0,28700036
9	8145	2,348434622	0,26093718
10	7983	2,396091695	0,23960917
11	7643	2,502682193	0,227516563
12	7588	2,520822351	0,210068529
20	6535	2,927008416	0,146350421
24	6093	3,139340226	0,130805843

-m 1600 -n 800 -k 2048

Threads	Time	Acceleration	Efficiency
1	84677	1	1
2	53520	1,582156203	0,791078102
3	42874	1,975019826	0,658339942
4	41300	2,050290557	0,512572639
5	40058	2,113859903	0,422771981

6	38753	2,185043739	0,364173956
7	35522	2,383790327	0,340541475
8	34331	2,466488014	0,308311002
9	33145	2,554744305	0,283860478
10	32983	2,567292241	0,256729224
11	32643	2,594032411	0,235821128
12	31988	2,647148931	0,220595744
20	28535	2,967478535	0,148373927
24	27093	3,12541985	0,130225827