###### 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 вычислений»

студента 2 курса, группы 20204

**Дронова Дениса Юрьевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2021

**СОДЕРЖАНИЕ**

[ЦЕЛИ 3](#_Toc88423664)

[ЗАДАНИЕ 3](#_Toc88423665)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc88423666)

[Приложение 1. *Без векторизации* 6](#_Toc88423667)

[Приложение 2. *AVX256* 11](#_Toc88423668)

[Приложение 3. *BLAS* 17](#_Toc88423669)

# ЦЕЛИ

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, реализующей̆ алгоритм из задания:

• вариант без ручной̆ 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 оптимизированной̆ библиотеки BLAS. Для элементов матриц использовать тип данных float.

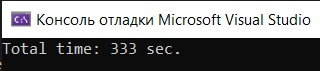
1. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.
2. Каждый̆ вариант программы оптимизировать по скорости, насколько это возможно.
3. Сравнить время работы трех вариантов программы для N=2048, M=10.

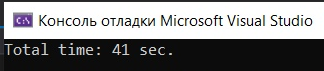
# ОПИСАНИЕ РАБОТЫ

В ходе выполнения практической работы было написано три варианта обращения матрицы: без векторизации, с применением ручной векторизации, с помощью библиотеки BLAS (Basic Linear Algebra Subprograms). Все приведенные варианты программ были протестированы на корректность обращения матрицы на примитивных тестах (задание матриц размеров 4\*4, 8\*8, 16\*16). Затем каждая программа была запущена на параметрах N = 2048 (размер матрицы), M = 10 (параметр, задающий точность). Для выполнения ручной векторизации был выбран вариант использования встроенных функций (intrinsics) библиотеки immintrin.h, дающей доступ к работе с векторами расширения AVX256. Были произведены замеры времени исполнения трёх программ при идентичных параметрах N и M. По результатам замеров были получены следующие значения времени:

1. Без векторизации: 333 секунды
2. AVX256: 41 секунда
3. BLAS: 10 секунд

Строка компиляции: g++ -I $HOME/Linux\_P4SSE2/include -L $HOME/Linux\_P4SSE2/lib -Ofast BLAS.cpp -lcblas -latlas







**ЗАКЛЮЧЕНИЕ**

В процессе выполнения практической работы были изучены принципы оптимизации программы с SIMD-расширений, а также получены навыки их использования при решении задач. Они необходимы на практике для ускорения выполнения многочисленных однотипных операций. По результатам работы нетрудно заметить, что вариант с использованием библиотеки BLAS является самым быстрым по времени работы, а использование встроенных функций компилятора даёт больший контроль над выполнением операций пользователю.

# Приложение 1. *Без векторизации*

#include <iostream>

#include <ctime>

#include <cstdlib>

const int M = 10;

const int N = 2048;

using namespace std;

void showMatrix(float \*matrix, size\_t size)

{

    size\_t i, j;

    for (i = 0; i < size; ++i)

    {

        for (j = 0; j < size; ++j)

        {

            cout << matrix[i \* size + j] << " ";

        }

        cout << endl;

    }

    cout << "\n"

         << "\n"

         << "\n";

}

void SubtractMatrix(float \*M1, float \*M2, float \*M3, int N)

{

    for (size\_t i = 0; i < N; i++) //selecting row

    {

        for (size\_t j = 0; j < N; j++) //selecting column

        {

            M3[i \* N + j] = M1[i \* N + j] - M2[i \* N + j];

        }

    }

}

float \*MultiplicateMatrix(float \*M1, float \*M2, int N)

{

    float \*M3 = new float[N \* N];

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

    {

        M3[i] = 0;

    }

    for (size\_t i = 0; i < N; i++)

    {

        for (size\_t k = 0; k < N; k++)

        {

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

            {

                M3[i \* N + j] += M1[i \* N + k] \* M2[k \* N + j];

            }

        }

    }

    return M3;

}

void SumMatrix(float \*M1, float \*M2, float \*result, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            result[i \* N + j] = M1[i \* N + j] + M2[i \* N + j];

        }

    }

}

void GenerateMatrix(float \*A, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            A[i \* N + j] = (float)(rand() % 5);

        }

    }

}

float MaxLineCount(float \*matrix, int N)

