Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика

ПРОЕКТ

по учебната дисциплина

Разпределени софтуерни архитектури

Тема: *"Детерминанта на матрица"*

Изготвил:
Никол Емануилова, 61987, Софтуерно инженерство, 3 курс, 2 група

Научен ръководител:
проф. д-р Васил Цунижев, ас. Христо Христов

Дата: 07.06.2019 **Проверка:** ……...

Съдържание

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 са следните:

- Размерността на матрицата се задава от подходящо избран команден параметър – например “-n 13”; Елементите на матрицата генерираме произволно с помощта на Math.random() (класа java.util.Random) или java.util.concurrent.ThreadLocalRandom; (Тоест матрицата може да има float или double елементи).

- Команден параметър указващ входен текстов файл, съдържащ матрицата, чийто детерминанта ще пресмятаме – например „-i m3x-data.in“. Параметрите „-n“ и „-i“ са взаимноизключващи се; Ако все пак бъдат зададени и двата решението как да реагира програмата е Ваше. Форматът на файла m3x-data.in е следният:

  === цитат ===
  n
  a11 a12 a13 … a1n
  a21 a22 a23 … a2n
  …
  an1 an2 an3 … ann
  === цитат ===
  Тоест:
  Първият ред съдържа единствено число, указващо размерността на матрицата.
  На оставащите n реда във файла са разположени редовете от матрицата чиято детерминанта ще пресмятаме. Елементите на всеки ред от матрицата са разделени със интервали.
- Команден параметър указващ изходен файл, съдържащ резултата от пресмятането – например „-o m3x-data.out“. Форматът на изходният файл можете да определите сами, стига файлът да е текстов; При липса на този команден параметър не се записва във файл резултата от пресмятането на детерминантата;

- Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на det A – например “–t 1” или “–tasks 3”;

- 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-<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.
- 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”;

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

2. Увод - цел на проекта, алгоритъм и функционалност

Намирането на детерминантата на матрица е важна тема, която е полезна в много области на математиката. Именно детерминантите са довели до изучаването на линейната алгебра. Въпреки че отношението към тях е променено, теоретичното им значение все още може да бъде намерено. През историята детерминантите намират много приложения за решаването на математически проблеми като решаване на системи от линейни уравнения, намиране на площта на триъгълник, тестване на колинеарни точки, намиране на уравнение на линия и други геометрични проблеми. Различни приложения има и в абстрактната математика, квантовата химия и други.

Съществуват различни начини за намиране на детерминанта, като най-често срещаният е чрез теоремата на Лаплас или развиване на детерминанта по даден ред или стълб. Правилото гласи, че всяка детерминанта е равна на сумата от произведенията на елементите от произволен ред (стълб) със съответните им адюнгирани количества.

$$|A| = \sum_{k=1}^{n} a_{ik} A_{ik} = \sum_{k=1}^{n} a_{kj} A_{kj}$$

При този алгоритъм рекурсивно се развива матрицата до подматрици, докато се стигне до базовия случай - матрица с размерност 2x2. Сложността му е O(n!).

Съществуват други методи за намиране на детерминанта чрез декомпозиция със сложност $O(n^3)$, като един от тях е LU-декомпозиция и именно той е използван за реализирането на текущия проект. LU-декомпозицията е модифицирана версия на Гаусовия метод за елиминация. Методът работи като се разбива голяма матрица A на произведение от две матрици - долната матрица L и горната U, които са по-прости за изчисляване.

$$|A| = |L| \cdot |U|$$

Декомпозицията се извършва чрез разделяне на матрицата A по диагонал, така че всички елементи над диагонала на матрицата L да са 0 и всички елементи под диагонала на матрица U също да са 0. Освен това, елементите по диагонала на матрицата L са 1-ци, за да бъде пресмятането по-лесно.

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{vmatrix} \cdot \begin{vmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{vmatrix}$$

За тази матрица резултатът от декомпозицията ще бъде 9 неизвестни числа и 9 разрешими уравнения от произведението на двете матрици. Общите формули за L и U са:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} u_{kj} \cdot l_{ik}$$

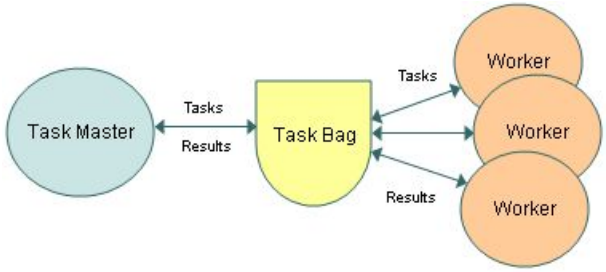
$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} u_{kj} \cdot l_{ik} \right)$$

Този алгоритъм е доста по-добър, както по сложност, така и по време за изпълнение.

