###### 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 ПРАКТИЧЕСКОЙ РАБОТЫ

Векторизация

студента 2 курса, группы 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 3](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 3](#__RefHeading___4)

[Приложение 1 (main) 3](#__RefHeading___5)

[Приложение 2 (usual\_matrix.h) 5](#__RefHeading___6)

[Приложение 3 (usual\_matrix.cpp) 6](#__RefHeading___7)

[Приложение 4 (blas\_matrix.h) 12](#__RefHeading___8)

[Приложение 5 (blas\_matrix.cpp) 13](#__RefHeading___9)

[Приложение 6 (intrinsic\_matrix.h) 18](#__RefHeading___10)

[Приложение 7 (intrinsic\_matrix.cpp) 20](#__RefHeading___11)

[Приложение 8 (CMakeLists.txt) 26](#__RefHeading___12)

# 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

# ЗАДАНИЕ

1. Написать три варианта программы, реализующей алгоритм из задания:

· Вариант без векторизации.

· Вариант с ручной векторизацией (выбрать любой вариант из возможных трех: ассемблерная вставка, встроенные функции компилятора, расширение GCC). При этом векторизованы должны быть как минимум операции перемножения двух матриц, сложения матриц, умножения матрицы на скаляр.

· Вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS. Для элементов матриц использовать тип данных float. Использовать минимум 2 BLAS-функции.

2. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.

3. Каждый вариант программы оптимизировать по скорости, насколько это возможно.

4. Сравнить время работы трех вариантов программы для N=2048, M=10.

# ОПИСАНИЕ РАБОТЫ

Было принято решение написать 3 класса под каждый из вариантов работы с матрицами (без оптимизаций, с использованием blas и с ручной векторизацией через встроенные функции). Интерфейс у них одинаковый, разница только в реализации.

В варианте с ручной оптимизацией были использованы YMM регистры из AVX. Они могут вмещать до 8 чисел float. Из интересного, перед умножением матриц, вторая из них транспонируется. Также внутри цикла применяется параллельное сложение.

Результаты по времени работы вычисления обратной матрицы получились следующими (M = 10):

|  |  |  |  |
| --- | --- | --- | --- |
| N | Usual | Blas | Intrinsic |
| 512 | 9.7 | 0.35 | 2.5 |
| 1024 | 74 | 3 | 20 |
| 2048 | 594 | 22 | 164 |

# ЗАКЛЮЧЕНИЕ

Казалось бы, с векторизацией скорость должна увеличиться в 8 раз, но на деле она увеличилась примерно в 4 раза. Так получается из за того, что несмотря на меньшее число повторений цикла, его тело становится более сложным (вместо 1 сложения – загрузка векторных регистров, параллельное сложение и загрузка обратно в память).

Вывод: выгоднее всего использовать готовые библиотеки (например blas)

# Приложение 1 (main)

#include "blas\_matrix.h"

#include "intrinsic\_matrix.h"

#include "usual\_matrix.h"

#include <iostream>

using namespace std;

#define debug 0

#if (debug)

const int N = 8;

const int M = 1000;

#else

const int N = 2048;

const int M = 10;

#endif

template <class MartixType>

float testMatrix()

{

    MartixType A(N);

    for (int j = 0; j < N; ++j)

    {

        for (int i = 0; i < N; ++i)

        {

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

        }

    }

    MartixType Res(N);

    clock\_t startTime = clock();

    A.reverse(Res, M);

    clock\_t finalTime = clock();

#if (debug)

    cout << "\n";

    Res.print();

#endif

    return float(finalTime - startTime) / CLOCKS\_PER\_SEC;

}

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

{

    float time;

    time = testMatrix<UsualMatrix>();

    cout << "Usual Time: " << time << "\n";

    time = testMatrix<BlasMatrix>();

    cout << "Blas Time: " << time << "\n";

    time = testMatrix<IntrinsicMatrix>();

    cout << "Intrinsic Time: " << time << "\n";

    return 0;

