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

ОТЧЁТ

по лабораторной работе № 2

по дисциплине «Параллельные алгоритмы»

Тема: «Реализация потокобезопасных структур данных с блокировками»

Студент гр.1304 _____ Поршнев Р.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2024

Задание

Реализовать итерационное (потенциально бесконечное) выполнение подготовки, обработки и вывода данных по шаблону “производитель-потребитель” (на основе лаб. 1 (части 1.2.1 и 1.2.2)). Количество производителей и потребителей должно быть изменяемым. Обеспечить параллельное выполнение потоков обработки готовой порции данных, подготовки следующей порции данных и вывода предыдущих полученных результатов. Использовать механизм “условных переменных”.

1. Использовать очередь с “грубой” блокировкой.

2. Использовать очередь с “тонкой” блокировкой.

Очередь должна иметь ограничение сверху по количеству элементов. Выполнить тестирование п. 2.1 и 2.2, убедиться в корректности результатов.

Выполнение работы.

1. Идея очереди с "грубой" блокировкой заключается в том, что при работе с методами данной структуры данных производится блокировка всей очереди. Данная идея в коде реализуется следующим образом: при вызове метода push(matrices) или wait_and_pop(matrices &) класса потокобезопасной очереди захватывается мьютекс X.

Для выполнения данного пункта был реализован класс threadsafe_queue, который содержит в себе следующие методы:

• threadsafe_queue(size_t) – данный конструктор предназначен для инициализации размера буфера очереди;

• threadsafe_queue() – конструктор, который инициализирует размер буфера числом 10;

• void push(matrices) – данный метод предназначен для добавления пары матриц в очередь, которые не добавятся до тех пор, пока в очереди не освободится место, а также для уведомления потока из метода wait_and_pop(matrices &) о том, что в очереди появились новые данные, которые необходимо извлечь;

• void wait_and_pop(matrices &) – данный метод предназначен для ожидания новых данных из очереди, а также их последующего извлечения из очереди и оповещения об этом потоку, который ожидает в методе push(matrices).

Класс threadsafe_queue содержит следующие приватные поля:

• mutable std::mutex mut – мьютекс;

• std::queue<matrices> dataQueue – очередь;

• std::condition_variable pushCond – условная переменная, которая нужна для уведомления потока о том, что в очередь добавлена пара матриц;

• std::condition_variable popCond – условная переменная, которая нужна для уведомления потока о том, что из очереди извлечены данные;

Остальная часть кода программы-входа в проект представлена ниже:

• int main() – данная функция предназначена для запуска функции ввода пользовательских данных, создания и ожидания потоков, отправки контрольных данных в очередь и измерения времени работы программы;

• void InputData(int &nProducers, int &nConsumers, int &nTasks) – данная функция предназначена для ввода числа потребителей, производителей и числа задач для каждого потребителя;

• void Consumer(int id) – данная функция предназначена для запуска бесконечного цикла, в котором запускается функция из класса потокобезопасной очереди для ожидания пары матриц из очереди, проверки условия окончания работы потока, запуска функции перемножения матриц и функции для записи пар матриц и результата их произведения в файл;

• void Producer(int id, int nTasks) – данная функция запускает функцию для генерации пары матриц и добавления их в очередь до тех пор, пока число задач не станет равным nTasks;

• matrices GenerateMatrices() – данная функция предназначена для запуска функций генерации двух матриц и их возврата в вызвавшую функцию;

• Matrix FillMatrix(int n, int m, int lowBorder, int highBorder) – данная функция предназначена для генерации элементов матрицы с учётом её параметров и возврата результата в вызвавшую программу;

• void MultiplyMatrices(matrices &someMatrices) – данная функция принимает на вход пару матриц и распределяет какие потоки будут перемножать последовательности строк левой матрицы на столбцы правой матрицы;

• void MultiplyOperation(Matrix &matrix1, Matrix &matrix2, Matrix &matrix, int fromRow, int toRow) – данная функция принимает на вход левую и правую матрицы, переменную результирующей матрицы, границы строк левой матрицы и предназначена для непосредственно операции перемножения последовательности строк левой матрицы на столбцы правой;

• void WriteMatrices(matrices &someMatrices) – данная функция принимает на вход три матрицы, захватывает мьютекс, который нужен для того, чтобы записать три матрицы в лог-файл, и вызывает функцию непосредственно для самой записи данных в файл;

• void WriteMatrix(Matrix &matrix) – данная функция принимает на вход матрицу, которую необходимо записать в лог-файл.