{

    float maximum = FLT\_MIN;

    for (size\_t i = 0; i < N; i++)

    {

        float temp = 0;

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

        {

            temp += matrix[i \* N + j];

        }

        if (temp > maximum)

        {

            maximum = temp;

        }

    }

    return maximum;

}

void TransposeMatrix(float \*transpMatrix, float \*matrix, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            transpMatrix[i \* N + j] = matrix[j \* N + i];

        }

    }

}

void GenerateMatrixB(float \*A, float \*B, int N)

{

    //showMatrix(A, N);

    float \*transposedA = new float[N \* N];

    TransposeMatrix(transposedA, A, N);

    float maxRowCount = MaxLineCount(A, N);

    float maxColumnCount = MaxLineCount(transposedA, N);

    float divider = 1 / (maxRowCount \* maxColumnCount);

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            B[i \* N + j] = transposedA[i \* N + j] \* divider;

        }

    }

}

void GenerateMatrixI(float \*I, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

        float \*matrix = I + i \* N;

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

        {

            if (i == j)

                matrix[j] = 1.0;

            else

                matrix[j] = 0.0;

        }

    }

}

void GenerateMatrixR(float \*A, float \*I, float \*B, float \*R, int N)

{

    float \*multed = new float[N \* N];

    multed = MultiplicateMatrix(B, A, N);

    SubtractMatrix(I, multed, R, N);

    delete[] multed;

}

float \*GetInversedMatrix(float \*A, int N, int M)

{

    float \*I = new float[N \* N];

    GenerateMatrixI(I, N);

    showMatrix(I, N);

    float \*B = new float[N \* N];

    GenerateMatrixB(A, B, N);

    showMatrix(B, N);

    float \*R = new float[N \* N];

    GenerateMatrixR(A, I, B, R, N);

    showMatrix(R, N);

    float \*result = new float[N \* N];

    result = I;

    showMatrix(result, N);

    float \*Rn = R;

    showMatrix(Rn, N);

    for (size\_t i = 0; i < M; i++)

    {

        SumMatrix(result, Rn, result, N);

        Rn = MultiplicateMatrix(Rn, R, N);

    }

    result = MultiplicateMatrix(result, B, N);

    delete[] I;

    delete[] B;

    delete[] R;

    delete[] Rn;

    return result;

}

int main()

{

    srand(time(0));

    //float \*A = new float[N \* N];

    //GenerateMatrix(A, N);

    /\*float A[N \* N] = {2, 4,

                      2, 3};\*/

    /\*float A[N \* N] = { 2, 4, 2, 3,

                       2, 1, 5, 9,

                       1, 3, 5, 7,

                       8, 9, 2, 1};\*/

    float A[N \* N] = {2, 4, 2, 3, 2, 1, 5, 0,

                      8, 9, 2, 1, 1, 4, 6, 7,

                      1, 3, 5, 7, 8, 9, 2, 1,

                      7, 9, 2, 5, 1, 4, 6, 7,

                      1, 4, 6, 7, 1, 5, 5, 7,

                      2, 4, 2, 3, 2, 4, 2, 3,

                      1, 4, 6, 7, 1, 3, 5, 7,

                      0, 4, 2, 3, 2, 4, 2, 6};

    showMatrix(A, N);

    float \*InversedA = GetInversedMatrix(A, N, M);

    showMatrix(InversedA, N);

    //delete[] A;

    delete[] InversedA;

    double total\_time = clock() / CLOCKS\_PER\_SEC;

    cout << "Total time: " << total\_time << " sec." << endl;

    return 0;

}

# Приложение 2. *AVX256*

include <iostream>

#include <ctime>

#include <cstdlib>

#include <float.h>

#include <immintrin.h>

#include <xmmintrin.h>

using namespace std;

void showMatrix(float \*matrix, size\_t size)

{

    size\_t i, j;

    for (i = 0; i < size; ++i)

    {

        for (j = 0; j < size; ++j)

        {

            cout << matrix[i \* size + j] << " ";

        }

        cout << endl;

    }

    cout << "\n"

         << "\n"

         << "\n";

}

void SubtractMatrix(float \*M1, float \*M2, float \*result, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

        for (size\_t j = 0; j < N; j += 16)

