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, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2

ЦЕЛЬ 3

ЗАДАНИЕ 3

ОПИСАНИЕ РАБОТЫ 4

Пошаговое описание выполненной работы 4

Команды для компиляции 6

Результаты измерения времени 6

ЗАКЛЮЧЕНИЕ 7

ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ) 8

src/default.cpp 8

src/manual.cpp 11

src/BLAS.cpp 14

CMakeLists.txt 17

ЦЕЛЬ

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

ЗАДАНИЕ

Алгоритм обращения матрицы A размером N×N с помощью разложения в ряд: $A^{-1} = (I + R + R^2 + \ldots)B$, *где* $R = I - BA$,

$B = \frac{A^T}{||A||_1 \cdot ||A||_\infty}$, $||A||_1 = max_j \sum_i \left|A_{ij}\right|$, $||A||_\infty = max_i \sum_j \left|A_{ij}\right|$,

I – единичная матрица. Параметры алгоритма: N – размер матрицы, M – число членов ряда.

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;
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;*
5. *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
   - *Титульный лист;*
   - *Цель лабораторной работы;*
   - *Результаты измерения времени работы трех программ;*
   - *Полный компилируемый листинг реализованных программ и команды для их компиляции;*
   - *Вывод по результатам лабораторной работы.*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Написать 3 варианта программы, реализующей алгоритм из задания:*
    a. *вариант без ручной векторизации;*
    b. *вариант с использованием встроенных функций компилятора;*
    c. *вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS;*
2. *Проверил правильность работы программ, изменив параметры (N = 4, M = 100000);*

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
A_1 check: 1.00001
A_infinity check: 1.00001
1 -3.57628e-07 2.74181e-06 2.05636e-06
-4.19095e-07 0.999998 -8.34465e-07 -7.59959e-07
2.30968e-06 -2.38419e-07 1 3.01003e-06
2.10013e-06 -9.53674e-07 3.15905e-06 1
Time without vectorization: 0.0559754 sec.
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/manual
A_1 check: 1
A_infinity check: 1
0.999996 6.25849e-07 -5.36442e-07 -8.64267e-07
8.34465e-07 0.99999 -3.21865e-06 2.20537e-06
-8.9407e-08 -3.57628e-06 0.999997 7.15256e-07
-1.12504e-06 2.01166e-06 5.06639e-07 0.999996
Time with vectorization: 0.0150537 sec.
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/BLAS
A_1 check: 1.00114
A_infinity check: 1.00115
1.00013 0.000132501 0.000151202 -0.000539578
0.000126794 1.00001 -5.67734e-06 -0.000136755
0.000147909 -8.58307e-06 0.999888 8.73804e-05
-0.000535585 -0.000136554 8.37967e-05 1.00039
Time with BLAS: 0.0315955 sec.
```

3. *Провёл несколько оптимизаций кода:*
    a. *Скомпилировал программу с уровнем оптимизации -O3;*
    b. *Изменил индексацию в цикле функции Multplication, чтобы получить последовательный обход памяти;*

Не последовательный обход памяти

```
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result)
{
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            for (int k = 0; k < N; ++k)
            {
                if (k == 0) result[N * i + j] = 0;
                result[N * i + j] += multiplier1[N * i + k] *
                                     multiplier2[N * k + j];
            }
}
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
Time without vectorization: 651.89 sec.
```

Последовательный обход памяти

```
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result)
{
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            for (int k = 0; k < N; ++k)
            {
                if (j == 0) result[N * i + k] = 0;
                result[N * i + k] += multiplier1[N * i + j] *
                                     multiplier2[N * j + k];
            }
}
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
Time without vectorization: 31.3678 sec.
```

c. *Избавился от условного оператора в цикле функции Multiplication;*

С условным оператором в цикле

```
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result)
{
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            for (int k = 0; k < N; ++k)
            {
```

```
                if (j == 0) result[N * i + k] = 0;
                result[N * i + k] += multiplier1[N * i + j] *
                                     multiplier2[N * j + k];
            }
    }
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
Time without vectorization: 31.2445 sec.
```

Без условного оператора в цикле

