###### -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

«Оптимизация матричных вычислений с помощью векторизации»

студента 2 курса, группы 21204

***Шальнева Тимофея Андреевича***

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель: Доцент, Кандидат технических наук

Власенко Андрей Юрьевич

Новосибирск 2022

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#__RefHeading___Toc18443921)

[ЗАДАНИЕ 3](#__RefHeading___Toc18443922)

[ОПИСАНИЕ РАБОТЫ 4](#__RefHeading___Toc18443923)

[ЗАКЛЮЧЕНИЕ 1](#__RefHeading___Toc18443924)0

[Приложения 1](#__RefHeading___Toc18443925)1

# ЦЕЛЬ

Целью этой практической работы является изучение векторных расширений процессора, применение SIMD-расширений в программах на языке С/C++.

# ЗАДАНИЕ

Написать три варианта программы, реализующей алгоритм из задания: вариант без ручной векторизации, вариант с ручной векторизацией (ассемблерная вставка, встроенные функции  
компилятора, расширение GCC), вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS. Проверить правильность работы программ на нескольких небольших  
тестовых наборах входных данных. Сравнить время работы трех вариантов программы для N=2048, M=10

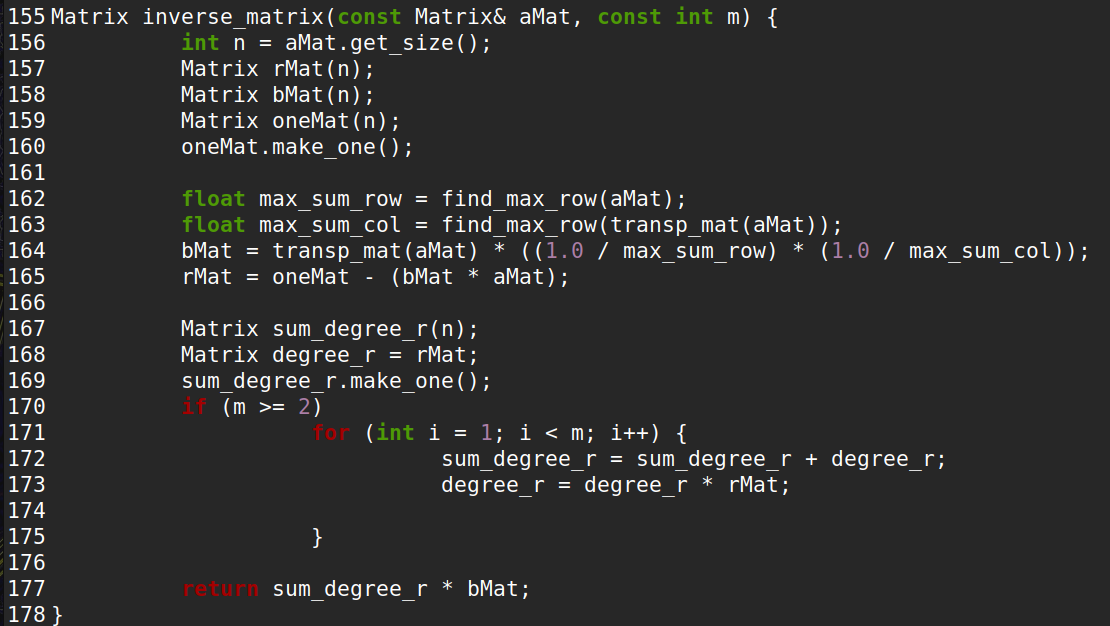
# ОПИСАНИЕ РАБОТЫ

Для программ с отсутствующей и ручной векторизацией был написан общий заголовочный файл matrix.h описывающий класс матрицы, его методы определены в файлах simple.cpp, vectorization.cpp. Программы написаны на языке С++, для векторизации во втором файле использовались ассемблерные вставки.

