###### 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, 22202 группы

**Бальчинова А.С.**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

В.А. Перепёлкин

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc18443921)

[ЗАДАНИЕ 3](#_Toc18443922)

[ОПИСАНИЕ РАБОТЫ](#_Toc18443923) 3-11

[ЗАКЛЮЧЕНИЕ 12](#_Toc18443924)

# 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, 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.

# 

# ОПИСАНИЕ РАБОТЫ

1. Реализация программ на языке программирования C++.

**Вариант без ручной векторизации**

#include <iostream>

#include <chrono>  
#include <climits>  
using namespace std;  
  
void add(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] + B[tmp + j];  
 }  
 }  
}  
  
void sub(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] - B[tmp + j];  
 }  
 }  
}  
  
void transposition(const float \*source, float \*res, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 res[i \* n + j] = source[j \* n + i];  
 }  
 }  
}  
  
void mult(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 float \*c = C + i \* n;  
 for (int j = 0; j < n; j++) {  
 c[j] = 0;  
 }  
 for (int k = 0; k < n; k++) {  
 const float \*b = B + k \* n;  
 const float a = A[i \* n + k];  
 for (int j = 0; j < n; j++) {  
 c[j] += a \* b[j];  
 }  
 }  
 }  
}  
  
float max1(const float \*matrix, const int n) {  
 float max1 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[i\*n + j];  
 }  
 max1 = max(sum, max1);  
 }  
 return max1;  
}  
  
float max2(const float \*matrix, const int n) {  
 float max2 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[j\*n + i];  
 }  
 max2 = max(sum, max2);  
 }  
 return max2;  
}  
  
void div(const float \*source, float \*res, const float k, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 res[tmp + j] = source[tmp + j] / k;  
 }  
 }  
}  
  
void identityMatrixInit(float \*matrix, const int n) {  
 memset(matrix, 0, sizeof(float) \* n \* n);  
 for (int i = 0; i < n; i++) {  
 matrix[i \* n + i] = 1;  
 }  
}  
  
void func(float \*source, float \*res, const int n, const int m) {  
 auto \*b = new float[n \* n];  
 auto \*sourceTransp = new float[n \* n];  
 auto \*identityMatrix = new float[n \* n];  
 auto \*r = new float[n \* n];  
 auto \*tmp = new float[n \* n];  
 transposition(source, sourceTransp, n);  
 div(sourceTransp, b, max1(source, n) \* max2(source, n), n);  
 identityMatrixInit(identityMatrix, n);  
 mult(b, source, tmp, n);  
 sub(identityMatrix, tmp, r, n);  
 memset(res, 0, sizeof(float) \* n \* n);  
 add(res, identityMatrix, res, n);  
 add(res, r, res, n);  
 float \*rPow = tmp;  
 tmp = sourceTransp;  
 memcpy(rPow, r, sizeof(float) \* n \* n);  
 for (int i = 2; i < m; i++) {  
 mult(r, rPow, tmp, n);  
 memcpy(rPow, tmp, sizeof(float) \* n \* n);  
 add(res, rPow, res, n);  
 }  
 mult(res, b, tmp, n);  
 memcpy(res, tmp, sizeof(float) \* n \* n);  
 delete[] b;  
 delete[] identityMatrix;  
 delete[] r;  
 delete[] tmp;  
 delete rPow;  
}  
  
void printMatrix(float \*matrix, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 cout << matrix[i \* n + j] << " ";  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
int main() {  
 constexpr int n = 2048;  
 auto \*a = new float[n \* n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if (i == j) {  
 a[i \* n + j] = 2;  
 }  
 else {  
 a[i \* n + j] = 1;  
 }  
 }  
 }  
 auto \*aInverted = new float[n \* n];  
 auto begin = chrono::high\_resolution\_clock::now();  
 func(a, aInverted, n, 10);  
 auto end = chrono::high\_resolution\_clock::now();  
 cout << "Time = " << chrono::duration\_cast<chrono::seconds> (end - begin).count() << " seconds" << endl;  
 delete[] a;  
 delete[] aInverted;  
}