3. Проектиране

Алгоритъмът за намиране на детерминанта чрез LU-декомпозиция е имплементиран на Java 11. Реализиран е паралелизъм по данни (SPMD - Single Program Multiple Data), тоест проблемът се разделя на задачи, които се пускат едновременно с различен вход от данни, с цел резултатът да бъде изчислен по-бързо.

Постигането на паралелизъм е осъществено чрез прилагането на шаблон (design pattern) за използването на нишки - Bag of Tasks. Този шаблон представлява набор от работници, които поддържат динамична "чанта" от хомогенни задачи. Всички задачи са независими една от друга, за да може всеки работник да поеме задача, да я изпълни и да се върне, за да вземе следваща. Това продължава, докато чантата остане празна, т.е. всички задачи са изпълнени и финалният резултат може да бъде получен.



Фиг. 1 Design Pattern "Bag of Tasks"

За ефективното смятане на детерминанта чрез LU-декомпозиция "чантата" се пълни с набор от подзадачи, чиито брой съответства на размерността на матрицата. Всяка една задача отговаря за изчисляването на собствена стойност. Изпълнението им се осъществява с помощта на **Executor Framework** на Java или по-точно обект от тип **ExecutorService.** Той се инициализира чрез статичния метод **Executors.newFixedThreadPool(nThreads)**, където **nThreads** е полученият като аргумент на програмата брой нишки. По подразбиране броят им е 1, т.е. програмата се изпълнява като серийна. Списъкът със задачите се подава на ExecutorService и той се грижи за разпределянето на задачите. По този начин се създава "басейн" от фиксиран брой нишки, които се преизползват за изпълнението на неограничен брой задачи. Следователно, ако броят нишки е по-малък от броя задачи, то при по-бързо приключване на дадена нишка тя веднага поема изпълнението на следващата задача.

Архитектурата на програмата следва принципите на обектно-ориентираното програмиране. Най-високо в йерархията е интерфейсът **Matrix**, който описва основните дейности, които могат да се извършват с матрица - да бъде инициализирана, да се намери нейната детерминанта и да се запише резултатът във файл. На следващото ниво е абстрактният клас **BaseMatrix**, който осигурява имплементацията на методите за изчисляване на детерминанта и записване на резултата във файл. Това са основните функционалности на програмата. Методът за инициализиране на матрица е абстрактен (без имплементация), защото има два възможни начини за това - чрез генериране на случайни числа или чрез прочитане на матрица от подаден файл. Съответно, двата класа **RandomMatrix** и **FileMatrix** наследяват **BaseMatrix** и имплементират нужния метод за задаване на матрица. Класът **DeterminantCalculator** е стартиращата точка за програмата, която обработва подадените аргументи и подава нужните данни на останалите обекти, за да бъде намерена детерминантата.