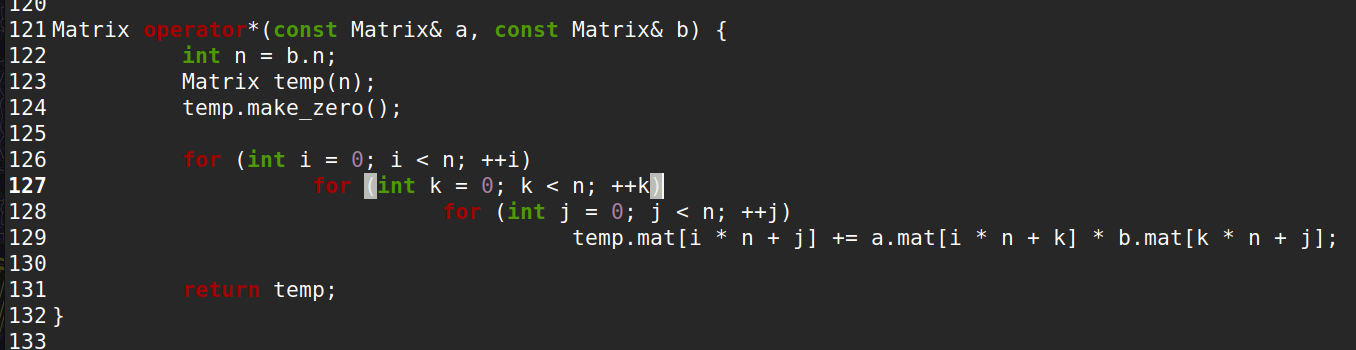
Листинги main.cpp, matrix.h, simple.cpp, vectorization.cpp и usingblas.c находятся в приложениях в конце документа, а также они загружены на [GitHub](https://github.com/extio1/FIT_NSU/tree/main/inverse matrix/src).

**Без векторизации.**

Рассмотрение simple.cpp начнем с функции, которая ищет обратную матрицу.

В ней дословно реализуется алгоритм для поиска обратной матрицы из задачи. Перегруженные операций над объектом матрицы реализованы довольно очевидно. Например метод make\_one() выставляет 1 по диагонали в цикле по размеру матрицы. find\_max\_row суммирует строки и находит максимальную из всех сумм для строк. Вычитание осуществляется проходом по матрицам с применением обычного «-» к данным, лежащим в ячейках, и записью в соответствующую ячейку выходной матрицы.

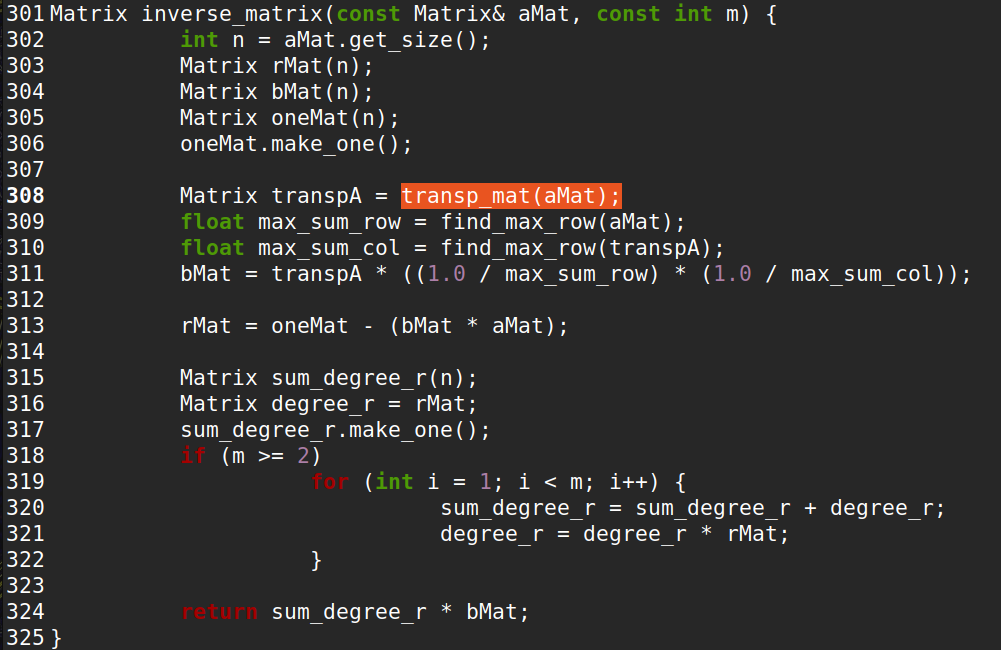
Стоит обратить внимание на реализацию умножения.