        {

            \_mm256\_storeu\_ps(result + i \* N + j, \_mm256\_sub\_ps(\_mm256\_loadu\_ps(M1 + i \* N + j), \_mm256\_loadu\_ps(M2 + i \* N + j)));

            \_mm256\_storeu\_ps(result + i \* N + j + 8, \_mm256\_sub\_ps(\_mm256\_loadu\_ps(M1 + i \* N + j + 8), \_mm256\_loadu\_ps(M2 + i \* N + j + 8)));

        }

    }

}

float \*MultiplicateMatrix(float \*M1, float \*M2, int N)

{

    float \*M3 = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    for (int i = 0; i < N; ++i)

    {

        float \*m3 = M3 + i \* N;

        for (size\_t j = 0; j < N; j += 8)

            \_mm256\_storeu\_ps(m3 + j, \_mm256\_setzero\_ps());

        for (size\_t k = 0; k < N; ++k)

        {

            float \*m2 = M2 + k \* N;

            \_\_m256 m1 = \_mm256\_set1\_ps(M1[i \* N + k]);

            for (size\_t j = 0; j < N; j += 16)

            {

                \_mm256\_storeu\_ps(m3 + j, \_mm256\_fmadd\_ps(m1, \_mm256\_loadu\_ps(m2 + j), \_mm256\_loadu\_ps(m3 + j)));

                \_mm256\_storeu\_ps(m3 + j + 8, \_mm256\_fmadd\_ps(m1, \_mm256\_loadu\_ps(m2 + j + 8), \_mm256\_loadu\_ps(m3 + j + 8)));

            }

        }

    }

    return M3;

}

void SumMatrix(float \*M1, float \*M2, float \*result, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

        for (size\_t j = 0; j < N; j += 16)

        {

            \_mm256\_storeu\_ps(result + i \* N + j, \_mm256\_add\_ps(\_mm256\_loadu\_ps(M1 + i \* N + j), \_mm256\_loadu\_ps(M2 + i \* N + j)));

            \_mm256\_storeu\_ps(result + i \* N + j + 8, \_mm256\_add\_ps(\_mm256\_loadu\_ps(M1 + i \* N + j + 8), \_mm256\_loadu\_ps(M2 + i \* N + j + 8)));

        }

    }

}

void GenerateMatrix(float \*A, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            A[i \* N + j] = (float)(rand() % 5);

        }

    }

}

float MaxLineCount(float \*matrix, int N)

{

    float maximum = FLT\_MIN;

    for (size\_t i = 0; i < N; i++)

    {

        float temp = 0;

        int row = i \* N;

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

        {

            temp += matrix[row + j];

        }

        if (temp > maximum)

        {

            maximum = temp;

        }

    }

    return maximum;

}

void TransposeMatrix(float \*transpMatrix, float \*matrix, int N)

{

    for (int i = 0; i < N; i += 4)

    {

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

        {

            \_\_m128 row0 = \_mm\_load\_ps(&matrix[i \* N + j]);

            \_\_m128 row1 = \_mm\_load\_ps(&matrix[(i + 1) \* N + j]);

            \_\_m128 row2 = \_mm\_load\_ps(&matrix[(i + 2) \* N + j]);

            \_\_m128 row3 = \_mm\_load\_ps(&matrix[(i + 3) \* N + j]);

            \_MM\_TRANSPOSE4\_PS(row0, row1, row2, row3);

            \_mm\_store\_ps(&transpMatrix[j \* N + i], row0);

            \_mm\_store\_ps(&transpMatrix[(j + 1) \* N + i], row1);

            \_mm\_store\_ps(&transpMatrix[(j + 2) \* N + i], row2);

            \_mm\_store\_ps(&transpMatrix[(j + 3) \* N + i], row3);

        }

    }

}

void GenerateMatrixB(float \*A, float \*B, int N)

{

    //showMatrix(A, N);

    float \*transposedA = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 16);

    TransposeMatrix(transposedA, A, N);

    //showMatrix(transposedA, N);

    float maxRowCount = MaxLineCount(A, N);

    float maxColumnCount = MaxLineCount(transposedA, N);

    float divider = 1 / (maxRowCount \* maxColumnCount);

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            B[i \* N + j] = transposedA[i \* N + j] \* divider;

