**МИНОБРНАУКИ РОССИИ**

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт математики, механики и компьютерных наук им. Воровича И. И.

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №1 (вариант 31)**

**по преобразованию программ**

Направление: Фундаментальная информатика и информационные технологии

Тема: «Блочное перемножение матриц».

Отчет студентки 1 курса магистратуры Пирумян Маргариты Рубеновны

Преподаватель: д.т.н., зав. каф. А и ДМ Штейнберг Б.Я.

Октябрь, 2016

Г.Ростов-на-Дону

# Содержание

[Содержание 1](#_Toc468276276)

[1. Постановка задачи: 2](#_Toc468276277)

[2. Описание алгоритма: 3](#_Toc468276278)

[2.1 Общие сведения 3](#_Toc468276279)

[2.2 Функциональное назначение 3](#_Toc468276280)

[2.3 Описание логической структуры 3](#_Toc468276281)

[2.4 Связи программы с другими программами 5](#_Toc468276282)

[2.5 Используемые технические средства 5](#_Toc468276283)

[2.6 Вызов и загрузка 5](#_Toc468276284)

[2.7 Входные данные 5](#_Toc468276285)

[2.8 Выходные данные 5](#_Toc468276286)

[3. Результаты расчетов: 6](#_Toc468276287)

[4. Литература: 9](#_Toc468276288)

[5. Текст программы: 10](#_Toc468276289)

# Постановка задачи:

**Написать стандартный и блочный алгоритмы перемножения матриц, распараллелить. Сравнить их производительность.**

Написать программу блочного умножения двух матриц С = А \* В.

Матрица А симметричная, хранится как верхнее - треугольная. Хранится в виде одномерного массива по блочным столбцам.

Матрица В верхнее - треугольная. Хранится в виде одномерного массива по блочным строкам.

Распараллелить блочную программу умножения двух матриц С = А \* В с использованием технологии OpenMp двумя способами

* Распараллелить каждое перемножение блоков.
* Параллельно выполнять операции над различными блоками.

Определить оптимальные размеры блоков в обоих случаях.

Провести численные эксперименты и построить таблицу сравнений времени выполнения различных программных реализаций решения задачи.

Определить лучшие реализации.

Проверить корректность (правильность) программ.

# Описание алгоритма:

* 1. Общие сведения

Программа имеет наименование «task1». Данная программа функционирует на любых версиях операционных систем Windows и Linux, специализированного программного обеспечения для работы не требуется. Алгоритм реализован с помощью языка программирования C в среде разработки Microsoft Visual Studio Express 2013, на этапе тестирования использовался компилятор: GNU GCC Compiler для проверки правильности и скорости работы программы.

* 1. Функциональное назначение

Программа предназначена для изучения зависимости производительности работы программы от выбранного алгоритма решения задачи. Для этого реализуются два алгоритма перемножения матриц: стандартный и блочный. Реализация программы демонстрирует, что деление матриц на блоки уменьшает количество кэш-промахов, что способствует возрастанию скорости выполнения.

* 1. Описание логической структуры

1. Алгоритм программы
2. **Стандартный алгоритм:**

1) \*Все матрицы хранятся в виде одномерных массивов.

Создание симметричной матрицы check\_ А (хранится как верхне - треугольная матрица), создание верхнее - треугольной матрицы check\_ В, создание матрицы check\_C, в которую будет помещен результат перемножения матриц check\_ А и check\_ В.

Заполнение элементов матриц check\_ А и check\_ В случайными числами.

Инициализация элементов матрицы check\_C нулями.

2) Передача параметров в функцию, представляющую собой стандартный алгоритм перемножения двух матриц.

а) Все элементы матрицы check\_C рассчитываются по формуле:

C[i \* size + j] += A[i \* size + k] \* B[k \* size + j]

1. **Блочный алгоритм:**

1) \*Все матрицы хранятся в виде одномерных массивов.

Создание матрицы А, хранится как верхнее – треугольная. Хранится в виде одномерного массива по блочным столбцам.

