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. 0381 _____ Соколов Д. В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Изучить работу с процессами и потоками на языке программирования C++.

Задание.

Выполнить поэлементное сложение 2х матриц M*N/ Входные матрицы вводятся из файла (или генерируются). Результат записывается в файл.

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).

Опционально: в этом процессе могут быть 2 потока ввода/генерации данных

Процесс 2: выполняет сложение.

Процесс 3: выводит результат.

1.2.1

Аналогично 1.1, используя потоки (threads)

1.2.2

Разбить сложение на P потоков.

Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы

Выполнение работы.

Основы:

Матрицы в программе представлена объектами классом Matrix с полями rows и columns, хранящими размеры матрицы (количество строк и столбцов) и двумерным вектором соответствующих размеров, хранящим непосредственно значения элементов матрицы.

Для класса Matrix были определены простые getter-методы, операторы ввода и вывода в поток, оператор сложения, метод проверки соответствия

(равенства) размеров матриц и статический метод частичного суммирования двух матриц (используется в пункте 3 работы).

Для выполнения задач по генерации матриц и записи/чтению их из файла были реализованы классы Generator и FileHandler, которые потом были объединены в DataHandler композицией.

1. Сложение матриц в разных процессах:

Был создан класс SharedMemory, отвечающий за хранение в совместной памяти, используемых матриц. При инициализации класс создает область shared_memory при помощи методов *sys/shm.h*. Написаны методы для записи и чтения из совместной памяти.

Был реализован класс ProcessHandler, отвечающий за распараллеливание сложения матриц в разные процессы. Он объединяет композицией SharedMemory и DataHandler. Для данного класса реализованы методы input, sum и output, отвечающие за задачи ввода, суммирования и вывода матриц соответственно. Данные методы оборачиваются в std::function и передаются в метод procees, которые отвечает за создание процессов. Последний реализованные метод run – алгоритм суммирования матриц в разных потоках, согласно заданию.

2. Сложение матриц в разных потоках

Был создан класс ThreadHandler, отвечающий за суммирование матриц в 3 разных потоках. Обладает объектом класса DataHandler для генерации и записи/чтения из файла. Логика реализации аналогична предыдущему пункту: методы input, sum и output также отвечают за ввод, суммирование и вывод, однако для создания потоков на основе данных методов пришлось реализовать соответствующие методы thread_input, thread_sum, thread_output. Обмен информацией между процессами реализован при помощи promise/future. Функция execute выполняет полностью суммирование двух матриц в соответствии с заданием, помечена ключевым

словом virtual, т.к. предполагалось наследование от данного класса и переопределение поведения некоторых методов для выполнения 3 пункта задания.

3. Сложение матриц в P – потоках:

Был реализован класс, MultiThread, который отвечает за сложение матриц, разбитое на P – потоков. Обладает объектом DataHandler и SessionTimer, реализующего замер и хранение времени выполнения операции сложения. Класс MultiThread также параметризуется значениями минимального и максимального допустимых числа потоков. Также можно изменять значение выделенных потоков для текущей операции (не во время выполнения операции). Ввод и вывод матриц аналогичен ThreadHandler. Суммирование же использует статический метод частичного суммирования матриц partial_sum. В самом методе определяются границы интервала, которые ограничивает те элементы, которые нужно сложить в текущем потоке путем деления общего количества элементов матрицы на количество выделенных потоков. Для каждой частичной суммы создается отдельный поток. Метод также вызывает методы SessionTimer для замера времени операций.

Завершение:

Также был реализован Data class configuration для более удобной конфигурации программы, а также в source.cpp разные пункты задания были вынесены в отдельные функции.

Было проведено исследование зависимости между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Результаты исследования

Система с 8 потоками процессора:

Зависимость времени сложения матриц размеров 20x20 от количества потоков P представлена в табл.1.

Количество потоков	Время (мс)
1	446
2	659
3	441
4	300
5	2725
6	427
7	404
8	376
9	432
10	1025
15	4512
20	633
25	903
30	3070
35	1264
40	4733

Таблица 1. Зависимость времени сложения 20x20 от количества потоков.

Зависимость времени сложения матриц размеров 1000x1000 от количества потоков представлена в таблице 2.

Количество потоков	Время (мс)
1	32602
2	18568
3	14014
4	10382
5	12396
6	11200
7	13280
8	12006
9	10605
10	12745

15	12336
20	10920
25	12838
30	10855
35	10603
40	11023

Таблица 2. Зависимость времени сложения 1000x1000 от количества потоков.

Выводы.

В ходе выполнения лабораторной работы была изучена работа с процессами и потоками на языке программирования C++.

Была исследована зависимость времени выполнения суммирования матриц фиксированного размера от количества используемых для распараллеливания потоков. По результатам исследования:

- Для относительно небольших по размерам матирц не имеет особого смысла. Иногда (особенно, если число программных потоков превышает половину числа потоков, поддерживаемых процессором) распараллеливание и вовсе приводит к худшему результату по времени выполнения. А если и получается выиграть по времени – то совсем незначительно.
- Для больших матриц – распределение вычисления суммы не просто дает лучший результат, но вероятно является залогом эффективного выполнения операции. Согласно полученным результатам время выполнения вычисления в одном потоке превышало в 3 раза время, затраченное на суммирование в 40 потоках (в 5 раз превышает количество, поддерживаемых процессором потоков, т.е. большая часть потоков ожидала в очереди).
- Выполнение программы не происходило в вакууме, т.е. потоки процессора могли заниматься и освобождаться другими программами или ОС. Однако результаты для большой по размерам матрицы при различных запусках сохраняли общую тенденцию.