        }

    }

    /\*\_\_m256\* mB = (\_\_m256\*)B;

    \_\_m256\* mTransposedA = (\_\_m256\*)transposedA;

    \_\_m256 mdivider = \_mm256\_load\_ps(&divider);

    for (size\_t j = 0; j < N; j++)

    {

        mB[j] = \_mm256\_mul\_ps(mTransposedA[j], mdivider);

    }\*/

    \_mm\_free(transposedA);

}

void GenerateMatrixI(float \*I, int N)

{

    for (size\_t i = 0; i < N; ++i)

    {

        float \*result = I + i \* N;

        for (size\_t j = 0; j < N; j += 8)

            \_mm256\_storeu\_ps(result + j, \_mm256\_setzero\_ps());

    }

    for (size\_t i = 0; i < N; i++)

        I[i \* N + i] = 1;

}

void GenerateMatrixR(float \*A, float \*I, float \*B, float \*R, int N)

{

    float \*multed = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    multed = MultiplicateMatrix(B, A, N);

    SubtractMatrix(I, multed, R, N);

    \_mm\_free(multed);

}

void CopyMatrix(float \*dest, float \*src, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            dest[i \* N + j] = src[i \* N + j];

        }

    }

}

float \*GetInversedMatrix(float \*A, int N, int M)

{

    float \*I = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    GenerateMatrixI(I, N);

    //showMatrix(I, N);

    float \*B = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    GenerateMatrixB(A, B, N);

    //showMatrix(B, N);

    float \*R = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    GenerateMatrixR(A, I, B, R, N);

    //showMatrix(R, N);

    //float\* result = I;

    //float\* result = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    //CopyMatrix(I, result, N);

    //showMatrix(result, N);

    float \*Rn = R;

    //float\* Rn = (float\*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    //CopyMatrix(R, Rn, N);

    //showMatrix(Rn, N);

    SumMatrix(I, Rn, I, N);

    for (size\_t i = 0; i < M - 1; i++)

    {

        Rn = MultiplicateMatrix(Rn, R, N);

        SumMatrix(I, Rn, I, N);

    }

    I = MultiplicateMatrix(I, B, N);

    \_mm\_free(B);

    \_mm\_free(R);

    return I;

}

int main()

{

    int M = 10;

    int N = 2048;

    srand(time(0));

    clock\_t a = clock();

    float \*A = (float \*)\_mm\_malloc(N \* N \* sizeof(float), 32);

    GenerateMatrix(A, N);

    /\*float A[N \* N] = { 2, 4, 2, 3, 2, 1, 5, 9, 2, 4, 2, 3, 2, 1, 5, 9,

                       1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7,

                       8, 9, 2, 1, 1, 4, 6, 7, 2, 4, 2, 3, 2, 4, 2, 3,

                       1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7,

                       8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1,

                       1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7,

                       2, 4, 2, 3, 2, 4, 2, 3, 1, 3, 5, 7, 8, 9, 2, 1,

                       1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7,

                       2, 4, 2, 3, 2, 1, 5, 9, 2, 4, 2, 3, 2, 1, 5, 9,

                       1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7,

                       8, 9, 2, 1, 1, 4, 6, 7, 2, 4, 2, 3, 2, 4, 2, 3,

                       1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7,

                       8, 9, 2, 1, 1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1,

                       1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7,

                       2, 4, 2, 3, 2, 4, 2, 3, 1, 3, 5, 7, 8, 9, 2, 1,

                       1, 4, 6 ,7, 1, 3, 5, 7, 8, 9, 2, 1, 1, 4, 6 ,7 };\*/

    /\*float A[N \* N] = { 2, 4, 2, 3, 2, 1, 5, 0,

                           8, 9, 2, 1, 1, 4, 6, 7,

                           1, 3, 5, 7, 8, 9, 2, 1,

                           7, 9, 2, 5, 1, 4, 6, 7,

                           1, 4, 6, 7, 1, 5, 5, 7,

                           2, 4, 2, 3, 2, 4, 2, 3,

                           1, 4, 6, 7, 1, 3, 5, 7,

                           0, 4, 2, 3, 2, 4, 2, 6 }; \*/

    //showMatrix(A, N);

    float \*InversedA = GetInversedMatrix(A, N, M);