Создание верхнее - треугольной матрицы В. Хранится в виде одномерного массива по блочным строкам.

Заполнение происходит числами из матриц для проверки (матрицы из стандартного алгоритма). Это требуется для того, чтобы затем сверить результаты перемножения двух алгоритмов.

Инициализация элементов матрицы C нулями.

2) Передача параметров в функцию, представляющую собой блочный алгоритм перемножения матриц.

а) Функция помимо самих матриц и размера матрицы на вход получает размер блока и внутри работает аналогично стандартному алгоритму, но уже в терминах блоков.

б)Так же функция в своем теле вызывает другие функции, вычисляющие индексы матриц в зависимости от их типа.

1. Структура программы

Программа представляет собой 14 вспомогательных подпрограмм и тело программы.

1. «fill\_check\_A» - заполнение матрицы check\_A для стандартного алгоритма, вызывается функцией «main».
2. «fill\_check\_B» - заполнение матрицы check\_B для стандартного алгоритма, вызывается функцией «main».
3. «initialize\_C» - инициализация матриц check\_С и С нулями, вызывается функцией «main».
4. «mult\_check\_A\_B» - реализация стандартного алгоритма перемножения матриц, вызывается функцией «main».
5. «fill\_A\_from\_arr» - заполнение матрицы A для блочного алгоритма, вызывается функцией «main».
6. «fill\_B\_from\_arr» - заполнение матрицы B для блочного алгоритма, вызывается функцией «main».
7. «multBlocks» - реализация перемножения пары блоков, вызывается функцией «mult\_block\_A\_B».
8. «findIndA» - вычисление адреса начала блока в симметричной блочной матрице А по заданным блочным индексам, вызывается функцией «mult\_block\_A\_B».
9. «findIndB» - вычисление адреса начала блока в верхне - треугольной блочной матрице B по заданным блочным индексам, вызывается функцией «mult\_block\_A\_B».
10. «findIndС» - вычисление адреса начала блока в блочной матрице С по заданным блочным индексам, вызывается функцией «mult\_block\_A\_B».
11. «mult\_block\_A\_B» - реализация блочного алгоритма перемножения матриц, вызывается функцией «main».
12. «check\_results» - функция, которая сверяет поэлементно матрицы полученные в обоих алгоритмах и выдает сообщение о их равенстве, вызывается функцией «main».
13. «print\_Arr\_lines» - простейшие функции печати матриц, хранящихся в виде одномерного массива, вызывается функцией «main».
14. «print\_Arr» - другой вид печать матриц, вызывается функцией «main».  
    1. Связи программы с другими программами

При работе программа связей с другими программными продуктами не имеет, но для экспериментального определения оптимального размера блока необходимы компиляторы или специализированные текстовые редакторы.

* 1. Используемые технические средства

При проектировании и тестировании использовались ЭВМ со следующими характеристиками:   
1) Процессор: Intel® Core™ i7-3632QM CPU 2.2GHz 2.2GHz  
ОЗУ: 8,00 Гб  
Тип системы: 64-разрядная ОС Windows 10

* 1. Вызов и загрузка

Вызывалась программа в среде разработки Microsoft Visual Studio Express 2013.

* 1. Входные данные

Входными данными является матрицы, определенные в пункте 2.3, размер матрицы и размер блока.

* 1. Выходные данные

Выходными данными являются матрицы, получающиеся после выполнения подпрограмм со стандартным и блочным алгоритмами, а также время выполнения каждого из методов.

