ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4  
«Оптимизация доступа к памяти»  
по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-813  
Бурдуковский Илья Александрович

Проверил: ст. преп. Кафедры ВС   
Токмашева Елизавета Ивановна

Новосибирск 2020

Содержание

[Постановка задачи 3](#__RefHeading___Toc663_3305851042)

[Выполнение работы 6](#__RefHeading___Toc665_3305851042)

[Запуск программы 7](#__RefHeading___Toc831_1292244855)

[Результат работы 8](#__RefHeading___Toc669_3305851042)

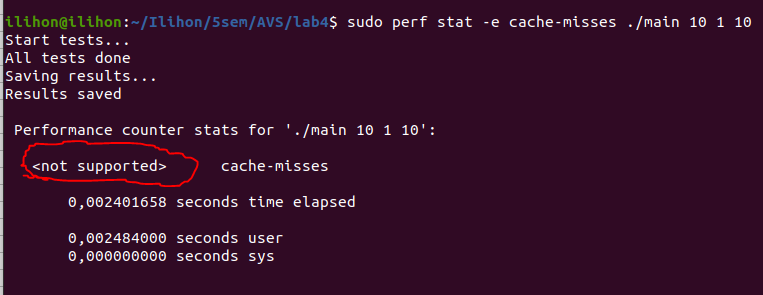
[Приложение №1 11](#__RefHeading___Toc671_3305851042)

[Приложение №2 13](#__RefHeading___Toc673_3305851042)

Выполнение работы

Перед выполнением работы было проведено ознакомление с предложенным в задании материалами, в которых были описаны алгоритмы оптимизации работы с матрицами, показан расчет ускорения и продемонстрирована работа профилировщика.

В связи с работой на виртуальной машине возникли трудности при работе профилировщика. Поэтому пришлось опираться на представленный в лекции материал для понимания работы.



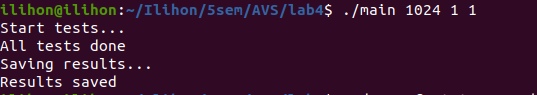
Постановка задачи

1. На языке С/С++/C# реализовать функцию DGEMM BLAS последовательное умножение двух квадратных матрицс элементами типа double. Обеспечить возможность задавать размерностиматриц 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. \* Реализовать дополнительную функцию DGEMM\_opt\_2, в которой выполняется оптимизация доступа к памяти, за счет блочного перебора элементов матриц. Обеспечить возможность задавать блока, в качестве аргументафункции.   
6. \*\* Реализовать дополнительную функцию DGEMM\_opt\_3, в которой выполняется оптимизация доступа к памяти, за счет векторизации кода.   
7. Оценить ускорение умножения для матриц фиксированного размера, например, 1000х1000, 2000х2000, 5000х5000, 10000х10000. \* Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.  
 8. С помощью профилировщика для исходной программы и каждого способа оптимизации доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).  
 9. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

Запуск программы

Запуск программы:

Здесь указывается размер матрица, количество повторений и размер блока.



Результат работы

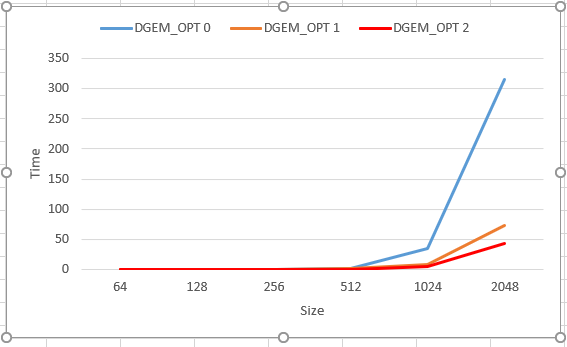
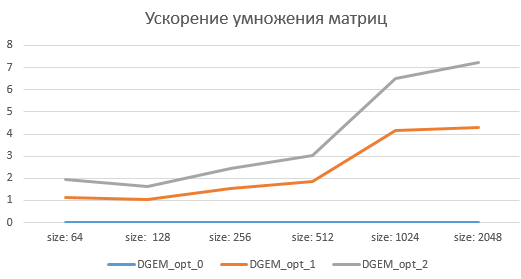


Рисунок 1. График зависимости времени выполнения программы от объёма входных данных при разных вариантах оптимизации.

