###### 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 №4

"ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ"

студента второго курса, 23203 группы

**Пучкова Даниила Дмитриевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук, доцент кафедры параллельных вычислений

А.Ю. Власенко

Новосибирск 2024

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#__RefHeading___Toc617_2968956038)

[ЦЕЛЬ 2](#__RefHeading___Toc619_2968956038)

[ЗАДАЧИ 2](#__RefHeading___Toc621_2968956038)

[ОПИСАНИЕ РАБОТЫ 3](#__RefHeading___Toc623_2968956038)

[ЗАКЛЮЧЕНИЕ 4](#__RefHeading___Toc625_2968956038)

[Приложение 1. (Исходный код программы на СИ) 5](#__RefHeading___Toc627_2968956038)

[Приложение 2. (Исходный код программы на СИ с ручной векторизацией) 10](#__RefHeading___Toc629_2968956038)

[Приложение 3. (Исходный код программы на СИ с BLAS) 16](#__RefHeading___Toc631_2968956038)

# ЦЕЛЬ

Написать программу на языке СИ, реализовать вариант с ручной векторизацией с использованием векторного расширения AVX512 и вариант с использованием библиотеки BLAS.

# ЗАДАЧИ

1. 1. Написать три варианта программы, реализующей алгоритм из задания:
2.  - Вариант без векторизации.
3.  - 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.
4.  - Вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS. Для элементов матриц использовать тип данных float. Использовать минимум 2 BLAS-функции.
5. 2. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.
6. 3. Каждый вариант программы оптимизировать по скорости, насколько это возможно.
7. 4. Сравнить время работы трех вариантов программы для N=2048, M=10.
8. 5. Составить отчет по лабораторной работе.

# ОПИСАНИЕ РАБОТЫ

**Вариант программы без векторизации:**

1. Был разработан код на языке СИ для инверсии матрицы.
2. Программа была протестирована на матрицах размером N = 2048.
3. Время работы программы было замерено функцией clock\_gettime.

**Вариант программы с ручной векторизацией:**

1. Адаптирована программа с использованием SIMD intrinsics для оптимизации процесса инверсии матрицы.
2. Проведено тестирование оптимизированной программы на тех же входных данных. Отмечено улучшение производительности по сравнению с неоптимизированным вариантом.
3. Время работы программы было замерено функцией clock\_gettime.

**Вариант программы с использованием библиотеки BLAS:**

1. Написана версия программы, использующая функции библиотеки BLAS для выполнения матричных операций.
2. Программа тестировалась на аналогичных входных данных. Благодаря использованию оптимизированных функций библиотеки BLAS, наблюдалось дальнейшее ускорение процесса обработки данных.
3. Время работы программы было замерено функцией clock\_gettime.

**Результаты измерения:**

****

# ЗАКЛЮЧЕНИЕ

В ходе данной практической работы были изучены и применены SIMD-расширения архитектуры x86/x86-64, что позволило освоить написание SIMD-интринсиков на языке СИ. Эти навыки играют важную роль при создании высокопроизводительных программ. Кроме того, был проведен анализ работы с оптимизированной библиотекой BLAS, что позволило на практике оценить преимущества специализированных библиотек для выполнения математических вычислений.

В процессе работы стало очевидно, что использование готовых решений, таких как BLAS, часто оказывается наиболее эффективным выбором благодаря их высокой производительности и оптимизации для различных операций. Однако ручная векторизация, хотя и способна дать дополнительный прирост скорости, требует глубоких знаний и значительных усилий для реализации. Следует также отметить, что использование BLAS имеет свои сложности, например, множество параметров, что может усложнить восприятие и поддержку кода.