Так как двумерные матрицы в языках С/C++ лежат в памяти последовательно, то есть каждая строка матрицы располагается после следующей в памяти, то неэффективно обращаться к элементам матрицы друг за другом по столбцам (что происходит при обычном умножении матриц, когда строка матрицы А скалярно умножается на столбец В). Неэффективность данного способа заключается в плохой работе с кэш-памятью компьютера. Так как при обращении к ячейке массива в кэш память подгружается строка (обычно 64Б, зависит от процессора) данных, она будет состоять из части строки матрицы, содержащей этот элемент и, быть может, элементов следующей строки. Таким образом, при обращении к следующему элементу столбца с большой вероятностью придется грузить новую кеш строку, а значит идти в оперативную память, что занимает много времени по сравнению со скоростью передачи данных из кеша в процессор. В случае, если строка матрицы больше кеш-строки, то подгрузка каждого следующего элемента столбца будет связана с обращением в оперативную память.

Именно для того чтобы решить эту проблему, был изменен порядок циклов. Теперь к каждая матрица читается построчно, что существенно уменьшает количество кеш-промахов.

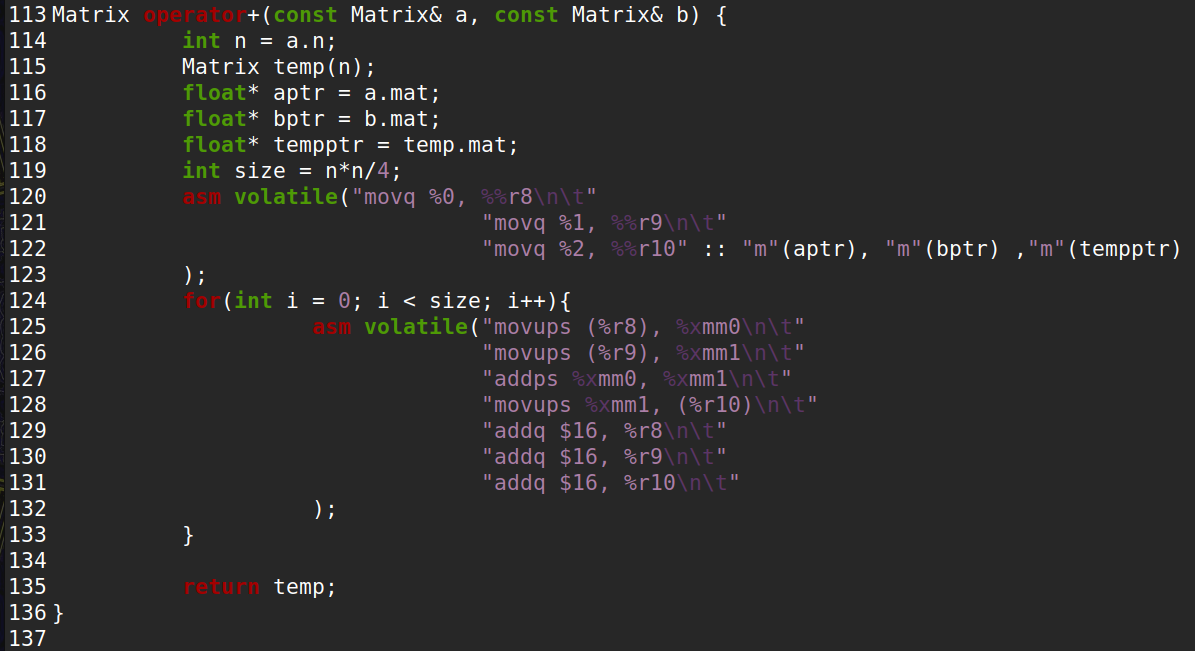
**Ручная векторизация.**

Функция поиска обратной матрицы имеет изменения в 308-311. Теперь транспонированная матрица сохранилась в памяти (308), таким образом избежали обхода матрицы по столбцам в 310-311 строках. Стоит заметить, что в 308 строке применяется перемещающее присваивание, а не копирующее, что затрачивает меньше ресурсов, так как просто переприсваивает указатели на массивы в приватных полях класса.

Ассемблерные вставки используются в функции сложения, вычитания, умножения, транспонирования матриц, присвоения к нулевой и единичной матрице.

