**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 потоками и процессами**

Студент гр. 1304 _____ Шаврин А.П.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2024

**Цель работы.**

Изучить основы работы с потоками и процессами. Провести 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.**

1. Для выполнения действий с матрицами был создан файл *matrix_operations.cpp,* который содержит следующие функции:

*void setDefaultMatricesParams(int& rowsA, int& columnsA, int& rowsB, int& columnsB, int& maxValue)* – функция, которая устанавливает значения параметров обоих матриц.

*std::vector<std::vector<int>> generateMatrix(const int rows, const int columns, const int maxValue = 10)* - функция генерации матрицы с заданными параметрами строк и столбцов, а также максимального по модулю значения для генерации. Функция возвращает двумерный вектор целых значений.

*std::vector<std::vector<int>> multiplyMatrices(const std::vector<std::vector<int>>& A, const std::vector<std::vector<int>>& B)* –

функция перемножения переданных матриц A и B. Функция возвращает двумерный вектор – результат перемножения.

*void writeMatrixInFile(const std::string& filename, const std::vector<std::vector<int>>& result)* – функция записи переданной матрицы в файл.

*void printMatrix(const std::vector<std::vector<int>>& result)* – функция вывода переданной матрицы в консоль.

2. Для выполнения пункта 1.1 был создан файл main.cpp в папке processes, который содержит следующие функции:

*void processMatrixWithPipe(int unpipe, std::vector<std::vector<int>>& matrix, int process)* – функция для чтения или записи в неименованный канал. Функция получает канал для работы, ссылку на матрицу для операции и численный параметр, определяющий операцию (0-чтение, 1-запись).

*int main()* – в данной функции создается три неименованых канала и три процесса, в которых вызываются соответствующие функции, также производится замер времени, синхронизация процессов осуществлялась, с помощью методов waitpid, дожидающихся завершения процессов.

3. Для выполнения пункта 1.2.1 был создан файл main.cpp в папке thread_1, который содержит следующие функции:

*void thGenerateMatrices(std::vector<std::vector<int>>& matrixA, std::vector<std::vector<int>>& matrixB, int maxValue)* – потоковая функция генерации обоих матриц.

*void thMultiplyMatrices(const std::vector<std::vector<int>>& matrixA, const std::vector<std::vector<int>>& matrixB, std::vector<std::vector<int>>& result)* – потоковая функция перемножения матриц.

*void thWriteMatrixInFile(const std::vector<std::vector<int>>& result)* – потоковая функция записи матрицы в файл.

*int main()* – в данной функции создается три потока, в которых вызываются соответствующие функции, также производится замер времени,

синхронизация потоков осуществлялась, с помощью методов join дожидающихся завершения потоков.

4. Для выполнения пункта 1.2.2 был создан файл main.cpp в папке thread_2, который содержит следующие функции:

*void thMultiplyMatrixBlock(const std::vector<std::vector<int>>& matrixA, const std::vector<std::vector<int>>& matrixB, std::vector<std::vector<int>>& result, std::vector<std::pair<int, int>> blockPositions)* – потоковая функция перемножения, вычисляющая значения ячеек результирующей матрицы по переданным индексам.

*int main()* – данная функция генерирует 2 матрицы, реализует их перемножение в p потоков (в соответствии с заранее определенным параметром) и записывает результирующую матрицу в файл. Также производится замер времени работы программы.

5. Сравнение 3х подходов к решению задачи.

По таблице тестирования 1 можно видеть, что использование подхода 1.2.1 во всех случаях показал себя лучше, чем 1.1, что можно обосновать легкостью потоков по отношению к процессам. Что касается использования подхода 1.2.2 с количеством подходом равным (или близким) к количеству элементов в результирующей матрице, то при небольшом увеличении потоков показывает значительно лучший результат, однако, когда разница в потоках увеличивается в разы то этот подход значительно проигрывает первым 2м, но для очень больших размеров матрицы (более 1000) большое количество потоков дает выигрыш во времени. Таким образом можно сделать выводы, что лучше всего по времени использовать 3 потока (при размере матрицы не больше 100).

