Умножение на матрици

Проект по Системи за паралелна обработка

Изготвил:

Божин Кацарски
Ф.н. 81291

Проверил:

……...
Ас. Христо Христов

Съдържание

Условие на задачата

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).

Решение

Идея

По дефиниция две матрици могат да бъдат умножени само ако броят редове на първата съвпада с броят колони на втората.
Алгоритъмът за умножение неформално наричаме “ред по стълб”, а формално всяка клетка от резултатната матрица намираме по следната формула:

$$c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj},$$

където m = броят колони в първата матрица = броят редове във втората.

От формулата следва важният факт, че **изчисляването на една клетка Cx,y не зависи от изчисляването на никоя друга клетка Cz,t**.

Следователно можем да разделим задачата на подзадачи, всяка от които е да се изчислят някаква част от клетките. Така задачите са напълно независими една от друга и могат да се изпълняват в произволен ред, както и частично или напълно паралелно.

За решаването на всяка подзадача имаме нужда от:
1. Достъп за четене от двете входни матрици
2. Достъп за писане в изходната матрица

Това не пречи на работата на няколко нишки, защото конкурентното четене дори и от едни и същи променливи е позволено, а конкурентно писане в едни и същи променливи нямаме, защото всяка клетка от резултатната матрица се изчислява от точно една подзадача (нишка).

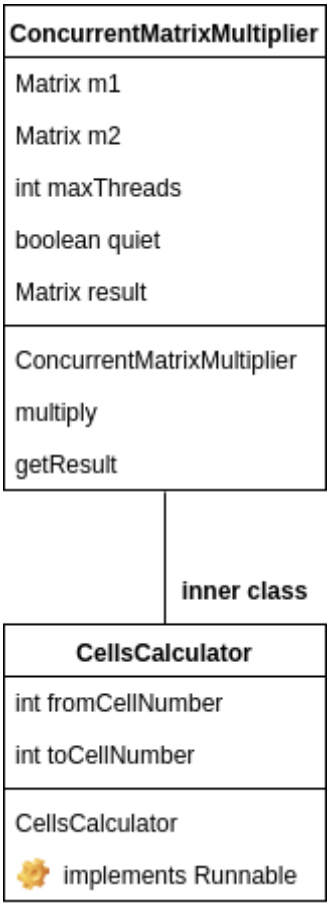
Реализация

Досега не срещнахме ограничения, които биха ни принудили да използваме конкретен език за програмиране, затова избираме например Джава.

Ще използваме следните класове:

- **Config** - отделя конфигурацията на програмата от същинския алгоритъм. Грижи се за обработване на командните аргументи, прочитане на матриците от файл или генериране на случайни такива, запазване на резултатната матрица в изходен файл.
- **Matrix** - реализира абстракцията матрица (от дробни числа) - нейните размерност, клетки, мутатори, селектори и др.
- **ConcurrentMatrixMultiplier** - реализира маркирания в предната точка алгоритъм. Отделянето му в самостоятелен клас предоставя модуларност на програмата и възможност за лесна смяна на имплементацията на умножаващия алгоритъм.
- **Main** - Входна точка на програмата - свързва останалите класове и измерва времето за изпълнение.

В повече подробности ще се спрем на класа **ConcurrentMatrixMultiplier**.



Конструкторът на класа приема като аргументи двете умножавани матрици, максималният брой нишки, които може да използва и опцията **quiet**, указваща тих режим без информационни съобщения на екрана.

Работата, която трябва да се извърши е да се изчислят **N** независими клетки, където **N** е броят клетки в резултатната матрица. Разделяме я възможно най-справедливо между всички нишки, като всяка ще изчисли по **N / maxThreads**, а евентуалния остатък ще бъде компенсиран от последната нишка.

Вътрешният подклас **CellsCalculator** имплементира интерфейса **Runnable** и в конструктора му се задават начален и краен номер на клетка, които трябва да изчисли.

