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

ОТЧЕТ

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

Студент гр. 9304 Ламбин А.В.

Преподаватель Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Изучить основы работы с процессами и потоками в языке программирования C++.

Задание.

Выполнить поэлементное сложение двух матриц $M \times N$. Входные матрицы вводятся из файла или генерируются, результат записывается в файл.

1. Выполнить задачу, разбив её на три процесса. Выбрать механизм обмена данными между процессами. Процесс 1 заполняет данными входные матрицы, процесс 2 выполняет сложение, процесс 3 выводит результат.
2. Выполнить задачу аналогично пункту 1, используя потоки.
3. Разбить сложение на P потоков. Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Выполнение работы.

Для решение поставленной задачи с помощью процессов были созданы классы *SharedMemory*, реализующий API для использования разделяемой памяти; *ReadingProcess*, *CalculationProcess* и *WritingProcess*, имеющие метод *run()*, при вызове которого начинается выполнение действий каждого алгоритма.

Для создания каждого из процесса использовалась функция *fork()*, возвращающая pid процесса. Если его значение равно 0, то текущий процесс является дочерним, а значение отличное от 0 и -1 говорит о родительском процессе. Таким образом, с помощью вложенных друг в друга операторах *switch* были определены текущие процессы, для каждого из которых были созданы экземпляры соответствующих классов и вызваны методы *run()*. После выполнения каждый из родительских процессов дожидался дочерние, после чего сам завершал работу.

Процесс, занимающийся чтением матриц, получает указатель на разделяемую память, после чего последовательно считывает каждую из матриц. Если на каком-то из этапов выясняется некорректность входных данных, переменная *isCrash* становится равной true, что говорит об ошибке и, соответственно, невозможности продолжать вычисления, выводится сообщение об ошибке и выполнение прерывается.

Процесс, занимающийся сложением матриц, получает указатель на разделяемую память, дожидается начала заполнения второй матрицы предыдущим процессом, после чего складывает строки матриц по мере появления новых строк в матрице B.

Процесс, занимающийся выводом, получает указатель на разделяемую память, дожидается момента, когда матрица C начнёт заполняться, после чего выводит строки матрицы по мере их вычисления.

Класс *SharedMemory* содержит приватный указатель на структуру *MemoryBlock*, которая является разделяемой памятью. Для удобства использования были реализованы методы, позволяющие читать и изменять данные, находящиеся в разделяемой памяти.

Для решения поставленной задачи на потоках был реализован класс *Threads*. В методе *run()* создаются три потока, которые могут обращаться к полям класса Threads, читая и изменяя их. Методы, реализующие логику потоков схожи с аналогичными методами, используемыми процессами.

Для вычисления суммы матриц с помощью P потоков были изменены методы, реализующие сложение матриц. В данных методах создаются P потоков, каждый из которых обрабатывает определённое число строк, заданные с помощью лямбда-выражения *fragmentation* и зависящие от количества потоков и индекса потока, обрабатывающего эти строки.

Был написан скрипт на языке программирования Python, вычисляющий зависимости между количеством потоков и временем выполнения программы и между размерами входных данных и временем выполнения программы. Результаты вычислений представлены на рисунках 1 – 2.

