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. 0304 _____ Алексеев Р.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2023

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 (“наивным” способом по по строкам-столбцам).

Выполнение работы.

1. Реализация умножения матриц, разбитого на 3 процесса, представлено в файле 111.cpp.

Для хранение данных о матрицах реализована структура matrixes:

```
struct matrixes {
     int left [MAT_Y][MAT_N];
     int right [MAT_N][MAT_X];
     int result [MAT_Y][MAT_X];
};
```

В данной структуре left — левая матрица в умножении, right — правая матрица в умножении, result — матрица-произведение.

Разбиение на отдельные потоки происходит при помощи fork(), которая создает процессы-потомки. Для определения того, в каком процессе находится программа используется switch и PID. Если PID = 0, то данный процесс — потомок, если PID > 0, то процесс — родитель.

В первом потомке происходит создание двух матриц, во втором умножение матриц. Вывод результата в консоль происходит в родителе, для ожидания завершения процессов-потомков используется waitpid().

Передача данных между потоками реализована при помощи FIFO.

2. Разделение процесса умножения на три потока реализовано в файле 121.cpp. Реализован ряд функций:

Создание матриц:

```
void createMatrixes(matrixes& matrixes) {
     for(int y = 0; y < MAT_Y; y++) {
          for(int x = 0; x < MAT_N; x++) {
               matrixes.left[y][x] = (y+1)*(x+1);
          }
     }

     for(int y = 0; y < MAT_N; y++) {
          for(int x = 0; x < MAT_X; x++) {
               matrixes.right[y][x] = (y+1)*(x+1);
          }
     }
};
```

Умножение матриц:

```
void multiplyMatrixes(matrixes& matrixes) {
     for(int y = 0; y < MAT_Y; y++) {
          for(int x = 0; x < MAT_X; x++) {
               for(int i = 0; i < MAT_N; i++) {
                    matrixes.result[y][x] += matrixes.left[y][i]
* matrixes.right[i][x];
               }
          }
```

```
        }
    };
```

Вывод результатов:

```
    void printResult(matrixes& matrixes) {
        for(int y = 0; y < MAT_Y; y++) {
            for(int x = 0; x < MAT_X; x++) {
                std::cout << matrixes.result[y][x] << " ";
            }
            std::cout << std::endl;
        }
    };
```

Для разделения на потоки используется библиотека thread и функция std::thread(), которая принимает функцию, которую необходимо выполнить в потоке и аргументы, которые необходимо передай этой функции. Ожидание завершения работы потоков реализовано при помощи join().

3. Реализация разбиения умножения на P потоков выполнена в файле 122.cpp. Для создания матриц и вывода результатов используются функции аналогичные функциям из 121.cpp. Для умножения при помощи P потоков изменена функция *void multiplyMatrixes()*:

```
    void multiplyMatrixes(matrixes& matrixes) {
        int matrix_size = MAT_X * MAT_Y;

        int hardware_threads = std::thread::hardware_concurrency();
        int num_threads = AMOUNT_P;

        int block_size = matrix_size / num_threads;

        std::vector<std::thread> threads(num_threads);

        for(int i = 0; i < (num_threads); i++) {
            threads[i] = std::thread(multiply, i, matrix_size,
block_size, std::ref(matrixes));
        }

        if(matrix_size % num_threads) {
            int y = num_threads*block_size / MAT_X;
            int x = num_threads*block_size % MAT_X;

            for(int j = 0; j < matrix_size % num_threads; j++) {
                for(int i = 0; i < MAT_N; i++) {
                    matrixes.result[y][x] += matrixes.left[y][i]
* matrixes.right[i][x];
                }
```

```
                x++;
                if(x >= MAT_X) {
                      x = 0;
                      y++;
                }

          }
     }

     for(int i = 0; i < (num_threads); i++) {
          threads[i].join();
     }

};
```

Общее количество умножений делится на блоки равного размера между P потоками. Если невозможно нацело разделить количество умножений, остаток операций выполняется в общем потоке умножения — *multiplyMatThread*. Ожидание завершения работы потоков реализовано при помощи join().

4. Для измерения производительности использовалась команда time в bash. Измерения проводились на системе с 8 процессорами. Полученные результаты представлены в таблицах 1-3.

Таблица 1. Разбиение умножения на 3 процесса.

Размер матриц	Время, мс
10x10, 10x10	2
25x25, 25x25	2
50x50, 50x50	3
100x100, 100x100	7

Таблица 2. Разбиение умножения на 3 потока.

Размер матриц	Время, мс
10x10, 10x10	1
25x25, 25x25	2
50x50, 50x50	3
100x100, 100x100	7

По таблицам видно, что умножение при разбиении на потоки требует меньше времени, чем при разбиении на процессы.

Таблица 3. Разбиение умножения на P потоков.

Размер матриц	Кол-во потоков	Время, мс
50x50, 50x50	2	2,5
	4	2,5
	8	2,5
	16	3
	24	3
	32	3
	48	3,5
	64	3,5
150x150	2	9
	4	6
	8	6
	16	7
	24	7
	32	7
	48	7,5
	64	7,5

По таблице видно, что наименьшее время достигается при количестве потоков равном количеству процессоров системы - 8. Если количество меньше, то часть процессоров простаивает. Если количество потоков превышает количество процессоров, то процессорам приходится переключаться между потоками и тратить на это дополнительное время.

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.