###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Лабораторная работа 7»

Студента 1 курса, 19210 группы

**Пирожков Андрей Константинович**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

(ученая степень, звание)

Д.С.Иванишкин

Новосибирск 2020

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc58682233)

[ЗАДАНИЕ 3](#_Toc58682234)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc58682235)

[ЗАКЛЮЧЕНИЕ 6](#_Toc58682236)

[Приложение 1 *Листинг файла Lab7\_1.c* 7](#_Toc58682237)

[Приложение 2 *Листинг файла Lab7\_2.c* 10](#_Toc58682238)

[Приложение 3 *Листинг файла Lab7\_3.c* 14](#_Toc58682239)

# ЦЕЛЬ

* Изучение SIMD-расширений архитектуры x86/x86-64.
* Изучение способов использования SIMD-расширений в программах на языке Си.
* Получение навыков использования SIMD-расширений.

# ЗАДАНИЕ

1. Написать три варианта программы, реализующей алгоритм из задания:
   * вариант без ручной векторизации,
   * вариант с ручной векторизацией (выбрать любой вариант из возможных трех: ассемблерная вставка, встроенные функции компилятора, расширение GCC),
   * вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS.

Для элементов матриц использовать тип данных float.

1. Для программы на языке Си Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных
2. Каждый вариант программы оптимизировать по скорости, насколько это возможно.
3. Сравнить время работы трех вариантов программы для N=2048, M=10.
4. Составить отчет по лабораторной работе.

**Вариант задания:**

Алгоритм обращения матрицы размером с помощью разложения в ряд: , где , , , , -единичная матрица (на главной диагонали – единицы, остальные - нули). Параметры алгоритма: N – размер матрицы, M – число членов ряда (число итераций цикла в реализации алгоритма).

# **ОПИСАНИЕ РАБОТЫ**

Изначально просто создал программу, которая считает обратную матрицу по этому алгоритму. Однако решил оптимизировать умножение матриц путём перестановки двух последних циклов, что уменьшило вероятность пробуксовки кэша и дала небольшой прирост к скорости вычисления.

К моему сожалению, программа считала неверно обратную матрицу. Я решил подробнее изучить это разложение матрицы и, к сожалению, ничего подобного не увидел в интернете. Как будто её просто придумали для этой задачи. И как я не упирался найти ошибки в алгоритме моей программы, мне ничего не удалось исправить так, чтоб ответ был верный и совпадал с онлайн калькуляторами в интернете. Такую проблему слышал у некоторых одногруппников, и я успокоился и начал оптимизировать программу.

Далее я начал применять ручную векторизацию и остановился на выборе встроенных SIMD-функций компилятора, т.к. ассемблерные вставки не хотели работать на моём компьютере. Да и встроенные функции были более понятны.

В коде программы я сделал выделение памяти с выравнивание и последующие методы SIMD-функций использовал выровненными по адресу. Я переделал 3 функции вычисления матриц: сложения, вычитания, умножения. Функции загружали по 4 значения в 128-и битную переменную и делали одновременное вычисления. Также я предусмотрел случай если в вектор не помещались все 4 значения. Я сделал отдельный цикл в каждой функции (сложения, вычитания, умножения), который работает если исходные данные матрицы не кратны четырём (чтобы при вычислениях не попал мусор из выделенной памяти).

У меня осталась старая версия функции умножения матриц, но я решил применить метод перестановки цикла, чтоб вычисления были ещё быстрее. Поэтому у меня две функции: «mul» и «mul\_2». Использую последнюю, потому что она с применением перестановки циклов. А ту оставил на память, ведь компилятор её удалит с оптимизацией О2 как мёртвый код.

Ну и третий мой код программы использовал библиотеку BLAS. Её я использовал только для вычисления умножения матриц. Как оказалось этого хватило для отличного прироста скорости вычисления.

Скорость вычисления матриц засекаю таймером системного времени через библиотечную функцию «clock\_gettime». Я считаю его самым адекватным и точным таймером.