**Вариант с ручной векторизацией (встроенные функции компилятора)**

#include <iostream>

#include <chrono>  
#include <climits>  
#include <intrin.h>  
using namespace std;  
  
void mult(float \*A, float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; ++i) {  
 float \*c = C + i \* n;  
 for (int j = 0; j < n; j += 8) {  
 \_mm256\_storeu\_ps(c + j + 0, \_mm256\_setzero\_ps());  
 }  
 for (int k = 0; k < n; ++k) {  
 const float \*b = B + k \* n;  
 \_\_m256 a = \_mm256\_set1\_ps(A[i\*n + k]);  
 for (int j = 0; j < n; j += 8)  
 \_mm256\_storeu\_ps(c + j + 0, \_mm256\_fmadd\_ps(a, \_mm256\_loadu\_ps(b + j + 0), \_mm256\_loadu\_ps(c + j + 0)));  
 }  
 }  
}  
  
void add(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] + B[tmp + j];  
 }  
 }  
}  
  
void sub(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] - B[tmp + j];  
 }  
 }  
}  
  
void transposition(const float \*source, float \*res, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 res[i \* n + j] = source[j \* n + i];  
 }  
 }  
}  
  
float max1(const float \*matrix, const int n) {  
 float max1 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[i\*n + j];  
 }  
 max1 = max(sum, max1);  
 }  
 return max1;  
}  
  
float max2(const float \*matrix, const int n) {  
 float max2 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[j\*n + i];  
 }  
 max2 = max(sum, max2);  
 }  
 return max2;  
}  
  
void div(const float \*source, float \*res, const float k, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 res[tmp + j] = source[tmp + j] / k;  
 }  
 }  
}  
  
void identityMatrixInit(float \*matrix, const int n) {  
 memset(matrix, 0, sizeof(float) \* n \* n);  
 for (int i = 0; i < n; i++) {  
 matrix[i \* n + i] = 1;  
 }  
}  
  
void func(float \*source, float \*res, const int n, const int m) {  
 auto \*b = new float[n \* n];  
 auto \*sourceTransp = new float[n \* n];  
 auto \*identityMatrix = new float[n \* n];  
 auto \*r = new float[n \* n];  
 auto \*tmp = new float[n \* n];  
 transposition(source, sourceTransp, n);  
 div(sourceTransp, b, max1(source, n) \* max2(source, n), n);  
 identityMatrixInit(identityMatrix, n);  
 mult(b, source, tmp, n);  
 sub(identityMatrix, tmp, r, n);  
 memset(res, 0, sizeof(float) \* n \* n);  
 add(res, identityMatrix, res, n);  
 add(res, r, res, n);  
 float \*rPow = tmp;  
 tmp = sourceTransp;  
 memcpy(rPow, r, sizeof(float) \* n \* n);  
 for (int i = 2; i < m; i++) {  
 mult(r, rPow, tmp, n);  
 memcpy(rPow, tmp, sizeof(float) \* n \* n);  
 add(res, rPow, res, n);  
 }  
 mult(res, b, tmp, n);  
 memcpy(res, tmp, sizeof(float) \* n \* n);  
 delete[] b;  
 delete[] identityMatrix;  
 delete[] r;  
 delete[] tmp;  
 delete rPow;  
}  
  
void printMatrix(float \*matrix, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 cout << matrix[i \* n + j] << " ";  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
int main() {  
 constexpr int n = 2048;  
 auto \*a = new float[n \* n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if (i == j) {  
 a[i \* n + j] = 2;  
 }  
 else {  
 a[i \* n + j] = 1;  
 }  
 }  
 }  
 auto \*aInverted = new float[n \* n];  
 chrono::high\_resolution\_clock::time\_point begin = chrono::high\_resolution\_clock::now();  
 func(a, aInverted, n, 10);  
 chrono::high\_resolution\_clock::time\_point end = chrono::high\_resolution\_clock::now();  
 cout << "Time difference = " << chrono::duration\_cast<chrono::seconds> (end - begin).count() << "[s]" << endl;  
 delete[] a;  
 delete[] aInverted;  
}

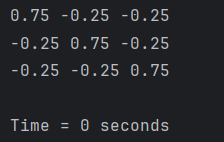
**Вариант с использованием оптимизированной библиотеки BLAS**