```
    void Multiplication(const float * multiplier1,
                        const float * multiplier2, float * result)
    {
        for (int i = 0; i < N * N; ++i)
            result[i] = 0;

        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                for (int k = 0; k < N; ++k)
                    result[N * i + k] += multiplier1[N * i + j] *
                                         multiplier2[N * j + k];
    }
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
Time without vectorization: 29.7316 sec.
```

Команды для компиляции

```
evmpu@comrade:~$ /usr/bin/cmake
-DCMAKE_BUILD_TYPE=Release
-DCMAKE_C_COMPILER=/usr/bin/gcc
-DCMAKE_CXX_COMPILER=/usr/bin/g++
-G "CodeBlocks - Unix Makefiles"
-S /home/evmpu/21211/s.petrov1/lab7
-B /home/evmpu/21211/s.petrov1/lab7/cmake-build-release

evmpu@comrade:~$ /usr/bin/cmake
--build /home/evmpu/21211/s.petrov1/lab7/cmake-build-release
--target default

evmpu@comrade:~$ /usr/bin/cmake
--build /home/evmpu/21211/s.petrov1/lab7/cmake-build-release
--target manual

evmpu@comrade:~$ /usr/bin/cmake
--build /home/evmpu/21211/s.petrov1/lab7/cmake-build-release
--target BLAS
```

Результаты измерения времени

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/default
A_1 check: 0.235952
A_infinity check: 0.235952
Time without vectorization: 40.394 sec.
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/manual
A_1 check: 0.233539
A_infinity check: 0.233539
Time with vectorization: 40.136 sec.
```

```
/home/evmpu/21211/s.petrov1/lab7/cmake-build-release-remote-host-evmpu/BLAS
A_1 check: 0.235847
A_infinity check: 0.235847
Time with BLAS: 1.76284 sec.
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

- *Изучил SIMD-расширения архитектуры x86/x86-64;*
- *Изучил способы использования SIMD-расширений в программах на языке Си;*
- *Получил навыки использования SIMD-расширений;*
- *Изучил работу оптимизированной библиотеки линейной алгебры BLAS;*

По результатам проведённых исследований можно сделать следующие выводы:

- *Прирост производительности с SIMD расширениями был незначительным, т.к. компилятор в варианте без векторизации использовал векторизацию простых циклов;*
- *BLAS лучше всего использовать в вычислениях с матрицами;*

ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)

src/default.cpp

```
#include <cfloat>       // FLT_MIN
#include <cmath>        // fabs()
#include <ctime>
#include <iostream>
#define N 2048
#define M 10

void Inverse(const float * matrix, float * result);
void GetNorms(float & A_1, float & A_infinity, const float * matrix);
void FillB(const float * matrix, float * B);
void FillI(float * I);
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result);
void Addition(const float * addend1, const float * addend2,
              float * result);
void Subtraction(const float * minuend, const float * subtrahend,
                 float * result);
void Copy(float * dest, const float * src);
void Print(const float * matrix);

int main()
{
    srandom(time(nullptr));
    auto * matrix = new float [N * N];
    auto * result = new float [N * N];
    auto * check = new float [N * N];
    timespec start = { 0, 0 };
    timespec end = { 0, 0 };

    for (int i = 0; i < N * N; ++i)
    {
        matrix[i] = float(random());
        matrix[i] *= (random() % 2) ? 1 : -1;
        result[i] = 0;
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    Inverse(matrix, result);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);

    Multiplication(matrix, result, check);

    float A_1, A_infinity;
    GetNorms(A_1, A_infinity, check);

    std::cout << "A_1 check: " << A_1 << std::endl;
    std::cout << "A_infinity check: " << A_infinity << std::endl;
//  Print(check);

    std::cout << "Time without vectorization: "
              << (double)end.tv_sec - (double)start.tv_sec + 1e-9 *
                 ((double)end.tv_nsec - (double)start.tv_nsec)
```

```
                << " sec." << std::endl;

    delete []matrix;
    delete []result;
    delete []check;

    return EXIT_SUCCESS;
}

void Inverse(const float * matrix, float * result)
{
    auto * B = new float[N * N];
    auto * I = new float[N * N];
    auto * tmp = new float[N * N];
    auto * R = new float[N * N];
    bool flag = true;