}

# Приложение 2 (usual\_matrix.h)

#ifndef USUAL\_MATRIX\_H

#define USUAL\_MATRIX\_H

class UsualMatrix

{

private:

    float \*arr\_;

    int size\_;

    // Возвращает индекс элемента в матрице с координатами (x, y)

    int id(int x, int y) const noexcept;

public:

    // Создаёт матрицу размером size \* size

    UsualMatrix(int size);

    // Деструктор

    ~UsualMatrix() noexcept;

    // Заполняет матрицу нулями

    void makeZeroMatrix() noexcept;

    // Заполняет элементы так, чтобы получилась единична матрица

    void makeUnitMatrix() noexcept;

    // Копирование this в other

    void copy(UsualMatrix &other) const noexcept;

    // Выводит матрицу в консоль

    void print() const noexcept;

    // Для обращения вида: matrix[y][x]

    float \*operator[](int y) noexcept;

    // Максимальная сумма столбцов

    float mod\_1() const noexcept;

    // Максимальная сумма строк

    float mod\_inf() const noexcept;

    // Транспонирование: this^T = res

    void transpose(UsualMatrix &res) const noexcept;

    // Добавить единичную матрицу

    void addUnitMatrix() noexcept;

    // Разница: this - other = res

    void sub(const UsualMatrix &other, UsualMatrix &res) const noexcept;

    // Умножение на число: this \* alpha = this

    void mult(const float alpha) noexcept;

    // Умножение матриц: this \* other = res

    void mult(const UsualMatrix &other, UsualMatrix &res) const noexcept;

    // Обращение маорицы: this^(-1) = res. Точность зависит от количества членов в ряде (iterations)

    void reverse(UsualMatrix &res, const int iterations) const noexcept;

};

#endif

# Приложение 3 (usual\_matrix.cpp)

#include "usual\_matrix.h"

#include <iostream>

UsualMatrix::UsualMatrix(int size)

{

    arr\_ = new float[size \* size];

    size\_ = size;

}

UsualMatrix::~UsualMatrix() noexcept

{

    delete[] arr\_;

}

void UsualMatrix::makeZeroMatrix() noexcept

{

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

        arr\_[i] = 0;

}

void UsualMatrix::makeUnitMatrix() noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            if (i == j)

                arr\_[id(i, j)] = 1;

            else

                arr\_[id(i, j)] = 0;

        }

    }

}

void UsualMatrix::copy(UsualMatrix &other) const noexcept

{

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

        other.arr\_[i] = arr\_[i];

}

void UsualMatrix::print() const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            std::cout << arr\_[id(i, j)] << "\t";

        }

        std::cout << "\n";

    }

    std::cout << "\n";

}

int UsualMatrix::id(const int x, const int y) const noexcept

{

    return y \* size\_ + x;

}

float \*UsualMatrix::operator[](int y) noexcept

{

    return arr\_ + (y \* size\_);

}

float UsualMatrix::mod\_1() const noexcept

{

    float ans = FLT\_MIN;

    for (int i = 0; i < size\_; ++i)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

float UsualMatrix::mod\_inf() const noexcept

{

    float ans = FLT\_MIN;

    for (int j = 0; j < size\_; ++j)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

void UsualMatrix::transpose(UsualMatrix &res) const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            res.arr\_[id(i, j)] = arr\_[id(j, i)];

        }

    }

}

void UsualMatrix::sub(const UsualMatrix &other, UsualMatrix &res) const noexcept

{

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

        res.arr\_[i] = arr\_[i] - other.arr\_[i];

}

void UsualMatrix::mult(const float alpha) noexcept

{

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

        arr\_[i] \*= alpha;

}

void UsualMatrix::mult(const UsualMatrix &other, UsualMatrix &res) const noexcept

