МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

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

Студент гр. 0304 _____ Максимов Е.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучить основные способы работы с процессами и потоками.

Постановка задачи.

Выполнить умножение двух матриц. 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.

- 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.

Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Выполнение работы.

Описание функций математической модели.

Был создан синоним типа `Matrix`, эквивалентный `vector<vector<int>>` — двумерный массив, хранящий в себе данные матрицы.

Были созданы функции, реализующие взаимодействия с матрицами, перечисленные ниже.

- `int multiplyVectors(vector<int> a, vector<int> b)` — скалярное произведение векторов.
- `Matrix getTransposedMatrix(Matrix matrix)` — транспонирование матрицы.

• `void multiplyMatrices(vector<Matrix>* matricesPointer)` — произведение матриц; полученная матрица добавляется в конец вектора `matricesPointer`.

Данные функции используются на этапе подсчёта в нижеперечисленных вариантах решения задачи.

Исполнение процессами.

Функции, взаимодействующие с процессами, перечислены ниже.

• `void runProcess(const function<void()> &function)` — создаёт и исполняет процессы-потомки. Чтобы определить тип запускаемого процесса (потомок или родительский процесс), используется конструкция switch-case и PID процесса: если PID = 0, то процесс является потомком, если PID > 0, то родительским процессом, и он ожидает выполнение процесса-потомка в соответствующем блоке при помощи вызова `wait(&pid)`.

• `void readMatrices(), void calculate(), void writeMatrix()` — функции, выполняющие этапы работы программы (чтение, подсчёт и запись соответственно) последовательно в процессах при помощи функции выше.

• `void* getSharedMemoryPointer(),`

• `void writeToSharedPointer (int* sharedPointer,`

• `vector<int> buffer), vector<int> readFromSharedPointer(int* sharedPointer)` — функции, которые взаимодействуют с общей памятью (*shared memory*) для межпроцессорной коммуникации (получение указателя на сегмент памяти, чтение и запись).

• `vector<int> serializeManyMatrices(vector<Matrix> matrices),`

• `vector<Matrix> deserializeManyMatrices(vector<int> buffer)` — функции для сериализации и десериализации матриц для взаимодействия с общей памятью; функции сериализуют вектор матриц в одномерный массив, удобный для сохранения в общей памяти, и обратно.

Основная функция программы запускает последовательно 3 процесса: чтение из файла, подсчёт и запись в файл. Процессы обмениваются между

собой данными при помощи общей памяти, используя после чтения и перед записью функции для де-/сериализации данных.

Исполнение потоками.

Функции, взаимодействующие с потоками, перечислены ниже.

- `void readMatrices(vector<Matrix>* matricesPointer)`,
- `void calculate(vector<Matrix>* matricesPointer)`,
- `void writeMatrix(vector<Matrix>* matricesPointer)` —

функции, выполняющие этапы работы программы (чтение, подсчёт и запись соответственно) последовательно в потоках.

Основная функция создаёт хранилище матриц `vector<Matrix> matrices` и запускает последовательно 3 потока: чтение из файла, подсчёт и запись в файл. Потоки обмениваются между собой данными при помощи указателя на `matrices`.

Поток (`std::thread`) инициализируется функцией, которая выполняется в потоке, и набором аргументов для этой функции. Поток запускается при помощи метода `join()`.

Исполнение при помощи множества потоков для подсчёта.

Для данного варианта подсчёта была написана функция `void multiplyMatricesMultithread(vector<Matrix>* matricesPointer, int indexFirst, int indexLast)`, которая в отличие от функции `void multiplyMatrices(vector<Matrix>* matricesPointer)` из математической модели подсчитывает значения элементов матрицы в заданном промежутке (при этом используя другие функции математической модели).

Основная функция отличается от предыдущего варианта тем, что вместо одного потока для подсчёта матрицы создаётся множество объектов потока,

помещаемые в вектор `vector<std::thread> threads`, которые затем запускаются (при помощи цикла `for`).

Исследование.

Для варианта подсчёта одним потоком была исследована зависимость времени работы программы от объёма входных данных. Результаты тестирования представлены в таблице A1. По полученным данным был построен график, см. рис. 1 ниже.