# **Приложение 1.** (Исходный код программы на СИ)

#include <bits/time.h>

#include <float.h>

#include <math.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define RAND\_FACTOR 10

#define N 2048

#define M 10

#define max(a, b) ((a > b) ? a : b)

void

print(float \*matrix) {

for (int i = 0; i < N; i++) {

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

printf("%f ", matrix[i \* N + j]);

}

printf("\n");

}

puts("");

}

float \*make\_matrix(void) {

float \*new\_matrix = calloc(N \* N, sizeof(float));

if (!new\_matrix)

return NULL;

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

new\_matrix[i] = rand() % RAND\_FACTOR;

}

return new\_matrix;

}

float \*make\_I\_matrix(void) {

float \*I\_matrix = calloc(N \* N, sizeof(float));

if (!I\_matrix)

return NULL;

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

I\_matrix[i \* N + i] = 1.0;

}

return I\_matrix;

}

float \*transpose(float \*matrix) {

float \*result = calloc(N \* N, sizeof(float));

if (!result)

return NULL;

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

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

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

}

}

return result;

}

float max\_sum\_row(float \*matrix) {

float max\_sum = FLT\_MIN;

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

float sum = 0;

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

sum += fabs(matrix[i \* N + j]);

}

max\_sum = max(sum, max\_sum);

}

return max\_sum;

}

void matrix\_scalar\_mult(float \*matrix, float scalar) {

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

matrix[i] \*= scalar;

}

}

float \*make\_B\_matrix(float \*transposed, float \*matrix) {

float max\_row = max\_sum\_row(matrix);

float max\_column = max\_sum\_row(transposed);

matrix\_scalar\_mult(transposed, 1 / (max\_row \* max\_column));

return transposed;

}

void mult\_matrices(float \*res, float \*mat1, float \*mat2) {

memset(res, 0, N \* N \* sizeof(float));

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

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

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

res[i \* N + j] += mat1[i \* N + k] \* mat2[k \* N + j];

}

}

}

}

void sum\_matrices(float \*res, float \*mat1, float \*mat2) {

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

res[i] = mat1[i] + mat2[i];

}

}

float \*make\_R\_matrix(float \*I, float \*B, float \*matrix) {

float \*R = calloc(N \* N, sizeof(float));

if (!R)

return NULL;

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

mult\_matrices(tmp, B, matrix);

matrix\_scalar\_mult(tmp, -1);

sum\_matrices(R, I, tmp);

free(tmp);

return R;

}

float \*find\_inverse\_matrix(float \*matrix) {

float \*result = make\_I\_matrix();

float \*transposed = transpose(matrix);

float \*I = make\_I\_matrix();

float \*B = make\_B\_matrix(transposed, matrix);

float \*R = make\_R\_matrix(I, B, matrix);

float \*Rn = calloc(N \* N, sizeof(float));

if (!Rn)

return NULL;

memcpy(Rn, R, N \* N \* sizeof(float));

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

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

sum\_matrices(result, result, Rn);

mult\_matrices(tmp, Rn, R);

memcpy(Rn, tmp, N \* N \* sizeof(float));

}

mult\_matrices(tmp, result, B);

memcpy(result, tmp, N \* N \* sizeof(float));

free(I);

free(B);

free(R);

free(Rn);

free(tmp);

return result;

}

int main(void) {

srand(time(NULL));

struct timespec start, end;

float \*matrix = make\_matrix();

float \*inverse\_matrix;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

inverse\_matrix = find\_inverse\_matrix(matrix);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

printf("%f %f\n\n", inverse\_matrix[0], inverse\_matrix[N \* N - 1]);

printf("Preformance time: %lf sec\n",

end.tv\_sec - start.tv\_sec +

0.000000001 \* (end.tv\_nsec - start.tv\_nsec));

free(matrix);

free(inverse\_matrix);

return 0;

}

# **Приложение 2.** (Исходный код программы на СИ с ручной векторизацией)

#include <bits/time.h>

#include <float.h>

#include <math.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <immintrin.h>

#pragma clang target("avx512f")

#define RAND\_FACTOR 10

#define N 2048

#define M 10

#define max(a, b) ((a > b) ? a : b)

void

print(float \*matrix) {

for (int i = 0; i < N; i++) {

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

printf("%f ", matrix[i \* N + j]);

}

printf("\n");

}

puts("");

}

float \*make\_matrix(void) {

float \*new\_matrix = calloc(N \* N, sizeof(float));

if (!new\_matrix)

return NULL;

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

new\_matrix[i] = rand() % RAND\_FACTOR;

}

return new\_matrix;