{

    res.makeZeroMatrix();

    for (int j = 0; j < size\_; j++)

    {

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

        {

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

            {

                res.arr\_[id(k, j)] += arr\_[id(i, j)] \* other.arr\_[id(k, i)];

            }

        }

    }

}

void UsualMatrix::addUnitMatrix() noexcept

{

    for (int i = 0; i < size\_; ++i)

        ++arr\_[id(i, i)];

}

void UsualMatrix::reverse(UsualMatrix &res, const int iterations) const noexcept

{

    UsualMatrix B(size\_);

    // B = A^T

    this->transpose(B);

    // B = A^T / (mod\_1 \* mod\_inf)

    B.mult(float(1) / (this->mod\_1() \* this->mod\_inf()));

    UsualMatrix I(size\_);

    // I = единичная матрица

    I.makeUnitMatrix();

    UsualMatrix Tmp(size\_);

    // Tmp = B \* A

    B.mult(\*this, Tmp);

    UsualMatrix R(size\_);

    // R = I

    R.makeUnitMatrix();

    // R = I - Tmp = I - BA

    I.sub(Tmp, R);

    // Tmp = I

    I.copy(Tmp);

    bool flag = true;

    for (int i = 0; i < iterations - 1; ++i)

    {

        if (flag)

        {

            Tmp.mult(R, I);

            I.addUnitMatrix();

        }

        else

        {

            I.mult(R, Tmp);

            Tmp.addUnitMatrix();

        }

        flag = !flag;

    }

    if (flag)

        Tmp.mult(B, res);

    else

        I.mult(B, res);

}

# Приложение 4 (blas\_matrix.h)

#ifndef BLAS\_MATRIX\_H

#define BLAS\_MATRIX\_H

class BlasMatrix

{

private:

    float \*arr\_;

    int size\_;

    // Возвращает индекс элемента в матрице с координатами (x, y)

    int id(int x, int y) const noexcept;

public:

    // Создаёт матрицу размером size \* size

    BlasMatrix(int size);

    // Деструктор

    ~BlasMatrix() noexcept;

    // Заполняет матрицу нулями

    void makeZeroMatrix() noexcept;

    // Заполняет элементы так, чтобы получилась единична матрица

    void makeUnitMatrix() noexcept;

    // Копирование this в other

    void copy(BlasMatrix &other) const noexcept;

    // Выводит матрицу в консоль

    void print() const noexcept;

    // Для обращения вида: matrix[y][x]

    float \*operator[](int y) noexcept;

    // Максимальная сумма столбцов

    float mod\_1() const noexcept;

    // Максимальная сумма строк

    float mod\_inf() const noexcept;

    // Транспонирование: this^T = res

    void transpose(BlasMatrix &res) const noexcept;

    // Добавить единичную матрицу

    void addUnitMatrix() noexcept;

    // Разница: this - other = res

    void sub(const BlasMatrix &other, BlasMatrix &res) const noexcept;

    // Умножение на число: this \* alpha = this

    void mult(const float alpha) noexcept;

    // Умножение матриц: this \* other = res

    void mult(const BlasMatrix &other, BlasMatrix &res) const noexcept;

    // Обращение маорицы: this^(-1) = res. Точность зависит от количества членов в ряде (iterations)

    void reverse(BlasMatrix &res, const int iterations) const noexcept;

};

#endif

# Приложение 5 (blas\_matrix.cpp)

#include "blas\_matrix.h"

#include "../openblas\_x64-windows/include/openblas/cblas.h"

#include <iostream>

BlasMatrix::BlasMatrix(int size)

{

    arr\_ = new float[size \* size];

    size\_ = size;

}

BlasMatrix::~BlasMatrix() noexcept

{

    delete[] arr\_;

}

void BlasMatrix::makeZeroMatrix() noexcept

{

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

        arr\_[i] = 0;

