**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

|  |
| --- |
| **РОССИЙСКОЙ ФЕДЕРАЦИИ** |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ЮЖНО-РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (НПИ) имени М.И. Платова»**

|  |
| --- |
|  |

**ФАКУЛЬТЕТ ФИТУ**

**КАФЕДРА ПОВТ**

**НАПРАВЛЕНИЕ ИВТ**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНЫМ РАБОТАМ ПО КУРСУ**

Параллельные вычисления

***Студента***  *1* ***курса, шифр*** *2м Руденко А.С. Иванов Р.А.*

*Тангатаров А.Р. Бурмистр А.С.*

**Фамилия, имя, отчество**

***Преподаватель*** *к.т.н.**доцент каф.ПОВТ Бузало Г.А.*

**Должность, звание Фамилия, имя, отчество**

***Отчет принят******« »***г.

**Подпись**

**Новочеркасск 2020г.**

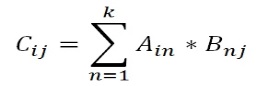
Содержание

[Лабораторная работа №1 3](#_Toc41909586)

[Лабораторная работа №2 9](#_Toc41909587)

# Лабораторная работа №1

**Задание:** написать программу реализующую перемножение двух квадратных матриц размерностями NxN в один поток, используя самописный код и сторонние библиотеки(MKL, OpenBLAS) и сравнить полученные результаты.

Классическое перемножение матриц , для его реализации достаточно использование двумерных массивов и дальнейшее перемножения строки одного массива на строку другой.

Для квадратных матриц код будет выглядеть так:

for (int y = 0; y < matrixDemension; y++)

{

for (int j = 0; j < matrixDemension; j++)

{

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[y, j] += myMatrixA[y, k] \* myMatrixB[k, j];

}

}

}

Не самый быстрый способ перемножения матриц (самый медленный) из-за способа хранения данных ­– двумерный массив (рис 1).

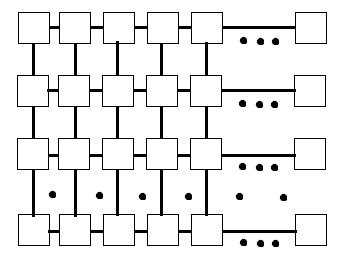


Рисунок 1.1. – Схема хранения данных в двумерном массиве

Добиться ускорения можно изменением способа хранения из двумерного массива в одномерный (рис 2).

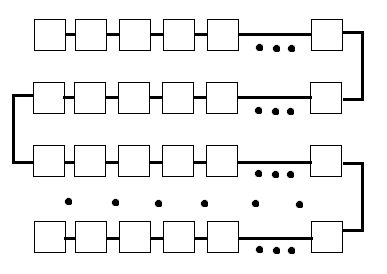


Рисунок 1.2. – Схема хранения данных в одномерном массиве

Код данного метода будет выглядеть так:

for (int i = 0; i < matrixDemension; i++)

{

for (int j = 0; j < matrixDemension; j++)

{

myOneDemMatrixC[i \* matrixDemension + j] = 0;

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[i , j] += myOneDemMatrixA[i \* matrixDemension + k] \*

myOneDemMatrixB[k \* matrixDemension + j];

}

}

}

Также для ускорения можно транспонировать вторую матрицу, чтобы доступ в памяти был последовательным(рис 3).

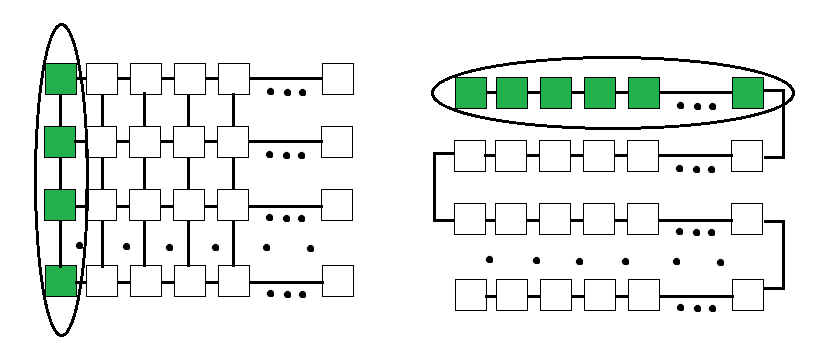


Рисунок 1.3. – Схема транспонирования данных в одномерном массиве

Код данного метода будет выглядеть так:

//Транспонирование матрицы

for (int y = 0; y < matrixDemension; y++)

{

for (int x = 0; x < matrixDemension; x++)

{

myOneDemMatrixB[y \* matrixDemension + x] = mathMatrixB[x, y];

}

}

for (int i = 0; i < matrixDemension; i++)

{

for (int j = 0; j < matrixDemension; j++)

{

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[i,j] += myOneDemMatrixA[i \* matrixDemension + k] \*

myOneDemMatrixB[j \* matrixDemension + k];

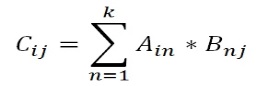
}

}

}

Для перемножения матриц можно использовать не только классический метод но и различные алгоритмы. Например алгоритм Штрассена.

Алгоритм Штрассена предназначен для быстрого [умножения матриц](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86). Он был разработан [Фолькером Штрассеном](https://ru.wikipedia.org/wiki/%D0%A8%D1%82%D1%80%D0%B0%D1%81%D1%81%D0%B5%D0%BD,_%D0%A4%D0%BE%D0%BB%D1%8C%D0%BA%D0%B5%D1%80) в 1969 году и является обобщением [метода умножения Карацубы](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%9A%D0%B0%D1%80%D0%B0%D1%86%D1%83%D0%B1%D1%8B) на матрицы.

В отличие от традиционного алгоритма умножения матриц (по формуле {\displaystyle c\_{i,k}=\sum a\_{i,j}b\_{j,k}}), работающего за время Θ({\displaystyle \Theta (n^{\log \_{2}8})=\Theta (n^{3})}n3), алгоритм Штрассена умножает матрицы за время Θ({\displaystyle \Theta (n^{\log \_{2}8})=\Theta (n^{3})}n2.81){\displaystyle \Theta (n^{\log \_{2}7})=O(n^{2.81})}, что даёт выигрыш на больших плотных матрицах начиная, примерно, от *64×64*.

Несмотря на то, что алгоритм Штрассена является асимптотически не самым быстрым из существующих алгоритмов быстрого умножения матриц, он проще программируется и эффективнее при умножении матриц относительно малого размера, поэтому именно он чаще используется на практике.

Для работы будем использовать модификацию алгоритма Штрассена – алгоритм Винограда–Штрассена. Данный алгоритм использует 7 умножений и 15 сложений (вместо 18 для обычного алгоритма Штрассена).

Матрицы A, B и C делятся на блочные подматрицы. Подматрицы a и b переводятся в одномерный массив, подматрица b транспонируется

Реализация алгоритма Винограда–Штрассена:

double[] VinShtr(double A11, double A12, double A21, double A22, double B11, double B12, double B21, double B22)

{

double[] result = new double[4];

double s1 = (A21 + A22);

double s2 = (s1 - A11);

double s3 = (A11 - A21);

double s4 = (A12 - s2);

double s5 = (B12 - B11);

double s6 = (B22 - s5);

double s7 = (B22 - B12);

double s8 = (s6 - B21);

double p1 = s2 \* s6;

double p2 = A11 \* B11;

double p3 = A12 \* B21;

double p4 = s3 \* s7;

double p5 = s1 \* s5;

double p6 = s4 \* B22;

double p7 = A22 \* s8;

double t1 = p1 + p2;

double t2 = t1 + p4;

result[0] = p2 + p3;

result[1] = t1 + p5 + p6;

result[2] = t2 - p7;

result[3] = t2 + p5;

return result;

}

Реализация алгоритма для перемножения матриц:

double[] tempmatrixA = new double[matrixDemension \* matrixDemension];

int xA = 0;

for (int y = 0; y < matrixDemension; y += 2)

{

for (int x = 0; x < matrixDemension; x += 2)

{

tempmatrixA[xA] = myOneDemMatrixA[y \* matrixDemension + x];

tempmatrixA[xA + 1] = myOneDemMatrixA[y \* matrixDemension + x + 1];

tempmatrixA[xA + 2] = myOneDemMatrixA[(y + 1) \* matrixDemension + x];

tempmatrixA[xA + 3] = myOneDemMatrixA[(y + 1) \* matrixDemension + x + 1];

xA += 4;

}

}

double[] tempmatrixB = new double[matrixDemension \* matrixDemension];

int xB = 0;

for (int y = 0; y < matrixDemension; y += 2)

{

for (int x = 0; x < matrixDemension; x += 2)

