Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе № 5

Название: Многопоточная реализация конвейера

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) Д.Р. Жигалкин (И.О. Фамилия)

Преподователь _____ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Система конвейерной обработки – это система, основаная на разделении подлежащей выполнению задачи на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры или потока. При этом конвейеризацию можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Но не каждую задачу можно разделить на несколько ступеней, организовав передачу данных от одного этапа к следующему.

Целью данной лабораторной работы является реализация системы конвейерной обработки.

Задачи данной лабораторной работы:

1) описать алгоритмы конвейерной обработки;
2) реализовать алгоритмы конвейерной обработки;
3) провести замеры процессорного времени работы.

1 Аналитический раздел

В данном разделе будут рассмотрены основные теоритические понятия конвейерной обработки и параллельных вычислений.

1.1 Основные понятия

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.

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.

1.2 Оценка производительности конвейера

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

$$\tau_e = \sum_{i=1}^{n} \tau_i \tag{1.1}$$

где

n — количество последовательных этапов;

τ_i — время выполнения i-го этапа;

Быстродействие одного процессора, выполняющего только эту операцию, составит

$$S_e = \frac{1}{\tau_e} = \frac{1}{\sum\limits_{i=1}^{n} \tau_i} \tag{1.2}$$

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

$$S_{max} = \frac{1}{\tau_T} \tag{1.3}$$

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

$$\frac{n}{n/2} \leqslant \frac{S_{max}}{S_e} \leqslant n \tag{1.4}$$

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.

1.3 Вывод

В данном разделе были рассмотрены основы конвейерной обработки и технологии параллельного программирования.

2 Конструкторский раздел

В данном разделе будут рассмотрены описание системы, требования к функциональности ПО, определены способы тестирования и представлена схема алгоритма.

2.1 Описание системы

Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на независимые этапы и организацией параллельного выполнения во времени различных этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. Поэтому основой разработки конвейера является разбиение процесса на независимые этапы.

Конвейер состоит из трех лент, которые называются PreProc, Proc и PostProc. Каждый объект проходит три этапа обработки на каждой из лент. Объект представляет собой экземпляр специально созданного класса MyObject. В связи с тем, что одной из задач данной работы является проектирование ПО, реализующего конвейерную обработку (а не реализация каких-либо определенных алгоритмов), Объекты класса MyObject по сути являются абрстракцией объектов, которые обрабатывались бы в реальной конкретной системе с конвейерными вычислениями. Три ленты конвейера представляют собой три отдельных класса, каждый из которых имеет свое заранее заданное время обработки. Ленты лишь имитируют обработку объектов, приостанавливая выполнение программы на время обработки, заданное для каждой ленты. Каждая лента запускается в отдельном потоке.

В программе N объектов генерируются и помещаются в очередь первой ленты (PreProc). После того, как i-й объект (i = 1, ..., N) был обработан на первой ленте, он передается в очередь второй ленты (Proc). После обработки на второй ленте объект передается в очередь третьей ленты (PostProc). После обработки на третьей ленты объект помещается в контейнер обработанных объектов. Объект считается обработанным, если он прошел все три ленты конвейера. Эти действия выполняются для каждого из N сгенерированных объектов.

Для ленты PreProc время обработки было установлено в 50, для ленты Proc - 70, для ленты PostProc - 20.

Ниже изображена схема алгоритма обработки объектов класса MyObject (рисунок 2.1).

Принцип обработки объектов на ленте PreProcessing (рисунок 2.2).

Принцип обработки объектов на ленте Processing (рисунок 2.3).

Принцип обработки объектов на ленте PostProcessing (рисунок 2.4).

2.2 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения:

1) предоставить возможность ввода количества генерируемых элементов в системе;
2) обеспечить вывод времени получения и отправки элемента конвейера.

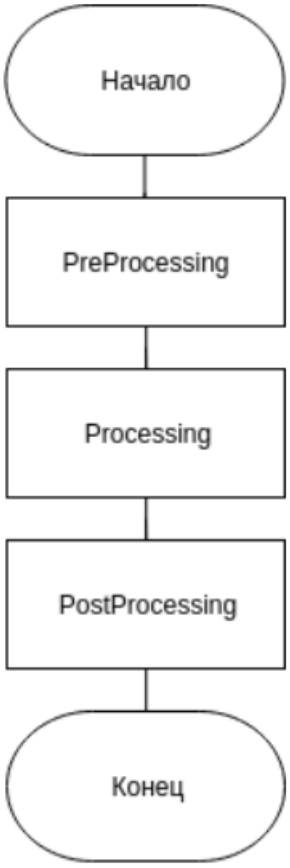

Рисунок 2.1 — Алгоритм обработки объектов класса MyObject

Кроме этого, должен быть создан log-file, куда должно быть записано общее время работы конвейера.