# Результаты расчетов:

Для определения разницы в скорости выполнения стандартного и блочного алгоритмов решения задачи было выполнено несколько тестов для разных входных данных:

1. Для размера матрицы 1440х1440:

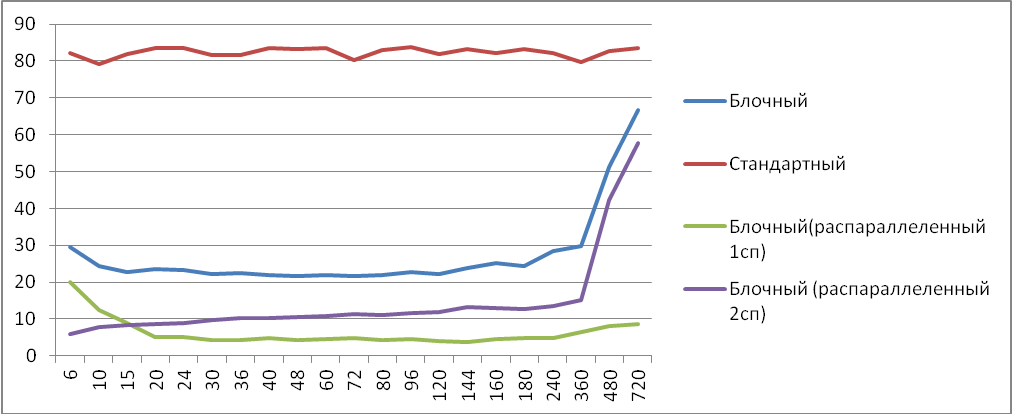


Рисунок 1 – графики выполнения (в секундах) программы для стандартного, блочного, распараллеленного блочного 1м и 2м способом алгоритмов при размере матрицы 1440х1440 от размера блока.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер блока | Стандартный | Блочный | Блочный (распар 1сп) | Блочный (распараллеленный 2сп) |
| 6 | 82,052 | 29,634 | 20,135 | 6,027 |
| 10 | 79,222 | 24,398 | 12,437 | 7,776 |
| 15 | 81,917 | 22,682 | 8,811 | 8,25 |
| 20 | 83,548 | 23,57 | 5,227 | 8,736 |
| 24 | 83,614 | 23,208 | 5,186 | 8,929 |
| 30 | 81,67 | 22,165 | 4,288 | 9,616 |
| 36 | 81,677 | 22,594 | 4,275 | 10,126 |
| 40 | 83,412 | 21,893 | 4,845 | 10,167 |
| 48 | 83,234 | 21,643 | 4,27 | 10,471 |
| 60 | 83,454 | 22,034 | 4,634 | 10,812 |
| 72 | 80,348 | 21,671 | 4,823 | 11,205 |
| 80 | 82,868 | 21,845 | 4,304 | 11,19 |
| 96 | 83,865 | 22,626 | 4,673 | 11,583 |
| 120 | 81,988 | 22,26 | 3,949 | 11,881 |
| 144 | 83,299 | 23,687 | 3,734 | 13,37 |
| 160 | 82,223 | 25,308 | 4,639 | 12,951 |
| 180 | 83,27 | 24,391 | 4,8 | 12,673 |
| 240 | 82,131 | 28,494 | 4,74 | 13,525 |
| 360 | 79,68 | 29,893 | 6,55 | 15,091 |
| 480 | 82,58 | 51,269 | 7,955 | 42,276 |
| 720 | 83,596 | 66,601 | 8,489 | 57,627 |

Таблица 1 – зависимость скорости выполнения (в секундах) программы для стандартного, блочного, распараллеленного блочного 1м и 2м способом алгоритмов при размере матрицы 1440х1440 от размера блока.

2) Для размера матрицы 2880х2880:

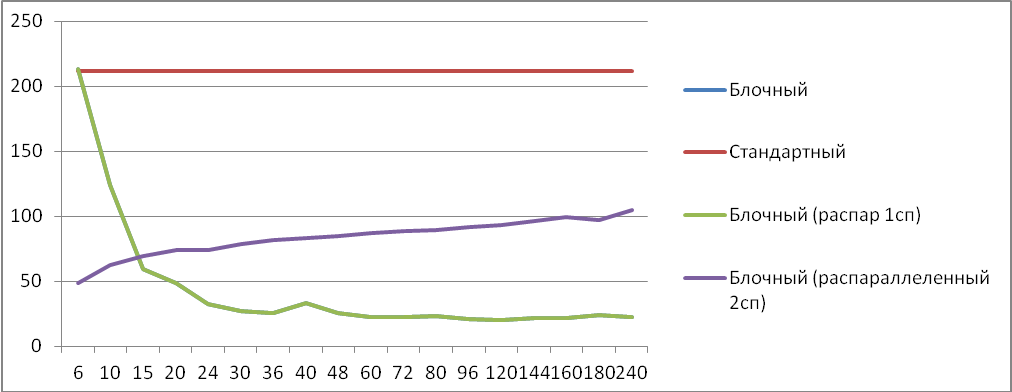


Рисунок 2 – графики скорости выполнения (в секундах) программы для стандартного, блочного, распараллеленного блочного 1м и 2м способом алгоритмов при размере матрицы 2880х2880 от размера блока.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер блока | Стандартный | Блочный | Блочный (распар 1сп) | Блочный (распараллеленный 2сп) |
| 6 | 212 | 50,537 | 212,875 | 48,459 |
| 10 | 212 | 50,537 | 124,156 | 62,163 |
| 15 | 212 | 50,537 | 59,364 | 69,119 |
| 20 | 212 | 60,537 | 48,469 | 73,929 |
| 24 | 212 | 60,537 | 32,254 | 73,767 |
| 30 | 212 | 60,537 | 27,037 | 78,289 |
| 36 | 212 | 70,537 | 25,555 | 81,606 |
| 40 | 212 | 70,537 | 33,14 | 83,119 |
| 48 | 212 | 70,537 | 25,573 | 85 |
| 60 | 212 | 71,478 | 22,523 | 87,292 |
| 72 | 212 | 92,672 | 22,128 | 88,945 |
| 80 | 212 | 92,487 | 23,277 | 89,731 |
| 96 | 212 | 95,339 | 21,132 | 92,054 |
| 120 | 212 | 92,675 | 20,393 | 93,047 |
| 144 | 212 | 99,689 | 21,74 | 96,563 |
| 160 | 212 | 107,714 | 21,962 | 99,739 |
| 180 | 212 | 101,463 | 23,678 | 97,224 |
| 240 | 212 | 215,694 | 22,188 | 104,657 |

Таблица 2 – зависимость скорости выполнения (в секундах) программы для стандартного, блочного, распараллеленного блочного 1м и 2м способом алгоритмов при размере матрицы 2880х2880 от размера блока.

# Литература:

1. Гергель В. П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем: Учебник – М.: Издательство Московского университета, 2010. – 544 с., ил.
2. Штейнберг Б.Я.: Информационные зависимости и высокоуровневые распараллеливающие преобразования программ, Ростов-на-Дону, 2007. – 167 с.
3. М.В. Васильева, П.Е. Захаров,И.К. Сирдитов, П.А. Попов, М.С. Еремеева: Учебное пособие. Параллельное программирование на основе библиотек, (веб-сайт: <http://edu.chpc.ru/parallel/main.html> )
4. Юрушкин М.В.: Автореферат. Оптимизация размещения массивов   
   в общей памяти

# Текст программы:

Программа вычисления скорости выполнения стандартного и блочного алгоритмов перемножения матриц.

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <iomanip> //setw()

#include <math.h>

#include <omp.h>

using namespace std;

#define N 1440 //размер матрицы

#define BLOCK\_N 8 //размер блока

#define BLOCK\_ELEM BLOCK\_N \* BLOCK\_N

#define NUM\_BLOCKS N/BLOCK\_N

#define BLOCK\_TRIANGLE\_N N \*( N + BLOCK\_N) / 2

#pragma warning(disable: 4996)

//функции для проверки перемножения(стандартный алгоритм)

void fill\_check\_A(double \*arr, int size)

{

srand(time(NULL));

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

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

if (i > j)

arr[i \* size + j] = arr[j \* size + i];

else

arr[i \* size + j] = (rand() % 100) / 10.0;

