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, 24203 группы

Анисимова Льва Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент, кандидат технических наук
А. Ю. Власенко

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ.....3
ЗАДАНИЕ 3
ОПИСАНИЕ РАБОТЫ.....4
ЗАКЛЮЧЕНИЕ.....6
Приложение 1. *Код программы «1.cpp» на C++*.....7
Приложение 2. *Код программы «2.cpp» на C++*.....10
Приложение 3. *Код программы «3.cpp» на C++*.....13
Приложение 4. *RESULT.TXT*.....15

ЦЕЛЬ

Изучение методов оптимизации численных вычислений в линейной алгебре средствами векторизации на архитектуре x86, а также с помощью специализированных библиотек. Получение навыков анализа эффективности различных подходов к вычислению обратных матриц.

ЗАДАНИЕ

Реализовать три варианта программ для вычисления обратной матрицы по заданному методу: без векторизации, с ручной векторизацией (SSE/AVX), с использованием библиотеки BLAS.

Векторизовать операции умножения матриц, их сложения и умножения на скаляр в ручном варианте.

Провести экспериментальное сравнение эффективности (по времени работы) трёх подходов на одинаковом наборе данных.

Проверить корректность результатов на тестовых наборах.

Подготовить и оформить отчёт по результатам лабораторной работы.

ОПИСАНИЕ РАБОТЫ

В рамках работы были реализованы и протестированы три программных метода вычисления обратной матрицы на основе разложения $A^{-1} = (I + R + R^2 + \cdots)B$, где вычисляются параметры R и B согласно алгоритму из задания.

1. **Вариант без векторизации:** все основные операции над матрицами реализованы с учётом оптимального порядка прохода по памяти, что обеспечивает более эффективное использование кеша процессора. Для компиляции использовалась следующая инструкция: `g++ -O0 1.cpp -o 1`

2. **Вариант с ручной векторизацией (SSE)**: для ускорения операций над матрицами применялись SIMD-инструкции SSE позволяющие выполнять вычисления над 4 элементами одновременно. Это особенно ускоряет умножение матриц и работу с большими данными.
   В реализации использовались следующие ключевые инструкции:
   - **`_mm_set1_ps`** — загрузка одного значения и его размножение по всем элементам 128-битного регистра (broadcast). Применялась для умножения строки матрицы на один элемент.
   - **`_mm_loadu_ps`** — загрузка невыровненных данных из памяти в SIMD-регистр (4 float). Использовалась для чтения строк матриц.
   - **`_mm_mul_ps`** — векторное умножение: каждый элемент одного регистра умножается на соответствующий элемент другого регистра параллельно.
   - **`_mm_add_ps`** — векторное сложение: параллельное суммирование элементов двух регистров.
   - **`_mm_sub_ps`** — векторное вычитание для операции вычисления матрицы $R = I - BA$.
   - **`_mm_storeu_ps`** — сохранение результата из SIMD-регистра обратно в память.

   Эти инструкции позволяют за один такт процессора обработать сразу 4 (SSE) значений типа float, что кратно увеличивает скорость вычислений по сравнению со скалярным подходом.
   Для компиляции использовалась следующая инструкция: `g++ -O0 -msse -msse2 2.cpp -o 2`

3. **Вариант с использованием BLAS**: для линейной алгебры применялась библиотека OpenBLAS, реализующая оптимизированные матричные операции. Были использованы функции `cblas_sgemm` (умножение матриц) и `cblas_saxpy` (сложение матриц).
   Для компиляции использовалась следующая инструкция: `g++ -O0 3.cpp -lopenblas -o 3`

Эксперименты проводились при размере матриц N = 1024 и количестве членов ряда M = 10. Для корректного тестирования все варианты запускались на одинаковой машине, с равными ключами компиляции. Корректность подтверждалась совпадением элементов обратных матриц на тестовых наборах.

Результаты измерения времени (*См. Приложение 4*) показывают стабильный выигрыш по скорости для вариантов с векторизацией и использованием BLAS, по сравнению с базовым вариантом без оптимизации.

ЗАКЛЮЧЕНИЕ

В ходе работы была продемонстрирована значительная роль векторных расширений и оптимизированных библиотек для ускорения матричных вычислений.

