МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Параллельные алгоритмы»
Тема: Основы работы с процессами и потоками

Студент гр. 0303 Парамонов В.В.

Преподаватель Сергеева Е. И.

Санкт-Петербург
2023

Цель работы.

Исследовать работу с потоками и процессами на языке c++. Вывести закономерности, связанные с числом потоков и размером данных.

Постановка задачи.

Лабораторная состоит из 3-х подзадач, которые выполняют одинаковую задачу с использованием процессов или потоков:

a) Выполнить умножение 2-х матриц.
b) Входные матрицы вводятся из файла (или генерируются).
c) Результат записывается в файл.

Сами подзадачи;

1.1. Выполнить задачу, разбив её на 3 процесса. Выбрать механизм обмена данными между процессами.

Процесс 1: заполняет данными входные матрицы (читает из файла или генерирует их некоторым образом).

Процесс 2: выполняет умножение.

Процесс 3: выводит результат.

1.2.1. Аналогично 1.1, используя потоки (*std::threads*).

1.2.2. Разбить умножение на P потоков (“наивным” способом по строкам или столбцам).

Выполнение задач.

1. **Для решения задач была создана программа со следующей структурой**:
   1) В main.cpp происходит выбор основных параметров запуска подзадач, таких как:
      a) число строк и столбцов первой и второй матрицы (fRowsStart, fRowsEnd, fCols, sRows, sCols);
      b) количество перезапусков задачи (numRestarts) для получения усредненного времени выполнения;
      c) для задачи 1.2.2 количество потоков (threadsNum);

d) записывать ли в консоль какие данные в матрицах и результат умножения (printToConsole);
e) путь к файлу для вывода полученной матрицы в ходе умножения (outFileName);
f) Выбор задачи (choice), если равно 1, то задача 1.1, если 2, то задача 1.2.1, если 3, то задача 1.2.2;

Так же в данном файле происходит запуск выбранной задачи с разными значениями строк первой таблицы (параметры fRowsStart, fRowsEnd) и вывод усредненного времени на выполнение задачи.

2) В tasks.h находятся объявление функций, выполняющих задачи лабораторной:
   a) int processesStageSep(int fRows, int fCols, int sRows, int sCols, std::string& outFileName, bool printToConsole), функция выполняет задачу 1.1 лабораторной, в ней от родительского процесса происходит с помощью fork() выделение нескольких дочерних процессов, в качестве механизма обмена данных между процессами было выбрано использование pipe.
   b) int threadsStageSep(int fRows, int fCols, int sRows, int sCols, std::string& outFileName, bool printToConsole), функция выполняет задачу 1.2.1 лабораторной, в ней с помощью std::thread создаются отдельные процессы, которые выполняют разные задачи: инициализацию, умножение, вывод результата.
   c) int threadsMultiply(int fRows, int fCols, int sRows, int sCols, std::string& outFileName, bool printToConsole, int threadsNum), функция выполняет задачу 1.2.1 лабораторной, в ней происходит разделение на потоки процесса умножения двух матриц. Если количество строк первой матрицы не кратно числу потоков, то происходит выход из функции с кодом ошибки.
3) В matrix.h и matrix.cpp реализован класс матрицы, которая может создаваться пустой, когда в конструктор передаются только число строк и

столбцов, а может заполняться случайными числами, если передать так же пределы возможных случайных значений в матрице (так же есть отдельная функция случайной инициализации void randomInit(int rndMin, int rndMax)). Для матрицы реализованы функции:

a) Для вывода матрицы в консоль: void printMatrix().
b) Получение одной строки результата умножения матриц: static std::vector<int> multiplyRow(Matrix& firstMat, Matrix& secondMat, int rowNum).
c) Сериализация и десериализация: int* serialize() и void deserialize(int* buf).
d) Запись матрицы в файл: void writeMatrix(std::string& fileName).
e) Получение числа строк и столбцов: int getRows() и int getCols().
f) Установка одной из строк матрицы на другую: void setRow(int rowNum, std::vector<int>& row).

2. Исследование получаемых с использованием программы результатов:

1) Исследуем скорость работы задач 1.1 и 1.2.1 в зависимости от количества данных (количества строк первой таблицы), результаты измерений времени работы, усредненные для 100 запусков, представлены в таблицах 1 и 2:

Таблица 1 – Измерение времени работы программы из 3-х процессов (1.1)

Размер первой матрицы	Размер второй матрицы	Время выполнения задачи(мкс)
(12,128)	(128,128)	2479
(24,128)	(128,128)	3755
(36,128)	(128,128)	5102
(48,128)	(128,128)	6435
(60,128)	(128,128)	7868
(72,128)	(128,128)	9102
(84,128)	(128,128)	10371
(96,128)	(128,128)	11641
(108,128)	(128,128)	12906

Таблица 2 – Измерение времени работы программы из 3-х потоков (1.2.1)

Размер первой матрицы	Размер второй матрицы	Время выполнения задачи (мкс)
(12,128)	(128,128)	2119
(24,128)	(128,128)	3508
(36,128)	(128,128)	4828
(48,128)	(128,128)	6134
(60,128)	(128,128)	7430
(72,128)	(128,128)	9302
(84,128)	(128,128)	10595
(96,128)	(128,128)	11596
(108,128)	(128,128)	12652