}

void fill\_check\_B(double \*arr, int size)

{

//srand(time(NULL));

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

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

if (i > j)

arr[i \* size + j] = 0.0;

else

arr[i \* size + j] = (rand() % 100) / 10.0;

}

void initialize\_C(double\* arr, int size)

{

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

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

arr[i \* size + j] = 0.0;

}

void mult\_check\_A\_B(double \*A, double \*B, double \*C, int size)

{

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

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

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

C[i \* size + j] += A[i \* size + k] \* B[k \* size + j];

}

//функции для блочного перемножения

void fill\_A\_from\_arr(double \*A, double \*arr, int size, int b\_s)

{

int num\_blocks = size / b\_s;

//в терминах блоков

//по столбцам: слева-направо

for (int dj = 0; dj < num\_blocks; ++dj)

{

//сверху-вниз

for (int di = 0; di <= dj; ++di)

//внутри блока по строкам

for (int k = 0; k < b\_s; ++k) //~i

for (int l = 0; l < b\_s; ++l) //~j

{

int start\_block = di \* b\_s \* b\_s + (dj \* (dj + 1) / 2) \* b\_s \* b\_s;

A[start\_block + k \* b\_s + l] = arr[(di \* b\_s + k) \* size + dj \* b\_s + l];

if (di == dj && k > l)

A[start\_block + k \* b\_s + l] = A[start\_block + l \* b\_s + k];

}

}

}

void fill\_B\_from\_arr(double \*B, double \*arr, int size, int b\_s)

{

int num\_blocks = size / b\_s;

//в терминах блоков

//по строкам: сверху-вниз

for (int di = 0; di < num\_blocks; ++di)

// слева-направо

for (int dj = di; dj < num\_blocks; ++dj)

//внутри блока по строкам

for (int k = 0; k < b\_s; ++k) //~i

for (int l = 0; l < b\_s; ++l) //~j

{

int start\_block = (dj - di) \* b\_s \* b\_s + (di \* (size)\* b\_s) - (di \* (di - 1) / 2) \* b\_s \* b\_s;

B[start\_block + k \* b\_s + l] = arr[(di \* b\_s + k) \* size + dj \* b\_s + l];

if (di == dj && k > l)

B[start\_block + k \* b\_s + l] = 0.0;

}

}

void fill\_A(double \*arr, int size, int b\_s)

{

double count = 1.0;

int num\_blocks = size / b\_s;

int wrong\_lines = 0;

//в терминах блоков

//по столбцам: слева-направо

for (int dj = 0; dj < num\_blocks; dj++)

{

//сверху-вниз

for (int di = 0; di <= dj; ++di)

{

//внутри блока по строкам

for (int k = 0; k < b\_s; ++k) //~i

{

for (int l = 0; l < b\_s; ++l) //~j

{

arr[di \* b\_s \* b\_s + (dj \* (dj + 1) / 2) \* b\_s \* b\_s + k \* b\_s + l] = count;

count += 1.0;

//условие для диагональных блоков

if (di == dj && k > l)

arr[di \* b\_s \* b\_s + (dj \* (dj + 1) / 2) \* b\_s \* b\_s + k \* b\_s + l] = arr[di \* b\_s \* b\_s + (dj \* (dj + 1) / 2) \* b\_s \* b\_s + l \* b\_s + k];

}

}

}

}

}

void fill\_B(double \*arr, int size, int b\_s)

{

double count = 1.0;

int num\_blocks = size / b\_s;

//в терминах блоков

//по строкам: сверху-вниз

for (int di = 0; di < num\_blocks; ++di)

// слева-направо

for (int dj = di; dj < num\_blocks; ++dj)

//внутри блока по строкам

for (int k = 0; k < b\_s; ++k) //~i

for (int l = 0; l < b\_s; ++l) //~j

{

arr[(dj - di) \* b\_s \* b\_s + (di \* N \* b\_s) - di \* (di - b\_s) / 2] = count;

count += 1.0;