Ручная векторизация SSE и применение BLAS значительно улучшили производительность без потери точности.

Проведённое сравнение вариантов подтвердило актуальность современных технологий оптимизации для решения вычислительных задач линейной алгебры.

Полученные практические знания будут полезны при разработке высокопроизводительных программ для инженерных и научных расчетов.

Приложение 1. *Код программы «1.cpp» на C++*

```
#include <iostream>
#include <vector>
#include <cmath>
#include <chrono>
using namespace std;

// Транспонирование
void transpose(const vector<vector<float>>& A, vector<vector<float>>&
AT, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            AT[j][i] = A[i][j];
}

// Норма ||A||_1
float norm1(const vector<vector<float>>& A, int N) {
    float maxSum = 0;
    for (int j = 0; j < N; j++) {
        float sum = 0;
        for (int i = 0; i < N; i++)
            sum += fabs(A[i][j]);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

// Норма ||A||_inf
float normInf(const vector<vector<float>>& A, int N) {
    float maxSum = 0;
    for (int i = 0; i < N; i++) {
        float sum = 0;
        for (int j = 0; j < N; j++)
            sum += fabs(A[i][j]);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

// Умножение матриц C = A * B (оптимизированный порядок i-k-j)
void matmul(const vector<vector<float>>& A, const
vector<vector<float>>& B,
            vector<vector<float>>& C, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            C[i][j] = 0;

    for (int i = 0; i < N; i++)
        for (int k = 0; k < N; k++)
            for (int j = 0; j < N; j++)
                C[i][j] += A[i][k] * B[k][j];
}

// Умножение матрицы на скаляр
void matscal(const vector<vector<float>>& A, float s,
             vector<vector<float>>& C, int N) {
    for (int i = 0; i < N; i++)
```

```
        for (int j = 0; j < N; j++)
            C[i][j] = A[i][j] * s;
}

// Сложение матриц C = A + B
void matadd(const vector<vector<float>>& A, const
vector<vector<float>>& B,
            vector<vector<float>>& C, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            C[i][j] = A[i][j] + B[i][j];
}

// Единичная матрица
void Identity(vector<vector<float>>& I, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            I[i][j] = (i == j) ? 1.0f : 0.0f;
}

// Вычитание матриц C = A - B
void matsub(const vector<vector<float>>& A, const
vector<vector<float>>& B,
            vector<vector<float>>& C, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            C[i][j] = A[i][j] - B[i][j];
}

int main() {
    int N = 1024;
    int M = 10;

    // Инициализация матрицы A
    vector<vector<float>> A(N, vector<float>(N));
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i][j] = (i == j) ? 2.0f : 0.1f;

    auto start = chrono::high_resolution_clock::now();

    // Вычисление B = A^T / (||A||_1 * ||A||_inf)
    vector<vector<float>> AT(N, vector<float>(N));
    transpose(A, AT, N);
    float n1 = norm1(A, N);
    float ninf = normInf(A, N);
    float scalar = 1.0f / (n1 * ninf);

    vector<vector<float>> B(N, vector<float>(N));
    matscal(AT, scalar, B, N);

    // R = I - BA
    vector<vector<float>> I(N, vector<float>(N));
    vector<vector<float>> BA(N, vector<float>(N));
    vector<vector<float>> R(N, vector<float>(N));
    Identity(I, N);
    matmul(B, A, BA, N);
    matsub(I, BA, R, N);
```

```
    // Вычисление суммы ряда: Sum = I + R + R^2 + ...
    vector<vector<float>> Sum(N, vector<float>(N));
    vector<vector<float>> Rn(N, vector<float>(N));
    Identity(Sum, N);  // Sum = I
    Identity(Rn, N);   // R^0 = I

    for (int m = 1; m <= M; m++) {
        vector<vector<float>> temp(N, vector<float>(N));
        matmul(Rn, R, temp, N);  // R^n = R^(n-1) * R
        Rn = temp;
        matadd(Sum, Rn, Sum, N);  // Sum += R^n
    }

    // A^(-1) = Sum * B
    vector<vector<float>> Ainv(N, vector<float>(N));
    matmul(Sum, B, Ainv, N);

    auto end = chrono::high_resolution_clock::now();

    // Вывод результата
    cout << "Inverse elements:\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            cout << Ainv[i][j] << " ";
        cout << "\n";
    }

    cout << "time: "
         << chrono::duration_cast<chrono::milliseconds>(end -
start).count()
         << " ms\n";

    return 0;
}
```