{

tempmatrixB[xB] = myOneDemMatrixB[x \* matrixDemension + y];

tempmatrixB[xB + 1] = myOneDemMatrixB[x \* matrixDemension + y + 1];

tempmatrixB[xB + 2] = myOneDemMatrixB[(x + 1) \* matrixDemension + y];

tempmatrixB[xB + 3] = myOneDemMatrixB[(x + 1) \* matrixDemension + y + 1];

xB += 4;

}

}

int ii;

int ii1;

int jj;

for (int i = 0; i < matrixDemension; i += 2)

{

for (int j = 0; j < matrixDemension; j += 2)

{

ii = i \* matrixDemension;

ii1 = (i + 1) \* matrixDemension;

jj = j \* matrixDemension;

for (int k = 0; k < matrixDemension + matrixDemension; k += 4)

{

var t = VinShtr2(

tempmatrixA[ii + k],

tempmatrixA[ii + k + 1],

tempmatrixA[ii + k + 2],

tempmatrixA[ii + k + 3],

tempmatrixB[jj + k],

tempmatrixB[jj + k + 1],

tempmatrixB[jj + k + 2],

tempmatrixB[jj + k + 3]);

myMatrixC[i, j] += t[0];

myMatrixC[i, j + 1] += t[1];

myMatrixC[i + 1, j] += t[2];

myMatrixC[i + 1, j + 1] += t[3];

}

}

}

Проверим данные методы на практике.

Вычисления будут проводиться на компьютере с:

Процессор AMD Ryzen 5 2600 3.4 ГГц.

Оперативная память DDR4 2x8 Гб 2666МГц.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название метода | Размерность матриц | | | |
| 500x500 | 1000x1000 | 1500x1500 | 2000x2000 |
| Время выполнения | | | |
| Метод 1 | 635 мс | 5626 мс | 31263 мс | 67825 мс |
| Метод 2 | 622 мс | 5440 мс | 24155 мс | 44374 мс |
| Метод 3 | 568 мс | 4709 мс | 15730 мс | 37297 мс |
| Метод 4 | 472 мс | 3776 мс | 12868 мс | 31438 мс |

Таблица 1.1.-Результаты перемножения матриц самописным кодом

Из таблицы 1.1 видно, что методы оптимизации (в случае матриц 2000х2000) привели к ускорению умножения матриц в 2.16 раза.

Существуют специальные библиотеки, оптимизированные под архитектуру процессора и выполняющие перемножения матриц гораздо быстрее способов, описанных выше.

Для демонстрации работы библиотек будем использовать MathNet.Numerics скачанную с NuGet. Данная библиотека имеет расширения MathNet.Numerics.MKL и MathNet.Numerics.OpenBLAS, для справедливости будем запускать их в 1 поток.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название метода | Размерность матриц | | | |
| 500x500 | 1000x1000 | 1500x1500 | 2000x2000 |
| Время выполнения | | | |
| MathNet.Numerics | 52 мс | 127 мс | 430 мс | 1111 мс |
| MKL | 16 мс | 130 мс | 433 мс | 1013 мс |
| OpenBLAS | 32 мс | 116 мс | 386 мс | 913 мс |

Таблица 1.2.-Результаты перемножения матриц с помощью библиотек

Из таблицы 1.2 видно, что использование специальных библиотек ускоряет перемножение матриц в десятки раз.

# Лабораторная работа №2

**Задание:** написать программу реализующую перемножение двух квадратных матриц размерностями NxN в несколько потоков, используя самописный код и сторонние библиотеки(MKL, OpenBLAS) и сравнить полученные результаты.

Для распараллеливания умножения воспользуемся методом Parallel.For, что даст нам возможность обрабатывать один из циклов многопоточно.

Для того чтобы не было конфликтов с доступом к памяти распараллеливать будем внешние циклы.