Входные данные на всех матрицах одинаковые: N=1200, K=10. Я сделал максимальный N, чтобы не возникало ошибки: «segmentation fault». Причины её возникновения я не нашел, как и способ от неё избавиться. Элементы матрицы заполняются рандомно.

Вот основная таблица с данными:

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Команда для компиляции | Название файлов | Листинг |
| Без векторизации | gcc -Wall -O2 -o Lab7\_1.exe Lab7\_1.c | Lab7\_1.c | [Приложение 1](#_Приложение_1) |
| Встроенные SIMD-функции | gcc -Wall -msse2 -O2 -o Lab7\_2.exe Lab7\_2.c | Lab7\_2.c | [Приложение 2](#_Приложение_2) |
| Библиотека BLAS | gcc -I/usr/local/atlas/include -L/usr/local/atlas/lib \-O2 -o Lab7\_3.exe Lab7\_3.c -lcblas -latlas | Lab7\_3.c | [Приложение 3](#_Приложение_3) |

А вот некоторые результаты:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N=1000; K=10 | | | N=1000; K=20 | | | N=1200; K=10 | | |
| \_1 | \_2 | \_3 | \_1 | \_2 | \_3 | \_1 | \_2 | \_3 |
| O1 | 36.60 | 26.09 | 3.15 | 70.06 | 49.68 | 5.88 | 64.78 | 49.39 | 5.24 |
| O2 | 32.72 | 22.40 | 3.17 | 63.46 | 42.18 | 5.98 | 60.49 | 42.43 | 5.48 |
| O3 | 24.15 | 22.26 | 3.19 | 45.46 | 42.58 | 5.85 | 45.69 | 42.92 | 5.79 |

* \_1 – без векторизации
* \_2 – встроенные SIMD-функции
* \_3 – библиотека BLAS

Благодаря этим результатам можно сделать несколько выводов:

* Оптимизации компилятора gcc (O1, O2, O3) работают отлично только для программы без ручной векторизации (\_1).
* Для программы с встроенными SIMD-функциями (\_2) О2 и О3 практически неотличимы. Видимо на этих оптимизациях компилятор сам начинает применять векторизацию.
* А вот оптимизация с библиотекой BLAS (\_3) как-то вообще ведёт себя наоборот. Возможно, для неё оптимизаций вообще не предназначено. И так очень быстро работает

# ЗАКЛЮЧЕНИЕ

В ходе этой лабораторной работы я изучил и применил SIMD-расширения архитектуры x86/x86-64. Также я изучил способы их использования в программах на языке си и получил навыки в работе с SIMD-расширениями. Написал 3 варианта программы. Оптимизировал каждый вариант насколько возможно. А также замерил время выполнения программ и сделал некоторые выводы.

# Приложение 1

*Листинг файла Lab7\_1.c*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 1200

#define M 10

void random\_matrix(float\* A)

{

for (int i = 0; i < N \* N; i++)

{

A[i] = (float)(rand() % 18 - 9);

}

}

void print\_matrix(float\* A)

{

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

{

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

{

printf("%g ", A[i \* N + j]);

}

printf("\n");

}

printf("\n");

}

float\* create\_I()

{

float\* I = (float\*)malloc(N \* N \* sizeof(float));

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

{

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

{

if (i != j)

{

I[i \* N + j] = 0;

}

else

{

I[i \* N + j] = 1;

}

}

}

return I;

}

float\* transposition(float\* A)

{

float\* A\_t = (float\*)malloc(N \* N \* sizeof(float));

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

{

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

{

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

}

}

return A\_t;

}

float\* create\_B(float\* A)

{

float i\_sum;

float j\_sum;

float i\_sum\_max = 0;

float j\_sum\_max = 0;

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

{

i\_sum = 0;

j\_sum = 0;

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

{

i\_sum += abs(A[j \* N + i]);

j\_sum += abs(A[i \* N + j]);