Приложение 2. *Код программы «2.cpp» на C++*

```
#include <xmmintrin.h>  // SSE
#include <emmintrin.h>  // SSE2
#include <iostream>
#include <vector>
#include <cmath>
#include <chrono>
using namespace std;

void transpose(const float* A, float* AT, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            AT[j*N + i] = A[i*N + j];
}

float norm1(const float* A, int N) {
    float maxSum = 0;
    for (int j = 0; j < N; j++) {
        float sum = 0;
        for (int i = 0; i < N; i++)
            sum += fabs(A[i*N + j]);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

float normInf(const float* A, int N) {
    float maxSum = 0;
    for (int i = 0; i < N; i++) {
        float sum = 0;
        for (int j = 0; j < N; j++)
            sum += fabs(A[i*N + j]);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

// Векторизованное умножение матриц
void matmul_sse(const float* A, const float* B, float* C, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            C[i*N + j] = 0;

    for (int i = 0; i < N; i++) {
        for (int k = 0; k < N; k++) {
            __m128 av = _mm_set1_ps(A[i*N + k]);
            int j = 0;
            for (; j <= N - 4; j += 4) {
                __m128 bv = _mm_loadu_ps(&B[k*N + j]);
                __m128 cv = _mm_loadu_ps(&C[i*N + j]);
                cv = _mm_add_ps(cv, _mm_mul_ps(av, bv));
                _mm_storeu_ps(&C[i*N + j], cv);
            }
            for (; j < N; j++)
                C[i*N + j] += A[i*N + k] * B[k*N + j];
        }
```

```
    }
}

// Векторизованное умножение на скаляр
void matscal_sse(const float* A, float s, float* C, int N) {
    __m128 sv = _mm_set1_ps(s);
    int i = 0;
    int total = N * N;
    for (; i <= total - 4; i += 4) {
        __m128 av = _mm_loadu_ps(&A[i]);
        __m128 cv = _mm_mul_ps(av, sv);
        _mm_storeu_ps(&C[i], cv);
    }
    for (; i < total; i++)
        C[i] = A[i] * s;
}

// Векторизованное сложение
void matadd_sse(const float* A, const float* B, float* C, int N) {
    int total = N * N;
    int i = 0;
    for (; i <= total - 4; i += 4) {
        __m128 av = _mm_loadu_ps(&A[i]);
        __m128 bv = _mm_loadu_ps(&B[i]);
        __m128 cv = _mm_add_ps(av, bv);
        _mm_storeu_ps(&C[i], cv);
    }
    for (; i < total; i++)
        C[i] = A[i] + B[i];
}

void matsub_sse(const float* A, const float* B, float* C, int N) {
    int total = N * N;
    int i = 0;
    for (; i <= total - 4; i += 4) {
        __m128 av = _mm_loadu_ps(&A[i]);
        __m128 bv = _mm_loadu_ps(&B[i]);
        __m128 cv = _mm_sub_ps(av, bv);
        _mm_storeu_ps(&C[i], cv);
    }
    for (; i < total; i++)
        C[i] = A[i] - B[i];
}

void Identity(float* I, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            I[i*N + j] = (i == j) ? 1.0f : 0.0f;
}

int main() {
    int N = 1024;
    int M = 10;

    vector<float> A(N*N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i*N + j] = (i == j) ? 2.0f : 0.1f;
```

```
    auto start = chrono::high_resolution_clock::now();

    vector<float> AT(N*N), B(N*N), BA(N*N), I(N*N), R(N*N);
    transpose(A.data(), AT.data(), N);
    float scalar = 1.0f / (norm1(A.data(), N) * normInf(A.data(), N));
    matscal_sse(AT.data(), scalar, B.data(), N);

    Identity(I.data(), N);
    matmul_sse(B.data(), A.data(), BA.data(), N);
    matsub_sse(I.data(), BA.data(), R.data(), N);

    vector<float> Sum(N*N), Rn(N*N), temp(N*N);
    Identity(Sum.data(), N);
    Identity(Rn.data(), N);

    for (int m = 1; m <= M; m++) {
        matmul_sse(Rn.data(), R.data(), temp.data(), N);
        Rn = temp;
        matadd_sse(Sum.data(), Rn.data(), Sum.data(), N);
    }

    vector<float> Ainv(N*N);
    matmul_sse(Sum.data(), B.data(), Ainv.data(), N);

    auto end = chrono::high_resolution_clock::now();

    cout << "Элементы обратной матрицы:\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            cout << Ainv[i*N + j] << " ";
        cout << "\n";
    }

    cout << "Время: " <<
chrono::duration_cast<chrono::milliseconds>(end - start).count() << "
мс\n";
    return 0;
}
```

