###### 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 21206

**Балашова Вячеслава Вадимовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2022

**Содержание**

Цель3

Задание3

Описание работы4

Заключение5

Приложение 1. Исходный код программы без ручной векторизации6

Приложение 2. Исходный код программы с ручной векторизацией10

Приложение 3. Исходный код программы с библиотекой BLAS15

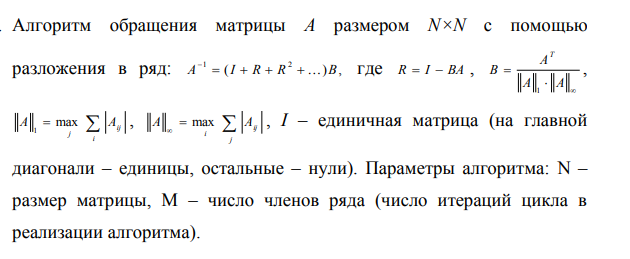
**Цель**

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:
   1. вариант без векторизации,
   2. вариант с ручной векторизацией (выбрать любой вариант из возможных трех: ассемблерная вставка, встроенные функции компилятора, расширение GCC),
   3. вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS.
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.

**Вариант задания**



**Описание работы**

Ход выполнения работы:

1. Был написан первый вариант программы на языке C++, реализующий алгоритм без какой-либо векторизации вычислений (см. Приложение 1). Была проверена правильность работы функций программы на нескольких небольших входных данных. После проверки было измерено время работы программы для N=2048 и M=10. Было измерено время работы программы с помощью библиотеки ctime. Итоговое время представлено на Рис. 1.



Рис 1. Время работы программы без векторизации

1. Был написан второй вариант программы на языке C++, реализующий алгоритм с применением ручной векторизацией с помощью расширения системы команд AVX и AVX2 и встроенных SIMD-функций компилятора (см. Приложение 2). Была также проверена правильность работы функций на нескольких небольших входных данных. Также было замерено время работы программы с помощью библиотеки ctime для входных данных N=2048 и M=10. Результат измерений представлен на Рис. 2.

Рис 2. Время работы программы с ручной векторизацией



1. Был написан третий вариант программы на языке С++, реализующий алгоритм с использованием библиотеки BLAS для входных данных N=2048, M=10 (см. Приложение 3). Было измерено время с помощью библиотеки ctime и результат измерения представлен на Рис. 3.

Рис 3. Время работы программы с библиотекой BLAS



1. Было произведено сравнения полученных результатов

**Заключение**

В ходе выполнения практической работы были изучены варианты векторизации вычислений в языках Си/С++.

Были найдены все возможные способы векторизации вычислений на процессоре AMD Ryzen 7 5700u и выбран самый оптимальный из существующих.

Были написаны три программы на языке C++, две из которых выполняются с использованием различных способов векторизации: ручная векторизация с помощью встроенных SIMD функций компилятора и с использованием библиотеки BLAS. Еще одна программа не имеет никаких ускорений.

В результате замеров времени работы было выявлено, что программа без векторизации вычислений работала 625.88 секунд, или же примерно 10.43 минут. С ручной векторизацией программа выполнялась 2.3 минуты, что примерно в 4.5 раз быстрее, чем без векторизации. Программа, использующая библиотеку BLAS выполнялась 41.5 секунд.

Можно сделать вывод: Вариант с выполнением матричных вычислений с использованием библиотеки BLAS дает наибольший прирост скорости вычислений.

**Приложение 1. Исходный код программы без ручной векторизации**

#include <iomanip>

#include <iostream>

#include <random>

#include <ctime>

using namespace std;

static int MATRIX\_SIZE = 2048;

static int NUM\_OF\_ITERATIONS = 10;

static int MAX\_MATRIX\_VALUE = 10;

inline void MakeIdentityMatrix(float \* Matrix)

{

fill(Matrix, Matrix + (MATRIX\_SIZE \* MATRIX\_SIZE), 0);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

Matrix[i \* MATRIX\_SIZE + i] = 1.;

}

}

inline void TransposeMatrix(const float \* Matrix, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

Result[i \* MATRIX\_SIZE + j] = Matrix[j \* MATRIX\_SIZE + i];

}

}

}

inline float MaxAbsRawSum(float \* Matrix)

{

float forRet = Matrix[0];

for (int i = 0; i < MATRIX\_SIZE; i++)

{

float sum = 0;

for (int j = 0; j < MATRIX\_SIZE; j++)

{

sum += abs(Matrix[i \* MATRIX\_SIZE + j]);