}

if (i\_sum > i\_sum\_max)

{

i\_sum\_max = i\_sum;

}

if (j\_sum > j\_sum\_max)

{

j\_sum\_max = j\_sum;

}

}

float max\_mul = i\_sum\_max \* j\_sum\_max;

float\* A\_t = transposition(A);

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

{

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

{

A\_t[i \* N + j] /= max\_mul;

}

}

return A\_t;

}

void add(float\* Res, float\* X, float\* Y)

{

for (int i = 0; i < N \* N; i++)

{

Res[i] = X[i] + Y[i];

}

}

void sub(float\* Res, float\* X, float\* Y)

{

for (int i = 0; i < N \* N; i++)

{

Res[i] = X[i] - Y[i];

}

}

void mul(float\* Res, float\* X, float\* Y)

{

float\* Result = (float\*)malloc(N \* N \* sizeof(float));

for (int i = 0; i < N \* N; i++)

{

Result[i] = 0;

}

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

{

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

{

//Result[i \* N + j] = 0;

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

{

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

}

}

}

for (int i = 0; i < N \* N; i++)

{

Res[i] = Result[i];

}

free(Result);

}

float\* inverse(float\* A)

{

float\* I = create\_I(); //I - единичная матрица

float\* B = create\_B(A); //B = A\_t / (max\_mul)

float\* BA = (float\*)malloc(N \* N \* sizeof(float));

mul(BA, B, A); //B \* A

float\* R = (float\*)malloc(N \* N \* sizeof(float));

sub(R, I, BA); //R = I - BA

float\* R\_i = (float\*)malloc(N \* N \* sizeof(float));

sub(R\_i, I, BA); //R\_i = R = I - BA (для R^2, R^3, ...)

float\* A\_ = (float\*)malloc(N \* N \* sizeof(float));

add(A\_, I, R); //Обратная матрица A^-1

for (int i = 1; i < M; i++)

{

mul(R\_i, R\_i, R);

add(A\_, A\_, R\_i);

}

mul(A\_, A\_, B);

free(I);

free(B);

free(BA);

free(R\_i);

free(R);

free(A);

return A\_;

}

int main()

{

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

float\* A = (float\*)malloc(N \* N \* sizeof(float));

random\_matrix(A);

//print\_matrix(A);

A = inverse(A);

//print\_matrix(A);

free(A);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

printf("Time: %lf\n\n", (finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec)));

return 0;

}

# Приложение 2

*Листинг файла Lab7\_2.c*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <xmmintrin.h>

#define N 1200

#define M 10

void random\_matrix(float\* A)

{

for (int i = 0; i < N \* N; i++)

{

A[i] = (float)(rand() % 18 - 9);

}

}

void print\_matrix(float\* A)

{

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

{

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

{

printf("%g ", A[i \* N + j]);

}

printf("\n");

}

printf("\n");

}

float\* create\_I()

{

float\* I = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

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

{

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

{

if (i != j)

{

I[i \* N + j] = 0;

}

else

{

I[i \* N + j] = 1;

}

}

}

return I;

}

float\* transposition(float\* A)

{

float\* A\_t = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

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

{

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

{

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

}

}

return A\_t;

}

float\* create\_B(float\* A)

{

float i\_sum;

float j\_sum;

float i\_sum\_max = 0;

float j\_sum\_max = 0;

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

{

i\_sum = 0;

j\_sum = 0;

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

{

i\_sum += abs(A[j \* N + i]);

j\_sum += abs(A[i \* N + j]);

}

if (i\_sum > i\_sum\_max)

{

i\_sum\_max = i\_sum;

}

if (j\_sum > j\_sum\_max)

{

j\_sum\_max = j\_sum;

}

}

float max\_mul = i\_sum\_max \* j\_sum\_max;

float\* A\_t = transposition(A);

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

{

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

{

A\_t[i \* N + j] /= max\_mul;

}

}

return A\_t;

}

void add(float\* Res, float\* X, float\* Y)