Приложение 3. *Код программы «3.cpp» на C++*

```
#include <cblas.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <chrono>
using namespace std;

float norm1(const float* A, int N) {
    float maxSum = 0;
    for (int j = 0; j < N; j++) {
        float sum = cblas_sasum(N, &A[j], N);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

float normInf(const float* A, int N) {
    float maxSum = 0;
    for (int i = 0; i < N; i++) {
        float sum = cblas_sasum(N, &A[i*N], 1);
        if (sum > maxSum) maxSum = sum;
    }
    return maxSum;
}

void identity(float* I, int N) {
    for (int i = 0; i < N*N; i++) I[i] = 0;
    for (int i = 0; i < N; i++) I[i*N + i] = 1.0f;
}

int main() {
    int N = 1024;
    int M = 10;

    vector<float> A(N*N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i*N + j] = (i == j) ? 2.0f : 0.1f;

    auto start = chrono::high_resolution_clock::now();

    // B = A^T / (||A||_1 * ||A||_inf)
    vector<float> B(N*N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            B[j*N + i] = A[i*N + j];

    float scalar = 1.0f / (norm1(A.data(), N) * normInf(A.data(), N));
    cblas_sscal(N*N, scalar, B.data(), 1);

    // R = I - BA
    vector<float> I(N*N), BA(N*N), R(N*N);
    identity(I.data(), N);
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                N, N, N, 1.0f, B.data(), N, A.data(), N, 0.0f,
```

```
BA.data(), N);

    for (int i = 0; i < N*N; i++)
        R[i] = I[i] - BA[i];

    // Sum = I + R + R^2 + ...
    vector<float> Sum(N*N), Rn(N*N), temp(N*N);
    identity(Sum.data(), N);
    identity(Rn.data(), N);

    for (int m = 1; m <= M; m++) {
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                    N, N, N, 1.0f, Rn.data(), N, R.data(), N, 0.0f,
temp.data(), N);
        Rn = temp;
        cblas_saxpy(N*N, 1.0f, Rn.data(), 1, Sum.data(), 1);
    }

    // A^(-1) = Sum * B
    vector<float> Ainv(N*N);
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                N, N, N, 1.0f, Sum.data(), N, B.data(), N, 0.0f,
Ainv.data(), N);

    auto end = chrono::high_resolution_clock::now();

    cout << "Элементы обратной матрицы:\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            cout << Ainv[i*N + j] << " ";
        cout << "\n";
    }

    cout << "Время: " <<
chrono::duration_cast<chrono::milliseconds>(end - start).count() << "
мс\n";
    return 0;
}
```

Приложение 4. *RESULT.TXT*

```
./1
Элементы обратной матрицы:
0.00192554 7.48967e-06 7.48967e-06
7.48967e-06 0.00192554 7.48967e-06
7.48967e-06 7.48967e-06 0.00192554
Время: 266123 мс

./2
Элементы обратной матрицы:
0.00192554 7.48967e-06 7.48967e-06
7.48967e-06 0.00192554 7.48967e-06
7.48967e-06 7.48967e-06 0.00192554
Время: 43512 мс

./3
Элементы обратной матрицы:
0.00192551 7.49092e-06 7.49092e-06
7.49092e-06 0.00192551 7.49092e-06
7.49092e-06 7.49092e-06 0.00192551
Время: 283 мс
```