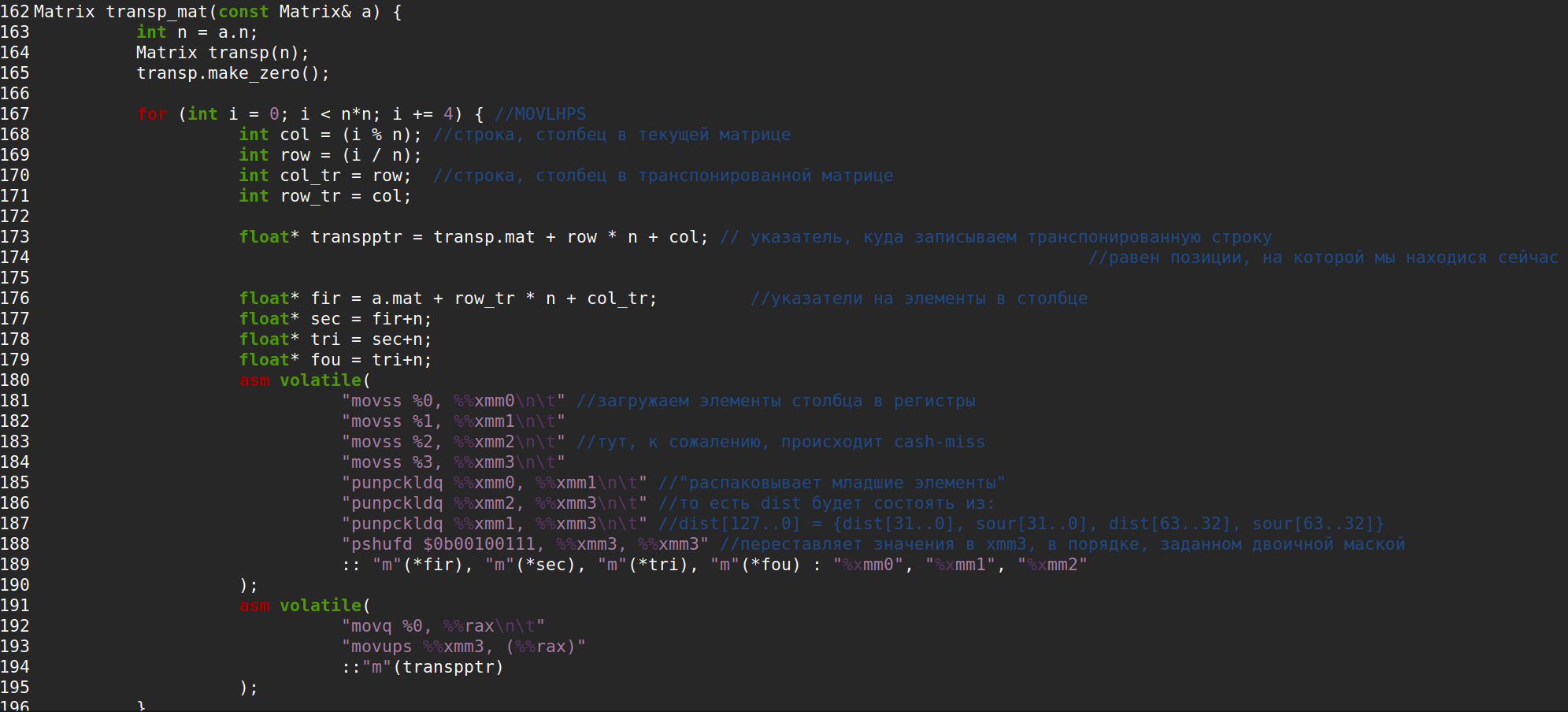
Прокомментируем каждую из них

*Сложение и вычитание.*

В основе функции цикл, по от 0 до n\*n/4 (количество векторов по 4 флоата, которое помещается в матрицу). Перед циклом в регистры r8, r9, r10 загружаются указатели на элементы матриц a, b, temp (она является результатом операции), которые в цикле (128-131 строка) меняются на 16 байт, таким образом перемещаюсь по матрице на 4 элемента вперед. В 125-126 из нужных ячеей матриц-операндов загружаются в векторные регистры xmm расширения sse загружаются 4 элемента, начиная с того, на который указывает aptr, bptr. Следующей командой регистры складываются и потом записываются, с помощью movups в адрес из r10, то есть в матрицу temp.\

