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 _____ Боблаков Д.С.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2022

Цель работы.

Ознакомиться с работой процессов и потоков в языке программирования C++.

Задание.

Выполнить поэлементное сложение 2х матриц M*N

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.
    a. Процесс 1: заполняет данными входные матрицы (читает из файла или генерирует их некоторым образом).
    b. Процесс 2: выполняет сложение.
    c. Процесс 3: выводит результат.
2. Выполнить задачу, разбив её на 3 потока.
    a. Поток 1: заполняет данными входные матрицы (читает из файла или генерирует их некоторым образом).
    b. Поток 2: выполняет сложение.
    c. Поток 3: выводит результат.
3. Разбить сложение на P потоков. Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Выполнение работы.

Вся работа состоит из 3 этапов:

1. Генерация двух матриц размера MxN.
2. Сложение матриц.
3. Запись результирующей матрицы в файл.

Сложение матриц с использованием процессов.

Для создания дочерних процессов необходимо было использовать fork(), который возвращает идентификатор процесса pid. По pid можно определить, кем должен выполняться код.

В ходе реализации программы была использована разделяемая память (shared memory).

Сложение матриц с помощью 3 потоков.

Для создания потоков использовался std::thread. Внутрь передается функция вызова и параметры для этой функции. Для корректной работы с объектами использовалась привязка std::ref(). После запуска дочернего потока, происходило ожидание его вызывающим потоком с помощью блокирующего метода join().

Сложение матриц с помощью P процессов.

Создание потоков происходит аналогично предыдущему пункту. Для выполнения сложения создается заданное количество потоков. Затем происходит разделение элементов матрицы. Для каждого потока определяются элементы, которые потоку следует посчитать. Внутри потоков выполняется работа, а далее они собираются в std::vector. В конце для каждого из них вызывается метод join().

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

Параметры вычислительной системы:

Двухъядерный процессор.

Физические ядра: 2.

Логические ядра: 4.

Таблица 1 – Сравнение полученных результатов для матрицы 1500x1500 для 2 ядерного процессора

Количество потоков	Время
1	0.080634
2	0.044857
3	0.046108
4	0.036343
5	0.042526
6	0.061943
7	0.059583
8	0.074313
9	0.063948
10	0.069507
100	0.074869
500	0.088036
1000	0.095811
1500	0.105915

Таблица 2 - Сравнение полученных результатов для матрицы 15x15 для 2 ядерного процессора.

Количество потоков	Время
1	0.000284
2	0.000207

3	0.000135
4	0.00037
5	0.000476
6	0.000487
7	0.000492
8	0.000494
9	0.000506
10	0.000551
100	0.003461
200	0.006953

Как можно заметить, с увеличением потоков до определенного числа, время выполнения уменьшается. Как можно понять максимальная производительность достигается при 4 потоках, что соответствует ожиданиям при 4 логических процессорах. После перехода оптимального числа потоков производительность падает.

Выводы.

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

В лабораторной работе были реализованы три программы:

Было реализовано три программы:

1. Сложение двух матриц с помощью 3 процессов.
2. Сложение двух матриц с помощью 3 потоков.
3. Сложение двух матриц с помощью P потоков.

Также было установлено, что:

1. На достаточно малом размере матриц более быстрый подсчёт обеспечивается одним потоком из-за отсутствия затрат на создание

потока и смену контекста.

2. На большом размере матриц наиболее быстрый подсчёт обеспечивается при использовании количества потоков приближенно равному количеству ядер процессора. При попытке использования большего количества потоком происходит увеличение времени выполнения.