Федеральное Государственное Бюджетное Образовательное Учреждение

«Сибирский Государственный Университет Телекоммуникаций и Информатики»

Лабораторная работа №4

по дисциплине

Автоматизация Вычислительных Систем

Выполнил: студент

группы ИП-814

Краснов Илья

Проверил: ст. преп. Кафедры ВС   
Ткачева Т. А.

Оглавление

[Постановка задачи 3](#_Toc56460824)

[Результаты тестирования 4](#_Toc56460825)

[Листинг 6](#_Toc56460826)

[Ссылки 8](#_Toc56460827)

# Постановка задачи

1. На языке С++ реализовать функцию DGEMM\_BLAS - последовательное умножение двух квадратных матриц с элементами типа double. Обеспечить возможность задавать размерности матриц в качестве аргумента командной строки при запуске программы. Инициализировать начальные значения матриц случайными числами.
2. Провести серию испытаний и построить график зависимости времени выполнения программы от объёма входных данных. Например, для квадратных матриц с числом строк/столбцов 1000, 2000, 3000, … 10000.
3. Оценить предельные размеры матриц, которые можно перемножить на вашем вычислительном устройстве.
4. Реализовать дополнительную функцию DGEMM\_opt\_1, в которой выполняется оптимизация доступа к памяти, за счет построчного перебора элементов обеих матриц.
5. Оценить ускорение умножения для матриц фиксированного размера, например,1000х1000, 2000х2000, 5000х5000,10000х10000.
6. С помощью профилировщика для исходной программы и каждого способа оптимизации доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).

# Результаты тестирования

Получаем следующее время выполнения умножения для матриц 100х100, 500х500 и 1000х1000:

