Отчёт по лабораторным работам 15-16

Корнеенко Егор, СКБ-231

**Вступление:**

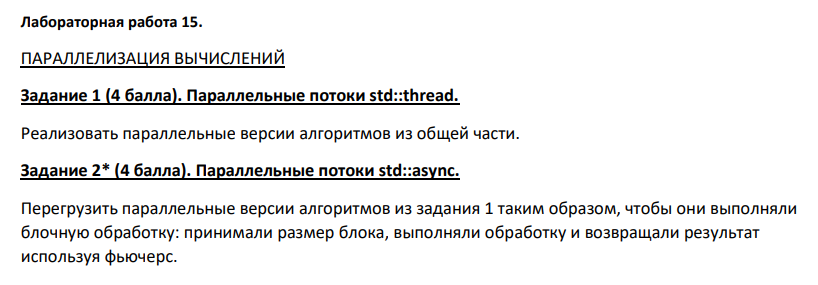
В лабораторной работе 15 я реализовал параллельные (асинхронные) версии алгоритмов класса “Matrix” с помощью стандартных библиотек std::thread и std::async. В лабораторной работе 16 измерил быстродействие этих алгоритмов, построил теоретическую оценку их быстродействия, а также сверил теоретические значения с практическими

**Фоновая музыка:**

Список треков, которые помогли мне выполнить эти лабораторные работы.

* [Adventure of a lifetime](https://music.yandex.ru/album/6349775/track/44523326)
* [V](https://music.yandex.ru/album/4446094/track/35587527)olga
* [Т](https://music.yandex.ru/album/3898328/track/32038112)ебе в прикол
* [X](https://music.yandex.ru/album/21173434/track/100727934)XX
* [SnD](https://music.yandex.ru/album/3332468/track/27898785)
* [4х4](https://music.yandex.ru/album/28255531/track/119359229)
* [Location](https://music.yandex.ru/album/4275374/track/34652974)

**Условия:**

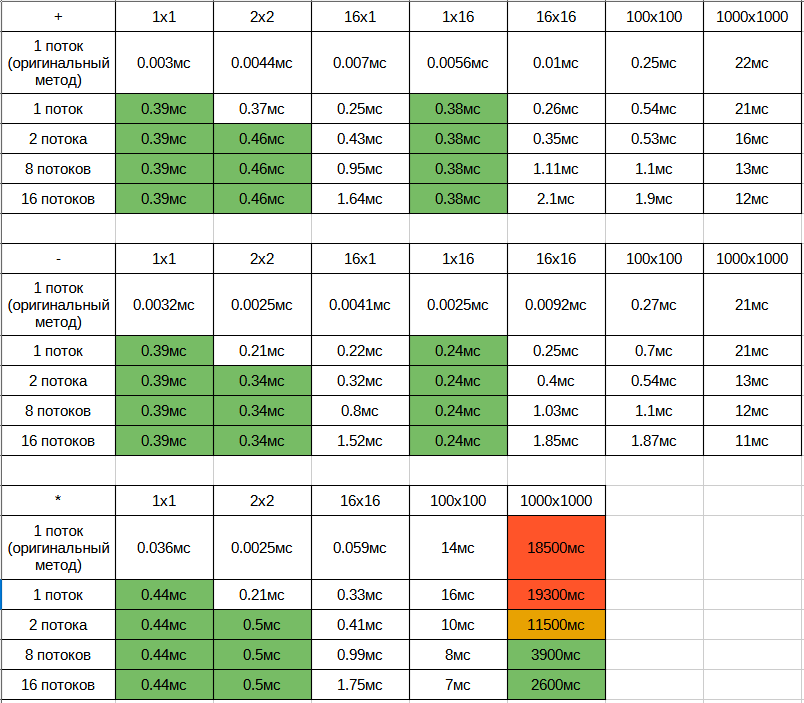
****

**Лабораторная работа 15:**

1. Для реализации параллельных вычислений я выбрал те алгоритмы (методы), которые одинаково действуют сразу на несколько элементов матриц. Этими алгоритмами стали сложение, вычитание и умножение матриц. В этот список не попало вычисление дискриминанта, так как в моём алгоритме внедрение параллельных потоков помешает последовательному сравнению и перестановке строк. Для того, чтобы грамотно использовать ресурсы компьютера, я решил, что каждый поток будет работать со своим определённым числом строк матрицы.
2. Количество используемых потоков я определял в соответствии с количеством строк в матрице (если столбцов больше, то я транспонировал матрицы в начале выполнения метода и в конце).
3. Методы add\_parallel, deduct\_parallel и multiply\_parallel я реализовал с помощью заголовочного файла std::thread путём создания стека и включения в него потоков с соответствующими задачами (задачи создавались с помощью лямюда-функций). В конце выполнения метода я «джоинил» потоки, как бы прекращая выполнение ими задач.
4. Для реализации асинхронных версий этих же методов я использовал заголовочный файл std::async, а именно std::future (для выполнения задач) и std::launch::async для запуска потока прямо после того, как мы дали ему задачу.

**Лабораторная работа 16:**

1. Для того, чтобы измерить время между началом и концом выполнения методов, я использовал возможности заголовочного файла std::chrono, а именно std::chrono::high\_resolution\_clock::now().
2. Далее я провёл серию тестов разных методов с разным числом потоков и разным размером/формой матриц. Усреднённые результаты тестов представлены в таблице ниже. Тесты проводились с случайными числами от 1 до 100.



1. Проводя тесты, я пришёл к выводу, что при маленьких размерах матриц распараллеливание вычислений не даёт прибавки к быстродействию программы, зато на масштабных матрицах прирост быстродействия значительный (яркий пример — умножение двух матриц 1000 на 1000).

**Код:**

<https://github.com/egoridze74/cpp-lab-15-16>

[**Заключение:**](https://github.com/egoridze74/cpp-lab-12)

В процессе выполнения этих лабораторных работ я изучил методы реализации многопоточного выполнения программ в С++ с помощью стандартных библиотек, а также изучил, как распараллеливание вычислений увеличивает быстродействие программы.