2.3 Тесты

Тестирование ПО будет проводиться методом чёрного ящика.

2.4 Вывод

В данном разделе были рассмотрены схема алгоритмов обработки элементов линии конвейера и описаны требования к функциональности ПО.

Рисунок 2.2 — Алгоритм обработки объектов на ленте PreProcessing

Рисунок 2.3 — Алгоритм обработки объектов на ленте Processing

Рисунок 2.4 — Алгоритм обработки объектов на ленте PostProcessing

3 Технологический раздел

В данном разделе будут выбраны средства реализации ПО и представлен листинг кода.

3.1 Средства реализации

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

— язык C++ имеет высокую вычислительную производительность;

— язык C++ поддерживает различные стили программирования;

— в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Для замеров времени использовался методы restart() и elapsed() класса QTime. Метод elapsed() возвращает количество миллисекунд, прошедших с момента последнего вызова start() или restart().

Для работы с мьютексами и потоками в лабораторной работе используются классы QMutex, QThread фреймворка Qt.

Для реализации очереди использовался класс queue из стандартной библиотеки C++.

3.2 Листинг программы

Ниже представлены листинги кода системы:

1) запуск конвейера (листинг 3.1);
2) реализация класса MyObject (листинг 3.2);
3) реализация класса PreProcessing (листинг 3.3);
4) реализация класса Processing (листинг 3.4);
5) реализация класса PostProcessing (листинг 3.5).

Листинг 3.1 — Запуск конвейера

```
1  const int COUNT = 10;
2
3  const string LOG_FILE = "/times";
4
5  int main(int argc, char *argv[])
6  {
7      QCoreApplication a(argc, argv);
8
9      QTime timer;
10     timer.restart();
11
12     int start_time = timer.elapsed();
```

```
13
14      vector<MyObject> dump;
15
16      QMutex *mutex2 = new QMutex;
17      QMutex *mutex3 = new QMutex;
18
19      PostProcessing *postproc = new PostProcessing(COUNT, &timer, &dump, mutex2);
20      Processing *proc = new Processing(COUNT, &timer, postproc, mutex2, mutex3);
21      PreProcessing *preproc = new PreProcessing(COUNT, &timer, proc, mutex3);
22
23      for (int i = 0; i < COUNT; ++i)
24      {
25          MyObject obj(i);
26          preproc->addToQueue(obj);
27      }
28
29      vector<thread> threads;
30      threads.push_back(thread(&PreProcessing::process, preproc));
31      threads.push_back(thread(&Processing::process, proc));
32      threads.push_back(thread(&PostProcessing::process, postproc));
33
34      for (unsigned int i = 0; i <  threads.size(); ++i)
35      {
36          if (threads.at(i).joinable())
37              threads.at(i).join();
38      }
39
40      int total_time = timer.elapsed() - start\_time;
41
42      ofstream fout(LOG_FILE);
43      if (fout.is_open())
44      {
45          for (unsigned int i = 0; i < dump.size(); ++i)
46              dump.at(i).timesToFile(fout);
47          fout << "Total time: " << total_time << endl;
48          cout << "Results in file: " << LOG_FILE << endl;
49      }
50      else
51          cout << "Unable to open file" << LOG_FILE << endl;
52      fout.close();
53
54
55      delete mutex2;
56      delete mutex3;
57      delete preproc;
58      delete proc;
59      delete postproc;
```

```
60
61      return 0;
62  }
```

Листинг 3.2 — Реализация класса MyObject

```
1   MyObject::MyObject(int id)
2   {
3       this->_id = id;
4   }
5
6   void MyObject::setTime(int time)
7   {
8       this->_times.push_back(time);
9   }
10
11  void MyObject::printTimes()
12  {
13      cout << "Object" << _id << "\t\t";
14      for (unsigned int i = 0; i < _times.size(); ++i)
15          cout << _times.at(i) << "  ";
16      cout << endl;
17  }
18
19  void MyObject::timesToFile(ofstream &fout)
20  {
21      fout << "Object" << _id << "\t\t";
22      for (unsigned int i = 0; i < _times.size(); ++i)
23      {
24          fout << _times.at(i) << "  ";
25      }
26      fout << endl;
27  }
```