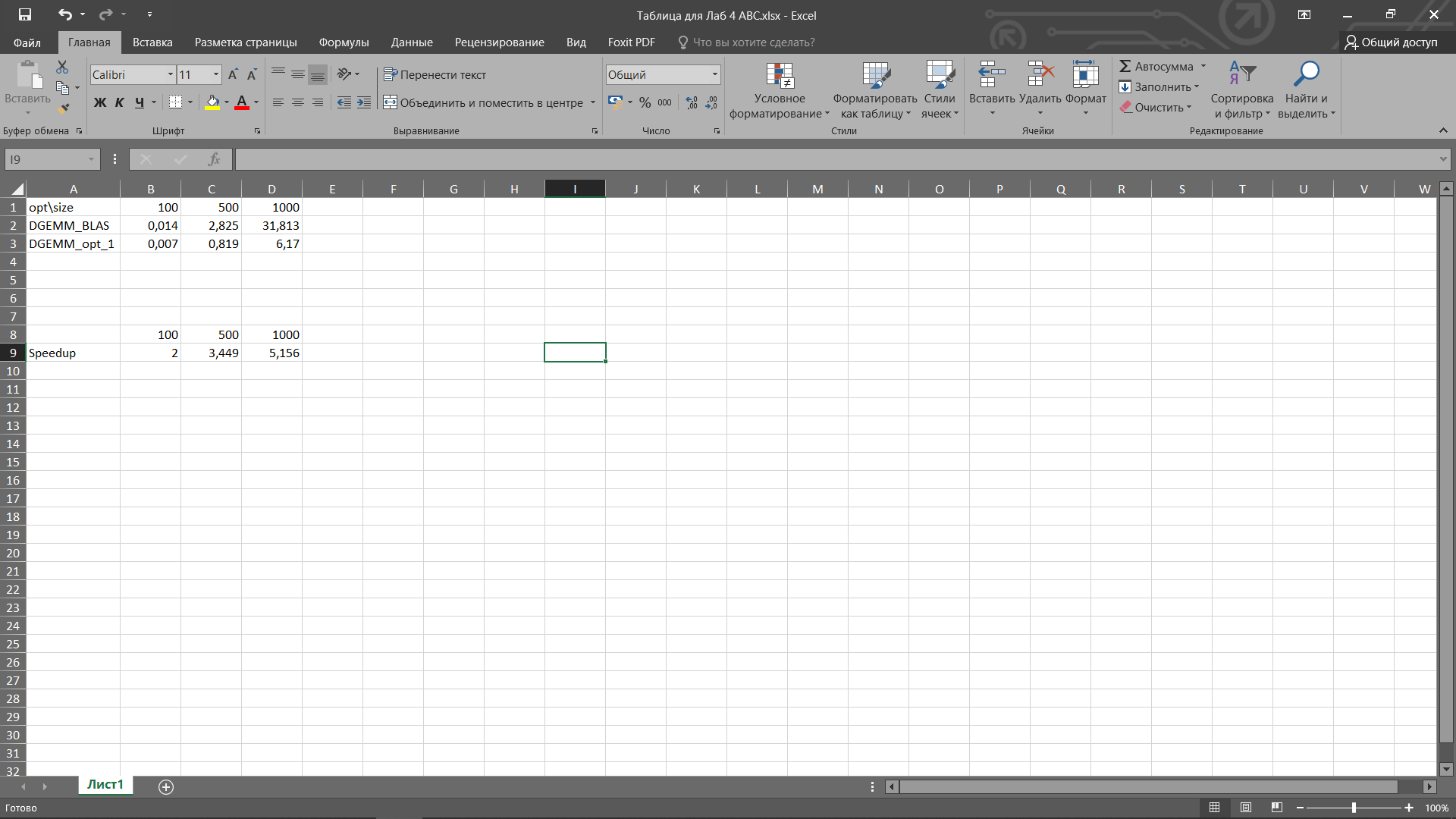


График зависимости времени выполнения от размера матрицы для DGEMM\_BLAS и DGEMM\_opt\_1:

Оценим предельны размеры матриц, которые можно перемножить на вычислительном устройстве, использованном в данном тестировании.

Оперативная память: 8 Гб = 8 388 608 \* 1024 байт.

При расчётах мы создаём три квадратных матрицы одинакового размера: две матрицы для перемножения и итоговая матрица с результатом.

Тип данных: double, значит размер одного элемента матрицы 8 байт.

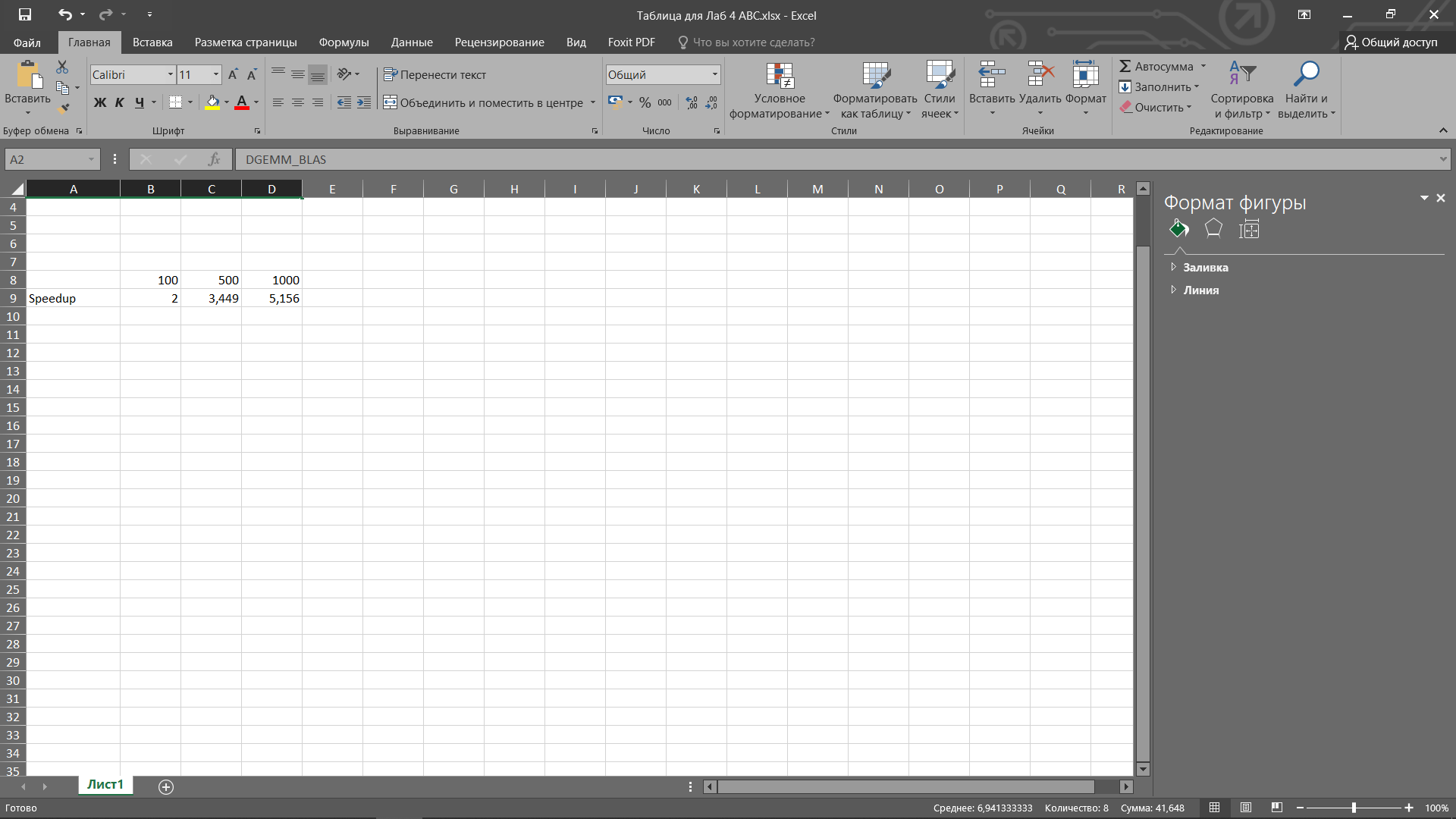
Пусть х – максимальное количество элементов в матрице.