}

forRet = max(sum, forRet);

}

return forRet;

}

inline void MulMatrixWithScalar(const float \* Matrix1, const float Scalar, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

Result[i] = Matrix1[i] \* Scalar;

}

}

inline void AddMatrices(const float \* Matrix1, const float \* Matrix2, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

Result[i] = Matrix1[i] + Matrix2[i];

}

}

inline void SubMatrices(const float \* Matrix1, const float \* Matrix2, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

Result[i] = Matrix1[i] - Matrix2[i];

}

}

inline void MulMatrices(const float \* Matrix1, const float \* Matrix2, float \* Result)

{

fill(Result, Result + (MATRIX\_SIZE \* MATRIX\_SIZE), 0);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int k = 0; k < MATRIX\_SIZE; k++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

Result[i \* MATRIX\_SIZE + j] += Matrix1[i \* MATRIX\_SIZE + k] \* Matrix2[k \* MATRIX\_SIZE + j];

}

}

}

}

inline void PrintMatrix(const float \* Matrix)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

cout << left << setw(15) << Matrix[i \* MATRIX\_SIZE + j] << ' ';

}

cout << endl;

}

cout << endl;

}

inline float \* FindInverse(float \* A)

{

auto \* I = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* A\_T = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* B = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* R = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Rn = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Res = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* buf = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

// Make I

MakeIdentityMatrix(I);

// Make A\_T

TransposeMatrix(A, A\_T);

// Count constants

float a\_1 = MaxAbsRawSum(A\_T);

float a\_inf = MaxAbsRawSum(A);

// Make B

MulMatrixWithScalar(A\_T, 1 / (a\_1 \* a\_inf), B);

// FIll R

MulMatrices(B, A, buf);

SubMatrices(I, buf, R);

// Make base of Result

copy(I, I + (MATRIX\_SIZE \* MATRIX\_SIZE), Res);

// FIll Rn

copy(R, R + (MATRIX\_SIZE \* MATRIX\_SIZE), Rn);

for (size\_t i = 0; i < NUM\_OF\_ITERATIONS; i++)

{

AddMatrices(Res, Rn, buf); // Adds Rn to Res and saves it to buf

swap(buf, Res); // swaps previous value with new in buf

MulMatrices(Rn, R, buf); // Makes Rn --> Rn+1 and saves to buf

swap(buf, Rn); // Swaps previous value of Rn and new in buf

}

MulMatrices(Res, B, buf); // Multiplies Res and buf and saves result to buf

swap(Res, buf); // Load result to Res from buf and saves previous in buf

delete[] I;

delete[] A\_T;

delete[] B;

delete[] R;

delete[] Rn;

delete[] buf;

return Res;

}

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

{

auto \* A = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

// Fill A

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

A[i] = static\_cast<float> (random() % MAX\_MATRIX\_VALUE);

}

if (argc > 1)

{

MATRIX\_SIZE = stoi(argv[1]);

if (argc > 2)

{

NUM\_OF\_ITERATIONS = stoi(argv[2]);

}

}

clock\_t start = clock();

auto \* Result = FindInverse(A);

clock\_t final = clock();

cout << (double(final - start)) / CLOCKS\_PER\_SEC << endl;

delete[] A;

delete[] Result;

return 0;

}

**Приложение 2. Исходный код программы с ручной векторизацией**

#include <iomanip>

#include <iostream>

#include <random>

#include <immintrin.h>

using namespace std;

static int MATRIX\_SIZE = 2048;

static int NUM\_OF\_ITERATIONS = 10;

static int MAX\_MATRIX\_VALUE = 10;

inline void AVX2\_FillZero(float \* Matrix)

{

auto reg0 = \_mm256\_setzero\_ps();

for (auto i = Matrix; i < Matrix + MATRIX\_SIZE \* MATRIX\_SIZE; i += 8)

{

\_mm256\_store\_ps(i, reg0);

}

}

inline void AVX2\_MakeIdentityMatrix(float \* Matrix)

{

AVX2\_FillZero(Matrix);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

Matrix[i \* MATRIX\_SIZE + i] = 1.;

}

}

inline void TransposeMatrix(const float \* Matrix, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

Result[i \* MATRIX\_SIZE + j] = Matrix[j \* MATRIX\_SIZE + i];

}

}

}

inline float MaxAbsRawSum(float \* Matrix)

{

float forRet = Matrix[0];

for (int i = 0; i < MATRIX\_SIZE; i++)

{

float sum = 0;

for (int j = 0; j < MATRIX\_SIZE; j++)

