**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, группы 23201

Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Матвеев Алексей Сергеевич

Новосибирск 2024

**ЦЕЛЬ**

Изучить SIMD-расширения архитектуры x86/x86-64 и способы использования SIMD-расширений в программах на языке C++. Получить навыки использования SIMD-расширений.

**ЗАДАНИЕ**

Написать три варианта программы, реализующей алгоритм обращения матрицы A размером N×N с помощью разложения в ряд:

* вариант без ручной векторизации
* вариант с ручной векторизацией
* вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS.

Для элементов матриц использовать тип данных float.

**Листинги**

**include/mat.h**

#ifndef MAT\_H

#define MAT\_H

#include <vector>

class Mat {

public:

int size;

alignas(16) std::vector<float> data;

explicit Mat(int n);

float& at(int row, int col);

void fillRandom();

void fillIdentity();

void transpose();

float findMaxAbsSumByRows();

float calcDiff(Mat& other) const;

void print();

};

#endif //MAT\_H

**mat.cpp**

#include "mat.h"

#include <iomanip>

#include <random>

#include <iostream>

Mat::Mat(int n) : size(n), data(n \* n, 0.0f) {}

float& Mat::at(int row, int col) {

return data[row \* size + col];

}

void Mat::fillRandom() {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(0, 10.0);

size\_t N = size;

for (std::size\_t i = 0; i < N; i++)

for (std::size\_t j = 0; j < N; j++)

data[i \* N + j] = dis(gen);

for (std::size\_t i = 0; i < N; i++) {

float sum = 0.f;

for (std::size\_t j = 0; j < N; j++)

sum += data[i \* N + j];

data[i \* N + i] += sum;

}

}

void Mat::fillIdentity() {

std::fill(data.begin(), data.end(), 0);

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

at(i, i) = 1.0f;

}

}

void Mat::transpose() {

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

for (int j = i + 1; j < size; ++j) {

std::swap(at(i, j), at(j, i));

}

}

}

float Mat::findMaxAbsSumByRows() {

float result = 0.0f;

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

float current\_sum = 0.0f;

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

current\_sum += std::fabs(data[i \* size + j]);

}

result = std::max(result, current\_sum);

}

return result;

}

float Mat::calcDiff(Mat& other) const {

float diff = 0.0f;

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

diff += std::fabs(data[i] - other.data[i]);

}

return diff;

}

void Mat::print() {

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

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

std::cout << std::fixed << std::setprecision(3) << at(i, j) << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

**src/include/novec.h**

#ifndef NOVEC\_H

#define NOVEC\_H

#include "mat.h"

class MatNovec : public Mat {

public:

explicit MatNovec(int n);

void subtract(MatNovec& other);

void add(const MatNovec& other);

void multiplyByNumber(float number);

MatNovec multiply(MatNovec& other);

static MatNovec findInverseMatrix(MatNovec& matrix, int iterations);

};

#endif //NOVEC\_H

**src/novec.cpp**

#include "novec.h"

MatNovec::MatNovec(int n): Mat(n) {}

void MatNovec::subtract(MatNovec& other) {

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

data[i] -= other.data[i];

}

}

void MatNovec::add(const MatNovec& other) {

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

data[i] += other.data[i];

}

}

void MatNovec::multiplyByNumber(float number) {

for (auto& element : data) {

element \*= number;

}

}

MatNovec MatNovec::multiply(MatNovec& other) {

MatNovec result(size);

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

float \*c = result.data.data() + i \* size; // указатель на начало строки в результирующей матрице

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

const float \*b = other.data.data() + k \* size; // указатель на начало строки во второй матрице (other)

float a = at(i, k); // элемент из текущей матрицы

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

c[j] += a \* b[j];

}

}

}

return result;

}

MatNovec MatNovec::findInverseMatrix(MatNovec& matrix, int iterations) {

int n = matrix.size;

MatNovec B = matrix;

B.transpose();

// нахождение коэффициентов для масштабирования B

// после масштабирования значения матрицы будут находиться в пределах, близких к 1, что

// минимизирует ошибки, возникающие при итерационном процессе

float A1 = matrix.findMaxAbsSumByRows();

float A2 = B.findMaxAbsSumByRows(); // так как B транспонирована, то A2 = сумма по столбцам

B.multiplyByNumber(1 / (A1 \* A2));

MatNovec R(n);

R.fillIdentity();

// вычисление BA и R = E - BA

MatNovec BA = B.multiply(matrix);

R.subtract(BA);

MatNovec result(n);

result.fillIdentity();

// Вычисление R^k и добавление к result.