//условие для диагональных блоков ! потом вынести за цикл - не эффективно

if (di == dj && k > l)

arr[(dj - di) \* b\_s \* b\_s + (di \* N \* b\_s) - di \* (di - b\_s) / 2] = 0.0;

}

}

void multBlocks(double\* blockA, double\* blockB, double\* blockZ, int b\_s, int simm)

{

int i, j, k, zi, zk, n = 0;

//для распараллеливания первым способом

#pragma omp parallel for shared(blockZ, blockA, blockB, simm) private(i, j, k, zi, zk) schedule(static) num\_threads (b\_s)

for (i = 0; i < b\_s; ++i)

{

zi = i \* b\_s;

for (j = 0; j < b\_s; ++j)

for (k = 0; k < b\_s; ++k)

{

zk = k \* b\_s;

blockZ[zi + j] += blockA[simm ? zk + i : zi + k] \* blockB[zk + j];

}

}

}

int findIndA(int di, int dj, int b\_s, int &simm)

{

//индекс начала блока

//ниже диагонали

if (di > dj)

{

simm = 1;

int z = di;

di = dj;

dj = z;

}

return di \* b\_s \* b\_s + (dj \* (dj + 1) / 2) \* b\_s \* b\_s;

}

int findIndB(int di, int dj, int b\_s)

{

return (dj - di) \* b\_s \* b\_s + (di \* N \* b\_s) - (di \* (di - 1) / 2) \* b\_s \* b\_s;

}

int findIndC(int di, int dj, int b\_s)

{

return di \* b\_s \* b\_s \* NUM\_BLOCKS + dj \* b\_s \* b\_s;

}

void mult\_block\_A\_B(double \*A, double \*B, double \*C, int size, int b\_s)

{

int num\_blocks = size / b\_s;

int block\_size = b\_s \* b\_s;

//выделяем память под блок результата произведения двух блоков

double \*z\_block = (double\*)malloc(BLOCK\_ELEM \* sizeof(double));

for (int i = 0; i < BLOCK\_ELEM; ++i)

z\_block[i] = 0;

int flag = 1, simm = 0;

int indA = 0, indB = 0, indC = 0;

int di, dj, dk, i;

//перемножение поблочное

//для распараллеливания вторым способом

//#pragma omp parallel for shared(C, A, B) private(di, dj, dk, i, indA, indB, indC, simm, flag, z\_block) schedule(static) num\_threads (num\_blocks)

for (di = 0; di < num\_blocks; ++di)

for (dj = 0; dj < num\_blocks; ++dj)

for (dk = 0; dk < num\_blocks; ++dk)

{

flag = 1;//не ноль блок

simm = 0;

for (i = 0; i < block\_size; ++i)

z\_block[i] = 0;

//вычисление адреса блока матрицы A

indA = findIndA(di, dk, b\_s, simm);

//вычисление адреса блока матрицы B

if (dk > dj)

flag = 0;//нулевой блок

else

{

indB = findIndB(dk, dj, b\_s);

//printf("indB=%d\n", indB);

}

//вычисление адреса блока матрицы C

indC = findIndC(di, dj, b\_s);

//printf("indC = %d, thread number %d\n", indC, omp\_get\_thread\_num());

//если не нулевой блок B, тогда перемножаем блоки и суммируем, а если нет, то останется тот же С

if (flag)

{

multBlocks(A + indA, B + indB, z\_block, b\_s, simm);

for (i = 0; i < block\_size; ++i)

{

C[indC + i] += z\_block[i];

//printf("indC = %d, thread number %d, z\_block[i]= %f\n", indC, omp\_get\_thread\_num(), z\_block[i]);

}

}

}

free(z\_block);

}

//проверка равенства двух полученных матриц

void check\_results(double \*C, double \*C\_block, int size, int b\_s)

{

printf("Are arrays C and C\_block equal? - ");

int num\_blocks = size / b\_s;

bool flag = false;

