**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. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Сергеева Е.И. |

Санкт-Петербург

2024

**Цель работы**

Изучение основ работы с процессами и потоками.

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

**Задание.**

Выполнить умножение 2х матриц.

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).

Процесс 2: выполняет умножение

Процесс 3: выводит результат

1.2.1 Аналогично 1.1, используя потоки (std::threads)

1.2.2 Разбить умножение на P потоков (можно “наивным” способом по по строкам-столбцам). Протестировать, сравнив результат вычислений с результатами из 1.2.1 / 1.1

**Выполнение работы.**

Для работы с матрицами как процессов, так и потоков, реализован класс Matrix. В данном классе хранятся размеры матрицы, а также двумерный вектор, хранящий значения матрицы. Реализованы методы to\_plain() и from\_plain(), позволяющие привести буфер с матрицы к одномерному представлению и обратно (необходимо для сериализации и десериализации).

Задача 1.1

В качестве механизма обмена данными между процессами были неименованные пайпы, однонаправленныe каналы межпроцессного взаимодействия. Создание дочерних процессов осуществляется с помощью функции fork(). Кроме того, в родительском процессе создаются два пайпа, посредством которых передаются матрицы для перемножения из одного дочернего процессы в другой, а затем передается результат умножения в третий процесс, который сохраняет полученную матрицу в файл. В каждый процесс- потомок передается функция-callback:

1) *void CreateMatrices(const int write\_fd)* – создает матрицы и

записывает их в первый пайп

2) *void MultiplyMatrices(const int read\_fd, const int write\_fd)* –

считывает матрицы из пайпа, перемножает их и записывает результат во второй пайп

3) *void SaveResult(const int read\_fd)* – считывает результат из второго пайпа, записывает его в файл.

Задача 1.2.1

В рамках выполнения данной задачи вместо процессов создавались потоки. Для этого применялся std::thread стандартной библиотеки. Кроме того, для получения результатов асинхронных операций использовались *std::promise* и *std::future*. Благодаря данным объектам потоки могут блокироваться для получения значений. Например, таким образом блокируется thread, выполняющий умножение матриц, до тех пор, пока поток-генератор не установит значение для операндов умножения.

Задача 1.2.2

Задача по созданию P потоков аналогична предыдущей за исключением процесса умножения. В данном случае умножение матриц разбивается на p-блоков, где размер блока равен отношению высоты левой матрицы к кол-ву запускаемых потоков. Если значение не целое, то остаток распределяется равномерно между потоками. Ожидание завершения работы потоков реализовано при помощи join()

Замеры времени выполнения программы

Для измерения производительности использовалась команда time в bash. Измерения проводились на системе с 6 ядрами, время представлено для 100 запусков. Полученные результаты представлены в таблицах 1-3.