После преобразования код методов из лабораторной работы №1 выглядит следующим образом:

Метод 1:

Parallel.For(0, matrixDemension,

y =>

{

for (int j = 0; j < matrixDemension; j++)

{

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[y, j] += myMatrixA[y, k] \* myMatrixB[k, j];

}

}

});

Метод 2:

Parallel.For(0, matrixDemension,

i =>

{

for (int j = 0; j < matrixDemension; j++)

{

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[i, j] += myOneDemMatrixA[i \* matrixDemension + k] \*

myOneDemMatrixB[k \* matrixDemension + j];

}

}

});

Метод 3:

//Транспонирование матрицы

Parallel.For(0, matrixDemension,

y =>

{

for (int x = 0; x < matrixDemension; x++)

{

myOneDemMatrixB[y \* matrixDemension + x] = mathMatrixB[x, y];

}

}

);

Parallel.For(0, matrixDemension,

i =>

{

for (int j = 0; j < matrixDemension; j++)

{

int ii = i \* matrixDemension;

for (int k = 0; k < matrixDemension; k++)

{

myMatrixC[i, j] += myOneDemMatrixA[i \* matrixDemension + k] \*

myOneDemMatrixB[j \* matrixDemension + k];

}

}

});

Метод 4:

double[] tempmatrixA = new double[matrixDemension \* matrixDemension];

int xA = 0;

for (int y = 0; y < matrixDemension; y += 2)

{

for (int x = 0; x < matrixDemension; x += 2)

{

tempmatrixA[xA] = myOneDemMatrixA[y \* matrixDemension + x];

tempmatrixA[xA + 1] = myOneDemMatrixA[y \* matrixDemension + x + 1];

tempmatrixA[xA + 2] = myOneDemMatrixA[(y + 1) \* matrixDemension + x];

tempmatrixA[xA + 3] = myOneDemMatrixA[(y + 1) \*matrixDemension+x+ 1];

xA += 4;

}

}

double[] tempmatrixB = new double[matrixDemension \* matrixDemension];

int xB = 0;

for (int y = 0; y < matrixDemension; y += 2)

{

for (int x = 0; x < matrixDemension; x += 2)

{

tempmatrixB[xB] = myOneDemMatrixB[x \* matrixDemension + y];

tempmatrixB[xB + 1] = myOneDemMatrixB[x \* matrixDemension + y + 1];

tempmatrixB[xB + 2] = myOneDemMatrixB[(x + 1) \* matrixDemension + y];

tempmatrixB[xB + 3] = myOneDemMatrixB[(x + 1) \*matrixDemension+y+ 1];

xB += 4;

}

}

Parallel.For(0, matrixDemension / 2,

c =>

{

int i = c \* 2;

for (int j = 0; j < matrixDemension; j += 2)

{

int ii = i \* matrixDemension;

int ii1 = (i + 1) \* matrixDemension;

int jj = j \* matrixDemension;

for (int k = 0; k < matrixDemension + matrixDemension; k += 4)

{

var t = VinShtr(

tempmatrixA[ii + k],

tempmatrixA[ii + k + 1],

tempmatrixA[ii + k + 2],

tempmatrixA[ii + k + 3],

tempmatrixB[jj + k],

tempmatrixB[jj + k + 1],

tempmatrixB[jj + k + 2],

tempmatrixB[jj + k + 3]);

myMatrixC[i, j] += t[0];

myMatrixC[i, j + 1] += t[1];

myMatrixC[i + 1, j] += t[2];

myMatrixC[i + 1, j + 1] += t[3];

}

}

});

Результаты многопоточной обработки приведены в таблице 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название метода | Размерность матриц | | | |
| 500x500 | 1000x1000 | 1500x1500 | 2000x2000 |
| Время выполнения | | | |
| Метод 1 | 223 мс | 916 мс | 4327 мс | 11085 мс |
| Метод 2 | 70 мс | 648 мс | 3265 мс | 7693 мс |
| Метод 3 | 71 мс | 533 мс | 1693 мс | 4227 мс |
| Метод 4 | 450 мс | 3560 мс | 8947 мс | 26722 мс |

Таблица 2.1.-Результаты перемножения матриц самописным кодом

Из таблицы 2.1 видно, что распараллеливание может привести к многократному ускорению вычислений, но большой объем инструкций может привести к конфликту между потоками и выполнение вычислений может быть не столь эффективным как в случае 4 метода.

Запустим библиотеку MathNet.Numerics в несколько потоков.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название метода | Размерность матриц | | | |
| 500x500 | 1000x1000 | 1500x1500 | 2000x2000 |
| Время выполнения | | | |
| MathNet.Numerics | 3 мс | 27 мс | 79 мс | 223 мс |
| MKL | 4 мс | 24 мс | 79 мс | 189 мс |
| OpenBLAS | 2 мс | 22 мс | 71 мс | 174 мс |

Таблица 2.2.-Результаты перемножения матриц с помощью библиотек

Из таблицы 2.2 видно, что использование нескольких потоков приводит решение матриц в считанные секунды.