}

void BlasMatrix::makeUnitMatrix() noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            if (i == j)

                arr\_[id(i, j)] = 1;

            else

                arr\_[id(i, j)] = 0;

        }

    }

}

void BlasMatrix::copy(BlasMatrix &other) const noexcept

{

    // Копирование (Размер массива, массив 1, шаг в массиве 1, массив 2, шаг в массиве 2)

    cblas\_scopy(size\_ \* size\_, arr\_, 1, other.arr\_, 1);

}

void BlasMatrix::print() const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            std::cout << arr\_[id(i, j)] << "\t";

        }

        std::cout << "\n";

    }

    std::cout << "\n";

}

int BlasMatrix::id(const int x, const int y) const noexcept

{

    return y \* size\_ + x;

}

float \*BlasMatrix::operator[](int y) noexcept

{

    return arr\_ + (y \* size\_);

}

float BlasMatrix::mod\_1() const noexcept

{

    float ans = FLT\_MIN;

    for (int i = 0; i < size\_; ++i)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

float BlasMatrix::mod\_inf() const noexcept

{

    float ans = FLT\_MIN;

    for (int j = 0; j < size\_; ++j)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

void BlasMatrix::transpose(BlasMatrix &res) const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            res.arr\_[id(i, j)] = arr\_[id(j, i)];

        }

    }

}

void BlasMatrix::sub(const BlasMatrix &other, BlasMatrix &res) const noexcept

{

    // Копирование (Размер массива, массив 1, шаг в массиве 1, массив 2, шаг в массиве 2)

    cblas\_scopy(size\_ \* size\_, arr\_, 1, res.arr\_, 1);

    // Сложение (Размер массива, множитель, массив 1, шаг в массиве 1, массив 2, шаг в массиве 2)

    cblas\_saxpy(size\_ \* size\_, -1.0, other.arr\_, 1, res.arr\_, 1);

}

void BlasMatrix::mult(const float alpha) noexcept

{

    // Умножение на скаляр (Размер массива, множитель, массив 1, шаг в массиве 1)

    cblas\_sscal(size\_ \* size\_, alpha, arr\_, 1);

}

void BlasMatrix::mult(const BlasMatrix &other, BlasMatrix &res) const noexcept

{

    // Умножение матриц (... размеры матриц, alpha, A, локальный размер lda, B, локальный размер ldb, beta, C, локальный размер ldc)

    // C = alpha \* op(A) \* op(B) + beta \* C

    cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, size\_, size\_, size\_, 1.0, arr\_, size\_, other.arr\_, size\_, 0.0, res.arr\_, size\_);

}

void BlasMatrix::addUnitMatrix() noexcept

{

    for (int i = 0; i < size\_; ++i)

        ++arr\_[id(i, i)];

}

void BlasMatrix::reverse(BlasMatrix &res, const int iterations) const noexcept

{

    BlasMatrix B(size\_);

    this->transpose(B);

    B.mult(float(1) / (this->mod\_1() \* this->mod\_inf()));

    BlasMatrix I(size\_);

    I.makeUnitMatrix();

    BlasMatrix Tmp(size\_);

    B.mult(\*this, Tmp);

    BlasMatrix R(size\_);

    R.makeUnitMatrix();

    I.sub(Tmp, R);

    I.copy(Tmp);

    bool flag = true;

    for (int i = 0; i < iterations - 1; ++i)

    {

        if (flag)

        {

            Tmp.mult(R, I);

            I.addUnitMatrix();

        }

        else

        {

            I.mult(R, Tmp);

            Tmp.addUnitMatrix();

        }

        flag = !flag;

    }

    if (flag)

        Tmp.mult(B, res);

    else

        I.mult(B, res);

}

# Приложение 6 (intrinsic\_matrix.h)

#ifndef INTRINSIC\_MATRIX\_H

#define INTRINSIC\_MATRIX\_H

class IntrinsicMatrix

{

private:

    float \*arr\_;

    int size\_;