MatNovec RSeries = R;

for (int i = 1; i < iterations; ++i) {

RSeries = RSeries.multiply(R);

result.add(RSeries);

}

// B.transpose();

return result.multiply(B);

// R = E - BA

// A^{-1} = B \* (E + R + R^2 + ... + R^{k})

}

**src/include/vec.h**

#ifndef VEC\_H

#define VEC\_H

#include "mat.h"

class MatVec : public Mat {

public:

explicit MatVec(int n);

void multiplyByNumber(float number);

MatVec multiply(MatVec& other);

static MatVec findInverseMatrix(MatVec& matrix, int iterations);

};

#endif //VEC\_H

**src/vec.cpp**

#include "vec.h"

#include <immintrin.h>

MatVec::MatVec(int n): Mat(n) {}

void MatVec::multiplyByNumber(float number) {

for (auto& element : data) {

element \*= number;

}

}

MatVec MatVec::multiply(MatVec& other) {

MatVec result(size);

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

float \*c = result.data.data() + i \* size; // указатель на начало строки в результирующей матрице

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

float \*b = other.data.data() + k \* size; // указатель на начало строки во второй матрице (other)

float a = data[i \* size + k]; // элемент из текущей матрицы

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

\_\_m128 b\_vec = \_mm\_load\_ps(b + j); // загружаем 4 элемента из b

\_\_m128 a\_vec = \_mm\_set1\_ps(a); // загружаем a в каждую позицию вектора

\_\_m128 c\_vec = \_mm\_load\_ps(c + j); // загружаем текущие значения результата

c\_vec = \_mm\_add\_ps(c\_vec, \_mm\_mul\_ps(a\_vec, b\_vec)); // c[j:j+4] += a \* b[j:j+4]

\_mm\_store\_ps(c + j, c\_vec); // сохраняем обратно в c

}

}

}

return result;

}

MatVec MatVec::findInverseMatrix(MatVec& matrix, int iterations) {

MatVec B = matrix;

B.transpose();

float A1 = matrix.findMaxAbsSumByRows();

float A2 = B.findMaxAbsSumByRows();

B.multiplyByNumber(1.0f / (A1 \* A2));

// инициализация матрицы ошибки R, изначально она единичная

MatVec R(matrix.size);

R.fillIdentity();

// BA = (A^{T}/(A1 \* A2)) \* A

MatVec BA = B.multiply(matrix);

// R = E - BA

for (int j = 0; j < matrix.size \* matrix.size; j += 4) {

\_\_m128 r\_vec = \_mm\_load\_ps(R.data.data() + j);

\_\_m128 ba\_vec = \_mm\_load\_ps(BA.data.data() + j);

r\_vec = \_mm\_sub\_ps(r\_vec, ba\_vec);

\_mm\_store\_ps(R.data.data() + j, r\_vec);

}

MatVec result(matrix.size);

result.fillIdentity();

// Вычисление R^k и добавление к result

MatVec RSeries = R;

for (int i = 1; i < iterations; i++) {

RSeries = RSeries.multiply(R);

// result += R^k (R^k = Rseries)

for (int j = 0; j < matrix.size \* matrix.size; j += 4) {

\_\_m128 a\_vec = \_mm\_load\_ps(result.data.data() + j);

\_\_m128 b\_vec = \_mm\_load\_ps(RSeries.data.data() + j);

a\_vec = \_mm\_add\_ps(a\_vec, b\_vec);

\_mm\_store\_ps(result.data.data() + j, a\_vec);

}

}

result = result.multiply(B);

return result;

}

**src/include/blas.h**

#ifndef BLAS\_H

#define BLAS\_H

#include "mat.h"

class MatBlas : public Mat {

public:

explicit MatBlas(int n);

void multiplyByNumber(float number);

MatBlas multiply(MatBlas& other);

static MatBlas findInverseMatrix(MatBlas& matrix, int M);

};

#endif //BLAS\_H

**src/blas.cpp**

#include "blas.h"