Графики зависимости времени исполнения от размера “данных” (кол-ва столбцов первой матрицы) см. на рисунке 1:

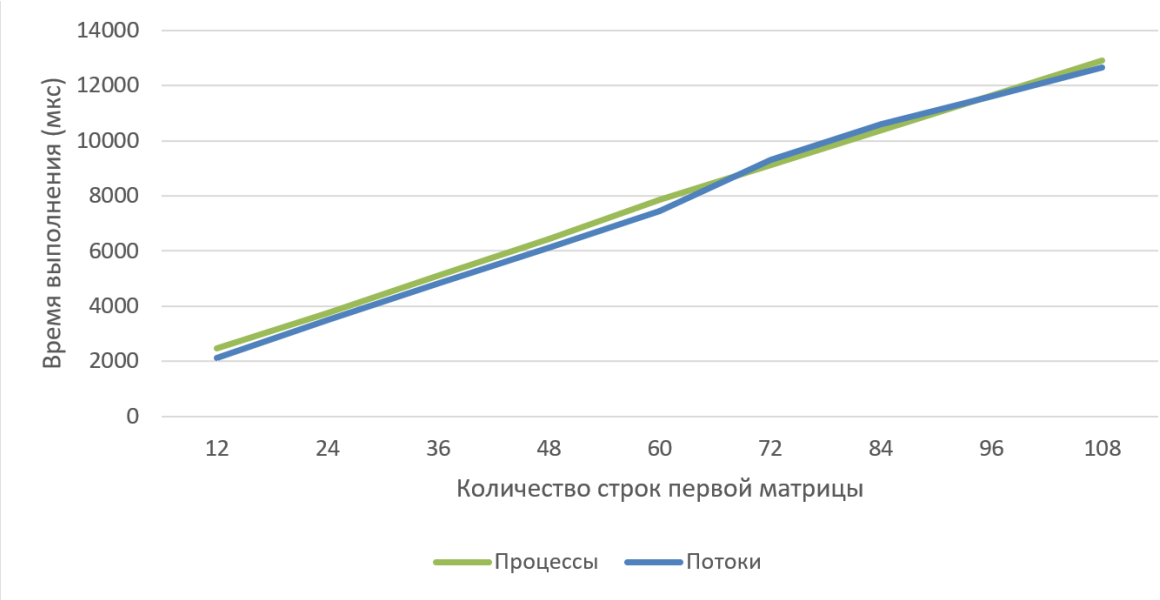

Рисунок 1 – Сравнение времени выполнения для программы на потоках и процессах с одинаковым алгоритмом работы.

Как видно по таблицам и графикам, умножение матрицы на процессах выполняется в среднем чуть медленнее, это связано с тем, что необходимо делать запросы в операционную систему на создание новых процессов и особенности передачи данных между разными процессами.

2) Теперь рассмотрим зависимость времени выполнение задачи 1.2.2 в зависимости от объема входных данных и от кол-ва потоков (см. таблицы 3 и 4):

Таблица 3 - Зависимость времени выполнения задачи многопоточного умножения матриц (1.2.2) от объема входных данных (при 12 потоках)

Размер первой матрицы	Размер второй матрицы	Время выполнения задачи (мкс)
(12,128)	(128,128)	1088
(24,128)	(128,128)	1457
(36,128)	(128,128)	1950
(48,128)	(128,128)	2354
(60,128)	(128,128)	2680
(72,128)	(128,128)	3064
(84,128)	(128,128)	3419
(96,128)	(128,128)	3805
(108,128)	(128,128)	4153
(120,128)	(128,128)	4585

Исходя из таблицы видно линейное увеличение скорости выполнения программы с ростом объема данных.

Таблица 4 - Зависимость времени выполнения задачи 1.2.2 от количества потоков (размер матриц (512,128) и (128,128))

Количество потоков	Время выполнения задачи (мкс)
1	55121
2	31009
4	19715
8	21180
16	**18811**
32	17203
64	17174
128	***17701***
256	***18980***
512	***21317***

Исходя из полученных данных в таблице 4 при увеличении потоков вплоть до 16 время выполнения программы падало (это число примерно соответствует числу физических потоков процессоров компьютера), далее время выполнения находится на примерно одном и том же уровне, до 128 потоков, когда количество потоков становится настолько большим, что они начинают мешать друг другу исполнять задачи из-за постоянного переключения физических потоков между ними.

Заключение.

В ходе работы была изучена работа с потоками и процессами на языке c++. Также были выведены закономерности, связанные с числом потоков и размером данных:

1) При увеличении числа потоков сначала идет рост производительности вплоть до значения близкого к количеству физических потоков процессоров, далее скорость выполнения идет на спад из-за постоянного переключения процессов.
2) При линейном росте объема данных (кол-ва строк одной из таблиц) скорость роста времени выполнения задачи умножения матриц растет линейно.
3) В среднем скорость потоков выше, чем скорость процессов.