

Детерминанта на матрица

Проект

по Системи за паралелна обработка

Изготвил: Здравко Петров, ф.н. 81279

1. Цел на проекта

Целта на проекта е реализация на паралелен алгоритъм пресмятащ детерминантата на матрица, посредством формулата



където A е квадратна матрица, която е произволно генерирана по зададен размер n или е предварително генерирана и записана във файл.

2. Изисквания към програмата

- Програмата трябва да използва паралелни процеси (нишки), за да разпредели работата по пресмятането на детерминантата на повече от един процесор.

- Размерността на матрицата се трябва да се зададе от подходящо избран команден параметър -n [Number].

- Команден параметър указващ входен текстов файл, съдържащ матрицата, чиято детерминанта ще пресмятаме –i [File]. Ако бъдат подадени и двата параметъра -n [Number] и -i [File] програмата по подразбиране ще ползва параметъра -n [Number]

- Команден параметър указващ изходен файл, съдържащ резултата от пресмятането -o [File].

- Команден параметър -t [Number] задаващ максималния брой нишки, с които програмата разполага за разпределянето на работата по пресмятането на детерминантата на матрицата A. Ако този параметър не бъде подаден, по поздразбиране програмата ще работи с -t 1, тоест една нишка.

- Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението.

- Да се осигури възможност за "quiet" режим на работата на програмата, при който се извежда само времето отделено за изчисление на детерминантата на матрицата, отново чрез команден параметър -q.

- Изчислението на детерминантата може да бъде извършено с помощта на адюнгирани количества и развитие на детерминантата по ред или стълб.

3. Реализация

Програмата е написана на езикът Java, версия 1.8. Програмата цели по най-оптимален начин да използва зададения от потребителя брой нишки. Тъй като те са ценен ресурс, ще бъде по-добре тяхната работа да се преизползва, тоест една нишка да работи по много задачи. За целта програмата използва **ExecutorService**, който е подинтерфейс на интерфейса **Executor.** Като имплементация на интерфейса ползваме **Executors.newFixedThreadPool(numberOfThreads)** който вътрешно ползва класа **ThreadPoolExecutor**. По този начин създаваме басейн с фиксиран брой нишки, които не завършват изпълнението си след приключване на дадена задача, което значи, че във всеки един момент на програмата, тя работи с по **numberOfThreads** задачи едновременно.

Чрез **ExecutorService** програмата пести ресурси, защото не създава нова нишка всеки път, а преизползва кешираните от басейна. Програмата ползва и други оптимизационни алгоритми на базата на броят нишки, с които тя разполага както и от големината на матрицата, с която тя работи. Чрез предварителни изчисления сме намерили, че намиране на детерминантата на матрицае с оптимален размер **11x11**(чрез повече от 1 нишка)**.** На базата на това и по броя подадени нишки програмата изчислява, кое е минималното ниво(големина на матрица), от което да започне за използва повече от една нишка, за да няма загуба на ресурси и overheating.

Основния клас на програмата е **Main**, чрез който тя се стартира. Първата стъпка на програмата е да обработи и валидира командните параметри които са подадени. Това става чрез класа **Parser** и метода **parse(String[] args);**



След като валидира, че всички командни аргументи са валидни, програмата вика метода **evaluateCommands(Map commands);**

чрез който започва същинското и изпълнение. След като програмата генерира матрицата, тя в зависимост от това дали ползваме повече от една нишка, инициализира басейна от нишки. Това става чрез статичния метод **ThreadPool.init(numberOfThreads);**



След приключване на програмата се вика статичния метод **ThreadPool.destroy();** чрез който работата на всички кеширани нишки приключва и освобождаваме техните ресурси.



Ако програмата не приключи за по-малко от **264 - 1** милисекунди, тя ще бъде принудително прекратена. Разбира се времето за изчакване може да бъде конфугурирано допълнително.

Ако програмата работи с една нишка басейна от нишки не се създава. За целта ползваме изцяло рекурсивен алгоритъм, който по метода на адюнгираните количества намира детерминантата на матрицата. Сложността на този алгоритъм е **O(n!)**;



Ако програмата работи с повече от една нишка ползваме същият рекурсивен алгоритъм, но паралелизиран използвайки басейна от нишки.



Чрез метода **buildSubMatrix(matrix, level);** програмата генерира подматрицата за дадено ниво, тоест премахва колоната за елемента от съответното ниво.



При изпълнение на дадена задача, програмата извежда в зависимост от командните параметри времето за нейното изпълнение както и нишката, която я е изпълнила. Също така програмата изчислява общото време на работа за всяка нишка от басейна по отделно, като по този начин, след нейното приключване извежда статистика на времената.





3. Стартиране на програмата

Програмата се стартира от командния ред, като трябва да се подадат следните параметри:

-n [Number], при подаване на този параметър казваме на програмата да генерира произволна матрица с подадения параметър

-i [FileName], при подаване на този параметър казваме на програмата да прочете матрицата и нейния размер от файл по подадено име

-o [FileName], при подаване на този параметър казваме на програмата да извежда информация от нейното изпълнение във файл по подадено име

-q, при подаване на този параметър програмата преминава в "тих" режим и извежда само общото време за пресмятане на детерминантата

-t [Number], при подаване на този параметър програмата определя максималния брой нишки, с които може да пресмята детерминантата

Задължителни параметри са -i или -n и -t. Ако параметърът -o не бъде подаден, то програмата ще извежда резултата само на стандартния изход. Ако параметърът -q не бъде подаден, програмата ще извежда подробра информация при пресмятането на детерминантата.

**Пример за стартиране на програмата в тих режим като записваме резултата във файл.**



**Пример за стартиране на програмата в нормален режим като резултата ще бъде изведен на стандартния изход.**



4. Резултати