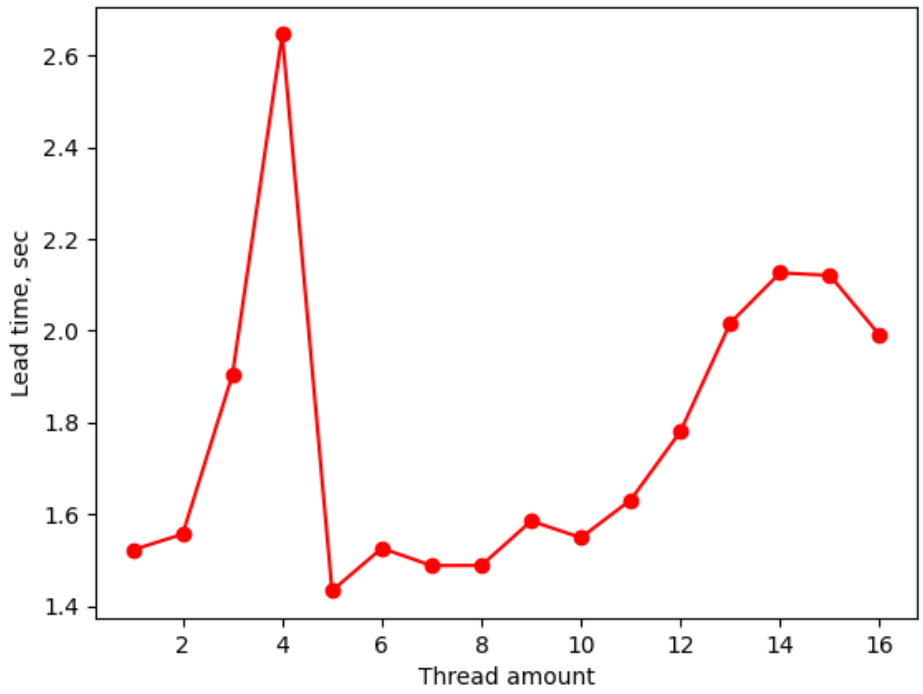

Рисунок 1 – Зависимость времени выполнения работы от количества потоков

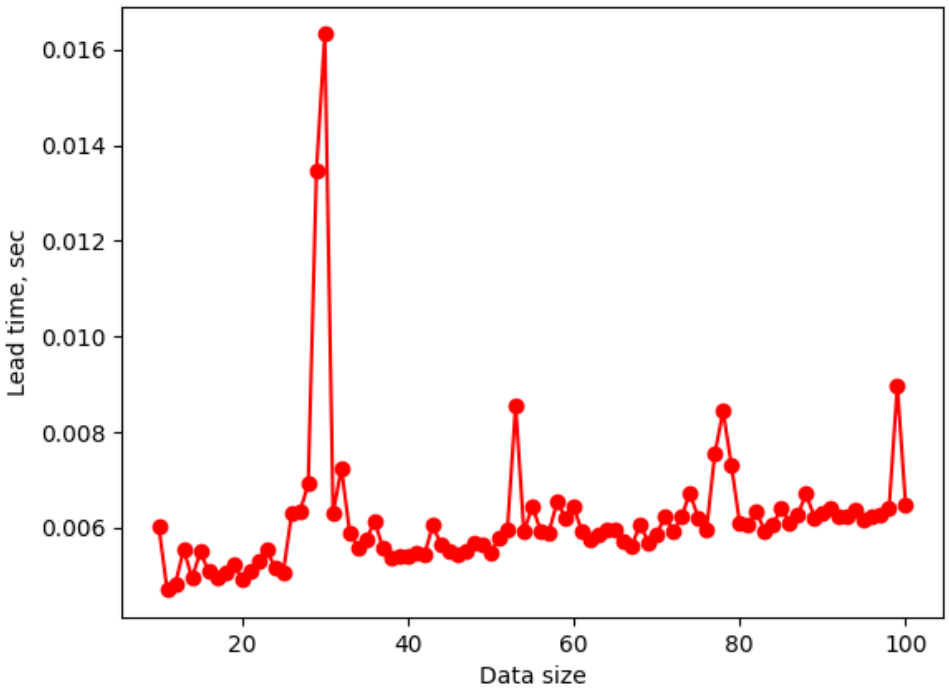

Рисунок 2 – Зависимость времени выполнения работы от размера входных данных

Выводы.

В ходе лабораторной работы были изучены основы работы с процессами и потоками в языке программирования C++. Были реализованы три программы, решающие одинаковую задачу с помощью процессов, потоков и нескольких потоков, решающих одну задачу. Для последней программы были построены зависимости времени работы от количества потоков и объёма входных данных.