4. Тестване

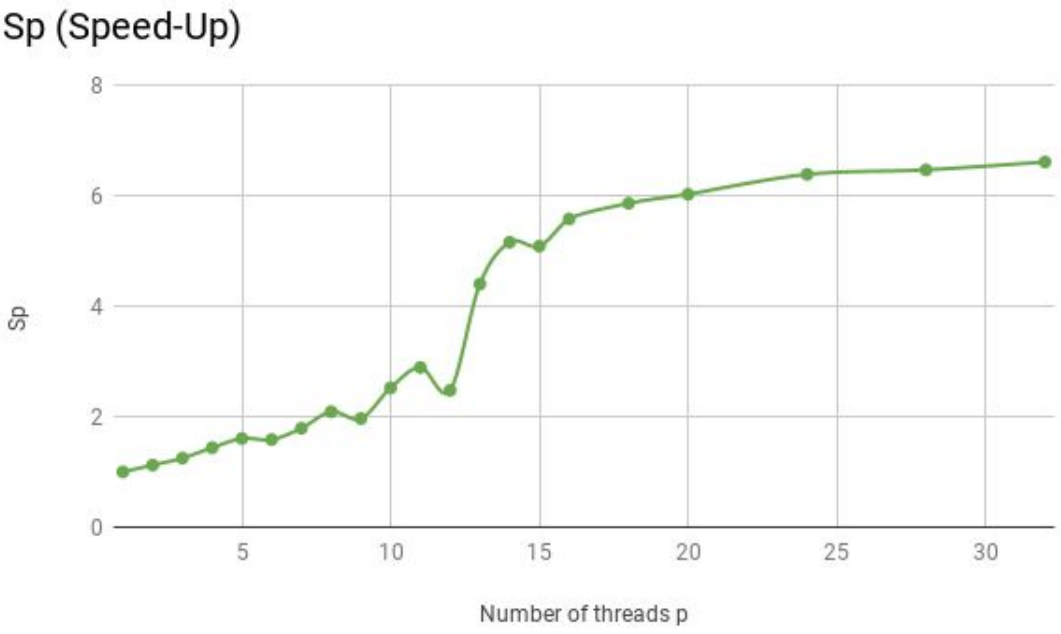
На предоставена машина за извършване на тестовете t5600.rmi.yaht.net са проведени тестове за намиране на детерминанта на матрица с размерност 160x160. Стойностите на матрицата са прочетени от входния файл m3x-data-2.txt. В следната таблица са представени данните получени при извършване на тестовете:

Брой нишки (p)	Време за изпълнение в ms (Tp)	Ускорение (Sp)	Ефективност (Ep)
1	1500	1	1
2	1335	1,123595506	0,5617977528
3	1201	1,248959201	0,4163197336
4	1043	1,438159156	0,3595397891
5	935	1,604278075	0,320855615
6	946	1,585623679	0,2642706131
7	838	1,789976134	0,2557108762
8	718	2,08913649	0,2611420613
9	763	1,965923984	0,2184359983
10	595	2,521008403	0,2521008403
11	519	2,89017341	0,2627430373
12	485	2,47628866	0,2063573883
13	341	4,398826979	0,3383713061
14	291	5,154639175	0,3681885125
15	295	5,084745763	0,3389830508
16	269	5,576208178	0,3485130112
18	256	5,859375	0,3255208333
20	249	6,024096386	0,3012048193
24	235	6,382978723	0,2659574468
28	232	6,465517241	0,23091133
32	227	6,607929515	0,2064977974

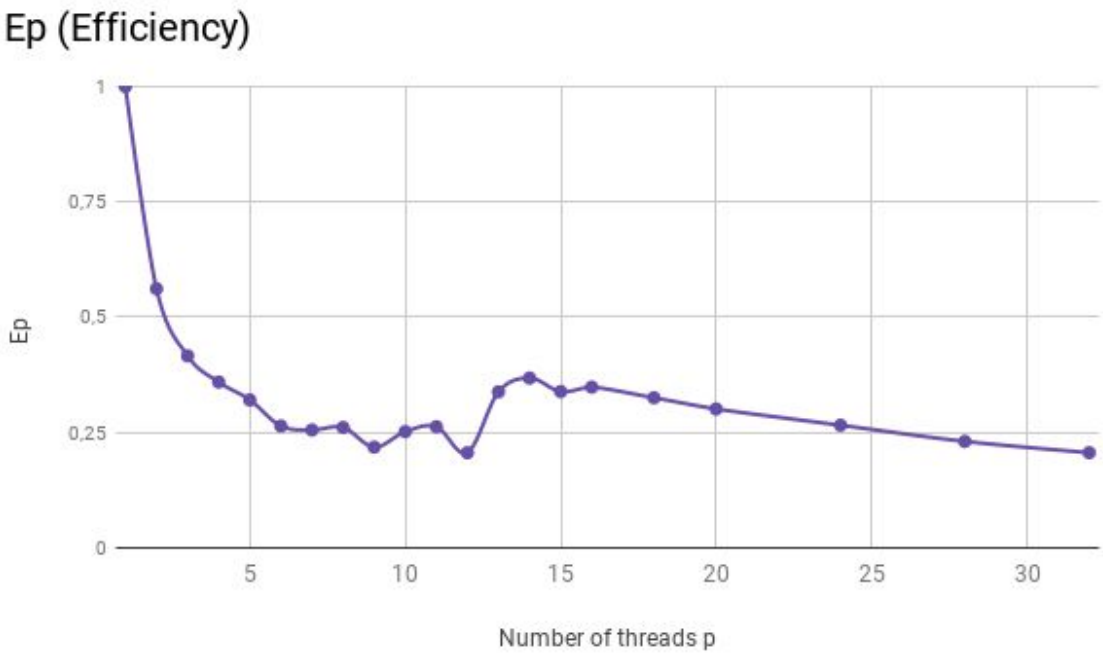
Фиг. 2 Таблица с резултати от тестването