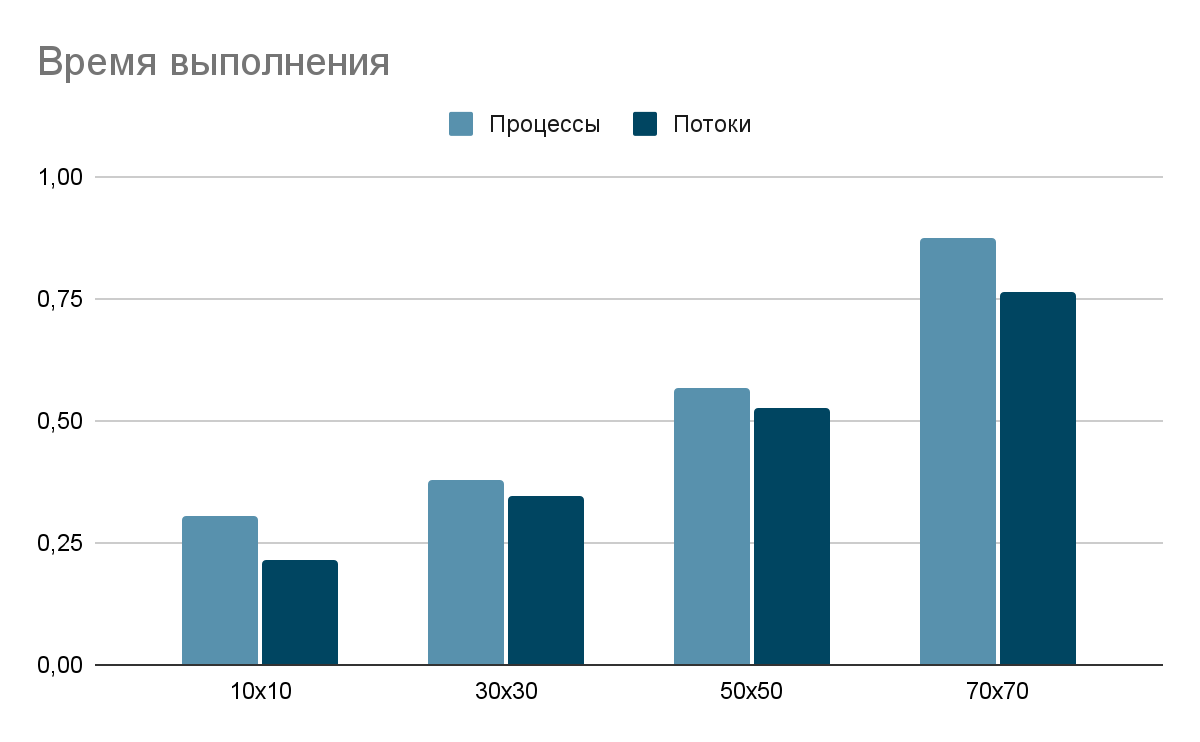
Таблица 1. Процессы

|  | Real, s | User, s | Sys, s |
| --- | --- | --- | --- |
| 10x10 | 0.305 | 0.274 | 0.036 |
| 30x30 | 0.377 | 0.344 | 0.038 |
| 50x50 | 0.569 | 0.528 | 0.048 |
| 70x70 | 0.874 | 0.826 | 0.055 |

Таблица 2. Три потока

|  | Real, s | User, s | Sys, s |
| --- | --- | --- | --- |
| 10x10 | 0.216 | 0.169 | 0.041 |
| 30x30 | 0.253 | 0.153 | 0.039 |
| 50x50 | 0.464 | 0.203 | 0.059 |
| 70x70 | 0.762 | 0.275 | 0.068 |

Гистограммы зависимости времени выполнения программы от размера матриц:



Исходя из представленных выше данных, легко видеть, что потоки более производительные, чем процессы. Это связано с тем, что они используют общее адресное пространство, их взаимодействие требует меньше системных вызовов.

Таблица 3. Зависимость времени выполнения от размера матриц и кол-ва потоков.

|  | Потоки, шт | Real, s | User, s | Sys, s |
| --- | --- | --- | --- | --- |
| 30x30 | 2 | 0,428 | 0,299 | 0,105 |
| 30x30 | 6 | 0,482 | 0,339 | 0,146 |
| 30x30 | 12 | 0,559 | 0,381 | 0,198 |
| 30x30 | 24 | 0,699 | 0,382 | 0,326 |
| 30x30 | 48 | 0,726 | 0,362 | 0,387 |
| 50x50 | 2 | 0,585 | 0,343 | 0,122 |
| 50x50 | 6 | 0,572 | 0,398 | 0,148 |
| 50x50 | 12 | 0,636 | 0,484 | 0,200 |
| 50x50 | 24 | 0,805 | 0,497 | 0,359 |
| 50x50 | 48 | 0,981 | 0,465 | 0,0,621 |
| 70x70 | 2 | 0,870 | 0,476 | 0,143 |
| 70x70 | 6 | 0,789 | 0,492 | 0,221 |
| 70x70 | 12 | 0,886 | 0,590 | 0,381 |
| 70x70 | 24 | 0,885 | 0,594 | 0,429 |
| 70x70 | 48 | 1,079 | 0,660 | 0,516 |

Исходя из представленных данных, нетрудно видеть, что для размеров матриц 50x50 и 70x70 наименьшее время выполнения достигнуто для кол-ва потоков, равного кол-ву ядер процессора.

Для 30x30 прироста с увеличением числа потоков не наблюдается. Увеличение размера матрицы приводит к большему количеству операций, что делает параллелизм более эффективным. Поэтому для матриц маленького размера может быть более эффективным выполнение в одном потоке.

Увеличение числа потоков может улучшить производительность на больших матрицах, так как каждая часть задачи может выполняться параллельно. Однако важно учитывать, что слишком большое количество потоков может привести к состоянию, когда расходы на переключение потоков будут превосходить полезную нагрузку.

**Выводы.**

В ходе работы были изучены основы работ с процессами и потоками. Были исследованы зависимости между количеством потоков и размерами входных данных. Было установлено, что наименьшее время умножения достигается при использовании количества потоков равного количеству процессоров системы.