#include <iostream>

#include <chrono>  
#include <climits>  
#include <cblas.h>  
using namespace std;  
  
void mult(float \*A, float \*B, float \*C, const int n) {  
 cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, n, n, n, 1.0, A, n, B, n, 0.0, C, n);  
}  
  
void add(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] + B[tmp + j];  
 }  
 }  
}  
  
void sub(const float \*A, const float \*B, float \*C, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 C[tmp + j] = A[tmp + j] - B[tmp + j];  
 }  
 }  
}  
  
void transposition(const float \*source, float \*res, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 res[i \* n + j] = source[j \* n + i];  
 }  
 }  
}  
  
float max1(const float \*matrix, const int n) {  
 float max1 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[i\*n + j];  
 }  
 max1 = max(sum, max1);  
 }  
 return max1;  
}  
  
float max2(const float \*matrix, const int n) {  
 float max2 = INT\_MIN;  
 for (int i = 0; i < n; i++) {  
 float sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += matrix[j\*n + i];  
 }  
 max2 = max(sum, max2);  
 }  
 return max2;  
}  
  
void div(const float \*source, float \*res, const float k, const int n) {  
 for (int i = 0; i < n; i++) {  
 const int tmp = i \* n;  
 for (int j = 0; j < n; j++) {  
 res[tmp + j] = source[tmp + j] / k;  
 }  
 }  
}  
  
void identityMatrixInit(float \*matrix, const int n) {  
 memset(matrix, 0, sizeof(float) \* n \* n);  
 for (int i = 0; i < n; i++) {  
 matrix[i \* n + i] = 1;  
 }  
}  
  
void func(float \*source, float \*res, const int n, const int m) {  
 auto \*b = new float[n \* n];  
 auto \*sourceTransp = new float[n \* n];  
 auto \*identityMatrix = new float[n \* n];  
 auto \*r = new float[n \* n];  
 auto \*tmp = new float[n \* n];  
 transposition(source, sourceTransp, n);  
 div(sourceTransp, b, max1(source, n) \* max2(source, n), n);  
 identityMatrixInit(identityMatrix, n);  
 mult(b, source, tmp, n);  
 sub(identityMatrix, tmp, r, n);  
 memset(res, 0, sizeof(float) \* n \* n);  
 add(res, identityMatrix, res, n);  
 add(res, r, res, n);  
 float \*rPow = tmp;  
 tmp = sourceTransp;  
 memcpy(rPow, r, sizeof(float) \* n \* n);  
 for (int i = 2; i < m; i++) {  
 mult(r, rPow, tmp, n);  
 memcpy(rPow, tmp, sizeof(float) \* n \* n);  
 add(res, rPow, res, n);  
 }  
 mult(res, b, tmp, n);  
 memcpy(res, tmp, sizeof(float) \* n \* n);  
 delete[] b;  
 delete[] identityMatrix;  
 delete[] r;  
 delete[] tmp;  
 delete rPow;  
}  
  
void printMatrix(float \*matrix, const int n) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 cout << matrix[i \* n + j] << " ";  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
int main() {  
 constexpr int n = 2048;  
 auto \*a = new float[n \* n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if (i == j) {  
 a[i \* n + j] = 2;  
 }  
 else {  
 a[i \* n + j] = 1;  
 }  
 }  
 }  
 auto \*aInverted = new float[n \* n];  
 chrono::high\_resolution\_clock::time\_point begin = chrono::high\_resolution\_clock::now();  
 func(a, aInverted, n, 10);  
 chrono::high\_resolution\_clock::time\_point end = chrono::high\_resolution\_clock::now();  
 cout << "Time difference = " << chrono::duration\_cast<chrono::seconds> (end - begin).count() << "[s]" << endl;  
 delete[] a;  
 delete[] aInverted;  
}

|  |  |  |
| --- | --- | --- |
| 2 | 1 | 1 |
| 1 | 2 | 1 |
| 1 | 1 | 2 |

Пример работы программы при n = 3, m = 1000 для матрицы A =

****

1. Запуск программ и результат.

|  |  |  |
| --- | --- | --- |
| **Без ручной векторизации** | **С ручной векторизацией** | **С использованием оптимизированной библиотеки BLAS** |
| **175 секунд** | **41 секунда** | **~0.6 секунд** |

Исходя из полученных значений, можно сделать вывод, что ручная векторизация может значительно ускорить работу программы.

В нашем случае при помощи векторизации функции умножения матриц удалось ускорить работу программы более чем в 4 раза.

В то же время, использование оптимизированной библиотеки BLAS может увеличить скорость выполнения программы в сотни раз.

# ЗАКЛЮЧЕНИЕ

Изучили SIMD-расширения архитектуры x86/x86-64.

Изучили способы использования SIMD-расширений в программах на языке Си/Си++. Получили навыки использования SIMD-расширений.