Листинг 3.3 — Реализация класса PreProcessing

```
1   {
2       this->_count = count;
3       this->_timer = timer;
4       this->_proc = p;
5       this->_mutex2 = mutex2;
6   }
7
8   void PreProcessing::addToQueue(MyObject obj)
9   {
10      _queue.push(obj);
11  }
12
```

```
13 void PreProcessing::process()
14 {
15     while (preprocessed != _count)
16     {
17         if (_queue.size() != 0)
18         {
19             // cout << "PreProcessing" << _timer->elapsed() << endl;
20             MyObject obj = _queue.front();
21             QThread thread;
22             obj.setTime(_timer->elapsed());
23             thread.msleep(_PROCESSING_TIME);
24             obj.setTime(_timer->elapsed());
25             _queue.pop();
26             _mutex2->lock();
27             _proc->addToQueue(obj);
28             _mutex2->unlock();
29             preprocessed++;
30         }
31     }
32 }
```

Листинг 3.4 — Реализация класса Processing

```
1  Processing::Processing(int count, QTime *timer, PostProcessing *p, QMutex *mutex2,
       QMutex *mutex3)
2  {
3      this->_count = count;
4      this->_timer = timer;
5      this->_proc = p;
6      this->_mutex2 = mutex2;
7      this->_mutex3 = mutex3;
8  }
9
10 void Processing::addToQueue(MyObject obj)
11 {
12     _queue.push(obj);
13 }
14
15 void Processing::process()
16 {
17     while (processed != _count)
18     {
19         if (_queue.size() != 0)
20         {
21             // cout << "Processing" << _timer->elapsed() << endl;
22             MyObject obj = _queue.front();
23             QThread thread;
```

```
24              obj.setTime(_timer->elapsed());
25              thread.msleep(_PROCESSING_TIME);
26              obj.setTime(_timer->elapsed());
27
28              _mutex2->lock();
29              _queue.pop();
30              _mutex2->unlock();
31
32              _mutex3->lock();
33              _proc->addToQueue(obj);
34              _mutex3->unlock();
35
36              processed++;
37          }
38      }
39  }
```

Листинг 3.5 — Реализация класса PostProcessing

```
1   PostProcessing::PostProcessing(int count, QTime *timer, vector<MyObject> *dump,
        QMutex *mutex3)
2   {
3       this->_count = count;
4       this->_timer = timer;
5       this->_dump = dump;
6       this->_mutex3 = mutex3;
7   }
8
9   void PostProcessing::addToQueue(MyObject obj)
10  {
11      _queue.push(obj);
12  }
13
14  void PostProcessing::process()
15  {
16      while (postprocessed != _count)
17      {
18          if (_queue.size() != 0)
19          {
20              // cout << "PostProcessing" << _timer->elapsed() << endl;
21              MyObject obj = _queue.front();
22              QThread thread;
23              obj.setTime(_timer->elapsed());
24              thread.msleep(_PROCESSING_TIME);
25              obj.setTime(_timer->elapsed());
26
27              _mutex3->lock();
```

```
28                  _queue.pop();
29                  _mutex3->unlock();
30
31                  _dump->push_back(obj);
32                  postprocessed++;
33              }
34          }
35      }
```

3.3 Тестирование

На рисунке 3.1 представлен результат работы программы. Проверялась работа и корректное завершение программы для разного количества объектов: для 1 - 5 объектов, от 10 до 100 с шагом 10

```
Object0     19  69  69  139  139  160
Object1     69  119  139  210  210  230
Object2     119  169  210  280  280  300
Object3     169  219  280  350  350  371
Object4     219  269  350  420  420  441
Object5     269  319  420  491  491  511
Object6     320  370  491  561  561  581
Object7     370  420  561  631  631  651
Object8     420  470  631  701  701  721
Object9     470  520  701  771  771  791
Total time: 772
```

Рисунок 3.1 — Результаты тестирования алгоритмов.

4 Экспериментальный раздел

В данном разделе будут проведены эксперименты для проведения сравнительного анализа на основе замеров времени работы алгоритмов.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данной работы был проведен эксперимент по вычислению времени работы системы в линейной и конвейерной реализациях (график 4.1).

Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i3-8130U CPU 2.20 GHz [1] под управлением Windows 10 с 8 Гб оперативной памяти.

В ходе эксперимента по замеру времени работы в линейной и конвейерной реализациях было установлено, что конвейерная модель обрабатывает элементы в ≈ 2.6 раза быстрее, чем линейная. Это объясняется тем, что одновременно обрабатываются разные элементы на разных этапах.

