СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

Курсов проект

Системи за паралелна обработка. Умножение на матрици

Изготвил:

Димитър Милев фак. № 81352

Специалност: Компютърни Науки

Научен ръководител: ас. Христо Христов

София
06.2018г.

Съдържание

1. Въведение

Целта на настоящия курсов проект е да се напише програма, която реализира умножаване на матрици, като работата на програмата трябва да се раздели по подходящ начин на две или повече нишки (задачи). В секцията '*1.Въведение*' се описва темата на проекта и се дават изискванията към задачата. В секция '*2. Проектиране*' е представен алгоритъмът, който се използва за решаването на задачата, и реализацията му. В '*3. Тестване и резултати*' се вижда колко добре се справя направеното решение като се измерват основните характеристики – време за изпълнение, ускорение и ефективност.

1.1 Тема на проекта

Темата на проекта е „Умножение на матрици". Разглеждат се 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.

1.2 Изисквания

Изискванията към програмата са следните:

- Размерността на матриците се задава от подходящо избрани командни параметри – **„-m 1024 -n 512 -k 2048"**;
- Команден параметър указващ входен текстов файл, съдържащ матриците, които ще се умножават –**„-i m3x-data.in"**. Който и да е от параметрите **„-m" „-n" „-k"** и параметърът **„-i"** са взаимно-изключващи се (приоритетни са **„-m" „-n" „-k"**);
- Команден параметър указващ изходен файл, съдържащ резултата от пресмятането – **„-o m3x-data.out"**;
- Друг команден параметър задава максималния брой нишки (задачи), на които се разделя работата по пресмятането елементите на C – **"–t 1" или "–tasks 3"**;
- Извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление;
- 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 – **"-q";**

2. Проектиране

Основният фокус при решаването на задачата ще е бързодействието и ускорението (съответно и ефективността) при увеличаване нивото на паралелизъм.

Един от най – използваните алгоритми за умножение на матрици е този на Volker Strassen, но поради специфичното му разделяне на матриците на степен на двойката части, няма да бъде използван, защото при нечетно ниво на паралелизъм няма да се постига значително ускорение. „Coppersmith–Winograd“ алгоритъмът също няма да бъде ползван заради практически невъзможната му имплементация.

За това, решението на задачата ще бъде дадено от класическия алгоритъм за умножение на матрици, тъй като то е много подходящо за разделяне на подзадачи.

2.1 Описание на алгоритъма

Класическият алгоритъм за умножаване на матрици на практика е реализация на математическата дефиниция.

Умножават се матрици

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}.$$

Резултатът е матрицата

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$$

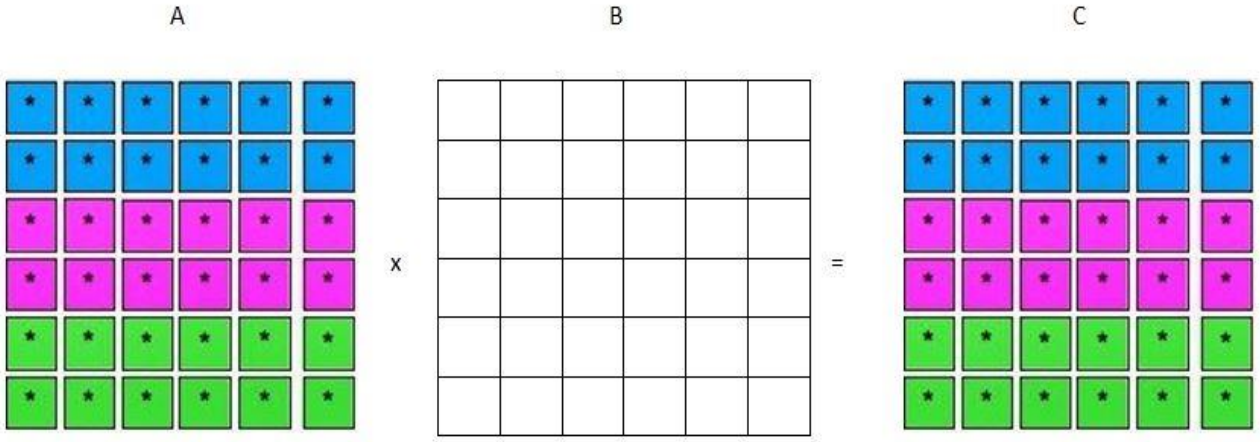
, където $c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj},$

for $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

Алгоритъмът, който ще се имплементира в този проект, ще е много подобен на класическия, но тъй като целта е да се търси ускорение при увеличаване на нивото на паралелизъм, ще има няколко модификации.