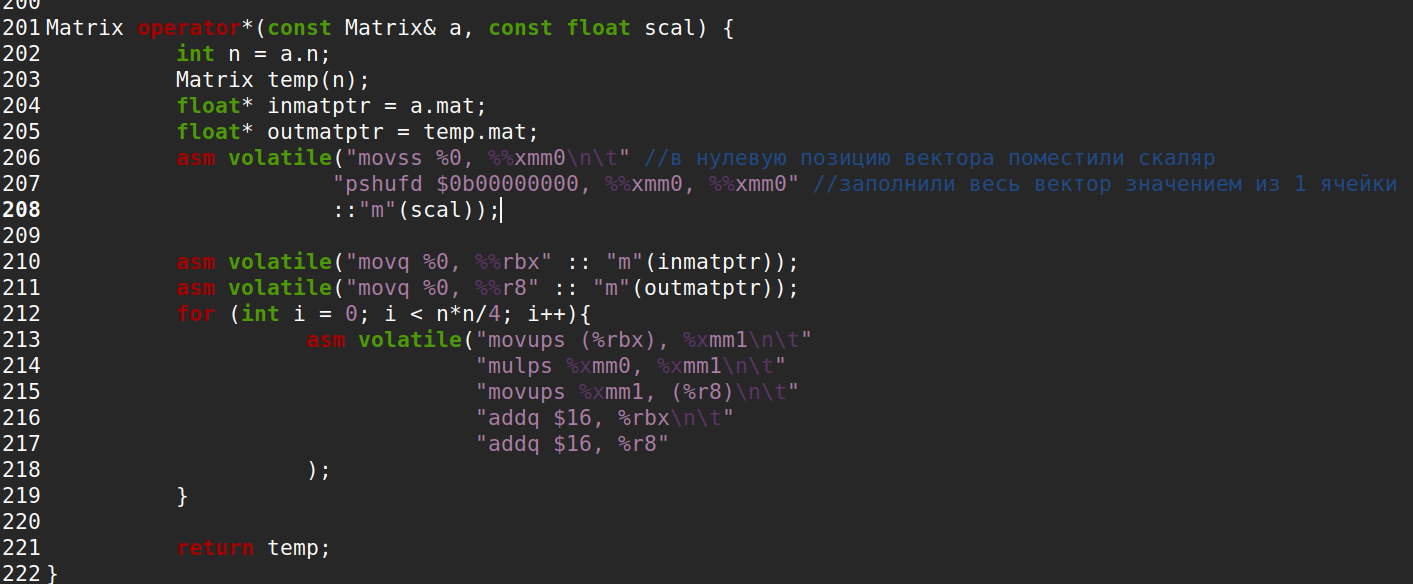
Вычитание отличается лишь командой в 128 строке, а именно всесто addps используется subps.