Използваме следната номерация на клетките:
Клетките на матрицата са номерирани стандартно от 0 до **N-1**, по редове и после по колони, като ако **C** е броят колони в резултатната матрица, то:

- По номер на клетка **n** намираме самата клетка **(n / C, n % C)**
- Клетката **(i, j)** намираме номера и **(i - 1) * C + j**.

Така нишка номер 0 изчислява клетките с номера **[0, N / maxThreads)**, нишка номер 1 изчислява клетките с номера **[N / maxThreads, 2 * N / maThreads)** и т.н. Нишката с номер **maxThreads - 1** изчислява необходимите клетки до края на матрицата.

Изпробване на програмата

Програмата може да приема няколко командни аргумента, които указват действието и.

1. Матриците, които се умножават, биха могли да се зададат по два начина:
    a. Да се прочетат от входен файл, зададен чрез опцията **-i**
    b. Да се генерират с произволни стойности по подадени измерения чрез опциите **-m**, **-n** и **-k**
2. Евентуално може да се подаде име на изходен файл с опцията **-o**, където да се запише резултатът от умножението.
3. Евентуално може да се подадат максималния брой подзадачи, на които програмта да раздели работата чрез опцията **-t**. По подразбиране е 1, тоест последователна обработка.
4. Евентуално може да се подаде опцията **-q**, която означава тих режим - извежда се само времето за умножение на двете матрици без други информационни съобщения.

Проведени тестове и измервания

На предоставената машина за извършване на тестове са проведени тестове за умножаване на две матрици, съдържащи произволно генерирани дробни числа в интервала (0; 1).

За размерност на матриците избрахме 4000 x 4000 - достатъчно големи, за да виждаме реална полза от използването на много паралелни нишки.
Бяха проведени тестове, в които броят използвани процесори ядра е 1 (последователна обработка), 4, 8, 12, 16, 20, 24, 28 и всичките 32 ядра на тестовата машина. При това беше измерено времето на същинско умножение и бяха изведени следните метрики - **време за изпълнение**, **ускорение** и **ефективност** спрямо броя използвани процесорни ядра.

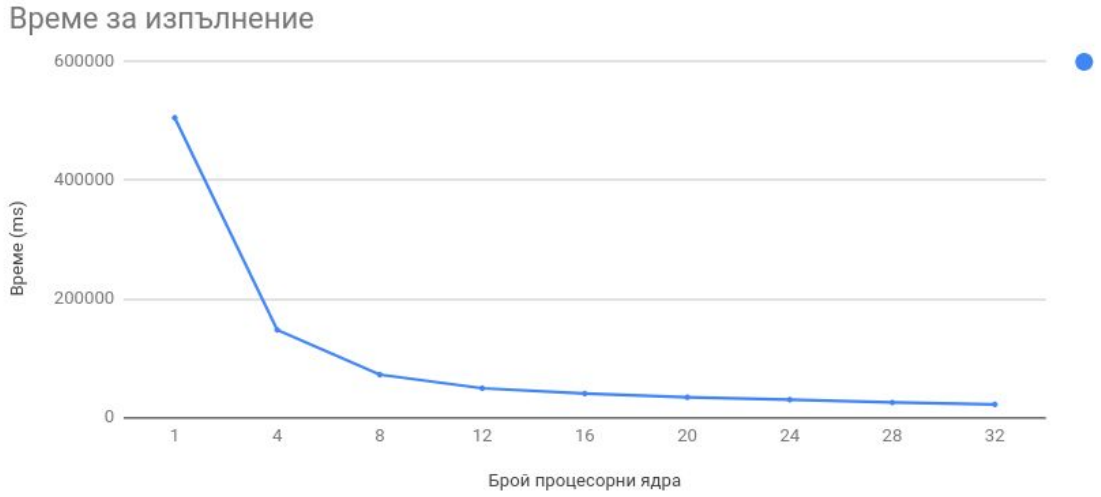
Тестовата машина беше използвана и от други потребители по време на тестовете, което означава, че не всички нейни ресурси са били на разположение за процеса на нашите тестове. Представените данни са осреднени от няколко опита и целят да представят в максимална степен данни, неповлияни от други потребители на системата.