    // Возвращает индекс элемента в матрице с координатами (x, y)

    int id(int x, int y) const noexcept;

public:

    // Создаёт матрицу размером size \* size

    IntrinsicMatrix(int size);

    // Деструктор

    ~IntrinsicMatrix() noexcept;

    // Заполняет матрицу нулями

    void makeZeroMatrix() noexcept;

    // Заполняет элементы так, чтобы получилась единична матрица

    void makeUnitMatrix() noexcept;

    // Копирование this в other

    void copy(IntrinsicMatrix &other) const noexcept;

    // Выводит матрицу в консоль

    void print() const noexcept;

    // Для обращения вида: matrix[y][x]

    float \*operator[](int y) noexcept;

    // Максимальная сумма столбцов

    float mod\_1() const noexcept;

    // Максимальная сумма строк

    float mod\_inf() const noexcept;

    // Транспонирование: this^T = res

    void transpose(IntrinsicMatrix &res) const noexcept;

    // Добавить единичную матрицу

    void addUnitMatrix() noexcept;

    // Разница: this - other = res

    void sub(const IntrinsicMatrix &other, IntrinsicMatrix &res) const noexcept;

    // Умножение на число: this \* alpha = this

    void mult(const float alpha) noexcept;

    // Умножение матриц: this \* other = res

    void mult(const IntrinsicMatrix &other, IntrinsicMatrix &res) const noexcept;

    // Обращение маорицы: this^(-1) = res. Точность зависит от количества членов в ряде (iterations)

    void reverse(IntrinsicMatrix &res, const int iterations) const noexcept;

};

#endif

# Приложение 7 (intrinsic\_matrix.cpp)

#include "intrinsic\_matrix.h"

#include <intrin.h>

#include <iostream>

IntrinsicMatrix::IntrinsicMatrix(int size)

{

    arr\_ = new float[size \* size];

    size\_ = size;

}

IntrinsicMatrix::~IntrinsicMatrix() noexcept

{

    delete[] arr\_;

}

void IntrinsicMatrix::makeZeroMatrix() noexcept

{

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

        arr\_[i] = 0;

}

void IntrinsicMatrix::makeUnitMatrix() noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            if (i == j)

                arr\_[id(i, j)] = 1;

            else

                arr\_[id(i, j)] = 0;

        }

    }

}

void IntrinsicMatrix::copy(IntrinsicMatrix &other) const noexcept

{

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

        other.arr\_[i] = arr\_[i];

}

void IntrinsicMatrix::print() const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            std::cout << arr\_[id(i, j)] << "\t";

        }

        std::cout << "\n";

    }

    std::cout << "\n";

}

int IntrinsicMatrix::id(const int x, const int y) const noexcept

{

    return y \* size\_ + x;

}

float \*IntrinsicMatrix::operator[](int y) noexcept

{

    return arr\_ + (y \* size\_);

}

float IntrinsicMatrix::mod\_1() const noexcept

{

    float ans = FLT\_MIN;

    for (int i = 0; i < size\_; ++i)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

float IntrinsicMatrix::mod\_inf() const noexcept

{

    float ans = FLT\_MIN;

    for (int j = 0; j < size\_; ++j)

    {

        float sum = 0;

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

            sum += fabs(arr\_[id(i, j)]);

        if (sum > ans)

            ans = sum;

    }

    return ans;

}

void IntrinsicMatrix::transpose(IntrinsicMatrix &res) const noexcept

{

    for (int j = 0; j < size\_; ++j)

    {

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

        {

            res.arr\_[id(i, j)] = arr\_[id(j, i)];

        }

    }

}

void IntrinsicMatrix::sub(const IntrinsicMatrix &other, IntrinsicMatrix &res) const noexcept

{

    // \_\_m128 = 4 float в SSE

    // \_\_m256 = 8 float в AVX

    for (int i = 0; i < size\_ \* size\_; i += 8)