    FillB(matrix, B);
    FillI(I);
    Multiplication(B, matrix, tmp);
    Subtraction(I, tmp, R);
    Addition(I, R, tmp);
    Copy(result, R);

    for (int i = 2; i < M; ++i)
    {
        Multiplication(flag ? result : I, R, flag ? I : result);
        Addition(tmp, flag ? I : result, tmp);
        flag = !flag;
    }

    Multiplication(tmp, B, result);

    delete[] I;
    delete[] B;
    delete[] tmp;
    delete[] R;
}

void GetNorms(float & A_1, float & A_infinity, const float * matrix)
{
    A_1 = FLT_MIN;
    A_infinity = FLT_MIN;
    float sum_row = 0;
    float sum_column = 0;

    for (int i = 0; i < N; i++) // rows
    {
        sum_row = 0;
        sum_column = 0;

        for (int j = 0; j < N; j++) // columns
        {
            sum_row += std::fabs(matrix[N * i + j]);
            sum_column += std::fabs(matrix[j * N + i]);
        }
```

```
            if (sum_row > A_1)
                A_1 = sum_row;
            if (sum_column > A_infinity)
                A_infinity = sum_column;
        }
    }

    void FillB(const float * matrix, float * B)
    {
        float A_1, A_infinity;
        GetNorms(A_1, A_infinity, matrix);

        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                B[N * i + j] = matrix[j * N + i] / (A_1 * A_infinity);
    }

    void FillI(float * I)
    {
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                I[N * i + j] = (float)(i == j);
    }

    void Multiplication(const float * multiplier1,
                        const float * multiplier2, float * result)
    {
        for (int i = 0; i < N * N; ++i)
            result[i] = 0;

        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                for (int k = 0; k < N; ++k)
                    result[N * i + k] += multiplier1[N * i + j] *
                                         multiplier2[N * j + k];
    }

    void Addition(const float * addend1, const float * addend2,
                  float * result)
    {
        for (int i = 0; i < N * N; ++i)
            result[i] = addend1[i] + addend2[i];
    }

    void Subtraction(const float * minuend, const float * subtrahend,
                     float * result)
    {
        for (int i = 0; i < N * N; i++)
            result[i] = minuend[i] - subtrahend[i];
    }

    void Copy(float * dest, const float * src)
    {
        for (int i = 0; i < N * N; i++)
            dest[i] = src[i];
    }
```

```
void Print(const float * matrix)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            std::cout << matrix[N * i + j] << " ";

        std::cout << std::endl;
    }
}
```

src/manual.cpp

```
#include <cfloat>       // FLT_MIN
#include <cmath>        // fabs()
#include <ctime>
#include <iostream>
#include <xmmintrin.h>
#define N 2048
#define M 10

void Inverse(const float * matrix, float * result);
void GetNorms(float & A_1, float & A_infinity, const float * matrix);
void FillB(const float * matrix, float * B);
void FillI(float * I);
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result);
void Addition(const float * addend1, const float * addend2,
              float * result);
void Subtraction(const float * minuend, const float * subtrahend,
                 float * result);
void Copy(float * dest, const float * src);
void Print(const float * matrix);

int main()
{
    srandom(time(nullptr));
    auto * matrix = new float [N * N];
    auto * result = new float [N * N];
    auto * check = new float [N * N];
    timespec start = { 0, 0 };
    timespec end = { 0, 0 };

    for (int i = 0; i < N * N; ++i)
    {
        matrix[i] = float(random());
        matrix[i] *= (random() % 2) ? 1 : -1;
        result[i] = 0;
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    Inverse(matrix, result);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);

    Multiplication(matrix, result, check);
```

```
    float A_1, A_infinity;
    GetNorms(A_1, A_infinity, check);

    std::cout << "A_1 check: " << A_1 << std::endl;
    std::cout << "A_infinity check: " << A_infinity << std::endl;
//  Print(check);

    std::cout << "Time with vectorization: "
              << (double)end.tv_sec - (double)start.tv_sec + 1e-9 *
                 ((double)end.tv_nsec - (double)start.tv_nsec)
              << " sec." << std::endl;

    delete []matrix;
    delete []result;
    delete []check;

    return EXIT_SUCCESS;
}

void Inverse(const float * matrix, float * result)
{
    auto * B = new float[N * N];
    auto * I = new float[N * N];
    auto * tmp = new float[N * N];
    auto * R = new float[N * N];
    bool flag = true;

