Лабораторная работа №5
По курсу: "Анализ алгоритмов"
По теме: "Конвейерная обработка"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Целью данной лабораторной работы является получение навыка организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Задачи работы

Задачи данной работы:

1. Поставить задачу стадийной обработки;
2. спроектировать программное обеспечение, реализующее конвейерную обработку;
3. описать реализацию программного обеспечения;
4. провести тестирование программного обеспечения по времени обработки;
5. интерпретировать данные лог файла.

1 Аналитическая часть

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени).

Один из самых простых и наиболее распространенных способов повышения быстродействия процессоров — конвейеризация процесса вычислений.

Конвейеризация – это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i+1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду.

На рисунке 1 показан простой пятиуровневый конвейер в RISC-процессорах. Условные обозначения:

1. IF (англ. Instruction Fetch) — получение инструкции;
2. ID (англ. Instruction Decode) — раскодирование инструкции;
3. EX (англ. Execute) — выполнение;
4. MEM (англ. Memory access) — доступ к памяти;
5. WB (англ. Register write back) — запись в регистр.

Вертикальная ось — последовательные независимые инструкции, горизонтальная — время. Зелёная колонка описывает состояние процессора в один момент времени, в ней самая ранняя, верхняя инструкция уже находится в состоянии записи в регистр, а самая последняя, нижняя инструкция — только в процессе чтения.

На рис. 1 представлена схема конвейера.

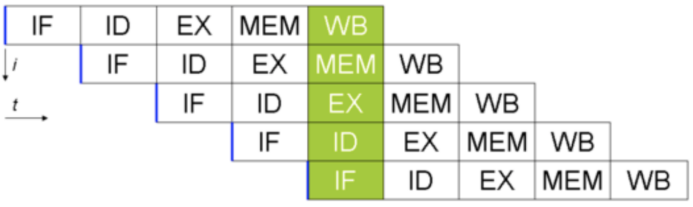

Рис. 1: Простой пятиуровневый конвейер в RISC-процессорах

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько

стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при последовательном выполнении каждой инструкции от начала до конца.

1.1 Оценка производительности конвейера

Пусть задана операция, выполнение которой разбито на n последовательных этапов. При последовательном их выполнении операция выполняется за время

$$\tau_e = \sum_{i=1}^{n} \tau_i \tag{1}$$

где

n – количество последовательных этапов;

τ_i – время выполнения i-го этапа;

Быстродействие одного процессора, выполняющего только эту операцию, составит

$$S_e = \frac{1}{\tau_e} = \frac{1}{\sum\limits_{i=1}^{n} \tau_i} \tag{2}$$

где

τ_e – время выполнения одной операции;

n – количество последовательных этапов;

τ_i – время выполнения i-го этапа;

Выберем время такта — величину $t_T = max\sum\limits_{i=1}^{n}(\tau_i)$ и потребуем при разбиении на этапы, чтобы для любого i = 1, ... , n выполнялось условие $(\tau_i + \tau_{i+1}) mod n = \tau_T$. То есть чтобы никакие два последовательных этапа (включая конец и новое начало операции) не могли быть выполнены за время одного такта.

Максимальное быстродействие процессора при полной загрузке конвейера составляет

$$S_{max} = \frac{1}{\tau_T} \tag{3}$$

где

τ_T – выбранное нами время такта;

Число n — количество уровней конвейера, или глубина перекрытия, так как каждый такт на конвейере параллельно выполняются n операций. Чем

больше число уровней (станций), тем больший выигрыш в быстродействии может быть получен.

Известна оценка

$$\frac{n}{n/2} \leq \frac{S_{max}}{S_e} \leq n \tag{4}$$

где

S_{max} – максимальное быстродействие процессора при полной загрузке конвейера;

S_e – стандартное быстродействие процессора;

n – количество этапов.

то есть выигрыш в быстродействии получается от n/2 до n раз [2].

