Государственное образовательное учреждение высшего
профессионального образования
“Московский государственный технический университет имени
Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 5

Конвейер

Гибадулин О.Н.
Студент группы ИУ7-52

2019 г.

Содержание

1 Аналитический раздел **3**
1.1 Конвейерная обработка данных 3
1.2 Вывод 3

2 Конструкторский раздел **4**
2.1 Разработка Алгоритма 4
2.2 Вывод 4

3 Технологический раздел **5**
3.1 Требования к программному обеспечению 5
3.2 Средства реализации 5
3.3 Листинг кода 5
3.4 Вывод 7

4 Экспериментальный раздел **8**
4.1 Сравнительное исследование 8
4.2 Вывод 9

Введение

Если задача заключается в применении одной последовательности операций ко многим независимым элементам данных, то можно организовать распараллеленный конвейер.

В данной работе требуется рассмотреть применение конвейерной обработки данных.

Цель работы: изучение конвейерной обработки данных.

Задачи работы:

1. разработка и реализация алгоритмов;
2. исследование работы алгоритма при различных параметрах;
3. описание и обоснование полученных результатов.

1 Аналитический раздел

В данном разделе будет описаны принципы конвейерной обработки данных.

1.1 Конвейерная обработка данных

Если задача заключается в применении одной последовательности операций ко многим независимым элементам данных, то можно организовать распараллеленный конвейер. Здесь можно провести аналогию с физическим конвейером: данные поступают с одного конца, подвергаются ряду операций и выходят с другого конца.

Для того, чтобы распределить работу по принципу конвейерной обработки данных, следует создать отдельный поток для каждого участка конвейера, то есть для каждой операции. По завершении операции элемент данных помещается в очередь, откуда его забирает следующий поток. В результате поток, выполняющий первую операцию, сможет приступить к обработке следующего элемента, пока второй поток трудится над первым элементом. Конвейеры хороши также тогда, когда каждая операция занимает много времени; распределяя между потоками задачи, а не данные, мы изменяем качественные показатели производительности[2].

1.2 Вывод

В данном разделе были описаны принципы конвейерной обработки данных.

2 Конструкторский раздел

В данном разделе в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будет рассмотрен способ организации конвейера.

2.1 Разработка Алгоритма

В данном пункте представлен способ организации конвейера, реализованный в данной работе.

Конвейер состоит из 3-х потоков для 3-х последовательных задач, 4-х очередей и 4 мьютексов, позволяющие реализовать синхронную работу с очередями.

Перед началом работы конвейера создаётся массив объектов, каждый из которых имеет свой индентификатор. Из этого массива с определённым промежутком времени объекты помещаются в 1-ю очередь, с которой начинается процесс обработки. Если какая-либо из очередей не является пустой и поток отвечающий за эту очередь свободен, то из данной очереди достаётся верхний объект, который отправляется на обратку в этот поток.

При помещении объекта в очередь или извлечении объекта из неё, происходит захват мьютекса, который не позволяет другим потокам изменить структуру очереди при её использовании. После каждой обработки объект помещается в следующую очередь или пул готовых объектов. По завершению работу конвейера, все объекты лежат в пуле готовых объектов.

2.2 Вывод

В данном разделе был рассмотрен способ организации конвейера.

3 Технологический раздел

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать конвейерную обработку данных. Пользователь должен иметь возможность ввода количества объектов, которые будут обырабатываться конвейером.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++[2], поддерживающий нативные потоки. Для реализации многопоточности была использована стандратная библиотека thred[2]. Проет был выполнен в среде XCode[3]. Для измерения времени в милисекундах была использована библиотека chrono[2].

3.3 Листинг кода

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка C++ было разработано программное обеспечение, содержащее реализацию конвейерной обработки данных. В данном пункте приведён листиннг этой реализации.