Важно е да се забележи, че при класическото изчисляване на умножение на матрици получаването на един елемент от резултатната матрица е абсолютно независимо от получаването на друг. Именно върху това наблюдение ще се изгради паралелния алгоритъм. Разбиването на задачата в подзадачи ще е върху входната матрица ***A***. Спрямо нивото на паралелизъм ***p*** матрицата ***A*** ще бъде разделяна вертикално на ***p*** сектора. Всяка подпрограма ще използва един от тези сектори и матрицата ***B*** и ще изчислява съответния сектор в резултатната матрица ***C***.

Графически, това се изобразява по следния начин:



2.2 Реализация

Реализацията на алгоритъма ще бъде направена на програмния език **C++** и по-конкретно ще бъде използвана версия **C++11** на стандарта. За компилатор ще се ползва **g++**, версия 4.8.5. Ще бъдат използвани стандартни библиотеки като **thread**, **string** и др. За "parse-ването" на командните аргументи ще се използва помощната библиотека **program_options** на **boost**.

Основната логика на програмата се случва в **main** функцията. Тя се грижи за прочитането на командните аргументи и за проверката им за коректност. Прочита / генерира матриците с помощта на **read_matrices()** / **generate_random_matrix()**. След това разделя матрицата **A** на равни по големина **t** брой сектора, където **t** е по-малкото от

редовете на матрицата **A,** т.е. командния параметър **n**, и командния параметър **tasks**, и пуска толкова на брой нишки **std::thread**, всяка от които изпълняващи функцията **matrix_mult()**.

3. Тестване и резултати

Програмата бе тествана на предоставения сървър с **hostname**: *t5600.yaht.net* и характеристики:

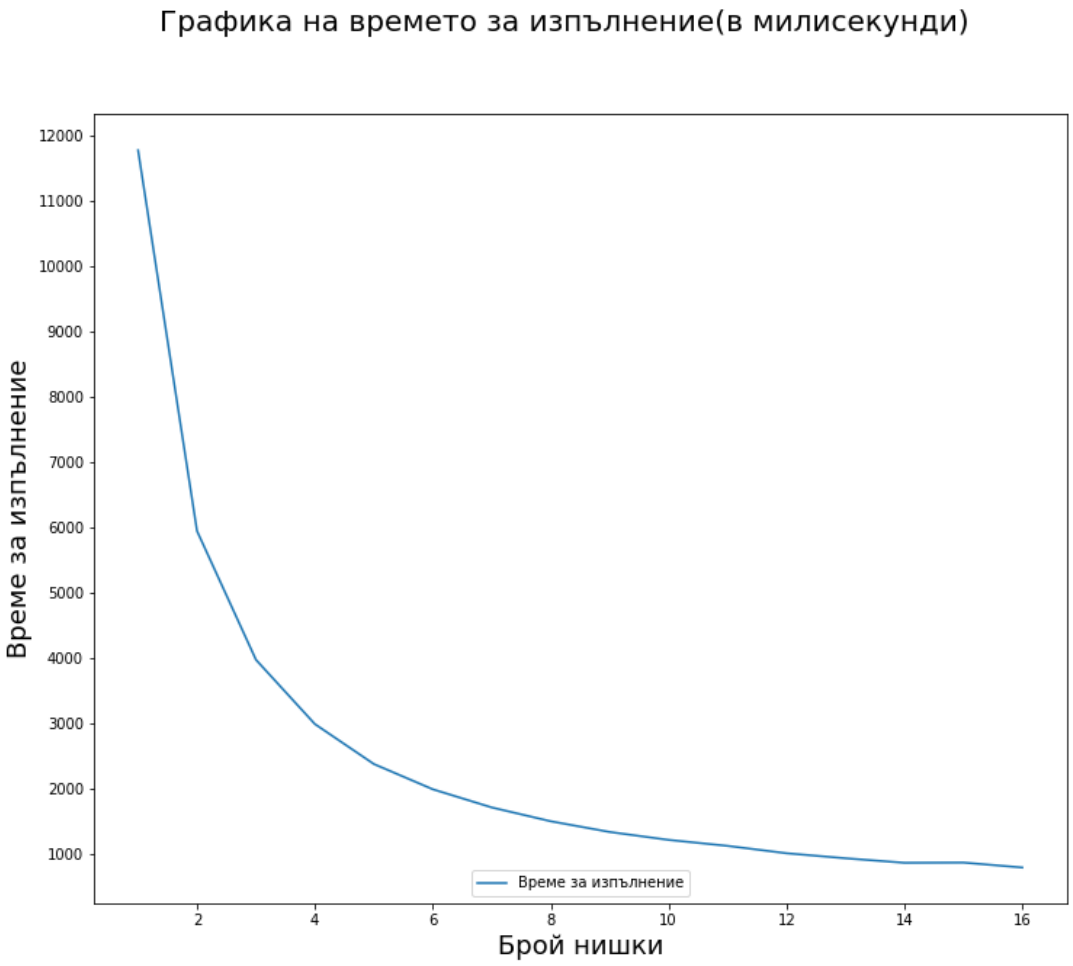
- два процесора Intel(R) Xeon(R) CPU E5-2660 2.20GHz с по 8 ядра всеки
- 62 GB RAM
- операционна система CentOS Linux 7
- ядро Linux 3.10.0-862.3.2.el7.x86_64