Реальный выигрыш в быстродействии оказывается всегда меньше, чем указанный выше, поскольку:

1) некоторые операции, например, над целыми, могут выполняться за меньшее количество этапов, чем другие арифметические операции. Тогда отдельные станции конвейера будут простаивать;

2) при выполнении некоторых операций на определённых этапах могут требоваться результаты более поздних, ещё не выполненных этапов предыдущих операций. Приходится приостанавливать конвейер;

3) поток команд(первая ступень) порождает недостаточное количество операций для полной загрузки конвейера.

Генератор подает на вход конвейера (первый уровень) созданные им объекты. Далее на каждом уровне осуществляется обработка данных, занимающая определенное время. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на следующий (следующую ленту). Для организации работы каждой ленты будет использована очередь задач, которые должны обработаться на этой ленте. Лента будет работать, пока в ее очереди есть задачи.

Задачи попадают в очередь только в том случае, если данные уже были обработаны на предыдущем уровне или если их только сформировал генератор (для первой ленты). При этом время обработки на какой-либо ленте не должно сильно отличаться от времени обработки на остальных лентах, так как в противном случае возможны ситуации простоя одной или нескольких лент конвейера. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач.

1.2 Вывод

Был рассмотрен вычислительный конвейер

2 Конструкторская часть

В данном разделе будет представлена схема работы конвейера.

2.1 Разработка конвейера

Генератор подает на вход конвейера (первый уровень) созданные им объекты. Далее на каждом уровне осуществляется обработка данных, занимающая определенное время. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на следующий (следующую ленту). Для организации работы каждой ленты будет использована очередь задач, которые должны обработаться на этой ленте. Лента будет работать, пока в ее очереди есть задачи.

Задачи попадают в очередь только в том случае, если данные уже были обработаны на предыдущем уровне или если их только сформировал генератор (для первой ленты). При этом время обработки на какой-либо ленте не должно сильно отличаться от времени обработки на остальных лентах, так как в противном случае возможны ситуации простоя одной или нескольких лент конвейера. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач.

Реализованнй конвейер состоит из трех уровней. На каждом этапе конвейера алгоритм обработки информации заменен задержкой выполняемой программы по времени. Каждой ленте конвейера выделен отдельный поток. В главном потоке main запускаются все три рабочих потока - ленты конвейера: first_line, second_line, third_line. Для каждого потока есть своя очередь, в которой содержатся индекс выполняемой задачи и время в миллисекундах, необходимое на выполнение задачи (время задержки). Также в главном потоке генерируются входные данные, которые помещаются в очередь для первого потока. После завершения работы всех рабочих потоков проверяется массив результирующих элементов и выводится сообщение о результате работы конвейера. В рабочих потоках извлекается очередной элемент из соответствующей очереди, вносится запись в лог файл о начале обработки очередного элемента, выполняется задержка по времени. После этого новое значение помещается в очередь для следующего потока (или в массив обработанных значений если обработка происходит уже в третьем потоке) и далее в лог файл заносится запись о завершении обработки очередного элемента на определенной ленте (в определенном потоке). После окончания работы генератора в первую очередь записывается значение -1, что говорит о завершении работы конвейера. Это значение передается из очереди в очередь по лентам.

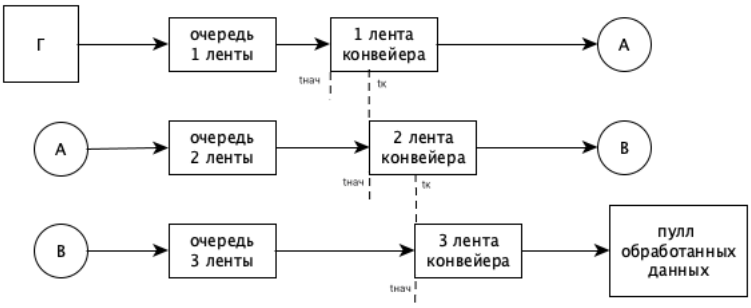

Рис. 2: Схема работы конвейера

2.2 Вывод

В данном разделе был кратко описан реализованный вычислительный конвейер.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

Для работы с потоками использовалась библиотека thread:

1. joinable - проверяет потенциальную возможность работы потока в параллельном контексте;
2. join - ожидает завершения потока.

