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

ОТЧЕТ

по лабораторной работе № 1

по дисциплине «Параллельные алгоритм»

Тема Основы работы с процессами и потоками

Студент гр. 0303 Сологуб Н.А.

Преподаватель Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучить принцип работы процессов и потоков на языке C++.

Задание.

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

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

2
Аналогично 1.1, используя потоки (std::threads)

3
Разбить умножение на P потоков (“наивным” способом по строкам-столбцам).

Выполнение работы.

1.1) Класс Matrix

Был реализован класс Matrix, который генерирует случайным образом матрицу размера N на M. Метод **void multiple(Matrix& second,Matrix& result)** умножает текущий объект на объект матрицы second и результат записывает в объект result. Метод **void multiple(Matrix& second,Matrix& result,int shiftRow,int threadCount)** перемножает матрицу текущего объекта с матрицей объекта second, результат записывает в матрицу result, но при этом совершает

перемножения со смещением относительно начала строк первой матрицы, в зависимости от количества потоков threadCount. Метод **int* toArray()** преобразует матрицу в массив целых чисел для записи данных в файловый дескриптор. Метод **void fromArray(int* buf,int N,int M)** преобразует из переданного массива целых чисел в матрицу. Метод **void fileOutput(char* fileName)** записывает матрицу в файл.

1.2) Класс Process

Был реализован класс Process, который запускает в методе **int ProceedProcess()** Три процесса: создание матриц, умножение матриц и вывод матриц в файл. Создание процессов происходит за счёт функции fork(). Если процесс ребёнка удалось создать, тогда fork() возвращает 0. Процессы ожидаются с помощью функции wait() которая ожидает код возврата процесса.

1.3) Класс Thread

Был реализован класс Thread, который разделяет выполнение программы на потоки. Метод **void Create(Matrix& first, Matrix& second,Matrix& result)** создаёт матрицы с помощью первого потока. Метод **void Multiple(Matrix& first, Matrix& second,Matrix& result,int shiftRow = 0,int threadCount = 1)** перемножает матрицы или в одном потоке или в P потоках в зависимости от выбора пользователя. Метод **void outPut(Matrix& result)** выводит результат умножения матриц в файл в одном потоке.Метод **void proceedThreads(int choice)** последовательно вызывает потоки с помощью thread() и ожидает их выполнения с помощью join().

2) Исследование зависимости между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Рассмотрим зависимость времени выполнения программы от размеров входных данных при фиксированном количестве потоков. Результаты представлены в табл. 1.

Таблица 1 - зависимость времени от входного количества данных

Количество потоков	Размер	Время, с
2	10x10	0.0012327
2	100x100	0.0036992
2	1000x1000	2.10919

Из таблицы видно, что с ростом количества данных растёт и время выполнения программы, что очевидно.

Рассмотрим зависимость времени выполнения программы от количества потоков при фиксированном количестве данных. Результаты представлены в табл. 2.

Количество потоков	Размер	Время
2	100x100	0.0041448
4	100x100	0.0037435
8	100x100	0.0035641
16	100x100	0.0034777
32	100x100	0.0027584

Из таблицы видно, что с ростом количества потоков, выделенных под умножение скорость работы программы линейно увеличивается, но это будет происходить до тех пор, пока количество потоков не превышает количество строк первой матрице, иначе в увеличении потоков не будет смысла.

Выводы.

В процессе выполнения работы были изучены принципы работы процессов и потоков.

Было изучено, что с ростом данных растёт и время выполнения программы. Количество потоков не всегда приводит к росту скорости выполнения ввиду спецификации выполняемой задачи(превышение количества поток над количеством строк не принесёт большего результата). Также существуют ограничения относительно выполнения потоков - умножение не может быть выполнено пока не создадутся матрицы, для которых умножение выполняется.