Тестовете бяха провеждани върху матрици с размерност *1024x1024*. За целта беше използван помощен скрипт написан на ***python***, който пуска програмата с 1-16 нишки по 5 пъти за всеки брой нишки и изчислява средното. Резултатите се записват във файл и след това се използват за визуалиция отново с програмния език ***python***(използвани библиотеки – ***matplotlib***, ***pandas*** и ***numpy)***. Резултатите от тестването са обобщени в следната таблица:

Num of threads	**Elapsed time**	**Speedup**	**Efficiency**
1	11776	1	1
2	5950	1.97916	0.98958
3	3979.2	2.959389	0.986463
4	2995	3.931886	0.982972
5	2384	4.939597	0.987919
6	1997.4	5.895664	0.982611
7	1719.4	6.848901	0.978414
8	1508.2	7.807983	0.975998
9	1344.2	8.760601	0.9734
10	1223	9.628782	0.962878
11	1131.4	10.40834	0.946213
12	1018.6	11.56097	0.963414
13	941.6	12.50637	0.962029
14	872.8	13.49221	0.963729
15	876	13.44292	0.896195
16	800.4	14.71264	0.91954

3.1 Време за изпълнение

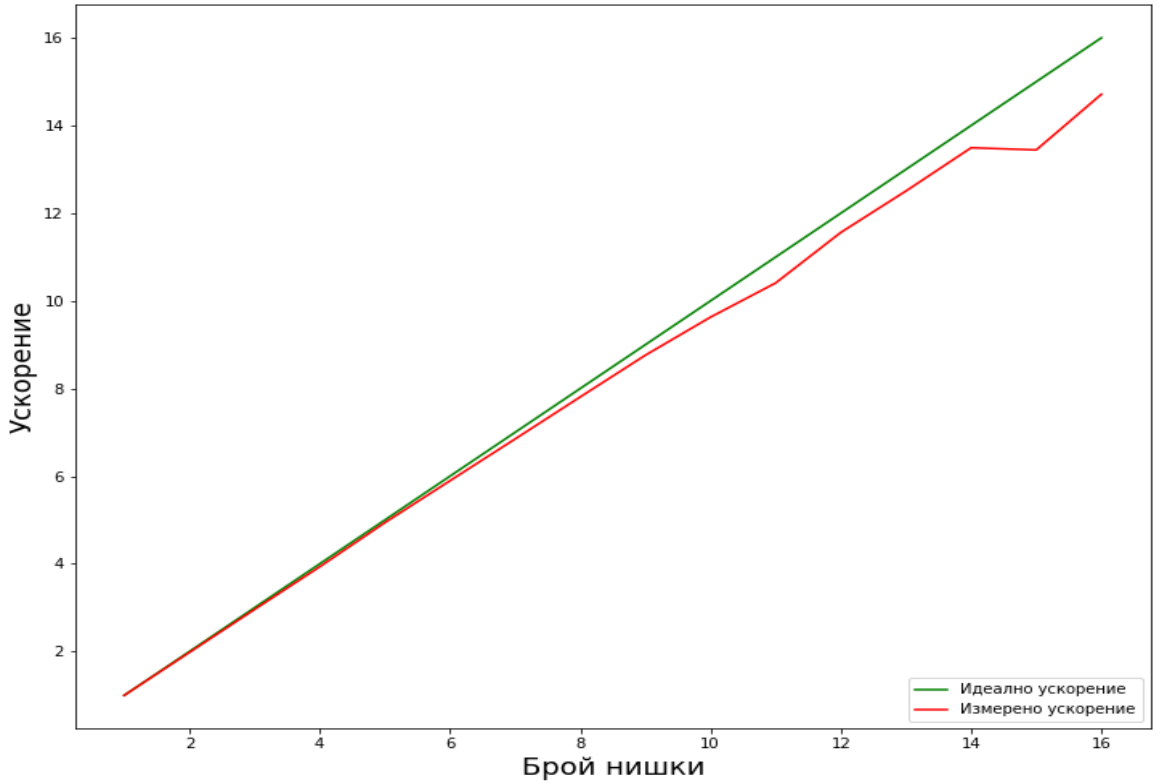
Следната графика представя времето за изпълнение на програмата спрямо нивото на паралелизъм:



3.2 Ускорение и ефективност

Съответно графики за ускорението и ефективността:

Графика на ускорението



Графика на ефективността