}

float \*make\_I\_matrix(void) {

float \*I\_matrix = calloc(N \* N, sizeof(float));

if (!I\_matrix)

return NULL;

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

I\_matrix[i \* N + i] = 1.0;

}

return I\_matrix;

}

float \*transpose(float \*matrix) {

float \*result = calloc(N \* N, sizeof(float));

if (!result)

return NULL;

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

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

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

}

}

return result;

}

float max\_sum\_row(float \*matrix) {

float max\_sum = FLT\_MIN;

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

float sum = 0;

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

sum += fabs(matrix[i \* N + j]);

}

max\_sum = max(sum, max\_sum);

}

return max\_sum;

}

void matrix\_scalar\_mult(float \*matrix, float scalar) {

\_\_m512 scalar\_vec = \_mm512\_set1\_ps(scalar);

for (size\_t i = 0; i < N \* N; i += 16) {

\_\_m512 vec = \_mm512\_loadu\_ps(&matrix[i]);

\_\_m512 res = \_mm512\_mul\_ps(vec, scalar\_vec);

\_mm512\_storeu\_ps(&matrix[i], res);

}

}

float \*make\_B\_matrix(float \*transposed, float \*matrix) {

float max\_row = max\_sum\_row(matrix);

float max\_column = max\_sum\_row(transposed);

matrix\_scalar\_mult(transposed, 1 / (max\_row \* max\_column));

return transposed;

}

void mult\_matrices(float \*res, float \*mat1, float \*mat2) {

memset(res, 0, N \* N \* sizeof(float));

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

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

\_\_m512 vec1 = \_mm512\_set1\_ps(mat1[i \* N + k]);

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

\_\_m512 vec2 = \_mm512\_loadu\_ps(&mat2[k \* N + j]);

\_\_m512 res\_vec = \_mm512\_loadu\_ps(&res[i \* N + j]);

res\_vec = \_mm512\_fmadd\_ps(vec1, vec2, res\_vec);

\_mm512\_storeu\_ps(&res[i \* N + j], res\_vec);

}

}

}

}

void sum\_matrices(float \*res, float \*mat1, float \*mat2) {

for (size\_t i = 0; i < N \* N; i += 16) {

\_\_m512 vec1 = \_mm512\_loadu\_ps(&mat1[i]);

\_\_m512 vec2 = \_mm512\_loadu\_ps(&mat2[i]);

\_\_m512 res\_vec = \_mm512\_add\_ps(vec1, vec2);

\_mm512\_storeu\_ps(&res[i], res\_vec);

}

}

float \*make\_R\_matrix(float \*I, float \*B, float \*matrix) {

float \*R = calloc(N \* N, sizeof(float));

if (!R)

return NULL;

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

mult\_matrices(tmp, B, matrix);

matrix\_scalar\_mult(tmp, -1);

sum\_matrices(R, I, tmp);

free(tmp);

return R;

}

float \*find\_inverse\_matrix(float \*matrix) {

float \*result = make\_I\_matrix();

float \*transposed = transpose(matrix);

float \*I = make\_I\_matrix();

float \*B = make\_B\_matrix(transposed, matrix);

float \*R = make\_R\_matrix(I, B, matrix);

float \*Rn = calloc(N \* N, sizeof(float));

if (!Rn)

return NULL;

memcpy(Rn, R, N \* N \* sizeof(float));

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

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

sum\_matrices(result, result, Rn);

mult\_matrices(tmp, Rn, R);

memcpy(Rn, tmp, N \* N \* sizeof(float));

}

mult\_matrices(tmp, result, B);

memcpy(result, tmp, N \* N \* sizeof(float));

free(I);

free(B);

free(R);

free(Rn);

free(tmp);

return result;

}

int main(void) {

srand(time(NULL));

struct timespec start, end;

float \*matrix = make\_matrix();

float \*inverse\_matrix;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

inverse\_matrix = find\_inverse\_matrix(matrix);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

printf("%f %f\n\n", inverse\_matrix[0], inverse\_matrix[N \* N - 1]);

printf("Preformance time: %lf sec\n",

end.tv\_sec - start.tv\_sec +

0.000000001 \* (end.tv\_nsec - start.tv\_nsec));

free(matrix);

free(inverse\_matrix);

return 0;