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

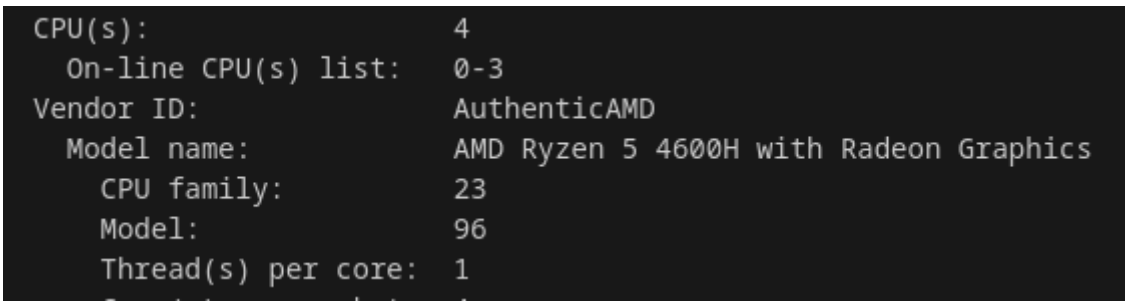


Рисунок 1. Параметры системы.

По таблице тестирования 2 можно видеть, что для малых размеров матрицы (2x2 – 10x10) оптимальным является число потоков равное двум. Однако для больших размеров матриц (100x100 и больше) оптимальное число потоков необходимо увеличивать, в частности для 100x100 до 25 потоков, а для 1000x1000 до 100 потоков. Таким образом, можно сделать вывод, что при данной мощности компьютера (4 процессора) оптимальное число потоков для небольших вычислений от 2 до 4, а для больших вычислений от 25 до 100. Также выявлено, что при увеличение входных данных время выполнения операции увеличивается.

Результаты тестирования см. в приложении А.

**Выводы.**

В результате работы были изучены основы работы с потоками и процессами, а также выявлено, что при данных параметрах целевой вычислительной системы оптимальное количество потоков от 2 до 4 для небольших вычислений и от 25 до 100 для больших вычислений.

# ПРИЛОЖЕНИЕ А
# ТЕСТИРОВАНИЕ

Таблица 1 – Сравнение подходов 1.1, 1.2.1 и 1.2.2.

| № п/п | Размеры входных матриц | Подход | Время выполнения, ms |
|---|---|---|---|
| 1. | A = 2 x 2<br>B = 2 x 2 | 3 процесса | 1.34491 |
| | | 3 потока | 1.25884 |
| | | 4 потока | 0.99221 |
| 2. | A = 5 x 5<br>B = 5 x 5 | 3 процесса | 1.54282 |
| | | 3 потока | 1.34971 |
| | | 25 потоков | 3.25326 |
| 3. | A = 7 x 7<br>B = 7 x 7 | 3 процесса | 1.75838 |
| | | 3 потока | 1.57523 |
| | | 49 потоков | 5.71752 |
| 4. | A = 50 x 50<br>B = 50 x 50 | 3 процесса | 6.51585 |
| | | 3 потока | 4.36115 |
| | | 2500 потоков | 244.919 |
| 5. | A = 100 x 100<br>B = 100 x 100 | 3 процесса | 22.1476 |
| | | 3 потока | 21.3965 |
| | | 10000 потоков | 1028.97 |
| 6. | A = 1000 x 1000<br>B = 1000 x 1000 | 3 процесса | 13046.3 |
| | | 3 потока | 12662.4 |
| | | 10000 потоков | 8965.36 |

Таблица 2 – Исследование зависимости между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

| № п/п | Размеры входных матриц | Кол-во потоков | Время выполнения, ms |
|---|---|---|---|
| 1. | A = 2 x 2<br>B = 2 x 2 | 4 | 1.031800 |
| | | 2 | 0.896544 |
| | | 1 | 0.903998 |
| 2. | A = 10 x 10<br>B = 10 x 10 | 100 | 10.29500 |
| | | 50 | 6.806060 |
| | | 25 | 4.895910 |
| | | 10 | 3.535100 |
| | | 5 | 1.447290 |
| | | 2 | 1.011760 |
| 3. | A = 100 x 100<br>B = 100 x 100 | 10000 | 994.1060 |
| | | 5000 | 492.6470 |
| | | 1000 | 115.4510 |
| | | 100 | 31.03960 |
| | | 25 | 19.53380 |
| | | 5 | 20.95480 |
| | | 2 | 22.03600 |
| 4. | A = 1000 x 1000<br>B = 1000 x 1000 | 10000 | 8506.37 |
| | | 5000 | 7492.77 |
| | | 1000 | 7176.24 |
| | | 100 | 6965.92 |
| | | 25 | 7455.85 |
| | | 5 | 7957.80 |
| | | 2 | 11797.4 |