**Ускорение умножения для матриц фиксированного размера**

Рисунок 2. График ускорения умножения выполнения программы от размера матриц

**\* Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.**

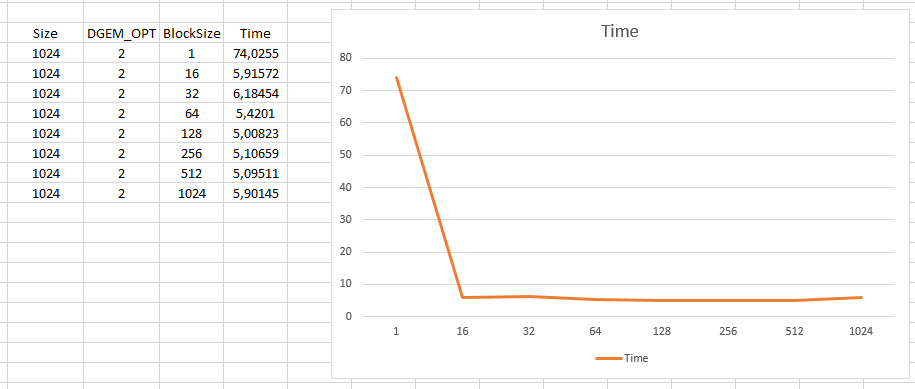


Рисунок 3. График времени программы от размера блока

В следствии данного графика можно сделать вывод, что максимальное ускорение достигается при размере блока 128, т.к. является размером L1 кэша.

**Оценить предельные размеры матриц, которые можно перемножить на вашем вычислительном устройстве.**

На виртуальную машину выделено 6 Гб оперативной памяти, из них ~4Гб (4 294 967 296 байт) свободно. Из этого количество 56 байт используется для вспомогательных переменных:   
10 переменных типа int = 40 байт

2 переменные типа double = 16 байт  
Оставшуюся память поровну делим на 6 массивов типа double (8 байт на элемент) размерности n\*n 900000000 элементов

n= 9 486

Предельный размер матрицы

Приложение №1

#include <iostream>

#include <fstream>

#include <time.h>

using namespace std;

void parse\_arguments(int argc, char \*argv[], int &n, int &launch\_num, int &block\_size);

void randmatrix(double \*\*M, int n);

void matrix\_to\_mas(double \*\*A, double \*B, int N);

void DGEMM\_opt\_0(double \*\*X, double \*\*Y, double \*\*C, int n, int launch\_num, double &result, double &result\_average);

void DGEMM\_opt\_1(double \*\*X, double \*\*Y, double \*\*C, int n, int launch\_num, double &result, double &result\_average);

void DGEMM\_opt\_2(double \*a, double \*b, double \*c, int n, int launch\_num, int block\_size, double &result, double &result\_average);

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

{

cout << "Start tests..." << endl;

int n, launch\_num, block\_size;

parse\_arguments(argc,argv,n, launch\_num, block\_size);

double \*\*x = new double\*[n];

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

x[i] = new double[n];

double \*\*y = new double\*[n];

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

y[i] = new double[n];

double \*\*c = new double\*[n];

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

c[i] = new double[n];

randmatrix(x, n);

randmatrix(y, n);

double \*X = new double[n\*n];

double \*Y = new double[n\*n];

double \*C = new double[n\*n];

matrix\_to\_mas(x,X,n);

matrix\_to\_mas(y,Y,n);

double result\_opt\_0;

double result\_opt\_0\_average;

double result\_opt\_1;

double result\_opt\_1\_average;

double result\_opt\_2;

double result\_opt\_2\_average;

DGEMM\_opt\_0(x, y, c, n, launch\_num, result\_opt\_0, result\_opt\_0\_average);

DGEMM\_opt\_1(x, y, c, n, launch\_num, result\_opt\_1, result\_opt\_1\_average);