{

int n = N \* N - N \* N % 4;

for (int i = 0; i < n; i += 4)

{

\_mm\_store\_ps(&Res[i], \_mm\_add\_ps(\_mm\_load\_ps(&X[i]), \_mm\_load\_ps(&Y[i])));

}

for (int i = n; i < N \* N; ++i)

{

Res[i] = X[i] + Y[i];

}

}

void sub(float\* Res, float\* X, float\* Y)

{

int n = N \* N - N \* N % 4;

for (int i = 0; i < n; i += 4)

{

\_mm\_store\_ps(&Res[i], \_mm\_sub\_ps(\_mm\_load\_ps(&X[i]), \_mm\_load\_ps(&Y[i])));

}

for (int i = n; i < N \* N; ++i)

{

Res[i] = X[i] - Y[i];

}

}

float mul\_M(int i, int j, float\* X, float\* Y)

{

float res = 0;

\_\_m128 sum = \_mm\_setzero\_ps();

\_\_m128 mul;

int n = N - N % 4;

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

{

mul = \_mm\_mul\_ps(\_mm\_loadu\_ps(&X[k]), \_mm\_loadu\_ps(&Y[k]));

sum = \_mm\_add\_ps(sum, mul);

}

if (n > 0)

{

mul = \_mm\_movehl\_ps(mul, sum);

sum = \_mm\_add\_ps(sum, mul);

mul = \_mm\_shuffle\_ps(sum, sum, 1);

sum = \_mm\_add\_ss(sum, mul);

\_mm\_store\_ss(&res, sum);

}

if (N % 4 != 0)

{

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

{

res += X[i] \* Y[i];

}

}

return res;

}

void mul(float\* Res, float\* X, float\* Y)

{

float\* Result = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

Y = transposition(Y);

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

{

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

{

Result[i \* N + j] = mul\_M(i, j, &X[i \* N], &Y[j \* N]);

}

}

for (int i = 0; i < N \* N; i++)

{

Res[i] = Result[i];

}

\_mm\_free(Result);

}

void mul\_2(float\* Res, float\* X, float\* Y)

{

float\* Result = (float\*)malloc(N \* N \* sizeof(float));

\_\_m128 zero = \_mm\_setzero\_ps();

int n = N \* N - N \* N % 4;

for (int i = 0; i < n; i += 4)

{

\_mm\_store\_ps(&Result[i], zero);

}

for (int i = n; i < N \* N; ++i)

{

Result[i] = 0;

}

int n\_ = N - N % 4;

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

{

for (int k = 0; k < n\_; k++)

{

//Result[i \* N + j] = 0;

for (int j = 0; j < n\_; j+=4)

{

//Result[i \* N + j] += X[i \* N + k] \* Y[k \* N + j];

\_mm\_store\_ps(&Result[i \* N + j], \_mm\_mul\_ps(\_mm\_set1\_ps(X[i \* N + k]), \_mm\_loadu\_ps(&Y[k \* N + j])));

}

}

}

for (int i = n\_; i < N; i++)

{

for (int k = n\_; k < N; k++)

{

for (int j = n\_; j < N; j++)

{

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

}

}

}

for (int i = 0; i < N \* N; i++)

{

Res[i] = Result[i];

}

free(Result);

}

float\* inverse(float\* A)

{

float\* I = create\_I(); //I - единичная матрица

float\* B = create\_B(A); //B = A\_t / (max\_mul)

float\* BA = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

mul\_2(BA, B, A); //B \* A

float\* R = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

sub(R, I, BA); //R = I - BA

float\* R\_i = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

sub(R\_i, I, BA);

float\* A\_ = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

add(A\_, I, R); //Обратная матрица A^-1

for (int i = 1; i < M; i++)

{

mul\_2(R\_i, R\_i, R);

add(A\_, A\_, R\_i);

}

mul\_2(A\_, A\_, B);

\_mm\_free(I);

\_mm\_free(B);

