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 алгоритма матричного умножения**

Допущена к защите  **Выполнил:**

студент группы 381706-2м

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шептунов В. О.

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

**Проверил:**

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

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

подпись

Рецензент:

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

ученая степень, ученое звание

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

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc532560347)

[Цель работы 4](#_Toc532560348)

[1. Постановка задачи 5](#_Toc532560349)

[2. Описание алгоритма матричного умножения 6](#_Toc532560350)

[3. Этапы оптимизации 7](#_Toc532560351)

[3.1 Этап 1. Выбор компилятора 7](#_Toc532560352)

[3.2 Этап 2. Оптимизация кэш памяти 7](#_Toc532560353)

[3.3 Этап 3. Векторизация. Параллелизм на уровне данных. AVX инструкции 8](#_Toc532560354)

[3.4 Этап 4. Параллельное вычисление 8](#_Toc532560355)

[4. Результат оптимизации 10](#_Toc532560356)

[Заключение 11](#_Toc532560357)

Введение

Данная работа посвящена исследованию, реализации и оптимизации алгоритма умножения двух квадратных матриц.

Цель работы

Требуется:

1. Реализовать тривиальный алгоритм матричного умножения.
2. Реализовать алгоритм проверки корректности работы матричного умножения.
3. Провести оптимизационный анализ. Найти способы улучшения работы тривиального алгоритма.
4. Оптимизировать тривиальный алгоритм.
5. Провести эксперименты с использованием реализованных алгоритмов с последующим получением временных результатов исполнения.
6. Постановка задачи

Исходные данные:

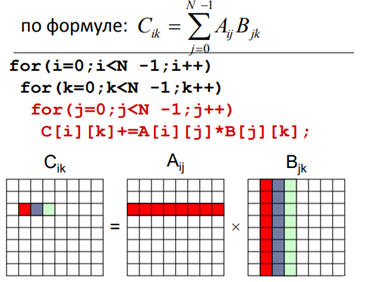
Матрицы и размера

Требуется:

Реализовать оптимальный алгоритм матричного умножения и применить к исходным данным вычислив матрицу размера такую, что

1. Описание алгоритма матричного умножения

Пусть даны две матрицы и размера . Результатом их произведения будет матрица размерностью элементы которой вычисляются



Это так называемый “тривиальный алгоритм”. Далее будет описано как можно его улучшить с точки зрения временных затрат.

1. Этапы оптимизации

## 3.1 Этап 1. Выбор компилятора

Реализация данного алгоритма проводилась в IDE **VisualStudio17** под OS **Windows10**

Аппаратная часть: процессор Intel® Core™ i7-7700U CPU @3.6GHz с уровнями кэша: .

Сравнение времени выполнения тривиального алгоритма в зависимости от компилятора:

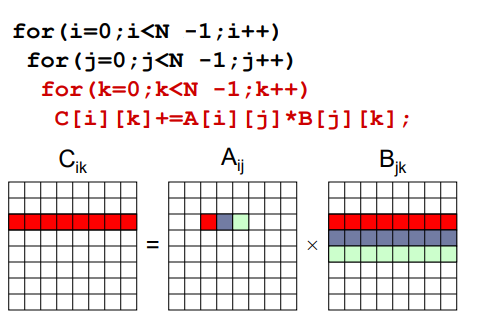


Исходя из целевой платформы и результата эксперимента, выбор пал на

**Intel C/C++ Compiler 19.0**

## 3.2 Этап 2. Оптимизация кэш памяти

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. Для оптимизации необходимо переставить циклы местами:



Сравнение оптимизированной версии алгоритма с версией тривиального алгоритма:



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 алгоритмов с 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 8 элементами матрицы за раз. Так же желательно использовать выравнивание по 64 байта.

Сравнение векторизованной версии алгоритма с версией тривиального алгоритма:



## 3.4 Этап 4. Параллельное вычисление

Каждый элемент матрицы вычисляется независимо от других и матрицы и не изменяются. Данный факт сказывается на времени выполнении алгоритма. Т.к целевая платформа поддерживает параллельные вычисления, то для оптимизации алгоритма необходимо  распределить вычислительную нагрузку на подсчет элементов матрицы . В данной лабораторной работе это достигается использованием технологии OpenMP (Open Multi-Processing) с директивой #pragma omp parallel fo

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 (на пике в 37 раз при размерах матриц 1000x1000).