###### 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, 18205 группы

Гайдамака Андрея Владиславовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А. Ю. Власенко

Новосибирск 2019

ЦЕЛЬ 3

ЗАДАНИЕ 3

ОПИСАНИЕ РАБОТЫ 3

ЗАКЛЮЧЕНИЕ 4

Приложение 1. Код программы 5

# ЦЕЛЬ

1. Изучение SIMD-расширений архитектуры x86/x86-64.
2. Изучение способов использования SIMD-расширений в программах на

языке Си.

1. 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.

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

# ОПИСАНИЕ РАБОТЫ

В ходе лабораторной работы было произведено обращение матриц тремя разными способами: вариант без векторизации, с ручной векторизацией, с помощью библиотеки BLAS. Данные программы были запущенны на сервере и время работы получено для матрицы размеров 1024х1024:

Без векторизации: 227 сек

С SSE: 16 сек

С CBLAS: 1.2 сек

# ЗАКЛЮЧЕНИЕ

В процессе выполнения работы была получена информация о SIMD-расширениях архитектуры x86. SIMD-расширения предназначены для ускорения вычислений однотипных данных. Также было проведено сравнение по времени работы одного и того же алгоритма, реализованного без векторизации на SIMD-расширениях, с ручной векторизацией и с векторизацией средствами библиотеки BLAS. По результатам нетрудно заметить, что самым оптимальным по времени является способ с использованием библиотеки BLAS, а самым читаемым для постороннего человека - без векторизации

**Приложение 1. Код программы**

**#include <cblas.h>**

**#include <time.h>**

**#include <string.h>**

**#include <iostream>**

**#include <stdlib.h>**

**#include <stdio.h>**

**#define SIZE 512**

**#define M 10**

**using namespace std;**

**float maxFirst(float \*matrix) {**

**float sums[SIZE];**

**for (int i = 0; i < SIZE; ++i) {**

**sums[i] = cblas\_sasum(SIZE, matrix + SIZE \* i, 1);**

**}**

**return sums[cblas\_isamax(SIZE, sums, 1)];**

**}**

**float maxInfinity(float \*matrix) {**

**float sums[SIZE];**

**for (int i = 0; i < SIZE; ++i) {**

**sums[i] = cblas\_sasum(SIZE, matrix + i, SIZE);**

**}**

**return sums[cblas\_isamax(SIZE, sums, 1)];**

**}**

**void identity(float\* result){**

**for (int i = 0; i < SIZE; ++i){**

**result[i \* (SIZE + 1)] = 1;**

**}**

**}**

**int main() {**

**float \*matrixA = (float \*)calloc(SIZE \* SIZE, sizeof(float));**

**float \*matrixR = (float \*)calloc(SIZE \* SIZE, sizeof(float));**

**float \*matrixI = (float \*)calloc(SIZE \* SIZE, sizeof(float));**

**float \*matrixM = (float \*)calloc(SIZE \* SIZE, sizeof(float));**

**float \*matrixT = (float \*)calloc(SIZE \* SIZE, sizeof(float));**

**if (matrixA == NULL || matrixR == NULL || matrixI == NULL || matrixM == NULL || matrixT == NULL){**

**printf("MALLOC RETURNED NULL\n");**

**return 1;**

**}**

**float max\_1, max\_inf;**

**for (int i = 0; i < SIZE; ++i) {**

**for (int j = 0; j < SIZE; ++j) {**

**matrixA[i \* SIZE + j] = (float)rand() / RAND\_MAX \* 10;**

**}**

**}**

**clock\_t start, end, start1, end1;**

**start = clock();**

**max\_1 = maxFirst(matrixA);**

**max\_inf = maxInfinity(matrixA);**

**cblas\_sgemm(CblasRowMajor, CblasTrans, CblasNoTrans, SIZE, SIZE, SIZE, 1.0 / (max\_1 \* max\_inf), matrixA, SIZE, matrixA, SIZE, 0, matrixR, SIZE);**

**identity(matrixI);**

**cblas\_saxpy(SIZE \* SIZE, (float)-1, matrixI, 1, matrixR, 1);**

**memcpy(matrixM, matrixR, SIZE \* SIZE \* sizeof(float));**

**start1 = clock();**

**for (int i = 0; i < M; ++i) {**

**cblas\_saxpy(SIZE \* SIZE, (float)1, matrixM, 1, matrixI, 1);**

**cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, SIZE, SIZE, SIZE, (float)1, matrixM, SIZE, matrixR, SIZE, 0, matrixT, SIZE);**

**matrixM = matrixT;**

**}**

**end1 = clock() - start1;**

**cblas\_sgemm(CblasRowMajor, CblasTrans, CblasNoTrans, SIZE, SIZE, SIZE, (float)1, matrixA, SIZE, matrixI, SIZE, 0, matrixT, SIZE);**

**end = clock() - start;**

**double circle = (double) end1 / CLOCKS\_PER\_SEC;**

**double all = (double) end / CLOCKS\_PER\_SEC;**

**cout.precision(6);**

**cout << "Time of circle (CBLAS): " << circle << endl;**

**cout << "All time (CBLAS): " << all << endl << endl;**

**free(matrixA);**

**free(matrixR);**

**free(matrixI);**

**free(matrixM);**

**return 0;**

**}**