}

# **Приложение 3.** (Исходный код программы на СИ с BLAS)

#include <float.h>

#include <math.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <cblas.h>

#define RAND\_FACTOR 10

#define N 2048

#define M 10

#define max(a, b) ((a > b) ? a : b)

void

print(float \*matrix) {

for (int i = 0; i < N; i++) {

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

printf("%f ", matrix[i \* N + j]);

}

printf("\n");

}

puts("");

}

float \*make\_matrix(void) {

float \*new\_matrix = calloc(N \* N, sizeof(float));

if (!new\_matrix)

return NULL;

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

new\_matrix[i] = rand() % RAND\_FACTOR;

}

return new\_matrix;

}

float \*make\_I\_matrix(void) {

float \*I\_matrix = calloc(N \* N, sizeof(float));

if (!I\_matrix)

return NULL;

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

I\_matrix[i \* N + i] = 1.0;

}

return I\_matrix;

}

float \*transpose(float \*matrix) {

float \*result = calloc(N \* N, sizeof(float));

if (!result)

return NULL;

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

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

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

}

}

return result;

}

float max\_sum\_row(float \*matrix) {

float max\_sum = FLT\_MIN;

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

float sum = 0;

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

sum += fabs(matrix[i \* N + j]);

}

max\_sum = max(sum, max\_sum);

}

return max\_sum;

}

void matrix\_scalar\_mult(float \*matrix, float scalar) {

cblas\_sscal(N \* N, scalar, matrix, 1);

}

float \*make\_B\_matrix(float \*transposed, float \*matrix) {

float max\_row = max\_sum\_row(matrix);

float max\_column = max\_sum\_row(transposed);

matrix\_scalar\_mult(transposed, 1 / (max\_row \* max\_column));

return transposed;

}

void mult\_matrices(float \*res, float \*mat1, float \*mat2) {

memset(res, 0, N \* N \* sizeof(float));

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0, mat1, N,

mat2, N, 0.0, res, N);

}

void sum\_matrices(float \*res, float \*mat1, float \*mat2) {

cblas\_scopy(N \* N, mat1, 1, res, 1);

cblas\_saxpy(N \* N, 1.0, mat2, 1, res, 1);

}

float \*make\_R\_matrix(float \*I, float \*B, float \*matrix) {

float \*R = calloc(N \* N, sizeof(float));

if (!R)

return NULL;

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

mult\_matrices(tmp, B, matrix);

matrix\_scalar\_mult(tmp, -1);

sum\_matrices(R, I, tmp);

free(tmp);

return R;

}

float \*find\_inverse\_matrix(float \*matrix) {

float \*result = make\_I\_matrix();

float \*transposed = transpose(matrix);

float \*I = make\_I\_matrix();

float \*B = make\_B\_matrix(transposed, matrix);

float \*R = make\_R\_matrix(I, B, matrix);

float \*Rn = calloc(N \* N, sizeof(float));

if (!Rn)

return NULL;

memcpy(Rn, R, N \* N \* sizeof(float));

float \*tmp = calloc(N \* N, sizeof(float));

if (!tmp)

return NULL;

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

sum\_matrices(result, result, Rn);

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

mult\_matrices(tmp, Rn, R);

memcpy(Rn, tmp, N \* N \* sizeof(float));

}

mult\_matrices(tmp, result, B);

memcpy(result, tmp, N \* N \* sizeof(float));

free(I);

free(B);

free(R);

free(Rn);

free(tmp);

return result;

}

int main(void) {

srand(time(NULL));

struct timespec start, end;

float \*matrix = make\_matrix();

float \*inverse\_matrix;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

inverse\_matrix = find\_inverse\_matrix(matrix);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

printf("%f %f\n\n", inverse\_matrix[0], inverse\_matrix[N \* N - 1]);

printf("Preformance time: %lf sec\n",

end.tv\_sec - start.tv\_sec +

0.000000001 \* (end.tv\_nsec - start.tv\_nsec));

free(matrix);

free(inverse\_matrix);

return 0;

}