    {

        \_\_m256 vec1 = \_mm256\_loadu\_ps(arr\_ + i);

        \_\_m256 vec2 = \_mm256\_loadu\_ps(other.arr\_ + i);

        \_\_m256 vecRes = \_mm256\_sub\_ps(vec1, vec2);

        \_mm256\_storeu\_ps(res.arr\_ + i, vecRes);

    }

}

void IntrinsicMatrix::mult(const float alpha) noexcept

{

    \_\_m256 vecAlpha = \_mm256\_set\_ps(alpha, alpha, alpha, alpha, alpha, alpha, alpha, alpha);

    for (int i = 0; i < size\_ \* size\_; i += 8)

    {

        \_\_m256 vec = \_mm256\_load\_ps(arr\_ + i);

        vec = \_mm256\_mul\_ps(vec, vecAlpha);

        \_mm256\_storeu\_ps(arr\_ + i, vec);

    }

}

void IntrinsicMatrix::mult(const IntrinsicMatrix &other, IntrinsicMatrix &res) const noexcept

{

    IntrinsicMatrix resT(size\_);

    resT.makeZeroMatrix();

    IntrinsicMatrix BT(size\_);

    other.transpose(BT);

    \_\_m256 vecA, vecBT;

    for (int i = 0; i < size\_; ++i)

    {

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

        {

            for (int k = 0; k < size\_; k += 8)

            {

                vecA = \_mm256\_load\_ps(&arr\_[id(k, i)]);

                vecBT = \_mm256\_load\_ps(&BT.arr\_[id(k, j)]);

                vecBT = \_mm256\_mul\_ps(vecA, vecBT);

                // Выполняем горизонтальное сложение всех элементов вектора vecBT через параллельное сложение

                vecA = \_mm256\_permute2f128\_ps(vecBT, vecBT, 1);

                vecBT = \_mm256\_add\_ps(vecBT, vecA);

                vecBT = \_mm256\_hadd\_ps(vecBT, vecBT);

                vecBT = \_mm256\_hadd\_ps(vecBT, vecBT);

                // Извлекаем сумму из вектора

                resT.arr\_[id(i, j)] += \_mm256\_cvtss\_f32(vecBT);

            }

        }

    }

    resT.transpose(res);

}

void IntrinsicMatrix::addUnitMatrix() noexcept

{

    for (int i = 0; i < size\_; ++i)

        ++arr\_[id(i, i)];

}

void IntrinsicMatrix::reverse(IntrinsicMatrix &res, const int iterations) const noexcept

{

    IntrinsicMatrix B(size\_);

    this->transpose(B);

    B.mult(float(1) / (this->mod\_1() \* this->mod\_inf()));

    IntrinsicMatrix I(size\_);

    I.makeUnitMatrix();

    IntrinsicMatrix Tmp(size\_);

    B.mult(\*this, Tmp);

    IntrinsicMatrix R(size\_);

    R.makeUnitMatrix();

    I.sub(Tmp, R);

    I.copy(Tmp);

    bool flag = true;

    for (int i = 0; i < iterations - 1; ++i)

    {

        if (flag)

        {

            Tmp.mult(R, I);

            I.addUnitMatrix();

        }

        else

        {

            I.mult(R, Tmp);

            Tmp.addUnitMatrix();

        }

        flag = !flag;

    }

    if (flag)

        Tmp.mult(B, res);

    else

        I.mult(B, res);

}

# Приложение 8 (CMakeLists.txt)

cmake\_minimum\_required(VERSION 3.10)

project(matrix)

set(SOURCES src/main.cpp src/usual\_matrix.cpp src/blas\_matrix.cpp src/intrinsic\_matrix.cpp)

set(HEADERS src/usual\_matrix.h src/blas\_matrix.h src/intrinsic\_matrix.h)

add\_executable(${PROJECT\_NAME} ${SOURCES})