МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: Фундаментальная информатика и информационные технологии

Магистерская программа: Инженерия программного обеспечения

**ОТЧЕТ**

по лабораторной работе

**Оптимизация нахождения произведения элементов вектора**

Допущена к защите  **Выполнил:**

студент группы 381706-1м

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пипикин О. И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись подпись

**Проверил:**

к.т.н., доцент кафедры МОСТ Мееров И. Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Рецензент:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc535927826)

[Цель работы 4](#_Toc535927827)

[1. Постановка задачи 5](#_Toc535927828)

[2. Описание алгоритма нахождения произведения элементов вектора 6](#_Toc535927829)

[3. Этапы оптимизации 7](#_Toc535927830)

[3.1 Этап 1. Выбор компилятора 7](#_Toc535927831)

[3.2 Этап 2. Оптимизация алгоритма и кэш памяти 7](#_Toc535927832)

[3.3 Этап 3. Векторизация. Параллелизм на уровне данных. AVX инструкции 8](#_Toc535927833)

[3.4 Этап 4. Параллельное вычисление 8](#_Toc535927834)

[4. Результат оптимизации 10](#_Toc535927835)

[Заключение 11](#_Toc535927836)

Введение

Данная работа посвящена исследованию, реализации и оптимизации алгоритма нахождения произведения элементов одномерного вектора (массива).

Цель работы

Требуется:

1. Реализовать тривиальный алгоритм произведения элементов вектора.
2. Провести оптимизационный анализ. Найти способы улучшения работы тривиального алгоритма.
3. Оптимизировать тривиальный алгоритм.
4. Провести эксперименты с использованием реализованных алгоритмов с последующим получением временных результатов исполнения.
5. Постановка задачи

Исходные данные:

Вектор А размера N, заполенный случайными элементами .

Требуется:

Реализовать оптимальный алгоритм нахождения произведения элементов вектора

1. Описание алгоритма нахождения произведения элементов вектора

Пусть дан вектор размера , заполненный случайными элементами. Тогда, результатом произведения его элементов будет считаться res, вычисляемый следующим образом:

res = 1;

    for (i = 0; i < N; i++) {

        res \*= A[i];

    }

1. Этапы оптимизации

## 3.1 Этап 1. Выбор компилятора

Реализация данного алгоритма проводилась в IDE VisualStudio17 под OS Windows10

Аппаратная часть: процессор Intel® Core™ i7-8700k CPU @3.7GHz с уровнями кэша: .

Сравнение времени выполнения тривиального алгоритма в зависимости от компилятора (в мкс):



Несмотря на лучшие результаты, показываемые компилятором Microsoft, выбор пал на Intel C/C++ Compiler 18.0, тк при дальнейшей оптимизации компилятор Intel показывает лучшие результаты.

## 3.2 Этап 2. Оптимизация алгоритма и кэш памяти

Для дополнительного ускорения необходимо несколько модифицировать алгоритм. К примеру, исключить лишние обращения к функции получения размера массива, исключить лишние операции разыменования указателей и т.д.

Известно, что память хранится и передается блоками. Даже если нужна часть блока, передается весь блок. Самый быстрый способ обращения к данным – “читать” память последовательно, а не в случайном порядке. В данном случае память уже уже считывается последовательно, никакой дополнительной оптимизации не требуется.

Для повышения эффективности загрузки данных из памяти в регистры и выгрузки из регистров в память выделенная в программе память должна быть выровнена по границе в 64 байта (архитектура x64).

Сравнение тривиального алгоритма с оптимизированным и выровненными данными:



## 3.3 Этап 3. Векторизация. Параллелизм на уровне данных. AVX инструкции

За один такт процессор выполняет одну инструкцию (сложение, умножения, запись в регистр) применительно к одному элементу матрицы. В случае с числом с плавающей точкой это 8 байт на x64. Для повышения эффективности необходимо использовать векторизацию, позволяющую за одну инструкцию выполнить сразу несколько операция над 64 байтами данных (AVX инструкции), т.е. над 4 элементами вектора. Так же желательно использовать выравнивание по 64 байта.

Сравнение векторизованной версии алгоритма с версией тривиального алгоритма:



## 3.4 Этап 4. Параллельное вычисление

Каждый элемент вектора A умножается независимо от других. Данный факт сказывается на времени выполнении алгоритма. Т.к целевая платформа поддерживает параллельные вычисления, то для оптимизации алгоритма необходимо  распределить вычислительную нагрузку перемножение элементов независимо. В данной лабораторной работе это достигается использованием технологии OpenMP (Open Multi-Processing).

Сравнение параллельной версии (OpenMP) алгоритма с версией тривиального алгоритма:



Были задействованы все физические ядра процессора.

Далее для улучшения производительности можно применить векторизацию вычислений.

Сравнение оптимизированной версии алгоритма (Векторизация + OpenMP) с версией тривиального алгоритма:



1. Результат оптимизации

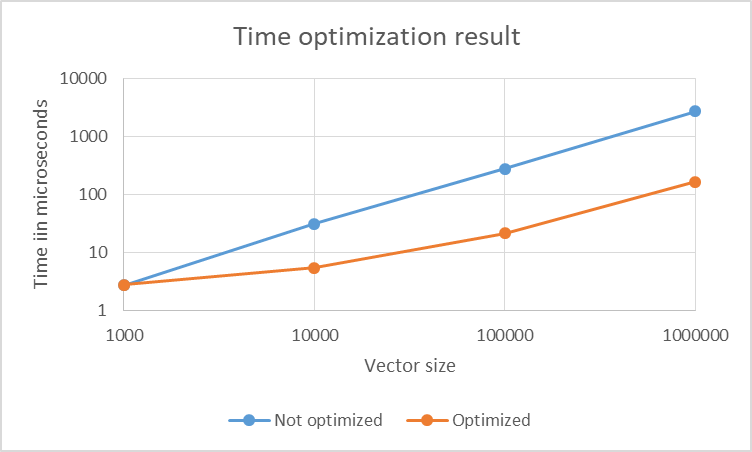


Рис.1 График времен выполнений

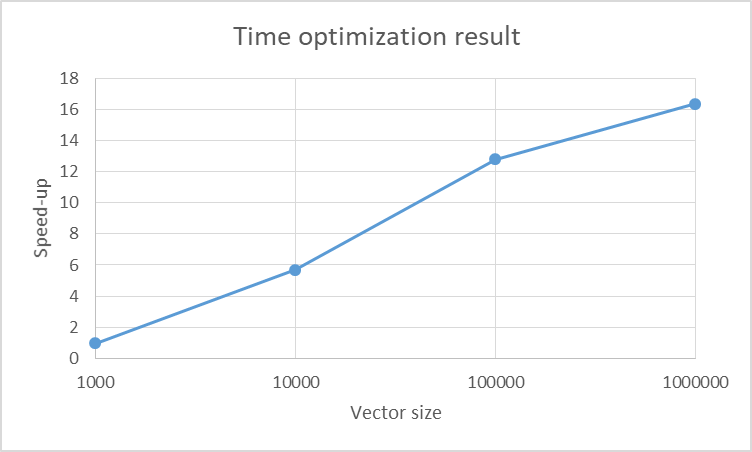


Рис.2 График изменения ускорения в зависимости от размера матриц

Заключение

Все поставленные цели достигнуты. Реализован оптимизированный алгоритм умножения квадратных матриц на языке C/C++ и использованием инструкций AVX и технологии OpenMP. Проведен сравнительный анализ тривиального и оптимизационного алгоритма. Полученные результаты подтверждают улучшение производительности алгоритма (в 16 раз при размере вектора 1000000).