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

# ОТЧЁТ

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

Студент гр.1304 _____ Поршнев Р.А.
Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2024

**Задание**

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, используя потоки (std::threads)

3. Разбить умножение на P потоков (можно “наивным” способом по по строкам-столбцам). Протестировать, сравнив результат вычислений с результатами из 1 и 2.

**Выполнение работы.**

1. Для выполнения данного задания в качестве механизма обмена сообщенями между процессами был выбран файловый обмен. Алгоритм работы программы следующий:

(a) Программе через CLI передаются названия файлов, в которые будет записана левая матрица, правая матрица, результат их умножения и флаг, который отвечает за то, нужно ли генерировать новую матрицу (0 – взять матрицы из файлов, которые были переданы на вход, в ином случае – предоставить возможность для генерации новых матриц).

(b) Главный процесс передаёт управление дочернему процессу, который предназначен для генерации матриц, и ждёт его завершения. Если требуется генерация новых матриц, то предоставляется возможность ввести их размерности, а также диапазон генерации чисел матриц. Сгенерированные матрицы записываются в соответствующие файлы, названия которых были переданы на вход.

(c) Затем главный процесс передаёт управление дочернему процессу, который считывает матрицы из файлов и перемножает их между собой. Результат

умножения записывается в файл, название которого было передано через CLI. Всё это время главный процесс ждёт окончания работы дочернего.

(d) Далее главный процесс передаёт управление дочернему процессу, который считывает матрицу-результат, и выводит её на экран. В то же время, главный процесс ждёт окончания работы дочеренего процесса.

Все реализованные программы используют одни и те же пользовательские функции, описанные ниже:

• *Matrix GetMatrix(std::string filename)* – данная функция принимает на вход пусть к текстовому файлу, из которого нужно загрузить матрицу в программу, и возвращает считанную матрицу;

• *void WriteMatrix(Matrix matrix, std::string filename)* – данная функция принимает на вход матрицу, которую нужно записать в файл, и путь к данному файлу;

• *void MultiplyMatrices(std::string filename1, std::string filename2, std::string filename)* – данная функция принимает на вход путь к файлу, в котором хранится левая матрица, правая матрица, а также путь файла, в котором будет храниться матрица-результат, полученная в результате перемножения левой и правой матриц внутри данной функции;

• *void Dialog(std::string nameMatrix, int &n, int &m, int &lowBorder, int &highBorder)* – данная функция предназначена для ввода пользовательских данных, таких число строк и столбцов матрицы, диапазон генерации чисел матрицы, а также её номер;

• *void FillMatrix(std::string &filename, int n, int m, int lowBorder, int highBorder)* – данная функция принимает на вход файл, в котором нужно сохранить сгенерированную в данной функции матрицу, её размеры и диапазон генерации значений матрицы;

• *void GenerateMatrices(std::string filename1, std::string filename2)* – данная функция принимает на вход пути к файлам, в которые будут записаны сгенерированные матрицы, а функция, в свою очередь, запускает все необходимые функции и процедуры для осуществления данной задачи;

• *void PrintMatrix(std::string filename)* – данная функция принимает на вход путь к файлу, из которого нужно загрузить матрицу и распечатать её в консоль.

2. В данном задании для обмена сообщениями так же использовались файлы для удобного сравнения с процессами. Алгоритм работы программы аналогичный, за исключением того, что главный поток ждёт завершения не процессов с помощью функции *waitpid*, а потоков с помощью метода *join*, который вызывается для переменных-потоков.

Также было проведено сравнение потоков и процессов относительно времени работы при одинаковых исходных данных. Так как третий процесс и третий поток выполняют одну и ту же функцию, а именно, вывод матрицы-результат на экран с помощью std::cout, что является системным прерыванием, то включать выполнение данного процесса и потока во время выполнения программы нецелесообразно. Первый процесс и поток также нецелесообразно включать во время работы программы, так как при тестировании программы брали исходные матрицы из текстовых файлов, следовательно, выполнять первый процесс и поток, отвечающие за генерацию исходных матриц, нет необходимости. Таким образом, производилось сравнение одного процесса и потока в рамках операции перемножения матриц. Результаты представлены в Таблице 1.

Таблица 1 – Результат вывода первых пяти записей датафрейма

| | Время выполнения, нс | | Конфигурация входных данных | | |
|---|---|---|---|---|---|
| № | Процесс | Поток | Порядок матрицы | Диапазон чисел первой матрицы | Диапазон чисел второй матрицы |
| 1. | 11236489014 | 10102158359 | 1000 | [-10, 10] | [-10, 10] |
| 2. | 11739377113 | 10686262542 | 1000 | [-100, 100] | [-10, 10] |
| 3. | 1366050689 | 1346216431 | 500 | [-10, 10] | [-10, 10] |
| 4. | 1231122858 | 1313059459 | 500 | [-50, 200] | [0, 10] |

Исходя из Таблицы 1, можно сделать вывод, что в большинстве случаев поток немногим быстрее процесса. Данное наблюдение подкрепляется тем, что для создания нового процесса нужно больше ресурсов, чем для создания потока, ведь процессу выделяется отдельное пространство под стек, данные и код.