{

sum += abs(Matrix[i \* MATRIX\_SIZE + j]);

}

forRet = max(sum, forRet);

}

return forRet;

}

inline void AVX2\_MulMatrixWithScalar(const float \* Matrix, const float scalar, float \* Result)

{

auto \* scalarVector = new float[8];

for (int i = 0; i < 8; i++)

{

scalarVector[i] = scalar;

}

auto reg0 = \_mm256\_load\_ps(scalarVector);

delete[] scalarVector;

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i+= 8)

{

auto reg1 = \_mm256\_load\_ps(Matrix + i);

reg1 = \_mm256\_mul\_ps(reg1, reg0);

\_mm256\_store\_ps(Result + i, reg1);

}

}

inline void AVX2\_AddMatrices(const float \* A, const float \* B, float \* Res)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i+=8)

{

auto reg0 = \_mm256\_load\_ps(A + i);

auto reg1 = \_mm256\_load\_ps(B + i);

reg0 = \_mm256\_add\_ps(reg0, reg1);

\_mm256\_store\_ps(Res + i, reg0);

}

}

inline void AVX2\_SubMatrices(const float \* A, const float \* B, float \* Res)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i+=8)

{

auto reg0 = \_mm256\_load\_ps(A + i);

auto reg1 = \_mm256\_load\_ps(B + i);

reg0 = \_mm256\_sub\_ps(reg0, reg1);

\_mm256\_store\_ps(Res + i, reg0);

}

}

inline void AVX2\_MulMatrices(const float \* A, const float \* B, float \* Res)

{

AVX2\_FillZero(Res);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int k = 0; k < MATRIX\_SIZE; k++)

{

float buf[8];

std::fill(buf, buf + 8, A[i \* MATRIX\_SIZE + k]);

auto reg0 = \_mm256\_load\_ps(buf);

for (int j = 0; j < MATRIX\_SIZE; j+=8)

{

auto reg1 = \_mm256\_load\_ps(B + (k \* MATRIX\_SIZE) + j);

auto reg2 = \_mm256\_load\_ps(Res + (i \* MATRIX\_SIZE) + j);

reg1 = \_mm256\_mul\_ps(reg0, reg1);

reg2 = \_mm256\_add\_ps(reg2, reg1);

\_mm256\_store\_ps(Res + (i \* MATRIX\_SIZE) + j, reg2);

}

}

}

}

inline void AVX2\_CopyMatrix(const float \* A, float \* Res)

{

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i+=8)

{

auto reg0 = \_mm256\_load\_ps(A + i);

\_mm256\_store\_ps(Res + i, reg0);

}

}

inline void PrintMatrix(const float \* Matrix)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

cout << left << setw(15) << Matrix[i \* MATRIX\_SIZE + j] << ' ';

}

cout << endl;

}

cout << endl;

}

inline float \* FindInverse(float \* A)

{

auto \* I = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* A\_T = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* B = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* R = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Rn = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Res = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* buf = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

// Make I

AVX2\_MakeIdentityMatrix(I);

// Make A\_T

TransposeMatrix(A, A\_T);

// Count constants

float a\_1 = MaxAbsRawSum(A\_T);

float a\_inf = MaxAbsRawSum(A);

// Make B

AVX2\_MulMatrixWithScalar(A\_T, 1 / (a\_1 \* a\_inf), B);

// FIll R

AVX2\_MulMatrices(B, A, buf);

AVX2\_SubMatrices(I, buf, R);

// Make base of Result

AVX2\_CopyMatrix(I, Res);

// FIll Rn

AVX2\_CopyMatrix(R, Rn);

for (size\_t i = 0; i < NUM\_OF\_ITERATIONS; i++)

{

AVX2\_AddMatrices(Res, Rn, buf); // Adds Rn to Res and saves it to buf

swap(buf, Res); // swaps previous value with new in buf

AVX2\_MulMatrices(Rn, R, buf); // Makes Rn --> Rn+1 and saves to buf

swap(buf, Rn); // Swaps previous value of Rn and new in buf

}

AVX2\_MulMatrices(Res, B, buf); // Multiplies Res and buf and saves result to buf

swap(Res, buf); // Load result to Res from buf and saves previous in buf

delete[] I;

delete[] A\_T;

delete[] B;

delete[] R;

delete[] Rn;

delete[] buf;

return Res;

}

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

{

auto \* A = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

// Fill A

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

A[i] = static\_cast<float> (random() % MAX\_MATRIX\_VALUE);