*Транспонирование матрицы.*

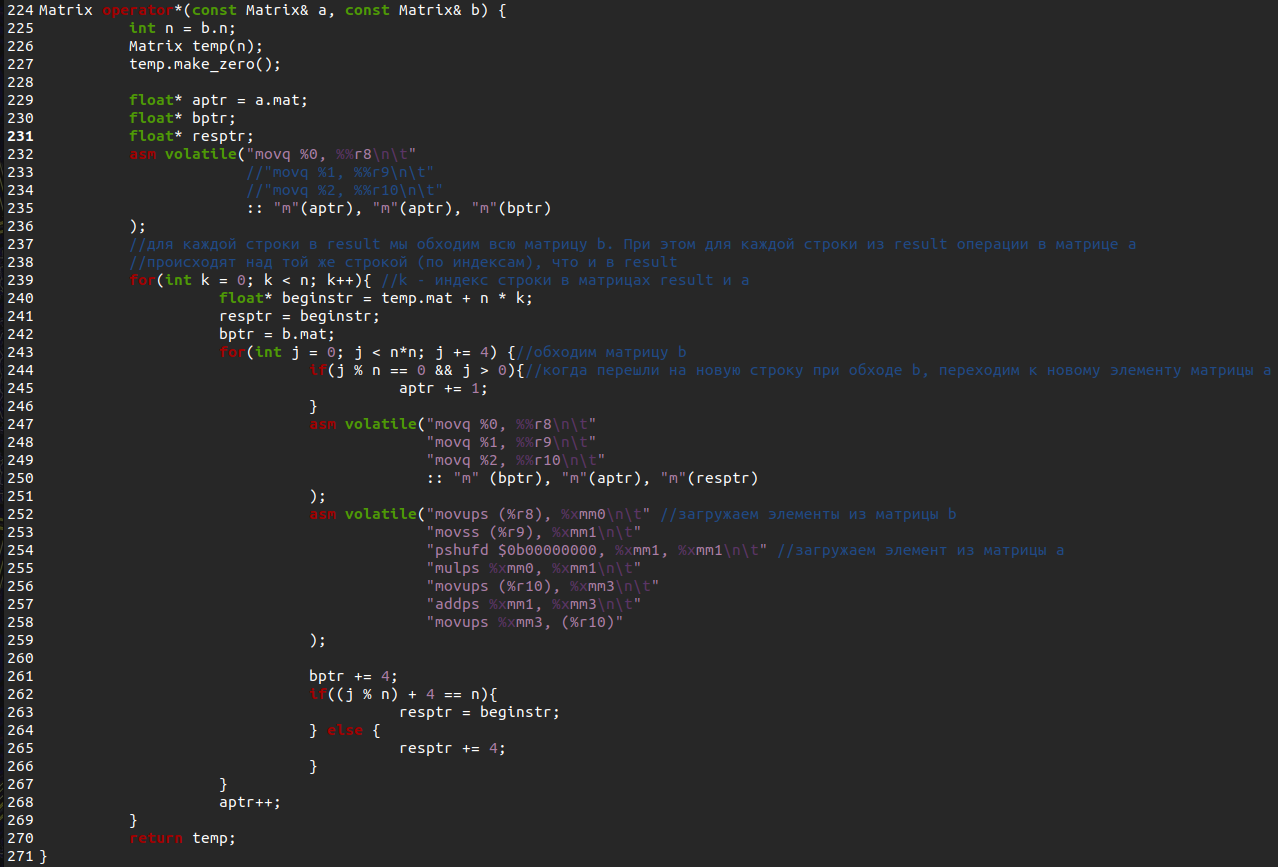
Оптимизация данной операции состоит в том, что за 1 операцию меняются 4 элемента местами (позиция в строке на позицию в столбце, позиция в столбце на позицию в строке). Но при осталась проблема обращения к данным по столбцам, что приводит к cash-miss.

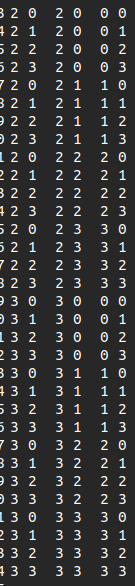
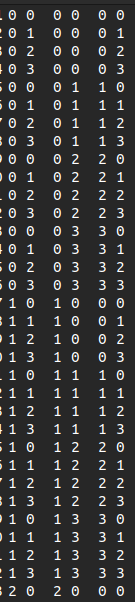
Идейно функция вычисляет позицию в строке, столбце, по ним загружает 4 элемента матрицы и помещает их в позицию с индексами по столбцу, строке.

*Умножение на скаляр.*

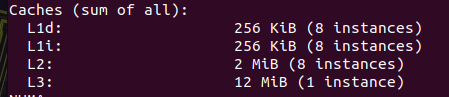
Функция помещает скаляр в вектор и распространяет его на все ячейки. Далее в цикле загружаются по 4 элемента, векторно умножатся на вектор скаляров, записываются в выходную матрицу.