    //showMatrix(InversedA, N);

    double total\_time = (clock() - a) / CLOCKS\_PER\_SEC;

    cout << "Total time: " << total\_time << " sec." << endl;

    //\_mm\_free(A);

    \_mm\_free(InversedA);

    return 0;

}

# Приложение 3. *BLAS*

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <float.h>

#include <immintrin.h>

#include <cblas.h>

using namespace std;

void showMatrix(float \*matrix, size\_t size)

{

    size\_t i, j;

    for (i = 0; i < size; ++i)

    {

        for (j = 0; j < size; ++j)

        {

            cout << matrix[i \* size + j] << " ";

        }

        cout << endl;

    }

    cout << "\n"

         << "\n"

         << "\n";

}

float \*GenerateMatrixI(int N)

{

    float \*I = new float[N \* N];

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            if (i == j)

                I[i \* N + j] = 1.0;

            else

                I[i \* N + j] = 0.0;

        }

    }

    return I;

}

float \*GenerateMatrix(int N)

{

    float \*A = new float[N \* N];

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            A[i \* N + j] = (float)(rand() % 5);

        }

    }

    return A;

}

float MaxRowCount(float \*A, const int N)

{

    float \*res = new float[N];

    for (int i = 0; i < N; i++)

    {

        res[i] = cblas\_sasum(N, &A[i], N);

    }

    return res[cblas\_isamax(N, res, 1)];

}

float MaxColumnCount(float \*A, const int N)

{

    float \*res = new float[N];

    for (int i = 0; i < N; i++)

    {

        res[i] = cblas\_sasum(N, &A[i \* N], 1);

    }

    return res[cblas\_isamax(N, res, 1)];

}

float \*GenerateMatrixB(float \*A, int N, float \*I)

{

    float \*B = new float[N \* N];

    float maxRowCount = MaxRowCount(A, N);

    float maxColumnCount = MaxColumnCount(A, N);

    //cout << maxRowCount << " " << maxColumnCount << "\n" << endl;

    float divider = 1 / (maxRowCount \* maxColumnCount);

    cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, N, N, N, divider, I, N, A, N, 0.0, B, N);

    return B;

}

float \*GenerateMatrixR(float \*A, float \*I, float \*B, int N)

{

    float \*multed = new float[N \* N];

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

        multed[i] = 0;

    cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, B, N, A, N, 0.0, multed, N);

    cblas\_saxpy(N \* N, -1.0, multed, 1, I, 1);

    return I;

}

void CopyMatrix(float \*dest, float \*src, int N)

{

    for (size\_t i = 0; i < N; i++)

    {

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

        {

            dest[i \* N + j] = src[i \* N + j];

        }

    }

}

float \*GetInversedMatrix(float \*A, int N, int M)

{

    float \*I = GenerateMatrixI(N);

    //showMatrix(I, N);

    float \*B = GenerateMatrixB(A, N, I);

    //showMatrix(B, N);

    float \*R = GenerateMatrixR(A, I, B, N);

    //showMatrix(R, N);

    float \*tmp = new float[N \* N];

    float \*res = new float[N \* N];

    CopyMatrix(tmp, R, N);

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

        res[i] = 0;

    float \*result = new float[N \* N];

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

        result[i] = 0;

    cblas\_saxpy(N \* N, 1.0, I, 1, result, 1);

    for (size\_t i = 0; i < M; i++)

    {

        cblas\_saxpy(N \* N, 1.0, tmp, 1, result, 1);

        cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, R, N, tmp, N, 0.0, res, N);

        CopyMatrix(tmp, res, N);

    }

    //showMatrix(result, N);

    cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, result, N, B, N, 0.0, res, N);

    delete[] B;

    delete[] R;

    return res;

}

int main()

{

    const int M = 10;

    const int N = 2048;

    clock\_t a = clock();

    srand(time(0));

    float \*A = GenerateMatrix(N);

    /\*float A[N \* N] = {1, 3, 1,

              1, 0, 3,

              2, 1, 3};\*/

    //showMatrix(A, N);

    float \*InversedA = GetInversedMatrix(A, N, M);

    //showMatrix(InversedA, N);

    cout << (float)((clock() - a) / CLOCKS\_PER\_SEC) << endl;

    delete[] A;

    delete[] InversedA;

    return 0;

}