

**Софийски Университет „Св. Климент Охридски”**

**Факултет по математика и информатика**

**Курсов проект**

**по Разпределени софтуерни архитектури**

**Тема:** *Детерминанта на матрица*

Изготвил: Велизар Връбчев, фак.№: 62098, 3 курс, Софтуерно инженерство

Ръководител: ас. Христо Христов

Дата: 13.06.2019г. Проверил: ………………….. (подпис)

1. Условие на задачата

Разглеждаме матрицата A с размерност (n, n). Да се напише програма която пресмята det A. Работата на програмата по пресмятането на детерминантата да се раздели по подходящ начин на две или повече нишки (задачи).

Изискванията към програмата :

(o) Размерността на матрицата се задава от подходящо избран команден параметър – например “-n 13”; Елементите на матрицата генерираме произволно.

(о) Команден параметър указващ входен текстов файл, съдържащ матрицата, чийто детерминанта ще пресмятаме – например „-i m3x-data.in“. Параметрите „-n“ и „-i“ са взаимноизключващи се. Форматът на файла m3x-data.in е следният:

=== цитат ===

n

a11 a12 a13 …a1n

a21 a22 a23 … a2n

an1 an2 an3 … ann

=== цитат ===

(о) Команден параметър указващ изходен файл, съдържащ резултата от пресмятането – например „-о m3x-data.out“.

(о) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на det A – например “–t 1”

(о) Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението; Примери за подходящи съобщения: „Thread- started.“, „Thread- stopped.“, „Thread- execution time was (millis): “, „Threads used in current run: “, „Total execution time for current run (millis): “ и т.н.

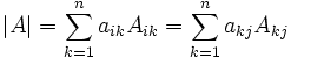
(o) Да се осигури възможност за „quiet“ режим на работа на програмата, при който се извежда само времето отделено за изчисление на det A, отново чрез подходящо избран друг команден параметър – например “-q”;

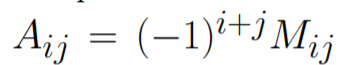
(о) Изчислението на det A може да бъде извършено с помощта на адюнгирани количества и развитието на детерминантата по ред или стълб;

1. Описание на решението

Проектът е написан на програмния език Java, като конзолно приложение. За пресмятането на детерминанта е приложено Правилото на Лаплас, което гласи : “Всяка детерминанта е равна на сумата от произведенията на елементите от произволен ред (стълб) със съответните им адюнгирани количества.“

Т.е. в сила следната формула :





Алгоритъмът използван, за решаването на задачата, е именно с тази формула. Като тук ще работим с елементите от 1-вия ред на матрицата и пресмятаме техните адюнгирани количества с рекурсивна функция, която прилага същата формула всеки път за нова матрица с една размерност по-малка от предходната(т.е с n-1).

В програмата има два основни класа - **Main** и **CalculateElementQuantitie**, който имплементира **Runnable** и се грижи за пресмятането на всяко адюн. количество на даден елемент. Тук забелязваме, че тези пресмятания са индивидуални за всеки елемент от реда и не се налагат от някаква синхронизация. За всеки елемент се пуска нишка(**new CalculateElementQuantitie**), която пресмята резултата от елемента \* дет. на въведената подматрица (без реда и стълба, в които се намира елемента, обвързан с тази нишка) чрез методите си - **determinantOfMatrix** и **getCofactor**. В **run** метода на **CalculateElementQuantitie** присвояваме на член данната **value** пресметнатото адюнгирано количество \* съответния елемент. Всеки новосъздаден обект от тип **CalculateElementQuantitie** се запазва в масив.Резултатите от всички пресмятанията се взимат циклично, от всеки един обект в масива, с **getValue** и се прилага сумирането по горепосочената формула.

За управлението на нишките се използва **ExecutorService** обект, който е **newFixedThreadPool(num\_of\_threads)**. T.e. по този начин може да преизползваме вече създадените **num\_of\_threads** на брой нишки, а не да създаваме нови. Ако броят на нишките е по-малко от броят на елементите, които трябва да бъдат пресметнати, се изчаква освобождаване на заета нишка, която да поеме пресмятанe на следващ елемент. Пускането на нишка и сумирането на получените стойности се случва в **calculateDeterminant** функцията на **Main** класа. В тази функция създаваме масив от новоизпълнените нишки, а също и **awaitTermination** метод на **ExecutorService** класа, за да може да се изчака приключването на всички нишки. Нужно e нишките да приключат с изчисления, за да може правилно да се сумират тези пресмятания. Останалите статични методи в **Main** класа са за четене от файл, записване във файл, генериране на матрица и изкарването и на екрана - **readFile**, **saveToFile**, **createMatrix**, **display**.**[1]**

В **main()** метода на **Main** класа се реализират основно проверките за входните параметри и съответните извиквания на функциите от **[1]** според тези параметри.