*Умножение на матрицу.*

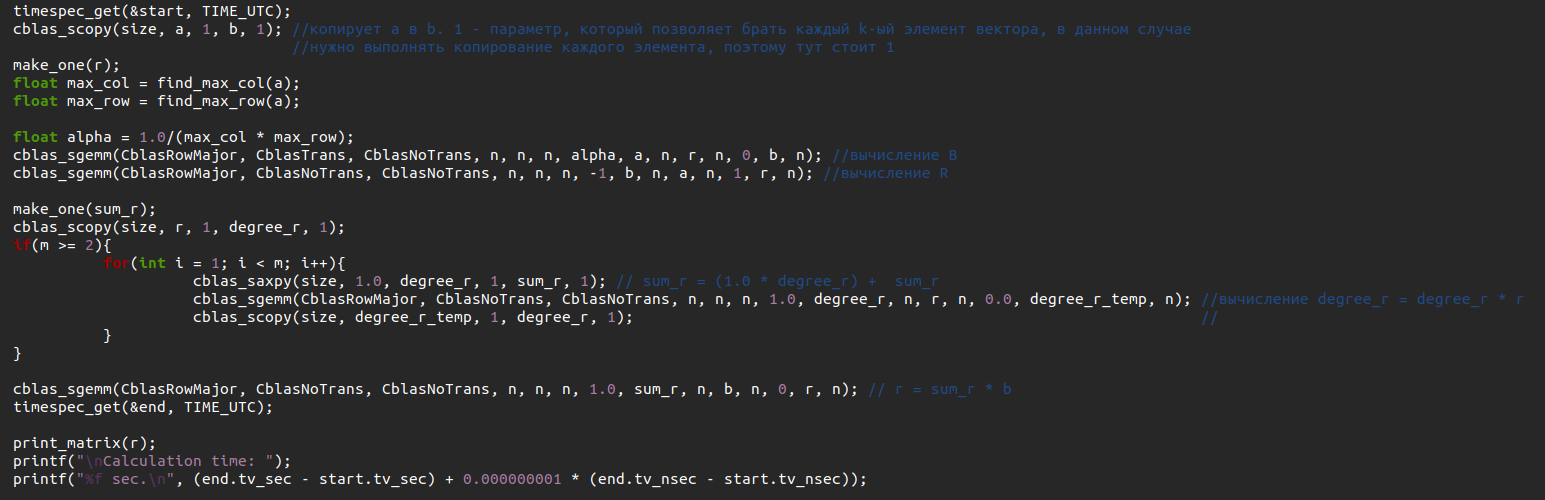
 Для того чтобы написать алгоритм умножения, можно вывести индексы матричного умножения из программы simple.cpp и заметить закономерность в обращении к матрицам. Именно такая логика обращения к ячейкам матрицы и реализована в этой функции.

 Пример: по таким индексам происходит обращения в к ячейкам в матрицах А, B, RESULT для перемножения матриц 4 на 4.

Видно, что матрица B проходится много раз по строкам целиком. Возможно ли, что матрица В влезет в кеш? Размер матрицы в мегабайтах 2048\*2048\*4/1024/1024 = 16Мбайт. Размер кэшей для Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz



**Векторизация с использованием библиотеки BLAS.**

Для вычисления использовались функции

|  |  |
| --- | --- |
| cblas\_sgemm(Order, TransA, TransB, m,n,k,α, &A, lda, &B, ldb, β, &C, ldc) | C = αAB + βC |
| cblas\_scopy(size, &A, incX, &B, incB) | Копирует из А в В |
| cblas\_saxpy(size, α, &A, incX, &B, incB) | αA + B |

\*incX — через сколько элементов вектора читать элементы.

**Замеры времени**

Программы компилировались следующими командами:







Программы тестировались на матрице размера 2048, с параметром m = 10. Матрица есть на [ЯД](https://disk.yandex.ru/d/Wb3ZdX1HaSszMg).

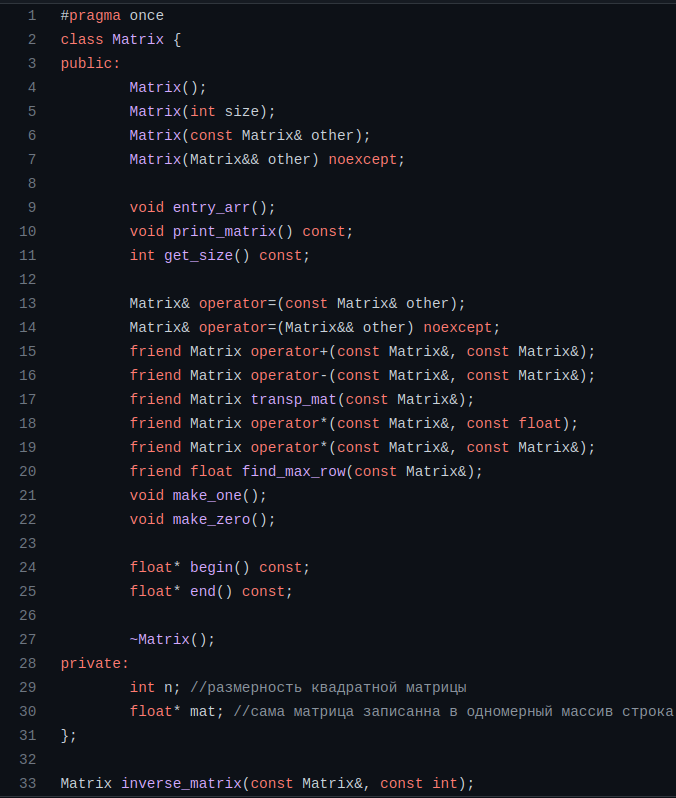
Результаты измерения таковы:

|  |  |
| --- | --- |
| Без векторизации | 304,175с |
| Ручная векторизация | 107.881с |
| BLAS | 0.474072c |

# ЗАКЛЮЧЕНИЕ

За время выполнения данной работы были изучены и применены на практике методы оптимизации вычислений с помощью векторизации. По итогам измерений скорости исполнения программ можно сделать вывод, что для того, чтобы написать программу не просто считающую что-либо правильно, но еще и быстро, нужно затратить дополнительные усилия и взглянуть на привычные алгоритмы под другим углом. Для практического результата можно использовать готовые решения для оптимизации некоторых операций. Например такие инструменты как библиотека BLAS, представляющая набор готовых и очень быстрых функций. В данной работе, в целях изучения в деталях работы с векторными операциями, были выбраны ассемблерные вставки. Такой выбор повлек, помимо изучения новых векторный команд, получение опыта в написании частей программы на языке Ассемблера. И хоть в повседневной жизни многим может и не придется никогда писать программы на Ассемблере, это умение поможет разобраться в ассемблерном листинге, что позволит рассмотреть программу в мельчайших деталях.

# Приложение 1. Заголовочный файл matrix.h



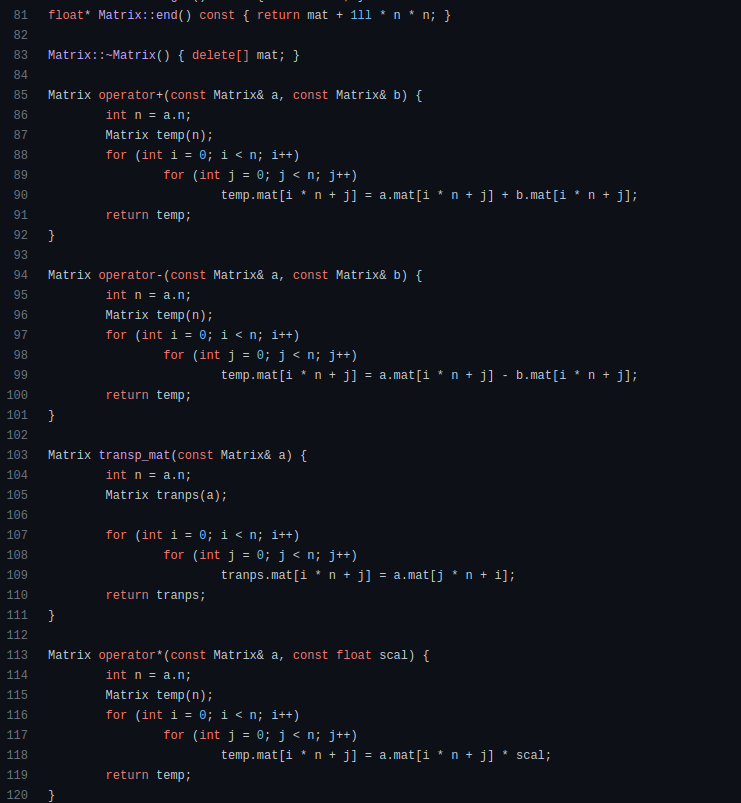
# Приложение 2. Без векторизации — simple.cpp

****

# Приложение 2 (продолжение)

****

# Приложение 2 (продолжение)



# Приложение 2 (продолжение)

# Приложение 2 (продолжение)

# Приложение 3 Ручная векторизация, с использованием ассемблерных вставок

# Приложение 3. (продолжение)

# Приложение 3. (продолжение)

# Приложение 3. (продолжение)

# Приложение 3. (конец)

# Приложение 4. Использование специальной библиотеки линейной алгебры для нахождения обратной матрицы.

# Приложение 4. (продолжение)