T1 – времето за изпълнение на серийната програма (използваща една нишка)
Tp – времето за изпълнение на паралелната програма, използваща p нишки
Sp = T1/Tp – ускорението, което програмата има при използването на p нишки
Ep = Sp/p – ефективността на програмата при използването на p нишки

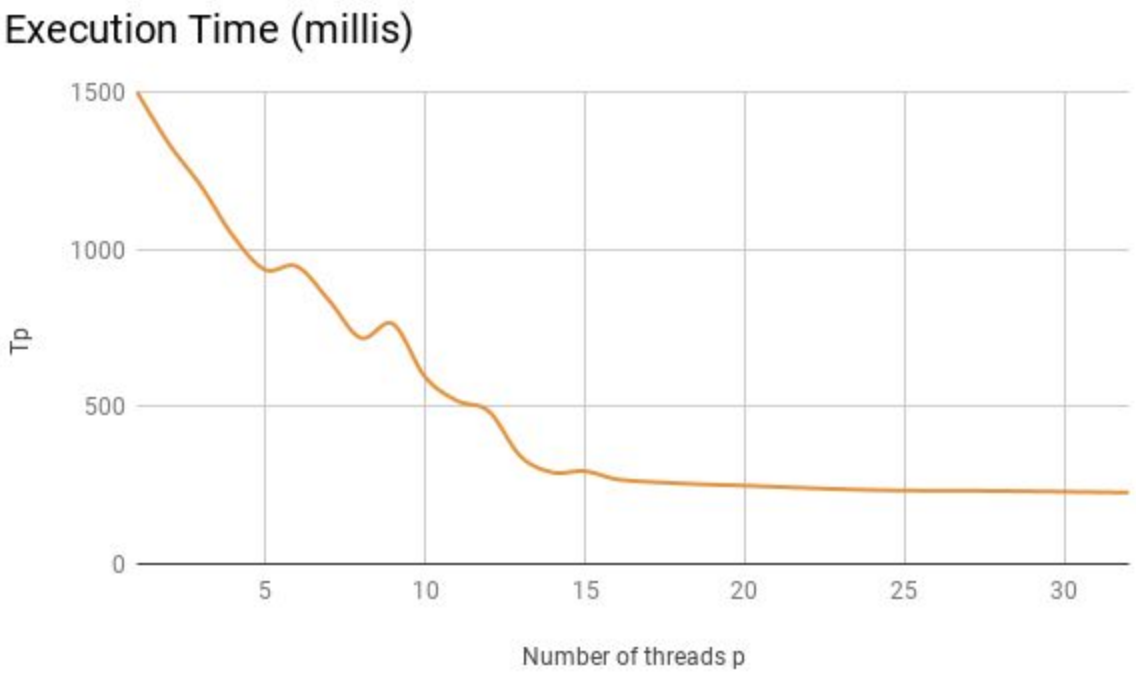
Данните от горната таблица са изобразени и графично чрез диаграми, които показват нагледно постигнатите резултати откъм ускорение, ефективност и време за изпълнение на програмата. Стойностите на тези критерии са сравнително еднакви при изчисляването на детерминанти на матрици с малка размерност, затова са демонстрирани резултатите при големи матрици NxN (в случая N=160). В общия случай това изисква големи изчислителни ресурси, но реализираният алгоритъм осъществява пресмятанията за доста кратко време.



Фиг. 3 Ускорение Sp при p нишки



Фиг. 4 Ефективност Ep при p нишки



Фиг. 5 Време за изпълнение Tp в ms при p нишки

5. Източници

[1] Determinant - Wikipedia
(https://en.wikipedia.org/wiki/Determinant)

[2] New parallel algorithms for finding determinants of N×N matrices, Sami Almalki
(https://www.researchgate.net/publication/261263046_New_parallel_algorithms_for_finding_determinants_of_NN_matrices)