Для реализации очереди выбрана библиотека queue:

1. front - предоставляет доступ к первому элементу;
2. back - предоставляет доступ к последнему элементу;
3. empty - проверяет отсутствие элементов в контейнере, используемом для реализации;
4. size - возвращает количество элементов в контейнере;
5. push -вставляет элемент в конец;
6. pop - удаляет первый элемент.

Так как в ходе работы программы возможно обращение разных потоков к одному участку памяти, используются объекты класса mutex:

1. lock - блокирует мьютекс, выполнение останавливается если мьютекс недоступен;
2. unlock - разблокирует мьютекс.

3.2 Листинги функций

В данном разделе представлены листинги реализованных алгоритмов.

В листинге 1 представлена основная функция программы.

Листинг 1: Функция main()

```
int main()
{
    auto start = chrono::high_resolution_clock::now();

    thread thread1 (first_line);
    thread thread2 (second_line);
    thread thread3 (third_line);
    fill_objects(4);

    for (int i = 0; i < input_objects.size(); i++)
    {
        mtx1.lock();
        queue1.push(input_objects[i]);
        mtx1.unlock();
        sleep(3);
    }

    cout << "Size: " << input_objects.size() << endl;

    if (thread1.joinable())
    {
        thread1.join();
    }

    if (thread2.joinable())
    {
        thread2.join();
    }

    if (thread3.joinable())
    {
        thread3.join();
    }

    cout << "All objects passed the conveyor" << endl;

    auto end = chrono::high_resolution_clock::now();
    auto dur = end - start;
    auto ms = chrono::duration_cast<std::chrono::seconds>(dur)
        .count();

    cout << "Time: " << ms << " sec" << endl;

    sort_output();
    ofstream file;
    file.open("log.txt");
    for (int i = 0; i < output_objects.size(); i++)
```

```
47      {
48          file << i << " " << output_objects[i].index <<
                output_objects[i].str;
49      }
50      file.close();
51      return 0;
52  }
```

В листинге 2 представлена первая лента конвейера.

Листинг 2: Вторая лента

```
1  void first_line(void)
2  {
3      bool finish = false;
4      while (!finish)
5      {
6          mtx1.lock();
7          if(!queue1.empty())
8          {
9              input obj = queue1.front();
10             if (obj.itemindex != -1)
11             {
12                 time(&obj.time);
13                 obj.index = 1;
14                 obj.str = " == start: ";
15                 resmtx.lock();
16                 output_objects.push_back(obj);
17                 resmtx.unlock();
18                 queue1.pop();
19                 sleep(1);
20                 mtx2.lock();
21                 queue2.push(obj);
22                 mtx2.unlock();
23                 obj.str = " == end: ";
24                 resmtx.lock();
25                 output_objects.push_back(obj);
26                 resmtx.unlock();
27             }
28             else
29             {
30                 mtx2.lock();
31                 queue2.push(obj);
32                 mtx2.unlock();
33                 finish = true;
34             }
35         }
36         mtx1.unlock();
37     }
38 }
```

В листинге 3 представлена вторая лента конвейера.

Листинг 3: Вторая лента

```
void second_line(void)
{
    bool finish = false;
    while (!finish)
    {
        mtx2.lock();
        if (!queue2.empty())
        {
            input obj = queue2.front();
            if (obj.itemindex != -1)
            {
                time(&obj.time);
                obj.index = 2;
                obj.str = " == start: ";
                resmtx.lock();
                output_objects.push_back(obj);
                resmtx.unlock();
                queue2.pop();
                sleep(1);
                mtx3.lock();
                queue3.push(obj);
                mtx3.unlock();
                obj.str = " == end: ";
                resmtx.lock();
                output_objects.push_back(obj);
                resmtx.unlock();
            }
            else
            {
                mtx3.lock();
                queue3.push(obj);
                mtx3.unlock();
                finish = true;
            }
        }
        mtx2.unlock();
    }
}
```

В листинге 4 представлена третья лента конвейера.