    FillB(matrix, B);
    FillI(I);
    Multiplication(B, matrix, tmp);
    Subtraction(I, tmp, R);
    Addition(I, R, tmp);
    Copy(result, R);

    for (int i = 2; i < M; ++i)
    {
        Multiplication(flag ? result : I, R, flag ? I : result);
        Addition(tmp, flag ? I : result, tmp);
        flag = !flag;
    }

    Multiplication(tmp, B, result);

    delete[] I;
    delete[] B;
    delete[] tmp;
    delete[] R;
}

void GetNorms(float & A_1,  & A_infinity, const float * matrix)
{
    A_1 = FLT_MIN;
    A_infinity = FLT_MIN;
    float sum_row = 0;
    float sum_column = 0;
```

```
    for (int i = 0; i < N; i++) // rows
    {
        sum_row = 0;
        sum_column = 0;

        for (int j = 0; j < N; j++) // columns
        {
            sum_row += std::fabs(matrix[N * i + j]);
            sum_column += std::fabs(matrix[j * N + i]);
        }

        if (sum_row > A_1)
            A_1 = sum_row;
        if (sum_column > A_infinity)
            A_infinity = sum_column;
    }
}

void FillB(const float * matrix, float * B)
{
    float A_1, A_infinity;
    GetNorms(A_1, A_infinity, matrix);

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            B[N * i + j] = matrix[j * N + i] / (A_1 * A_infinity);
}

void FillI(float * I)
{
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            I[N * i + j] = (float)(i == j);
}

void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result)
{
    __m128 * m128_result = (__m128 *)result;
    const __m128 * m128_multiplier2 = (const __m128 *)multiplier2;
    __m128 m128_multiplier1;
    __m128 tmp;

    for (int i = 0; i < N * N / 4; ++i)
        m128_result[i] = _mm_setzero_ps();

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
        {
            m128_multiplier1 = _mm_set1_ps(multiplier1[N * i + j]);
            for (int k = 0; k < N / 4; ++k)
            {
                tmp = _mm_mul_ps(m128_multiplier1,
                                 m128_multiplier2[N * j / 4 + k]);
                m128_result[N * i / 4 + k] =
                _mm_add_ps(m128_result[N * i / 4 + k], tmp);
```

```
            }
        }
}

void Addition(const float * addend1, const float * addend2,
              float * result)
{
    for (int i = 0; i < N * N; ++i)
        result[i] = addend1[i] + addend2[i];
}

void Subtraction(const float * minuend, const float * subtrahend,
                 float * result)
{
    for (int i = 0; i < N * N; i++)
        result[i] = minuend[i] - subtrahend[i];
}

void Copy(float * dest, const float * src)
{
    for (int i = 0; i < N * N; i++)
        dest[i] = src[i];
}

void Print(const float * matrix)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            std::cout << matrix[N * i + j] << " ";

        std::cout << std::endl;
    }
}
```

src/BLAS.cpp

```
#include <cfloat>       // FLT_MIN
#include <cmath>        // fabs()
#include <ctime>
#include <iostream>
#include <mkl_cblas.h>  // cblas_sgemm
#define N 2048
#define M 10

void Inverse(const float * matrix, float * result);
void GetNorms(float & A_1, float & A_infinity, const float * matrix);
void FillB(const float * matrix, float * B);
void FillI(float * I);
void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result);
void Addition(const float * addend1, const float * addend2,
              float * result);
void Subtraction(const float * minuend, const float * subtrahend,
                 float * result);
```

```
void Copy(float * dest, const float * src);
void Print(const float * matrix);

int main()
{
    srandom(time(nullptr));
    auto * matrix = new float [N * N];
    auto * result = new float [N * N];
    auto * check = new float [N * N];
    timespec start = { 0, 0 };
    timespec end = { 0, 0 };

    for (int i = 0; i < N * N; ++i)
    {
        matrix[i] = float(random());
        matrix[i] *= (random() % 2) ? 1 : -1;
        result[i] = 0;
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    Inverse(matrix, result);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);