DGEMM\_opt\_2(X, Y, C, n, launch\_num, block\_size, result\_opt\_2, result\_opt\_2\_average);

cout << "All tests done" << endl;

for (int i = 0; i < n; i++) {

delete x[i];

delete y[i];

delete c[i];

}

delete x;

delete y;

delete c;

delete X;

delete Y;

delete C;

cout << "Saving results..." << endl;

ofstream output;

output.open("result.csv", ios\_base::app);

output << "Size;DGEM\_OPT;Time;AvTime;LaunchNum;BlockSize" << endl;

output << n << ";0;" << result\_opt\_0 << ";" << result\_opt\_0\_average << ";" << launch\_num << ";" << block\_size << endl;

output << n << ";1;" << result\_opt\_1 << ";" << result\_opt\_1\_average << ";" << launch\_num << ";" << block\_size << endl;

output << n << ";2;" << result\_opt\_2 << ";" << result\_opt\_2\_average << ";" << launch\_num << ";" << block\_size << endl;

output.close();

cout << "Results saved" << endl;

return 0;

}

void parse\_arguments(int argc, char \*argv[], int &n, int &launch\_num, int &block\_size) {

n = 100;

launch\_num = 1;

block\_size = 64;

if (argc > 1) {

n = atoi(argv[1]);

if (argc > 2){

launch\_num = atoi(argv[2]);

if (argc > 3){

block\_size = atoi(argv[3]);

}

}

}

}

void randmatrix(double \*\*M, int N) {

for (int i = 0; i < N; i++) {

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

M[i][j] = rand() % N;

}

}

void matrix\_to\_mas(double \*\*A, double \*B, int N) {

for (int i = 0; i < N; i++) {

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

B[i\*N+j] = A[i][j];

}

}

void DGEMM\_opt\_0(double \*\*X, double \*\*Y, double \*\*C, int n, int launch\_num, double &result, double &result\_average) {

unsigned int start, stop;

result = 0;

for (int l = 0; l < launch\_num; l++){

start = clock();

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++){

C[i][j]=0;

for (int k = 0; k < n; k++){

C[i][j] += X[i][k] \* Y[k][j];

}

}

}

stop = clock();

result += stop - start;

}

result = (double)result / CLOCKS\_PER\_SEC;

result\_average = result / launch\_num;

}

void DGEMM\_opt\_1(double \*\*X, double \*\*Y, double \*\*C, int n, int launch\_num, double &result, double &result\_average) {

unsigned int start, stop;

result = 0;

for (int l = 0; l < launch\_num; l++){

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++){

C[i][j]=0;

}

}

start = clock();

for (int i = 0; i < n; i++) {

for (int k = 0; k < n; k++){

for (int j = 0; j < n; j++){

C[i][j] += X[i][k] \* Y[k][j];

}

}

}

stop = clock();

result += stop - start;

}

result = (double)result / CLOCKS\_PER\_SEC;

result\_average = result / launch\_num;

}

void DGEMM\_opt\_2(double \*a, double \*b, double \*c, int n, int launch\_num, int block\_size, double &result, double &result\_average) {

int i, j, k, i0, j0, k0;

double \*a0, \*b0, \*c0;

unsigned int start, stop;

result = 0;

for (int l = 0; l < launch\_num; l++){

start = clock();

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

for (j = 0; j < n; j += block\_size) {

for (k = 0; k < n; k += block\_size) {

for (i0 = 0, c0 = (c + i \* n + j), a0 = (a + i \* n + k); i0 < block\_size; ++i0, c0 += n, a0 += n) {

for (k0 = 0, b0 = (b + k \* n + j); k0 < block\_size; ++k0, b0 += n) {

for (j0 = 0; j0 < block\_size; ++j0) {

c0[j0] += a0[k0] \* b0[j0];

}

}

}

}

}

}

stop = clock();

result += stop - start;

}

result = (double)result / CLOCKS\_PER\_SEC;

result\_average = result / launch\_num;

}

Приложение №2

Файл 2 .csv