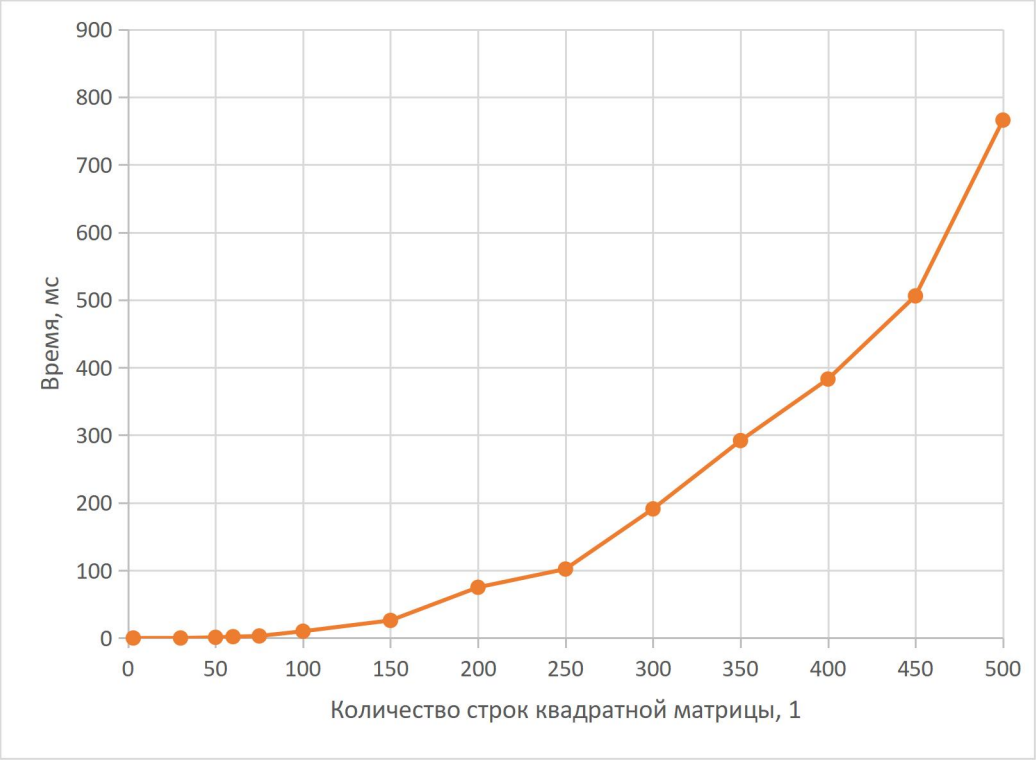

Рисунок 1 — Исследование зависимости времени работы программы от объёма входных данных

Исходя из полученных данных, можно сделать вывод о том, что при увеличении объёма данных время работы программы также увеличивается.

Для варианта подсчёта несколькими потоками была исследована зависимость времени работы программы от количества потоков. Результаты

тестирования представлены в таблице А2. По полученным данным был построен график, см. рис. 2-3 ниже.

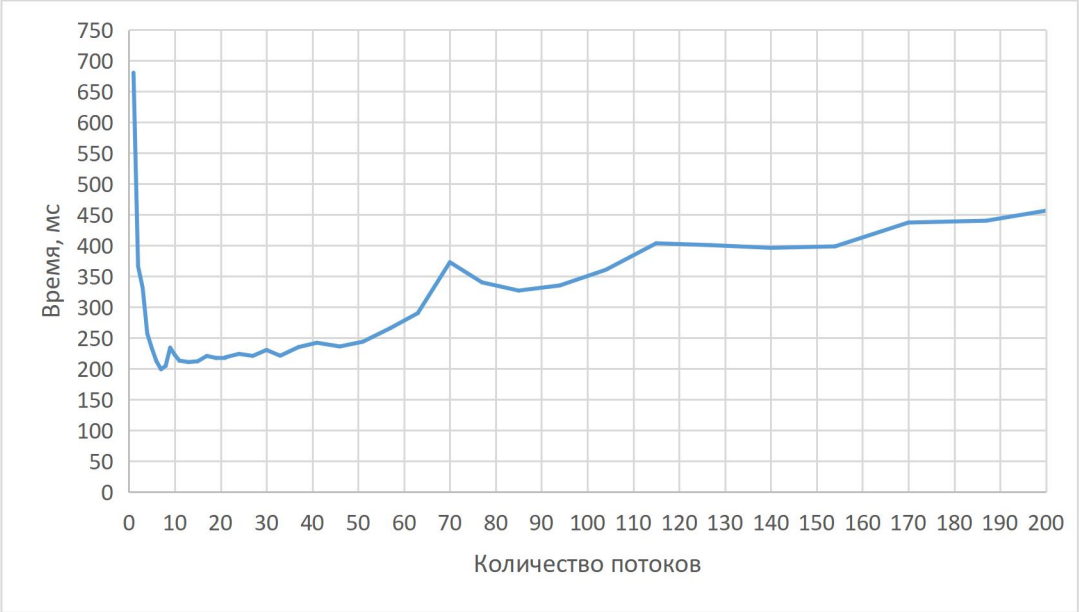

Рисунок 2 — Исследование зависимости времени работы программы от количества потоков, приближение у начала координат