2. Идея очереди с "тонкой" блокировкой заключается в том, что при работе с методами данной структуры данных производится блокировка отдельно метода push(matrices) и wait_and_pop. Данная идея в коде реализуется следующим образом: при вызове метода push(matrices) класса потокобезопасной очереди захватывается мьютекс X, а при вызове метода wait_and _pop – мьютекс Y. Остальная же часть кода полностью идентична по сравнению с первым заданием.

Также было проведено сравнение двух вариантов блокировок при следующей конфигурации:

• размеры матриц: 50 на 50;

• количество пар матриц для каждого потока: 100;

• размер буфера очереди: 10;

- матрицы состоят только из -1;
- умножение матриц производится с помощью четырёх потоков;

Зависимость времени умножения матриц от количества потребителей/производителей представлена ниже на Рисунке 1.

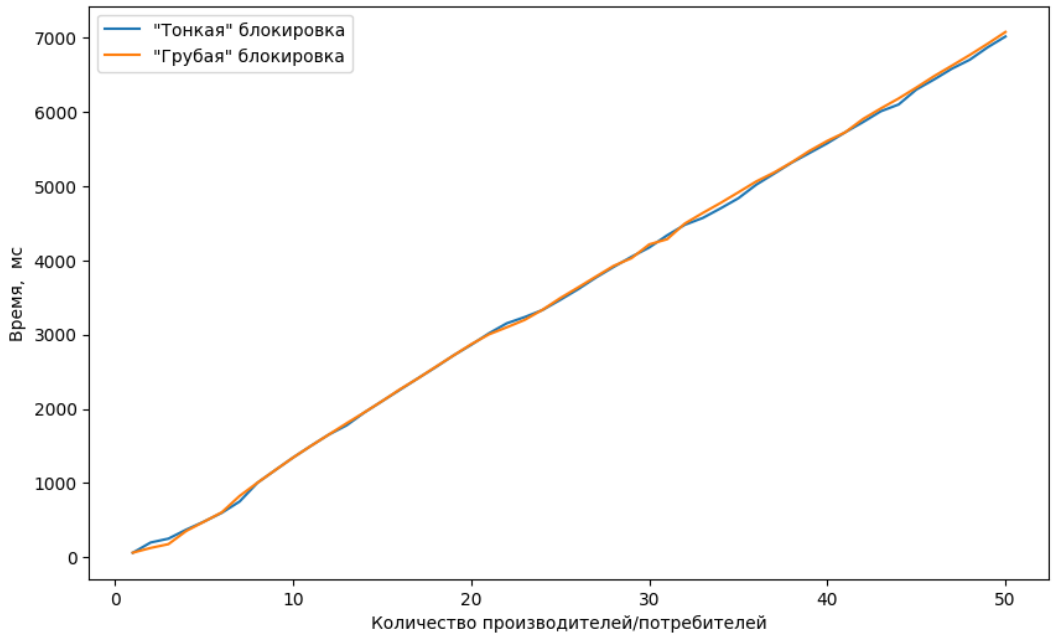

Рисунок 1 – Зависимость времени умножения матриц от количества потребителей/производителей

Исходя из данных на Рисунке 1, можно сделать вывод, что при увеличении числа потребителей и производителей, а также при большом количестве задач для каждого потока при меньшим по порядку размером буфера, использование "тонкой" блокировки окажется выгоднее, что связано с тем, что вместо одного мьютекса используется два, а с учётом конфигурации эксперимента, два мьютекса предпочтительнее, ведь при использовании всего одного мьютекса будет тратиться большее количество времени для захвата и освобождения мьютекса при заполненной очереди. При небольшом буфере и большом количестве производитилей и задач большую часть времени очередь будет заполнена, поэтому необходимо обеспечить мобильность при захвате и освобождении мьютекса, что эффективнее реализовать с помощью двух мьютексов: первый для добавления пары матриц в очередь, второй – для их извлечения.

Вывод

В ходе выполнения работы изучено применение потоков в UNIX-подобных системах для решения практической задачи: перемножение матриц. Также изучена зависимость времени перемножения матриц от числа потребителей/производителей при заданной конфигурации.

Реализованно две программы: первая реализует потокобезопасную очередь с "грубой" блокировкой, а вторая – с "тонкой". Вторая программа отличается от первой лишь тем, что она использует два мьютекса вместо одного, что благоприятно сказывается при большом количестве производителей и задач и при сравнительно небольшим размером буфера.