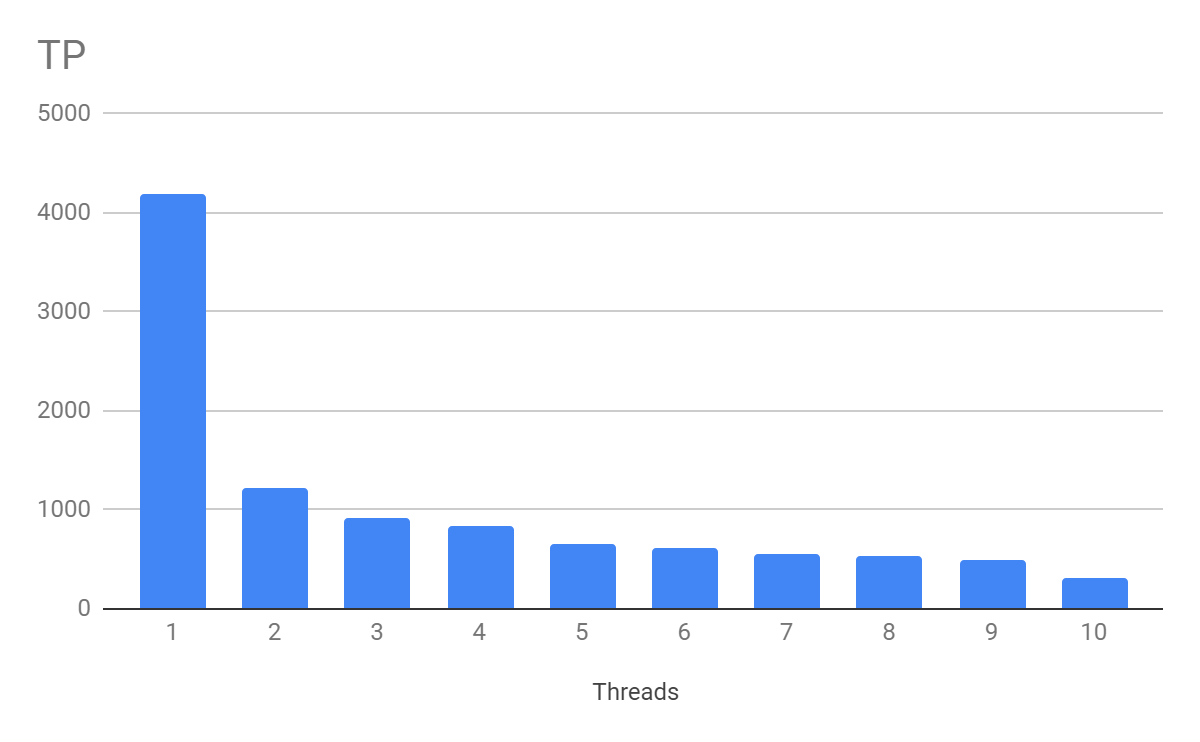
3. Тест за проверка на Ускорение, Ефективност и Бързодействие

За тестовете използвах матрица 10х10, като беше генерирана с **“-n 10”** команден параметър. Тестването беше проведено на сървъра [t5600.rmi.yaht.net](http://t5600.rmi.yaht.net/).

В таблицата са описани броя на нишките, **Тp** - време за изпълнение на програмата върху определен брой нишки, **Sp** - ускорението (Sp = T1 / Tp), **Ep** - ефикастнос (Ep = Sp / p), където p - брой нишки.

|  |  |  |  |
| --- | --- | --- | --- |
| Threads | TP | SP | EP |
| 1 | 4187 | 1 | 1 |
| 2 | 1217 | 3,44042728 | 1,72021364 |
| 3 | 926 | 4,521598272 | 1,507199424 |
| 4 | 836 | 5,008373206 | 1,252093301 |
| 5 | 664 | 6,305722892 | 1,261144578 |
| 6 | 605 | 6,920661157 | 1,153443526 |
| 7 | 558 | 7,503584229 | 1,071940604 |
| 8 | 543 | 7,710865562 | 0,9638581952 |
| 9 | 498 | 8,407630522 | 0,9341811691 |
| 10 | 305 | 13,72786885 | 1,372786885 |

Диаграма за изпълнение на програмата върху определен брой нишки. Стойностите по Y-оста е времето пресметнато в ms.



Диаграма за ускорението и ефикасността.