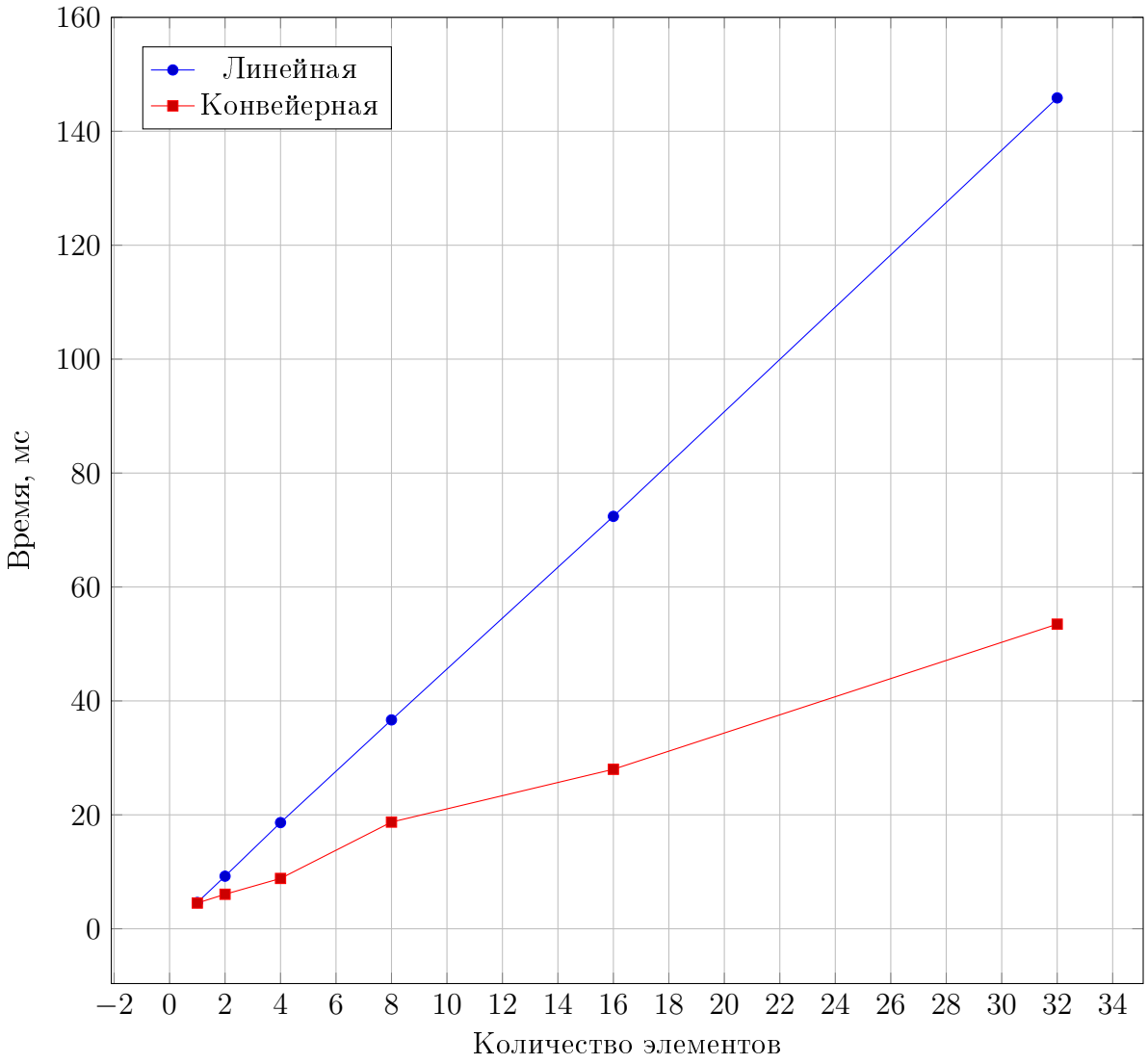

Рисунок 4.1 — График зависимости времени работы от модели системы

Заключение

В ходе выполнения лабораторной работы были описаны и реализованы алгоритмы конвейерной обработки. Разработанный конвейер, с параллельно работающими линиями, позволил ускорить работу программы в 2,6 раз по сравнению с линейной реализацией. Однако если одна из стадий намного более трудоемкая, чем остальные, то конвейерная обработка становится неэффективной, так как производительность всей программы будет упираться в производительность этой стадии, и разница между обычной обработкой и конвейерной будет малозаметна. В таком случае можно либо разбить трудоемкую стадию на набор менее трудоемких, либо выбрать другой алгоритм, либо отказаться от конвейерной обработки.

Список использованных источников

1. Intel® Core™ i3-8130U Processor. // [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/137977/intel-core-i3-8130u-processor-4m-cache-up-to-3-40-ghz.html, (дата обращения: 10.12.2020).