Листинг 4: Третья лента

```
void third_line(void)
{
    bool finish = false;
    while (!finish)
    {
        mtx3.lock();
        if (!queue3.empty())
        {
            input obj = queue3.front();
            if (obj.itemindex != -1)
            {
                time(&obj.time);
                obj.index = 3;
                obj.str = " == start: ";
                resmtx.lock();
                output_objects.push_back(obj);
                resmtx.unlock();
                queue3.pop();
                sleep(1);
                obj.str = " == end: ";
                resmtx.lock();
                output_objects.push_back(obj);
                resmtx.unlock();
            }
            else
            {
                finish = true;
            }
        }
        mtx3.unlock();
    }
}
```

3.3 Вывод

В данном разделе были представлены листинги лент конвейера.

4 Исследовательская часть

В данном разделе будет рассмотрен и интерпретирован лог файл.

4.1 Анализ лог файла

Для того, чтобы отследить работу потоков, необходимо получить лог файл, отражающий время начала работы ленты над очередным заданием и время окончания этой работы.

Рассмотрим лог файл для конвейера, работающего с 5 элементами.

```
22:42:31 line 1: start 0 item
22:42:32 line 1: end 0 item
22:42:32 line 2: start 0 item
22:42:32 line 1: start 1 item
22:42:35 line 2: end 0 item
22:42:35 line 3: start 0 item
22:42:35 line 1: end 1 item
22:42:35 line 1: start 2 item
22:42:35 line 2: start 1 item
22:42:37 line 3: end 0 item
22:42:38 line 2: end 1 item
22:42:38 line 3: start 1 item
22:42:38 line 1: end 2 item
22:42:38 line 2: start 2 item
22:42:38 line 1: start 3 item
22:42:40 line 3: end 1 item
22:42:41 line 2: end 2 item
22:42:41 line 3: start 2 item
22:42:41 line 1: end 3 item
22:42:41 line 2: start 3 item
22:42:41 line 1: start 4 item
22:42:43 line 3: end 2 item
22:42:44 line 3: start 3 item
22:42:44 line 2: end 3 item
22:42:44 line 1: end 4 item
22:42:44 line 2: start 4 item
22:42:46 line 3: end 3 item
22:42:47 line 2: end 4 item
22:42:47 line 3: start 4 item
22:42:49 line 3: end 4 item
```

Рис. 3: Лог файл

В первом столбце показано время записи, во втором номер ленты конвейера, в третьем начало или конец обработки и номер элемента, который обрабатывался.

Проанализировав лог файл, можно увидеть, что как только заканчивается обработка элемента на первой ленте, он тут же поступает наобработку на следующую ленту. Одновременно в это время первая лента начинает обрабатывать следующий элемент.

4.2 Время работы

В таблице 1 представлено время работы конвейера для различного числа объектов:

Таблица 1: Таблица результатов параметризации

Количество объектов	Время (секунды)
1	3
3	9
5	15
7	17

Время обработки одного входного элемента равно равно: 1 секунда * 3 ленты = 3 секунды. Для 7 входных элементов суммарное время конвейерной обработки становится меньше суммарного времени последовательной обработки: 17 секунд $<$ 21 секунды. Аналогичная ситуация возникает и при вхождении большего числа элементов.

4.3 Вывод

Был получен и проанализирован лог файл.

Заключение

В ходе выполнения лабораторной работы был изучен алгоритм работы конвейра с использованием методов распараллеливания процессов.

Список литературы

[1] Конвейерная обработка данных https://studfiles.net/preview/1083252/page:25/.