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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Параллельные алгоритмы»
Тема: Реализация параллельной структуры данных с тонкой блокировкой

Студентка гр. 9304 _____ Аксёнова Е.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Реализация параллельной структуры данных с тонкой блокировкой и использовать её для сложения потока матриц.

Задание.

Обеспечить структуру данных из лаб.2 как минимум тонкой блокировкой(сделать lock-free).

Протестировать доступ в случае нескольких потоков-производителей и потребителей.

Сравнить производительность со структурой с грубой синхронизацией.

В отчёте сформулировать инвариант структуры данных.

Выполнение работы.

Реализация объекта для буфера.

В классе Bucket содержится:

1. Три массива размера NxM (две для сложения генерируются рандомно при создании объекта, третья заполняется во время суммирования первых двух).
2. Номер обрабатываемого элемента по этому индексу.

Реализация буфера для решения проблем.

В классе Queue содержится:

1. Индекс считывания (head).
2. Индекс записи (tail).
3. Массив bucket’ов.

Принцип работы pop:

Достаем индекс “головы” (используется relaxed-семантика, которая обеспечивает атомарное чтение и то, что при каждом чтении не будет более раннее).

Пока операция не пройдет успешно:

1. Достаем bucket, на который указывала “голова”.
2. Достаем номер обрабатываемого элемента по этому индексу. (последующие операции с этой переменной будут начаты только после завершения данной загрузки, т.е. все увидят последнее значение)
3. Далее вычисляем разницу между “головой” и обрабатываемый элемент по индексу “головы”.
    a. Если их разница - 1 (Значит все успешно, у нас есть матрица в bucket, на который указывала “голова”).
    b. Если их разница - больше 1. (Значит нет значения в bucket, на который указывала “голова”). Меняем голову, пытаемся считать какой-нибудь следующий.
    c. Если их разница - меньше 1. (Значит значение в bucket, на который указывала “голова” уже кто-то начал считывать). А значит нет смысла менять голову, надо просто ждать.
4. Когда удалось считать, то записываем в ссылку значение, а номер обрабатываемого элемента по этому индексу увеличиваем на размер массива (предыдущие операции с памятью будут завершены до данной записи, т.е. это будет последнее записанное).

Принцип работы pop:

Достаем индекс “хвоста”.

Пока операция не пройдет успешно:

5. Достаем bucket, на который указывал “хвост”.
6. Достаем номер обрабатываемого элемента по этому индексу.
7. Далее вычисляем разницу между “хвостом” и обрабатываемый элемент по индексу “хвоста”.
    a. Если их разница - 0 (Значит все успешно, bucket, на который указывал “хвост”, свободен).

b. Если их разница - больше 0. (Значит уже записали в bucket, на который указывал “хвост”). Меняем хвост, пытаемся записать в следующий.
c. Если их разница - меньше 1. (Значит значение в bucket, на который указывала “голова” уже кто-то начал писать в bucket и изменил хвост). А значит нет смысла менять хвост, ждем, когда закончат записывать.

8. Когда удалось получить верный bucket, то записываем в него значение, а номер обрабатываемого элемента по этому индексу увеличиваем на 1.

Инвариант структуры данных.

1. $tail \geq head$ (циклически)

Нельзя считать больше, чем записано.

2. $(index\_near\_free[i \bmod size]) \bmod size$

$$\leq (index\_near\_free[(i + 1) \bmod size]) \bmod size$$

Нельзя увеличивать значение $index\_near\_free$ больше чем на $1 \bmod size$, т.е. нельзя записывать или считывать через ячейку

3. $tail = buffer[tail] \rightarrow index\_near\_free$

Можно записать, только если никто не перенес tail (начал писать) или никто не закончил писать (ячейка была пустой).

4. $head + 1 = buffer[head] \rightarrow index\_near\_free$

Можно считать, только если никто не перенес head (начал читать) или никто не закончил читать (или значение было записано).

Сравнение структуры данных со структурой данных из прошлой лабораторной работой.

Были сравнены две структуры в зависимости от количества итераций сложения. Результаты представлены в таблице 1.

Таблица 1 - Сравнение работы программы с lock-free структурой данных и структуры данных с грубой синхронизацией в зависимости от количества итераций сложения.

Количество потоков потребителей/производителей	Время работы, с	
	Структура данных с грубой синхронизацией	Lock-free структура данных
12	0.007490	0.010770
25	0.013388	0.098113
30	0.009422	0.010617
1000	0.215323	0.394122
10000	1.979846	2.737439

Были сравнены две структуры в зависимости от количества потоков производителей/потребителей при сложении 12 пар матриц. Результаты представлены в таблице 2.

Таблица 2 - Сравнение работы программы с lock-free структурой данных и структуры данных с грубой синхронизацией в зависимости от количества потоков производителей/потребителей.

Количество потоков потребителей/производителей	Время работы, с	
	Структура данных с грубой синхронизацией	Lock-free структура данных
2	0.005248	0.005402
3	0.005872	0.006938
4	0.006261	0.007919

Работа с файлом.

Для обеспечения взаимно исключающего доступа к файлу был использован std::unique_lock. В него передается std::mutex file_mtx, который нужно получить, чтобы писать в файл. Так как замок может быть единовременно только у одного потока, то только один поток можем писать в файл. Так гарантируется целостность данных при записи в файл.

Выводы.

В ходе выполнения лабораторной работы было написана программа на языке программирования C++ для попарного сложения потока матриц, использующая в качестве структуры данных lock-free очередь.