###### 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

«ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ»

студента 2 курса, 23208 группы

**Веретенникова Андрея Александровича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Д.С. Иванишкин

Новосибирск 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc18443921)

[ЗАДАНИЕ 3](#_Toc18443922)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc18443923)

[ЗАКЛЮЧЕНИЕ 5](#_Toc18443924)

[Приложение 1 (2, …). *Наименование приложения* 6](#_Toc18443925)

# ЦЕЛЬ

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

# ОПИСАНИЕ РАБОТЫ

*Пошаговое описание выполненной работы:*

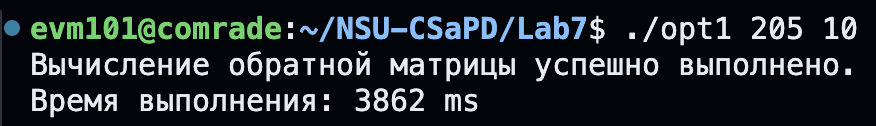
1. *Написать три варианта программы, реализующей алгоритм из задания:*
   1. *Вариант без ручной векторизации*
   2. *Вариант с ручной векторизацией*
   3. *Вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS*
2. *Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.*
3. *Сравнить время работы трех вариантов программы.*

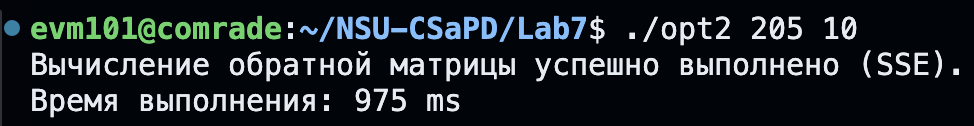
*Выбранная методика: векторное расширение SSE.*

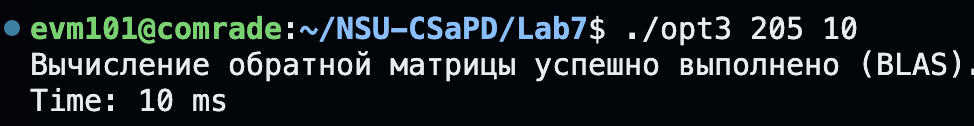
*Строки компиляции для третьего варианта:*

*g++ main.cpp -o main -lblas*

*Результаты для N = 205, M = 10:*

**

**

**

*Сравнение:*

*Вычисление с ручной векторизацией быстрее в ~3.96 раз, а с использованием BLAS — ~386.2*

# ЗАКЛЮЧЕНИЕ

*В ходе лабораторной работы я изучил SIMD-расширения архитектуры x86/x86-64 и способы использования SIMD-расширений в программах на языке Си , а так же получил навыки использования SIMD-расширений.*

# Приложение 1. *Листинг варианта без ручной векторизации*

#include <iostream>

#include <vector>

#include <cmath>

#include <chrono>

using namespace std;

// Умножение матриц A и B

vector<vector<float>> mulMatrices(const vector<vector<float>>& A, const vector<vector<float>>& B, int N) {

vector<vector<float>> result(N, vector<float>(N, 0));

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

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

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

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

}

}

}

return result;

}

// Сложение матриц A и B

vector<vector<float>> addMatrices(const vector<vector<float>>& A, const vector<vector<float>>& B, int N) {

vector<vector<float>> result(N, vector<float>(N, 0));

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

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

result[i][j] = A[i][j] + B[i][j];

}

}

return result;

}

// Вычитание матриц A - B

vector<vector<float>> subMatrices(const vector<vector<float>>& A, const vector<vector<float>>& B, int N) {

vector<vector<float>> result(N, vector<float>(N, 0));

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

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

result[i][j] = A[i][j] - B[i][j];

}

}

return result;

}

// Функция для нахождения обратной матрицы

void invertMatrix(const vector<vector<float>>& A, vector<vector<float>>& A\_inv, int N, int M) {

// Нормы ||A||\_1 и ||A||\_inf

float norm\_1 = 0, norm\_inf = 0;

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

float row\_sum = 0, col\_sum = 0;

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

row\_sum += fabs(A[i][j]);

col\_sum += fabs(A[j][i]);

}

norm\_inf = max(norm\_inf, row\_sum);

norm\_1 = max(norm\_1, col\_sum);

}

// Инициализация B и R

float B\_coef = 1.0f / (norm\_1 \* norm\_inf);

vector<vector<float>> B(N, vector<float>(N, 0));

vector<vector<float>> R(N, vector<float>(N, 0));

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

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

B[i][j] = A[j][i] \* B\_coef; // B = A^T / (||A||\_1 \* ||A||\_inf)

}

}

// R = I - BA

vector<vector<float>> I(N, vector<float>(N, 0));