#include <cblas.h>

MatBlas::MatBlas(int n): Mat(n) {}

void MatBlas::multiplyByNumber(float number) {

cblas\_sscal(size \* size, number, data.data(), 1);

}

MatBlas MatBlas::multiply(MatBlas& other) {

MatBlas result(size);

// size - количество строк/столбцов 1 матрицы

// other.size - количество столбцов 2 матрицы

// 1.0f - коэффицент умножения 1 (1 \* m1 \* m2)

// 0.0f - коэффицент умножения 2 (0 \* m3)

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,

size, other.size, size,

1.0f,

data.data(), size,

other.data.data(), other.size,

0.0f, result.data.data(), result.size);

return result;

}

MatBlas MatBlas::findInverseMatrix(MatBlas& matrix, int M) {

MatBlas B = matrix;

B.transpose();

float A1 = matrix.findMaxAbsSumByRows();

float A2 = B.findMaxAbsSumByRows();

B.multiplyByNumber(1.0f / (A1 \* A2)); // масштабируем B приближая его к обратной матрице

MatBlas R = MatBlas(matrix.size);

R.fillIdentity();

MatBlas result = R; // изначально result = E

MatBlas BA = B.multiply(matrix); // BA - приближение умножить на аргумент

cblas\_saxpy(matrix.size \* matrix.size, -1, BA.data.data(), 1, R.data.data(), 1); // R = E + (-1)BA

MatBlas RSeries = R;

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

RSeries = RSeries.multiply(R);

// result += RSeries

cblas\_saxpy(result.size \* result.size, 1.0f, RSeries.data.data(), 1, result.data.data(), 1); // result += 1 \* R\_series

}

return result.multiply(B);

}

**src/main.cpp**

#include <chrono>

#include <iostream>

#include "novec.h"

#include "vec.h"

#include "blas.h"

template <typename MatType>

void testMat(int N, int M) {

MatType matrix(N);

matrix.fillRandom();

// std::cout << "Original matrix: " << std::endl;

// matrix.print();

std::cout << "Calculating inverse matrix (" << typeid(MatType).name() << ")..." << std::endl;

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

MatType inverse = MatType::findInverseMatrix(matrix, M);

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

const double duration = std::chrono::duration\_cast<std::chrono::duration<double>>(end - start).count();

// std::cout << "Inverse matrix: " << std::endl;

// inverse.print();

MatType identity = inverse.multiply(matrix);

MatType trueIdentity(N);

trueIdentity.fillIdentity();

// std::cout << "Identity matrix: " << std::endl;

// identity.print();

// std::cout << "Expected identity matrix: " << std::endl;

// expectedIdentity.print();

std::cout << "Time taken: " << duration << " seconds" << std::endl;

std::cout << "Difference: " << identity.calcDiff(trueIdentity) << std::endl << std::endl;

}

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

int N = 2048;

int M = 10;

if (argc == 3) {

N = std::stoi(argv[1]);

M = std::stoi(argv[2]);

}

testMat<MatNovec>(N, M);

testMat<MatVec>(N, M);

testMat<MatBlas>(N, M);

return 0;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.14)

project(lab7)

set(CMAKE\_CXX\_STANDARD 17)

set(CMAKE\_CXX\_FLAGS "-O3 -fno-tree-vectorize")

add\_executable(${PROJECT\_NAME}

src/main.cpp

src/mat.cpp

src/novec.cpp

src/vec.cpp

src/blas.cpp

)

find\_package(CBLAS REQUIRED)

target\_link\_libraries(${PROJECT\_NAME} cblas)

target\_include\_directories(${PROJECT\_NAME} PRIVATE ./src/include)

**Результаты измерений**

|  |  |  |
| --- | --- | --- |
| **Вариант исполнения** | **Время O0, сек** | **Время O3, сек** |
| novec | 220.63 | 100.11 |
| vec | 142.58 | 44.99 |
| blas | 37.33 | 36.87 |

**ЗАКЛЮЧЕНИЕ**

В ходе работы были изучены SIMD-расширения архитектуры x86/x86-64 и их использование в коде, а также изучена работа оптимизированной библиотеки линейной алгебры BLAS.  
Как было выяснено, лучше всего пользоваться готовой библиотекой, хотя ручная векторизация и может дать ощутимый прирост в производительности. Единственный недостаток библиотеки BLAS составляет большое количество разных параметров, из-за которых код становится трудно понимаемым.