    Multiplication(matrix, result, check);

    float A_1, A_infinity;
    GetNorms(A_1, A_infinity, check);

    std::cout << "A_1 check: " << A_1 << std::endl;
    std::cout << "A_infinity check: " << A_infinity << std::endl;
//  Print(check);

    std::cout << "Time with BLAS: "
              << (double)end.tv_sec - (double)start.tv_sec + 1e-9 *
                 ((double)end.tv_nsec - (double)start.tv_nsec)
              << " sec." << std::endl;

    delete []matrix;
    delete []result;
    delete []check;

    return EXIT_SUCCESS;
}

void Inverse(const float * matrix, float * result)
{
    auto * B = new float[N * N];
    auto * I = new float[N * N];
    auto * tmp = new float[N * N];
    auto * R = new float[N * N];
    bool flag = true;

    FillB(matrix, B);
    FillI(I);
    Multiplication(B, matrix, tmp);
    Subtraction(I, tmp, R);
```

```
    Addition(I, R, tmp);
    Copy(result, R);

    for (int i = 2; i < M; ++i)
    {
        Multiplication(flag ? result : I, R, flag ? I : result);
        Addition(tmp, flag ? I : result, tmp);
        flag = !flag;
    }

    Multiplication(tmp, B, result);

    delete[] I;
    delete[] B;
    delete[] tmp;
    delete[] R;
}

void GetNorms(float & A_1, float & A_infinity, const float * matrix)
{
    A_1 = FLT_MIN;
    A_infinity = FLT_MIN;
    float sum_row = 0;
    float sum_column = 0;

    for (int i = 0; i < N; i++) // rows
    {
        sum_row = 0;
        sum_column = 0;

        for (int j = 0; j < N; j++) // columns
        {
            sum_row += std::fabs(matrix[N * i + j]);
            sum_column += std::fabs(matrix[j * N + i]);
        }

        if (sum_row > A_1)
            A_1 = sum_row;
        if (sum_column > A_infinity)
            A_infinity = sum_column;
    }
}

void FillB(const float * matrix, float * B)
{
    float A_1, A_infinity;
    GetNorms(A_1, A_infinity, matrix);

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            B[N * i + j] = matrix[j * N + i] / (A_1 * A_infinity);
}

void FillI(float * I)
{
    for (int i = 0; i < N; ++i)
```

```
        for (int j = 0; j < N; ++j)
            I[N * i + j] = (float)(i == j);
}

void Multiplication(const float * multiplier1,
                    const float * multiplier2, float * result)
{
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N,
                1.0, multiplier1, N, multiplier2, N, 0.0, result, N);
}

void Addition(const float * addend1, const float * addend2,
              float * result)
{
    for (int i = 0; i < N * N; ++i)
        result[i] = addend1[i] + addend2[i];
}

void Subtraction(const float * minuend, const float * subtrahend,
                 float * result)
{
    for (int i = 0; i < N * N; i++)
        result[i] = minuend[i] - subtrahend[i];
}

void Copy(float * dest, const float * src)
{
    for (int i = 0; i < N * N; i++)
        dest[i] = src[i];
}

void Print(const float * matrix)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            std::cout << matrix[N * i + j] << " ";

        std::cout << std::endl;
    }
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.3)

project(lab7)

add_executable(default
        src/default.cpp)

add_executable(manual
        src/manual.cpp)
target_compile_options(manual PUBLIC
        -msse -msse2 -msse3 -msse4)
```

```
add_executable(BLAS
        src/BLAS.cpp)
find_package(MKL CONFIG REQUIRED)
target_compile_options(BLAS PUBLIC
        $<TARGET_PROPERTY:MKL::MKL,INTERFACE_COMPILE_OPTIONS>)
target_include_directories(BLAS PUBLIC
        $<TARGET_PROPERTY:MKL::MKL,INTERFACE_INCLUDE_DIRECTORIES>)
target_link_libraries(BLAS PUBLIC
        $<LINK_ONLY:MKL::MKL>)
```