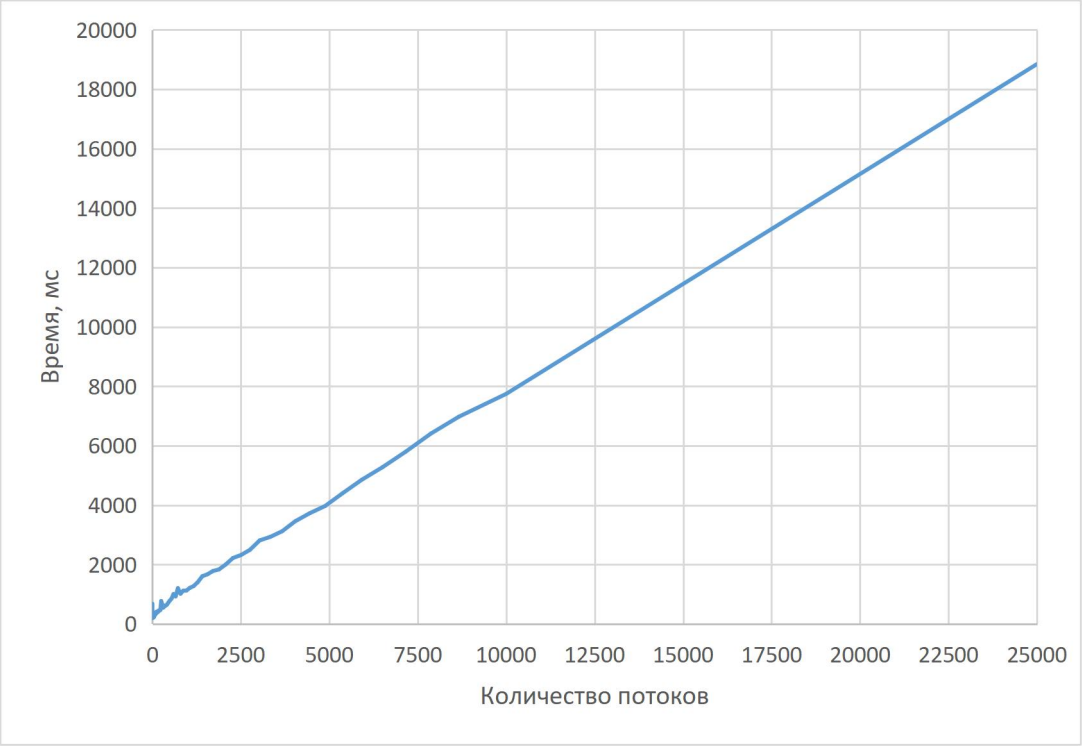

Рисунок 3 — Исследование зависимости времени работы программы от количества потоков, полный график

Исходя из полученных данных, можно сделать вывод о том, что при увеличении числа потоков, время работы программы сначала падает, но после какого-то значения оно начинает возрастать. Это связано с тем, что время, затрачиваемое на инициализацию потока становится больше, чем время подсчёта передаваемых в поток данных.

Выводы.

В ходе выполнения лабораторной работы были изучены принципы работы с процессами и потоками, способы их создания и их взаимодействие.

Практическим результатом лабораторной работы является набор приложений, которые выполняют матричное умножение при помощи процессов и потоков на языке программирования C++.

Также в ходе лабораторной работы был проведён ряд исследований скорости работы программы. После интерпретации полученных данных было выявлено, что:

1) при увеличении размерности матрицы время работы программы также увеличивается;

2) при увеличении числа потоков время работы программы падает до минимума при некотором числе потоков, и после преодоления этого значения время работы возрастает.

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

Таблица А1 — Исследование зависимости времени работы программы от объёма входных данных

Размерность квадратной матрицы	Время работы программы, мс	Размерность квадратной матрицы	Время работы программы, мс
3	< 1	200	75
30	< 1	250	102
50	1	300	191
60	2	350	292
75	3	400	383
100	10	450	506
150	26	500	766

Таблица А2 — Исследование зависимости времени работы программы от количества потоков

Число потоков	Время работы, мс	Число потоков	Время работы, мс	Число потоков	Время работы, мс	Число потоков	Время работы, мс
1	680	37	235	275	669	1881	1833
2	366	41	242	303	547	2070	1998
3	331	46	236	334	579	2277	2218
4	257	51	244	368	616	2505	2320
5	233	57	266	405	640	2756	2492
6	212	63	290	446	707	3032	2811
7	199	70	373	491	782	3336	2932
8	205	77	340	541	849	3670	3121
9	234	85	327	596	1003	4037	3455
10	223	94	335	656	924	4441	3722
11	213	104	361	722	1203	4886	3972
13	211	115	404	795	1016	5375	4398
15	212	127	401	875	1121	5913	4848
17	221	140	396	963	1124	6505	5275
19	218	154	399	1060	1215	7156	5794
21	219	170	437	1166	1273	7872	6407
24	224	187	440	1283	1406	8660	6973
27	221	206	464	1412	1606	10000	7743
30	231	227	478	1554	1669	25000	18842
33	221	250	769	1710	1780		