Тогда : х \* 8 \* 3 = 8 388 608 \* 1024

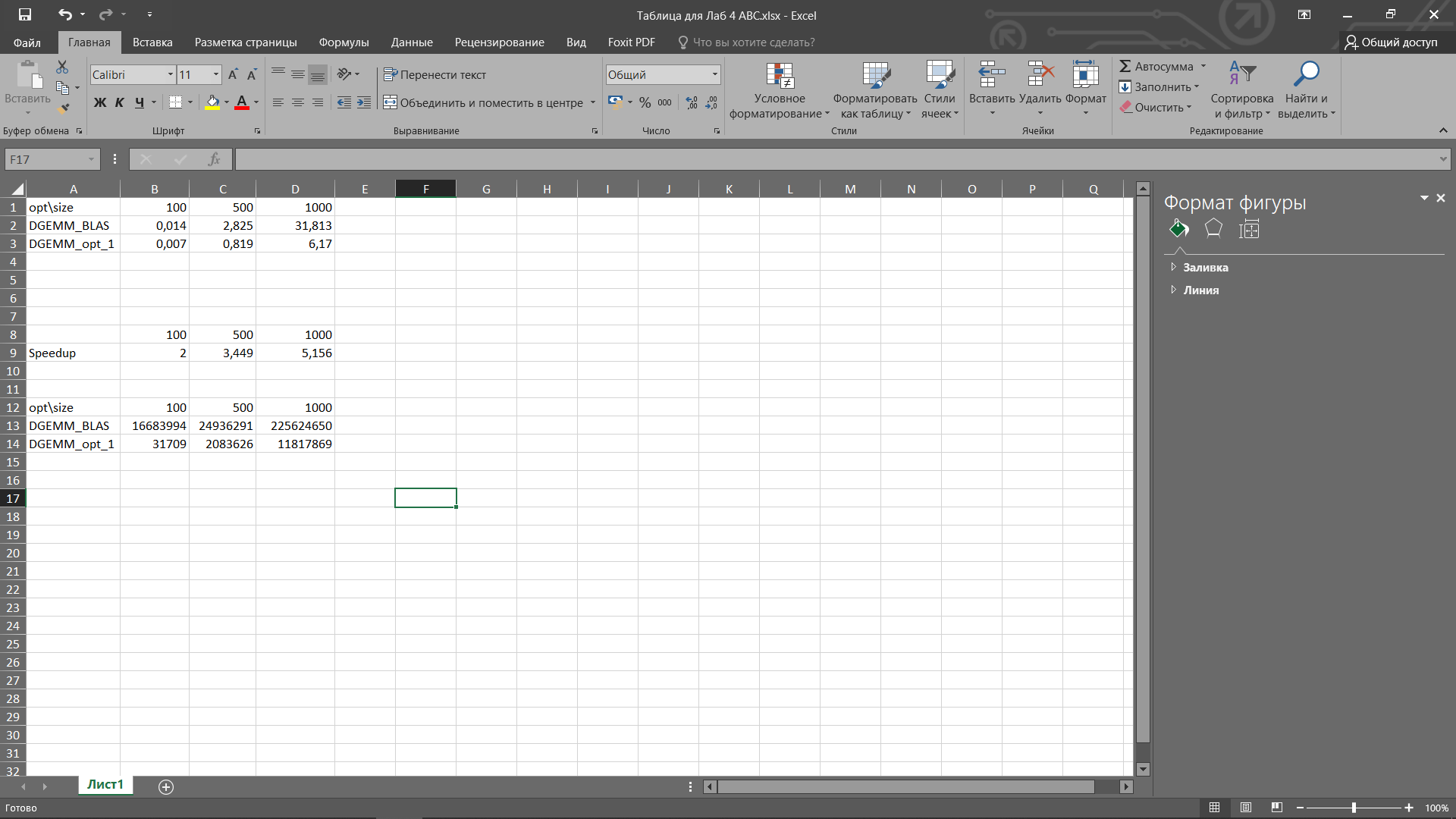
х = 8 388 608 / (3 \* 8) =(приблизительно) 357 913 941

