Задание на защиту лабораторной работы 4.
Подготовил Титов Дмитрий, Б22-505

Примеры полезных алгоритмов не требующих блокировок

1. Параллельное сканирование (prefix sum)

Задача:

Вычисление префиксных сумм (например, массив [1, 2, 3, 4] преобразуется в [1, 3, 6, 10]).

Описание алгоритма:

Метод основан на подходе с использованием дерева, который позволяет избежать синхронизации через блокировки.

1. **Фаза подъёма по дереву:**
    - Каждый поток параллельно обрабатывает определенную часть массива, суммируя элементы с шагом 2^i, где i — текущий уровень дерева.
    - Потоки взаимодействуют только с соседними индексами в своей области.
2. **Фаза спуска:**
    - Потоки пересчитывают значения, начиная с корня дерева, чтобы заполнить префиксные суммы.

Преимущества:

- Отсутствие необходимости в блокировках, так как каждый поток работает только со своими частями.
- Локальные данные остаются изолированными.

2. Неблокирующая сортировка (параллельный merge sort)

Задача:

Сортировка массива с использованием разделения данных между потоками.

Описание алгоритма:

1. Делим массив на части, равные числу потоков.
2. Каждый поток сортирует свою часть (например, с помощью встроенной `std::sort`).
3. Используем алгоритм слияния (merge), где каждая пара потоков объединяет свои отсортированные части.
4. На следующем шаге число потоков уменьшается вдвое, пока не останется один поток.

Преимущества:

- Потоки работают независимо на этапе сортировки.
- На этапе слияния данные каждого потока не пересекаются, что исключает необходимость в блокировках.

3. Параллельный подсчет частоты (Histogram)

Задача:

Построение гистограммы частот элементов в массиве.

Описание алгоритма:

1. Создаем локальные гистограммы для каждого потока.
2. Потоки обрабатывают свои части массива и обновляют локальные гистограммы.
3. После завершения параллельной фазы объединяем локальные гистограммы в одну глобальную.

Преимущества:

- Локальные гистограммы позволяют избежать блокировок.
- Конфликты записи минимизированы, так как каждая запись происходит в локальную память.

4. Параллельный поиск максимального элемента

При реализации лабораторной работы №4, на практике удалось показать, что алгоритм поиска максимума, плохо себя показывает при использовании блокировок.

Задача:

Поиск максимального значения в массиве.

Описание алгоритма:

1. Делим массив на части, равные числу потоков.
2. Каждый поток ищет максимальное значение в своей части.
3. Затем сравниваем результаты

Преимущества:

- Отсутствие блокировок за счет использования встроенной редукции.
- Высокая производительность за счёт минимального взаимодействия между потоками.

Полезные алгоритмы, где невозможно избежать блокировок

1. Параллельная очередь с несколькими производителями и потребителями (Producer-Consumer Problem)

Задача:

Реализация очереди, в которую несколько потоков добавляют элементы (producers), а другие потоки извлекают элементы (consumers).

Описание алгоритма:

1. Потоки производителей добавляют элементы в очередь.
2. Потоки потребителей извлекают элементы из очереди.
3. Блокировки используются для защиты операций добавления и удаления, чтобы избежать гонок данных.

Реализация:

- Используется мьютекс (`omp_set_lock` или `std::mutex`) для синхронизации доступа к очереди.
- Условные переменные (`std::condition_variable`) могут применяться для ожидания, если очередь пуста или заполнена.

2. Параллельное обновление глобальной таблицы (Hash Table или Map)

Задача:

Обновление общей таблицы значений несколькими потоками.

Описание алгоритма:

1. Потоки читают и обновляют ключи таблицы на основе вычислений.
2. Блокировки используются для защиты операций вставки, удаления и изменения.

Реализация:

- Мьютексы или спинлоки защищают доступ к частям таблицы.
- Возможно использование сегментирования таблицы (lock striping) для уменьшения конкуренции между потоками.

3. Параллельное построение дерева (например, бинарного дерева поиска)

Задача:

Добавление и удаление элементов в бинарное дерево несколькими потоками.

Описание алгоритма:

1. Каждый поток добавляет новый элемент или удаляет существующий.
2. Для поддержания структуры дерева требуется синхронизация.

Реализация:

- Используются мьютексы для защиты узлов дерева.
- Для оптимизации можно применять блокировки только на необходимой ветке дерева.

4. Параллельная обработка логов с общим счётчиком

Задача:

Анализ лог-файлов несколькими потоками, при этом общий счётчик событий синхронизируется между потоками.

Описание алгоритма:

1. Потоки читают свои части логов и обновляют общий счётчик.
2. Для корректного обновления общего счётчика используется блокировка.

Реализация:

- Атомарные операции или мьютексы используются для защиты счётчика.
- Если объём данных большой, можно использовать локальные копии счётчиков с последующим объединением.

5. Параллельное добавление данных в общую коллекцию

Задача:

Множество потоков добавляют элементы в общую коллекцию, например, вектор или список.

Описание алгоритма:

1. Каждый поток вычисляет свой элемент.
2. Для предотвращения конфликтов при добавлении используется блокировка.

Реализация:

- Используются блокировки или атомарные операции для управления доступом к коллекции.

6. Параллельный поиск с динамическим распределением задач

Задача:

Найти элемент в большом массиве, при этом задачи распределяются динамически.

Описание алгоритма:

1. Каждый поток выбирает блок элементов для обработки.
2. Глобальная переменная-флаг указывает, найден ли элемент.
3. Блокировки или атомарные операции используются для обновления флага.