3. В данном задании алгоритм аналогичен тому, который был представлен в 2, но с некоторыми изменениями:

• через CLI передаётся четыре параметра: название файла для первой матрицы, для второй, для результата, а также для количества потоков;

• каждый поток должен умножить $\lfloor \frac{n}{p} \rfloor$ строк левой матрицы на все столбцы правой матрицы, где $n$ – число строк левой матрицы, $p$ – количество потоков;

• оставшиеся $n - \lfloor \frac{n}{p} \rfloor$ строк перемножают $n - \lfloor \frac{n}{p} \rfloor$ первых потоков, то есть, каждому из вышеперечисленных потоков нужно перемножить одну строку левой матрицы на все столбцы правой матрицы;

• если количество потоков больше количества строк левой матрицы, то количество потоков полагается равным числу строк левой матрицы.

Также построена зависимость времени умножения двух матриц порядка 1000 от числа потоков, что представлено на Рисунке 1. Для получения данных была разработана программа, которая перебирует количество потоков от 1 до 100 и для каждого числа потоков запускает перемножение двух фиксированным матриц.

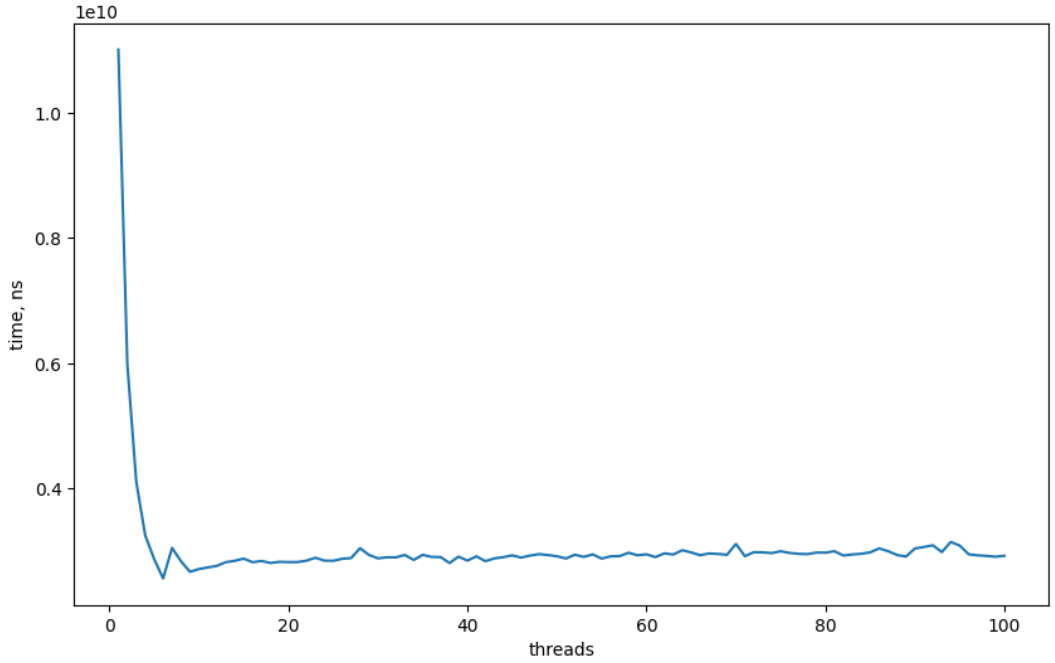


Рисунок 1 – Зависимость времени умножения матриц порядка 1000 от числа потоков

Исходя из данных на Рисунке 1, можно сделать вывод, что до определённого количества потоков чем больше количество потоков, тем меньше время перемножения матриц, причём после опредёлнного количества потоков время работы программы выходит на плато и практически не меняется.

**Тестирование**

Тестирование программ представлено в Таблице 2.

Таблица 2 – Результат вывода первых пяти записей датафрейма

| | | | Результат | | |
|---|---|---|---|---|---|
| № | Левая матрица | Правая матрица | Потоки | Процессы | 2 потока для перемножения |
| 1 | [[-7, -4]] | [[7], [5]] | [[-69]] | [[-69]] | [[-69]] |
| 2 | [[-7], [-4]] | [[7, 5]] | [[-49, -35], [-28, -20]] | [[-49, -35], [-28, -20]] | [[-49, -35], [-28, -20]] |
| 3 | [[-7]] | [[-4, 7, 5]] | [[28 -49 -35]] | [[28 -49 -35]] | [[28 -49 -35]] |
| 4 | [[-7]] | [[4]] | [[-28]] | [[-28]] | [[-28]] |
| 5 | [[-7], [-4], [-7]] | [[5]] | [[-35], [-20], [35]] | [[-35], [-20], [35]] | [[-35], [-20], [35]] |

**Вывод**

В ходе выполнения работы изучено применение процессов и потоков в UNIX-подобных системах для решения практической задачи: перемножение матриц. Также изучена зависимость времени перемножения матриц от числа потоков. Практическим методом установлено, что потоки в большинстве случаев быстрее процессов, что обусловлено большим количеством подготовительных действий при создании дочерних процессов.

Разработано 3 программы, каждая из которых либо генерирует две матрицы и записывает их в два отдельных текстовых файла, либо считывает их из файла, затем перемножает их, предварительно считав с двух файлов, а затем выводит матрицу-результат в консоль. Первая основана на процессах, вторая на потоках, а третья использует многопоточность непосредственно в операции умножения. Также разработана четвёртая программа, которая предназначена для фиксирования времени умножения двух матриц при разном количестве потоков.