}

if (argc > 1)

{

MATRIX\_SIZE = stoi(argv[1]);

if (argc > 2)

{

NUM\_OF\_ITERATIONS = stoi(argv[2]);

}

}

clock\_t start = clock();

auto \* Result = FindInverse(A);

clock\_t final = clock();

cout << (double(final - start)) / CLOCKS\_PER\_SEC << endl;

delete[] A;

delete[] Result;

return 0;

}

**Приложение 3. Исходный код программы с библиотекой BLAS**

#include <iomanip>

#include <iostream>

#include <random>

#include <cblas.h>

using namespace std;

static int MATRIX\_SIZE = 2048;

static int NUM\_OF\_ITERATIONS = 10;

static int MAX\_MATRIX\_VALUE = 10;

inline void MakeIdentityMatrix(float \* Matrix)

{

fill(Matrix, Matrix + (MATRIX\_SIZE \* MATRIX\_SIZE), 0);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

Matrix[i \* MATRIX\_SIZE + i] = 1.;

}

}

inline void TransposeMatrix(const float \* Matrix, float \* Result)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

Result[i \* MATRIX\_SIZE + j] = Matrix[j \* MATRIX\_SIZE + i];

}

}

}

inline float MaxAbsRawSum(float \* Matrix)

{

float forRet = Matrix[0];

for (int i = 0; i < MATRIX\_SIZE; i++)

{

forRet = max(cblas\_sasum(MATRIX\_SIZE, Matrix + i \* MATRIX\_SIZE, 1), forRet);

}

return forRet;

}

inline void MulMatrixWithScalar(float \* Matrix, const float Scalar)

{

cblas\_sscal(MATRIX\_SIZE \* MATRIX\_SIZE, Scalar, Matrix, 1);

}

inline void AddMatrices(float \* Matrix1, float \* Matrix2)

{

cblas\_saxpy(MATRIX\_SIZE \* MATRIX\_SIZE, 1, Matrix1, 1, Matrix2, 1);

}

inline void MulMatrices(const float \* Matrix1, const float \* Matrix2, float \* Result)

{

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, MATRIX\_SIZE, MATRIX\_SIZE,

MATRIX\_SIZE, 1.0, Matrix1, MATRIX\_SIZE, Matrix2, MATRIX\_SIZE, 0.0, Result, MATRIX\_SIZE);

}

inline void PrintMatrix(const float \* Matrix)

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

cout << left << setw(15) << Matrix[i \* MATRIX\_SIZE + j] << ' ';

}

cout << endl;

}

cout << endl;

}

inline float \* FindInverse(float \* A)

{

auto \* I = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* B = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* R = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* buf = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Res = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

auto \* Rn = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

MakeIdentityMatrix(I);

TransposeMatrix(A, B);

float a\_1 = MaxAbsRawSum(B);

float a\_inf = MaxAbsRawSum(A);

copy(I, I + MATRIX\_SIZE \* MATRIX\_SIZE, R);

MulMatrixWithScalar(B, 1 / a\_1 / a\_inf);

MulMatrices(B, A, buf);

MulMatrixWithScalar(buf, -1);

AddMatrices(R, buf);

swap(buf, R);

// Make base of Result

copy(I, I + (MATRIX\_SIZE \* MATRIX\_SIZE), Res);

// FIll Rn

copy(R, R + (MATRIX\_SIZE \* MATRIX\_SIZE), Rn);

for (size\_t i = 0; i < NUM\_OF\_ITERATIONS; i++)

{

AddMatrices(Rn, Res);

MulMatrices(Rn, R, buf);

swap(buf, Rn);

}

MulMatrices(Res, B, buf);

swap(Res, buf);

delete[] I;

delete[] B;

delete[] R;

delete[] Rn;

delete[] buf;

return Res;

}

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

{

auto \* A = new float[MATRIX\_SIZE \* MATRIX\_SIZE];

// Fill A

for (int i = 0; i < MATRIX\_SIZE \* MATRIX\_SIZE; i++)

{

A[i] = static\_cast<float> (random() % MAX\_MATRIX\_VALUE);

}

if (argc > 1)

{

MATRIX\_SIZE = stoi(argv[1]);

if (argc > 2)

{

NUM\_OF\_ITERATIONS = stoi(argv[2]);

}

}

clock\_t start = clock();

auto \* Result = FindInverse(A);

clock\_t final = clock();

cout << (double(final - start)) / CLOCKS\_PER\_SEC << endl;

delete[] A;

delete[] Result;

return 0;

}