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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Параллельные алгоритмы»

ТЕМА: РЕАЛИЗАЦИЯ ПОТОКОБЕЗОПАСНЫХ СТРУКТУР ДАННЫХ С БЛОКИРОВКАМИ

Студент гр. 0303 _____ Мыратгелдиев А. М.

Преподаватель _____ Сергеева Е. И.

Санкт-Петербург

2023

Цель работы.

Научиться реализовывать потокобезопасную очередь с грубой и мелкогранулярной блокировкой с использованием условных переменных.

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)).

Обеспечить параллельное выполнение потоков обработки готовой порции данных, подготовки следующей порции данных и вывода предыдущих полученных результатов.

Использовать механизм “условных переменных”.

2.1 Использовать очередь с “грубой” блокировкой.

2.2 Использовать очередь с “тонкой” блокировкой

Сравнить производительность 2.1. и 2.2 в зависимости от количества производителей и потребителей.

Выполнение работы.

Добавили класс *Matrix*, которая принимает на вход размерность матрицы и генерирует случайным образом матрицу данного размера. Для него были перегружены операторы присваивания (с копированием и перемещением), перенаправления в поток и оператор умножения, которая умножает матрицы в 4 потока.

2.1 Для решения данного пункта задачи, был реализован класс *MyQueue*. Этот класс – очередь с грубой блокировкой. Данный класс основан на использовании *std::queue*. Синхронизация потоков реализована путем использования мьютекса и условной переменной.

2.2 Для решения данного пункта задания, был реализован класс FGBQueue (Fine grained blocking queue) – очередь с мелкогранулярной блокировкой. Данный класс, отличается от предыдущего тем, что для блокировки поток используются

2 мьютекса (для «головы» и «хвоста» очереди). При вставке нового элемента блокируется только «хвост», а при удалении элемента из очереди – «голова» и «хвост» (для проверки условия в условной переменной).

Для решения каждой подзадачи, используются 2 очереди:

- в первую очередь кладем пары матриц, которые нужно умножить
- во второю очередь – результат умножения матриц для последующей записи в файл;

В нашей программе «producer» (производитель) генерирует матрицы и кладет их в очередь, а «consumer» (потребитель) - достает эту пару матриц, умножает и кладет в другую очередь.

Исследуем скорость работы подзадач в зависимости от количества потребителей и количества производителей. Так как, программа потенциально может работать бесконечно, посчитаем количество произведенных и умноженных матриц за 3 секунды.

Таблица 1. Результат работы программы при использовании очереди с грубой блокировкой.

Производитель	Потребитель	Произведенные пары матриц	Умноженные пары матриц
1	1	20332	4475
1	5	9245	8808
5	1	11929	2209
10	10	4389	4369

Таблица 2. Результат работы программы при использовании очереди с тонкой блокировкой.

Производитель	Потребитель	Произведенные пары матриц	Умноженные пары матриц
1	1	20885	4493
1	5	9550	8941
5	1	12428	2297
10	10	4322	4316

Из полученных данных видно, что очередь с «тонкой» блокировкой в большинстве случаев быстрее, чем очередь с «грубой» блокировкой. Это объясняется тем, что очередь с «грубой» блокировкой блокирует всю структуру данных. В каждый момент лишь 1 поток совершает какие-то действия.

Выводы.

В ходе выполнения данной лабораторной работы были реализованы потокобезопасные очереди с «грубой» и «тонкой» блокировкой для решения проблемы производитель-потребитель. За счет блокировки отдельных узлов («тонкая» блокировка) мы смогли повысить уровень параллелизма.