**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 _____ Арутюнян В.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

**Цель работы.**

Ознакомиться с работой с процессами и потоками 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.**

Сложение матриц происходит в 3 этапа:

1. Считывание двух матриц из файла matrices.txt. На отдельной строке была задана размерность матрицы (высота, ширина через пробел). Затем представлена матрица, элементы которой разделены пробелами, а строки матрицы выделяются переносами строк. Аналогично представлена вторая матрица в том же файле.

2. Поэлементное сложение матриц.
3. Вывод получившейся матрицы в файл out_matrix.txt в том же виде, как представлена матрица во входном файле.

**Сложение матриц с помощью 3 процессов.**

Создание дочерних процессов происходило с помощью fork(). По возвращаемому pid можно однозначно разделять выполняемый код между родительским и дочерним процессами.

Для передачи данных между процессами использовалась разделяемая память (shared memory). Для использования было необходимо несколько функций:

1. shmget(key, size, shmflg) – возвращает идентификатор shmid разделяемого сегмента памяти по key. С помощью size задается размер, shmflg позволяет задавать различное поведение, например, если сегмент для key уже существует.
2. shmat(shmid, shmaddr, shmflg) – позволяет подсоединиться к разделяемой памяти по shmid. Shmaddr – определяет адрес присоединения. Shmflg – флаг, позволяющий задавать определенное поведение.

**Сложение матриц с помощью 3 потоков.**

Для создания потоков используется конструкция std::thread, в которую передается функция для вызова, а также параметры для этой функции. Для получения возвращаемого значения из функции использовалась конструкция std::promise.

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 выбирается количество потоков, на которые равномерно разделяются элементы матрицы. Например, если для потока отобрано первых 10 элементов, то данный поток суммирует только эти 10 элементов и заканчивают работу. Потоки собираются в std::vector, а затем для каждого из них вызывается метод join().

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

Исследование зависимости происходило путём перебора количества потоков.

В таблице 1 представлено сравнение обработки матрицы 1500x1500 для разного кол-ва потоков. По приведенным результатам видно, что наиболее быстрое выполнение происходит при количестве потоков приближенно равному количеству ядер процессора (8 ядер). Также ясно, что в случае чрезмерного количества потоков время обработки начинает замедляться из-за чересчур частой смены контекста.

Таблица 1 – Сравнение полученных результатов для матрицы 1500x1500 для 8 ядерного процессора.

| Количество потоков | Затраченное время, сек. |
|---|---|
| 1 | 0.121501 |
| 2 | 0.079182 |
| 3 | 0.074381 |
| 4 | 0.070501 |
| 5 | 0.067692 |
| 6 | 0.065931 |
| 7 | 0.063737 |
| 8 | 0.062064 |
| 9 | 0.063012 |
| 10 | 0.064251 |
| 100 | 0.063195 |
| 500 | 0.072311 |
| 1000 | 0.097499 |

| | |
|---|---|
| 1500 | 0.189527 |

В таблице 2 представлено сравнение обработки матрицы 15x15 для разного кол-во потоков. По приведенным результатам видно, что на малом размере матрицы подсчёт, выполненный одним потоком, является наиболее быстрым способом, так как при увеличении кол-ва потоков время переключения контекста начинает превышать время выполнения полезных действий.

Таблица 2 – Сравнение полученных результатов для матрицы 15x15 для 8 ядерного процессора.

| Количество потоков | Затраченное время, сек. |
|---|---|
| 1 | 0.000181 |
| 2 | 0.000211 |
| 3 | 0.000224 |
| 4 | 0.000365 |
| 5 | 0.000435 |
| 6 | 0.000295 |
| 7 | 0.000269 |
| 8 | 0.000311 |
| 9 | 0.000303 |
| 10 | 0.000348 |
| 100 | 0.002141 |
| 200 | 0.002183 |

**Выводы**

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

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 экспериментальным путём 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.