//в терминах блоков

//по строкам: сверху-вниз

for (int di = 0; di < num\_blocks; ++di)

{

// слева-направо

for (int dj = 0; dj < num\_blocks; ++dj)

//внутри блока по строкам

for (int k = 0; k < b\_s; ++k) //~i

for (int l = 0; l < b\_s; ++l) //~j

{

int start\_block = di \* b\_s \* b\_s \* num\_blocks + dj \* b\_s \* b\_s;

if (fabs(C[(di \* b\_s + k) \* size + dj \* b\_s + l] - C\_block[start\_block + k \* b\_s + l]) > 0.000001)

flag = true;

}

}

if (flag)

printf("No!\n");

else

printf("Yes!\n");

}

//функции печати блочных матриц, которые расположены как одномерный массив

void print\_Arr\_lines(double \*arr, int size)

{

printf("Size =%d\n", size);

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

printf("%7.2f, ", arr[i]);

printf("\n");

}

//печать в двух циклах

void print\_Arr(double \*arr, int size)

{

printf("Size =%d\n", size \* size);

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

{

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

printf("%7.2f, ", arr[i \* size + j]);

printf("\n");

}

}

int main()

{

//матрицы для проверки

double \*check\_A = (double\*)malloc(N \* N \* sizeof(double)), //симметричная

\*check\_B = (double\*)malloc(N \* N \* sizeof(double)), //верхнетреугольная, внизу нули

\*check\_C = (double\*)malloc(N \* N \* sizeof(double));

//матрицы для блочного перемножения

double \*A = (double\*)malloc(BLOCK\_TRIANGLE\_N \* sizeof(double)), //симметричная, хранится как верхетреугольная

\*B = (double\*)malloc(BLOCK\_TRIANGLE\_N \* sizeof(double)), //верхнетреугольная

\*C = (double\*)malloc(N \* N \* sizeof(double));

fill\_check\_A(check\_A, N);

/\*printf("Check\_A:\n");

print\_Arr(check\_A, N);\*/

fill\_check\_B(check\_B, N);

/\*printf("\nCheck\_B:\n");

print\_Arr(check\_B, N);\*/

initialize\_C(check\_C, N);

clock\_t t\_start, t\_end;

t\_start = clock();

mult\_check\_A\_B(check\_A, check\_B, check\_C, N);

t\_end = clock();

printf("\nMultiplication of arrays done in %f seconds\n", ((double)(t\_end - t\_start)) / CLOCKS\_PER\_SEC);

/\*printf("\nCheck multiplic Matrix C\n");

print\_Arr(check\_C, N);//\*/

//блочное перемножение

/\*заполним матрицу А\*/

//fill\_A(A, N, BLOCK\_N);

fill\_A\_from\_arr(A, check\_A, N, BLOCK\_N);

/\*printf("\nMatrix A\n");

print\_Arr\_lines(A, BLOCK\_TRIANGLE\_N);//\*/

/\*заполним матрицу B\*/

//fill\_B(B, N, BLOCK\_N);

fill\_B\_from\_arr(B, check\_B, N, BLOCK\_N);

/\*printf("\nMatrix B\n");

print\_Arr\_lines(B, BLOCK\_TRIANGLE\_N);//\*/

initialize\_C(C, N);

clock\_t t\_block\_start, t\_block\_end;

t\_block\_start = clock();

mult\_block\_A\_B(A, B, C, N, BLOCK\_N);

t\_block\_end = clock();

printf("\nMultiplication of block arrays done in %f seconds\n", ((double)(t\_block\_end - t\_block\_start)) / CLOCKS\_PER\_SEC);

/\*printf("\nBlock multiplic Matrix C\n");

print\_Arr\_lines(C, N \* N);//\*/

check\_results(check\_C, C, N, BLOCK\_N);

free(A);

free(B);

free(C);

free(check\_A);

free(check\_B);

free(check\_C);

return 0;

}