\_mm\_free(BA);

\_mm\_free(R\_i);

\_mm\_free(R);

\_mm\_free(A);

return A\_;

}

int main()

{

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

float\* A = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 16);

random\_matrix(A);

//print\_matrix(A);

A = inverse(A);

//print\_matrix(A);

\_mm\_free(A);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

printf("Time: %lf\n\n", (finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec)));

return 0;

}

# Приложение 3

*Листинг файла Lab7\_3.c*

#include <stdio.h>

#include <stdlib.h>

#include <cblas.h>

#include <time.h>

#define N 1200

#define M 10

void random\_matrix(float\* A)

{

for (int i = 0; i < N \* N; i++)

{

A[i] = (float)(rand() % 18 - 9);

}

}

void print\_matrix(float\* A)

{

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

{

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

{

printf("%g ", A[i \* N + j]);

}

printf("\n");

}

printf("\n");

}

float\* create\_I()

{

float\* I = (float\*)malloc(N \* N \* sizeof(float));

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

{

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

{

if (i != j)

{

I[i \* N + j] = 0;

}

else

{

I[i \* N + j] = 1;

}

}

}

return I;

}

float\* transposition(float\* A)

{

float\* A\_t = (float\*)malloc(N \* N \* sizeof(float));

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

{

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

{

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

}

}

return A\_t;

}

float\* create\_B(float\* A)

{

float i\_sum;

float j\_sum;

float i\_sum\_max = 0;

float j\_sum\_max = 0;

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

{

i\_sum = 0;

j\_sum = 0;

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

{

i\_sum += abs(A[j \* N + i]);

j\_sum += abs(A[i \* N + j]);

}

if (i\_sum > i\_sum\_max)

{

i\_sum\_max = i\_sum;

}

if (j\_sum > j\_sum\_max)

{

j\_sum\_max = j\_sum;

}

}

float max\_mul = i\_sum\_max \* j\_sum\_max;

float\* A\_t = transposition(A);

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

{

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

{

A\_t[i \* N + j] /= max\_mul;

}

}

return A\_t;

}

void add(float\* Res, float\* X, float\* Y)

{

for (int i = 0; i < N \* N; i++)

{

Res[i] = X[i] + Y[i];

}

}

void sub(float\* Res, float\* X, float\* Y)

{

for (int i = 0; i < N \* N; i++)

{

Res[i] = X[i] - Y[i];

}

}

void mul(float\* Res, float\* X, float\* Y)

{

float\* Result = (float\*)malloc(N \* N \* sizeof(float));

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, X, N, Y, N, 0.0, Result, N);

for (int i = 0; i < N \* N; i++)

{

Res[i] = Result[i];

}

free(Result);

}

float\* inverse(float\* A)

{

float\* I = create\_I(); //I - единичная матрица

float\* B = create\_B(A); //B = A\_t / (max\_mul)

float\* BA = (float\*)malloc(N \* N \* sizeof(float));

mul(BA, B, A); //B \* A

float\* R = (float\*)malloc(N \* N \* sizeof(float));

sub(R, I, BA); //R = I - BA

float\* R\_i = (float\*)malloc(N \* N \* sizeof(float));

sub(R\_i, I, BA); //R\_i = R = I - BA (для R^2, R^3, ...)

float\* A\_ = (float\*)malloc(N \* N \* sizeof(float));

add(A\_, I, R); //Обратная матрица A^-1

for (int i = 1; i < M; i++)

{

mul(R\_i, R\_i, R);

add(A\_, A\_, R\_i);

}

mul(A\_, A\_, B);

free(I);

free(B);

free(BA);

free(R\_i);

free(R);

free(A);

return A\_;

}

int main()

{

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

float\* A = (float\*)malloc(N \* N \* sizeof(float));

random\_matrix(A);

//print\_matrix(A);

A = inverse(A);

//print\_matrix(A);

free(A);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

printf("Time: %lf\n\n", (finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec)));

return 0;

}