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

I[i][i] = 1.0f;

}

R = subMatrices(I, mulMatrices(B, A, N), N);

// Итерационный процесс

A\_inv = B; // Начальное приближение

vector<vector<float>> R\_pow = I; // R^0 = I

for (int m = 1; m <= M; m++) {

R\_pow = mulMatrices(R\_pow, R, N); // R^m = R^(m-1) \* R

A\_inv = addMatrices(A\_inv, mulMatrices(R\_pow, B, N), N); // A\_inv += R^m \* B

}

}

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

if (argc < 3) {

cerr << "Too few arguments!" << endl;

}

int N = stoi(string(argv[1]));

int M = stoi(string(argv[2]));

vector<vector<float>> A = {

{4, 1, 2, 3},

{3, 4, 1, 2},

{2, 3, 4, 1},

{1, 2, 3, 4}

};

vector<vector<float>> A\_inv(N, vector<float>(N, 0));

auto start = chrono::high\_resolution\_clock::now();

invertMatrix(A, A\_inv, N, M);

auto end = chrono::high\_resolution\_clock::now();

cout << "Вычисление обратной матрицы успешно выполнено." << endl;

cout << "Время выполнения: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " ms\n" << endl;

for (const auto& row : A\_inv) {

for (float val : row)

cout << val << " ";

cout << endl;

}

return 0;

}

# Приложение 2. *Листинг варианта с использованием ручной векторизации*

#include <iostream>

#include <vector>

#include <algorithm>

#include <chrono>

#include <xmmintrin.h>

#include <cmath>

using namespace std;

// Умножение матриц A и B

void mulMatrices(const float \*A, const float \*B, float \*result, int N) {

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

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

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

for (int k = 0; k < N; k += 4) {

\_\_m128 a = \_mm\_loadu\_ps(&A[i \* N + k]);

\_\_m128 b = \_mm\_set\_ps(B[(k + 3) \* N + j], B[(k + 2) \* N + j], B[(k + 1) \* N + j], B[k \* N + j]);

sum = \_mm\_add\_ps(sum, \_mm\_mul\_ps(a, b));

}

float partial\_sum[4];

\_mm\_storeu\_ps(partial\_sum, sum);

result[i \* N + j] = partial\_sum[0] + partial\_sum[1] + partial\_sum[2] + partial\_sum[3];

}

}

}

// Сложение матриц A и B

void addMatrices(const float \*A, const float \*B, float \*result, int N) {

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

for (int j = 0; j < N; j += 4) {

\_\_m128 a = \_mm\_loadu\_ps(&A[i \* N + j]);

\_\_m128 b = \_mm\_loadu\_ps(&B[i \* N + j]);

\_\_m128 sum = \_mm\_add\_ps(a, b);

\_mm\_storeu\_ps(&result[i \* N + j], sum);

}

}

}

// Вычитание матриц A и B

void subMatrices(const float \*A, const float \*B, float \*result, int N) {

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

for (int j = 0; j < N; j += 4) {

\_\_m128 a = \_mm\_loadu\_ps(&A[i \* N + j]);

\_\_m128 b = \_mm\_loadu\_ps(&B[i \* N + j]);

\_\_m128 diff = \_mm\_sub\_ps(a, b);

\_mm\_storeu\_ps(&result[i \* N + j], diff);

}

}

}

void invertMatrix(const float \*A, float \*A\_inv, int N, int M) {

// Нормы ||A||\_1 и ||A||\_inf

float norm\_1 = 0, norm\_inf = 0;

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

float row\_sum = 0, col\_sum = 0;

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

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

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

}

norm\_inf = max(norm\_inf, row\_sum);

norm\_1 = max(norm\_1, col\_sum);

}

// Инициализация B и I

float B\_coef = 1.0f / (norm\_1 \* norm\_inf);

vector<float> B(N \* N, 0.0f);

vector<float> I(N \* N, 0.0f);

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

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

B[i \* N + j] = A[j \* N + i] \* B\_coef; // B = A^T / (||A||\_1 \* ||A||\_inf)

}

I[i \* N + i] = 1.0f;

}

// R = I - BA

vector<float> R(N \* N, 0.0f);

vector<float> BA(N \* N, 0.0f);

mulMatrices(B.data(), A, BA.data(), N);

subMatrices(I.data(), BA.data(), R.data(), N);

// Итерационный процесс

copy(B.begin(), B.end(), A\_inv);

vector<float> R\_pow = I;

vector<float> temp(N \* N, 0.0f);

for (int m = 1; m <= M; m++) {

mulMatrices(R\_pow.data(), R.data(), temp.data(), N);