Тогда получаем размер матрицы 18 918 х 18 918.

Оценим ускорение. В программе использован только один метод оптимизации, потому сравним его ускорение по отношению к стандартному алгоритму перемножения матриц. Ускорение для DGEMM\_opt\_1:



С помощью профилировщика Linux оценили промахи кэша:



# Листинг

#include <cmath>

#include <ctime>

#include <iomanip>

#include <iostream>

*using* *namespace* std;

double timeStart;

double timeCounter;

void **BenchTimer**(string task)

{

*if* (task == "START"){

timeStart = clock();

} *else* *if*(task == "STOP"){

timeCounter = ((double)(clock() - timeStart)) / CLOCKS\_PER\_SEC;

}

}

void **FillMatrix**(double\*\* aMatrix, double\*\* bMatrix, int size)

{

srand(time(NULL));

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

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

aMatrix[i][j] = rand()%100;

bMatrix[i][j] = rand()%100;

}

}

}

void **PrintMatrix**(double\*\* matrix, int size)

{

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

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

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

double **MultipicMatrix**(double\*\* aMatrix, double\*\* bMatrix, int size, int x\_pos, int y\_pos)

{

double res = 0;

int i = 0, j = 0;

*while* (i < size){

res += aMatrix[x\_pos][i] \* bMatrix[j][y\_pos];

i++;

j++;

}

*return* res;

}

void **DGEMM\_BLAS**(double\*\* aMatrix, double\*\* bMatrix, double\*\* resMatrix, int size)

{

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

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

resMatrix[i][j] = MultipicMatrix(aMatrix, bMatrix, size, i, j);

}

}

}

void **DGEMM\_opt1**(double\*\* A , double\*\* B, double\*\* C, int size)

{

double r;

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

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

r = A[i][k];

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

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

}

}

int **main**(int argc, char \*argv[])

{

int size = 1000;

double\*\* aMatrix = *new* double\* [size];

double\*\* bMatrix = *new* double\* [size];

double\*\* resMatrix = *new* double\* [size];

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

aMatrix[i] = *new* double[size];

bMatrix[i] = *new* double[size];

resMatrix[i] = *new* double[size];

}

FillMatrix(aMatrix, bMatrix, size);

*//* *cout* *<<* *"A:"* *<<* *endl;*

*//* *PrintMatrix(aMatrix,* *size);*

*//* *cout* *<<* *"B:"* *<<* *endl;*

*//* *PrintMatrix(bMatrix,* *size);*

BenchTimer("START");

DGEMM\_opt1(aMatrix, bMatrix, resMatrix, size);

BenchTimer("STOP");

*//* *cout* *<<* *"Res:"* *<<* *endl;*

*//* *PrintMatrix(resMatrix,* *size);*

cout << setprecision(15) << "Time = " << timeCounter << endl;

*delete* aMatrix;

*delete* bMatrix;

*delete* resMatrix;

*return* 0;

}

# Ссылки

1. Архив курса по дисциплине «Высокопроизводительные вычислительные системы» (М.Г.Курносов) URL:https://mkurnosov.net/docs/hpcs-2015-2016.tar.bz2

* Лекция3«Оптимизация доступа к памяти»
* Лекция 4 «Векторизация кода»

1. Репозиторий с исходным кодом:

https://github.com/Krasnoffsky/Study/tree/master/AVS/Lab4/withQt/lab4\_AVS