Таблични данни

Брой процесорни ядра	**Време за изпълнение**	**Ускорение**	**Ефективност**
1	505564	1	1
4	147587	3.425532059	0.8563830148
8	72057	7.016167756	0.8770209695
12	49162	10.2836337	0.8569694751
16	40023	12.63183669	0.7894897934
20	33893	14.91647243	0.7458236214
24	29895	16.91132296	0.7046384568
28	25067	20.1685084	0.7203038713
32	21839	23.14959476	0.7234248363

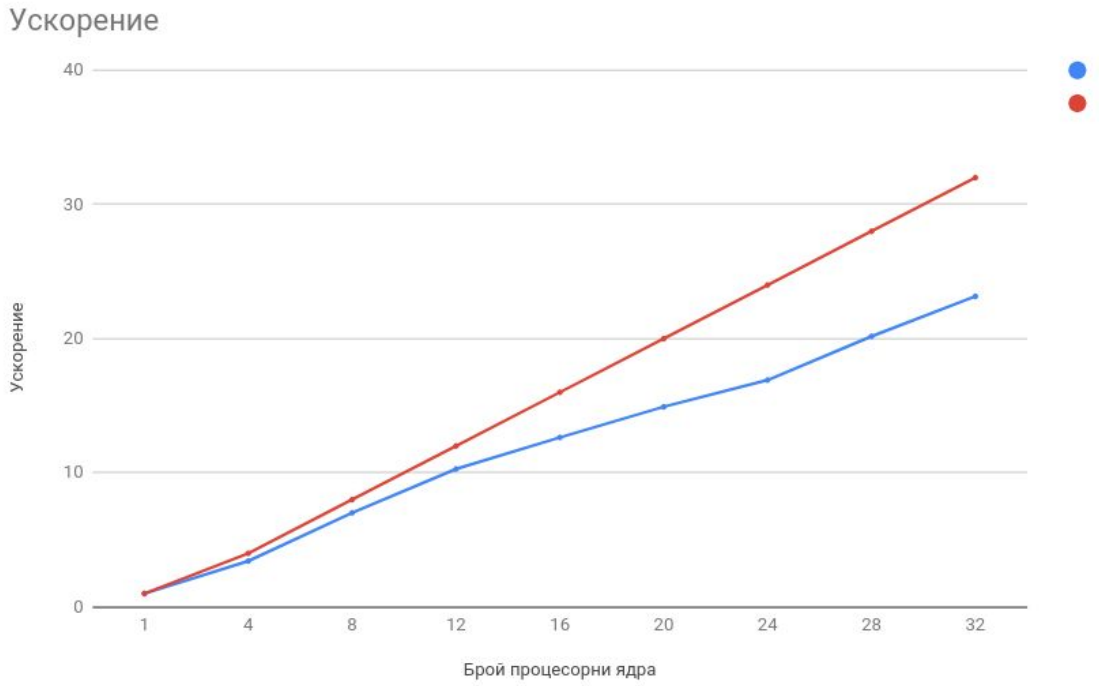
Време за изпълнение

Засеченото време е в милисекунди и включва само същинското умножение, без обработване на командни аргументи, входни и изходни файлове.
Бележим времето за изпълнение на програмата с **p** нишки с **Tp**.
T1 е времето за изпълнение на процеса с 1 нишка, тоест последователната реализация.



Ускорение

Бележим ускорението с **Sp = T1 / Tp** - отношението на времето за работа на серийния процес към времето за работа на процеса, използващ **p** нишки.
В синьо е показано реалното измерено ускорение, а в червено - перфектното теоретични такова.



Ефективност (ефикасност)

Ефективността се дефинира като отношението на ускорението на процеса, използващ **p** нишки към самото число **p** и се бележи с **Ep = Sp / p**.