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
- Volga
- Тебе в прикол
- XXX
- SnD
- 4x4
- Location

Условия:

Лабораторная работа 15.

ПАРАЛЛЕЛИЗАЦИЯ ВЫЧИСЛЕНИЙ

Задание 1 (4 балла). Параллельные потоки std::thread.

Реализовать параллельные версии алгоритмов из общей части.

Задание 2* (4 балла). Параллельные потоки std::async.

Перегрузить параллельные версии алгоритмов из задания 1 таким образом, чтобы они выполняли блочную обработку: принимали размер блока, выполняли обработку и возвращали результат используя фьючерс.

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.

+	1x1	2x2	16x1	1x16	16x16	100x100	1000x1000
1 поток (оригинальный метод)	0.003мс	0.0044мс	0.007мс	0.0056мс	0.01мс	0.25мс	22мс
1 поток	0.39мс	0.37мс	0.25мс	0.38мс	0.26мс	0.54мс	21мс
2 потока	0.39мс	0.46мс	0.43мс	0.38мс	0.35мс	0.53мс	16мс
8 потоков	0.39мс	0.46мс	0.95мс	0.38мс	1.11мс	1.1мс	13мс
16 потоков	0.39мс	0.46мс	1.64мс	0.38мс	2.1мс	1.9мс	12мс

-	1x1	2x2	16x1	1x16	16x16	100x100	1000x1000
1 поток (оригинальный метод)	0.0032мс	0.0025мс	0.0041мс	0.0025мс	0.0092мс	0.27мс	21мс
1 поток	0.39мс	0.21мс	0.22мс	0.24мс	0.25мс	0.7мс	21мс
2 потока	0.39мс	0.34мс	0.32мс	0.24мс	0.4мс	0.54мс	13мс
8 потоков	0.39мс	0.34мс	0.8мс	0.24мс	1.03мс	1.1мс	12мс
16 потоков	0.39мс	0.34мс	1.52мс	0.24мс	1.85мс	1.87мс	11мс

*	1x1	2x2	16x16	100x100	1000x1000
1 поток (оригинальный метод)	0.036мс	0.0025мс	0.059мс	14мс	18500мс
1 поток	0.44мс	0.21мс	0.33мс	16мс	19300мс
2 потока	0.44мс	0.5мс	0.41мс	10мс	11500мс
8 потоков	0.44мс	0.5мс	0.99мс	8мс	3900мс
16 потоков	0.44мс	0.5мс	1.75мс	7мс	2600мс

3. 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:

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