Листинг 1. Код реализации конвейерной обработки данных

```
        void doObjectParallelWork(Object curObject, queue
            <Object>& queue, size_t queueNum, mutex&
            mutex) {
        size_t deltaTime = getRandDeltaTime();
        this_thread::sleep_for(chrono::milliseconds(
            averegeTime + deltaTime));

        mutex.lock();
        queue.push(curObject);
        mutex.unlock();

        size_t end = getCurTime();
        fileMutex.lock();
        resTimeFile << "Object #" << curObject.number <<
             " from Queue #" << queueNum << ": STOP - "
            << end << endl;
        fileMutex.unlock();
        }
```

```
void executeParallel() {
resTimeFile.open("ParallelConveyor.txt");

queue <Object> objectsGenerator;

for (size_t i = 0; i < elementsCount; ++i) {
objectsGenerator.push(Object(i + 1));
}

vector <thread> threads(3);
vector <queue <Object> > queues(3);
queue <Object> objectsPool;
vector <mutex> mutexes(4);
size_t prevTime = getCurTime() - delayTime;

while (objectsPool.size() != elementsCount) {
size_t curTime = getCurTime();

if (!objectsGenerator.empty() && prevTime +
    delayTime < curTime) {
Object curObject = objectsGenerator.front();
objectsGenerator.pop();
queues[0].push(curObject);

prevTime = getCurTime();
}

for (int i = 0; i < queuesCount; ++i) {
if (threads[i].joinable()) {
threads[i].join();
}
if (!queues[i].empty() && !threads[i].joinable())
     {
mutexes[i].lock();
Object curObject = queues[i].front();
queues[i].pop();
mutexes[i].unlock();

size_t start = getCurTime();
fileMutex.lock();
resTimeFile << "Object #" << curObject.number <<
     " from Queue #" << i << ": START - " << start
     << endl;
fileMutex.unlock();

if (i == queuesCount - 1) {
threads[i] = thread(&Conveyor::
    doObjectParallelWork, this, curObject, ref(
    objectsPool), i, ref(mutexes[i + 1]));
```

```
        }
        else {
        threads[i] = thread(&Conveyor::
            doObjectParallelWork, this, curObject, ref(
            queues[i + 1]), i, ref(mutexes[i + 1]));
        }
        }
        }
        }

        for (int i = 0; i < queuesCount; ++i) {
        if (threads[i].joinable()) {
        threads[i].join();
        }
        }

        resTimeFile.close();
        }
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлен листинг реализации конвейерной обработки данных.

4 Экспериментальный раздел

В данном разделе будет проведен сравнительный анализ работы последовательной и конвейерной обработки данных при различных параметрах.

4.1 Сравнительное исследование

Для сравнения последовательной и конвейерной обработки замеры времени выполнялись при различном количестве обрабатываемых объектов (табл. 4.1.2, рис. 4.1.1).

Все замеры были произведены на 2-х ядерном процессоре 2,7 GHz Intel Core i5 с паматью 8 ГБ 1867 MHz DDR3.

Таблица 4.1.1. Сравнение среднего времени простоя объектов на каждой очереди в милисекундах

Количество объектов	1-ая очередь	2-ая очередь	3-ая очередь
100	3	2	3
200	2	2	2
300	3	3	2
400	3	3	3
500	2	3	3

Таблица 4.1.2. Сравнение времени работы последовательной и конвейерной обработки в милисекундах при различном количестве обрабатываемых объектов

Количество объектов	Последедовательная обработка	Конвейер
100	2598	928
200	5172	1839
300	7266	2704
400	9663	3584
500	12412	4512
600	14555	5354
700	17323	6038
800	19334	7000
900	22168	8155
1000	24551	8870

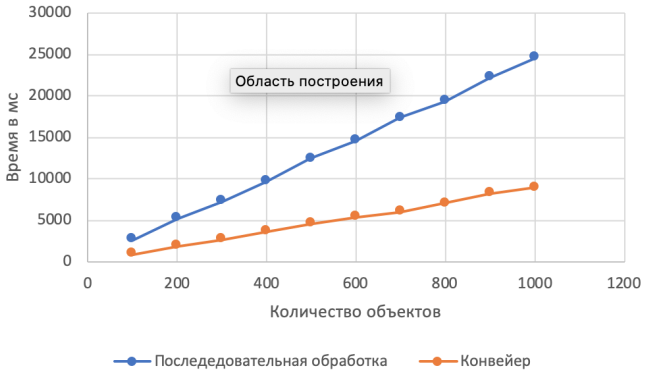

Рисунок 4.1.1. – график зависимости времени работы последовательной и конвейерной обработки при различном количестве обрабатываемых объектов

Из данной таблицы можно сделать вывод о том, что конвейерная обработка работает значительно быстрее чем последедовательная обработка (на 74% при количестве объектов равное 1000).

4.2 Вывод

В данном разделе был проведен сравнительный анализ работы последовательной и конвейерной обработки данных при различных параметрах, из которого можно сделать вывод, что конвейерная обработка работает значительно быстрее чем последедовательная обработка (на 74% при количестве объектов равное 1000).

Заключение

В ходе выполнения данной лабораторной работы были изучены принципы конвейерной обработки. Было проведено исследование работы алгоритма при различных параметрах, показавшее, что конвейерная обработка работает значительно быстрее чем последедовательная обработка (на 74% при количестве объектов равное 1000).

В аналитическом разделе было дано описание алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура. В конструкторском разделе были разработаны алгоритмы, описанные в аналитическом разделе. В технологическом разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы. В экспериментальном разеде было проведено сравнительно исследование, из которого следует, что алгоритм Бойера-Мура работает с меньшим количеством сравнений (для строки "erererererasfer"и подстроки "asf"на 50%) по сравнению с алгоритмом Кнута-Морриса-Пратта и исследование корректности работы алгоритмов.

Список литературы

[1] Энтони Уильямс Параллельное программирование на C++ в действии Практика разработки многопоточных программ ISBN: 978-5-94074-537-2

[2] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)

[3] Apple «Apple Developer Documentation» [Электронный ресурс]. – Режим доступа: https://developer.apple.com/documentation/, свободный. (Дата обращения: 29.09.2019 г.)