R\_pow.swap(temp);

mulMatrices(R\_pow.data(), B.data(), temp.data(), N);

addMatrices(A\_inv, temp.data(), A\_inv, N);

}

}

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

if (argc < 3) {

cerr << "Too few arguments!" << endl;

}

int N = stoi(string(argv[1]));

int M = stoi(string(argv[2]));

vector<float> A = {

4, 1, 2, 3,

3, 4, 1, 2,

2, 3, 4, 1,

1, 2, 3, 4

};

vector<float> A\_inv(N \* N, 0.0f);

auto start = chrono::high\_resolution\_clock::now();

invertMatrix(A.data(), A\_inv.data(), N, M);

auto end = chrono::high\_resolution\_clock::now();

cout << "Вычисление обратной матрицы успешно выполнено (SSE)." << endl;

cout << "Время выполнения: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " ms\n" << endl;

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

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

cout << A\_inv[i \* N + j] << ' ';

}

cout << endl;

}

return 0;

}

# Приложение 3. *Вариант с матричными операциями, выполненными с помощью библиотеки BLAS*

#include <iostream>

#include <vector>

#include <algorithm>

#include <xmmintrin.h>

#include <cmath>

#include <chrono>

#include <cblas.h>

using namespace std;

void mulMatrices(const float \*A, const float \*B, float \*result, int N) {

cblas\_sgemm(

CblasRowMajor,

CblasNoTrans,

CblasNoTrans,

N, N, N,

1.0f,

A, N,

B, N,

0.0f,

result, N

);

}

// Сложение матриц A и B

void addMatrices(const float \*A, const float \*B, float \*result, int N) {

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

for (int j = 0; j < N; j += 4) {

\_\_m128 a = \_mm\_loadu\_ps(&A[i \* N + j]);

\_\_m128 b = \_mm\_loadu\_ps(&B[i \* N + j]);

\_\_m128 sum = \_mm\_add\_ps(a, b);

\_mm\_storeu\_ps(&result[i \* N + j], sum);

}

}

}

// Вычитание матриц A и B

void subMatrices(const float \*A, const float \*B, float \*result, int N) {

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

for (int j = 0; j < N; j += 4) {

\_\_m128 a = \_mm\_loadu\_ps(&A[i \* N + j]);

\_\_m128 b = \_mm\_loadu\_ps(&B[i \* N + j]);

\_\_m128 diff = \_mm\_sub\_ps(a, b);

\_mm\_storeu\_ps(&result[i \* N + j], diff);

}

}

}

void invertMatrix(const float \*A, float \*A\_inv, int N, int M) {

// Нормы ||A||\_1 и ||A||\_inf

float norm\_1 = 0, norm\_inf = 0;

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

float row\_sum = 0, col\_sum = 0;

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

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

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

}

norm\_inf = max(norm\_inf, row\_sum);

norm\_1 = max(norm\_1, col\_sum);

}

// Инициализация B и I

float B\_coef = 1.0f / (norm\_1 \* norm\_inf);

vector<float> B(N \* N, 0.0f);

vector<float> I(N \* N, 0.0f);

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

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

B[i \* N + j] = A[j \* N + i] \* B\_coef; // B = A^T / (||A||\_1 \* ||A||\_inf)

}

I[i \* N + i] = 1.0f;

}

// R = I - BA

vector<float> R(N \* N, 0.0f);

vector<float> BA(N \* N, 0.0f);

mulMatrices(B.data(), A, BA.data(), N);

subMatrices(I.data(), BA.data(), R.data(), N);

// Итерационный процесс

copy(B.begin(), B.end(), A\_inv);

vector<float> R\_pow = I;

vector<float> temp(N \* N, 0.0f);

for (int m = 1; m <= M; m++) {

mulMatrices(R\_pow.data(), R.data(), temp.data(), N);

R\_pow.swap(temp);

mulMatrices(R\_pow.data(), B.data(), temp.data(), N);

addMatrices(A\_inv, temp.data(), A\_inv, N);

}

}

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

if (argc < 3) {

cerr << "Too few arguments!" << endl;

}

int N = stoi(string(argv[1]));

int M = stoi(string(argv[2]));

vector<float> A = {

4, 1, 2, 3,

3, 4, 1, 2,

2, 3, 4, 1,

1, 2, 3, 4

};

vector<float> A\_inv(N \* N, 0.0f);

auto start = chrono::high\_resolution\_clock::now();

invertMatrix(A.data(), A\_inv.data(), N, M);

auto end = chrono::high\_resolution\_clock::now();

cout << "Вычисление обратной матрицы успешно выполнено (BLAS)." << endl;

cout << "Time: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " ms\n" << endl;

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

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

cout << A\_inv[i \* N + j] << ' ';

}

cout << endl;

}

return 0;

}