МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Вычислительный конвейер

Работу выполнила: Овчинникова Анастасия, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Целью данной работы является получение навыка организации асинхронного взаимодействия между потоками на примере конвейерных вычислений.

Задачи лабораторной работы:

1. поставить задачу стадийной обработки данных;
2. спроектировать ПО, реализующее конвейерную обработку;
3. описать реализацию;
4. провести тестирование ПО (по времени обработки) на основании журнала(лога);
5. интерпретировать данные лога.

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.

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

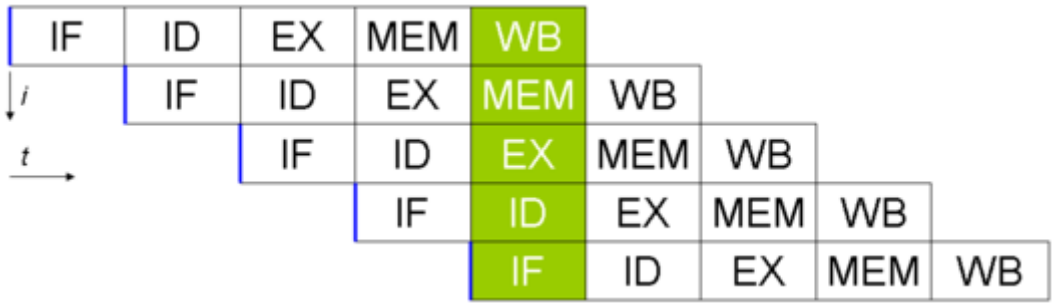

Рис. 1: Простой пятиуровневый конвейер в RISC-процессорах [1]

получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при последовательном выполнении каждой инструкции от начала до конца.

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.

Конструкторская часть

В данном разделе будут описаны принципы работы выбранных решений и их блоксхемы.

Разработка реализаций алгоритмов

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 называеются PreProcessing, Processing и PostProcessing. 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 (PreProcessing). После того, как i-й объект (i = 1, ..., N) был обработан на первой ленте, он передается в очередь второй ленты (Processing). После обработки на второй ленте объект передается в очередь третьей ленты (PostProcessing). 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 PreProcessing время обработки было установлено в 50, для ленты Processing - 70, для ленты PostProcessing - 20.

На рисунке 2 изображена схема алгоритма обработки объектов класса MyObject.

Принцип обработки объектов на ленте PreProcessing изображен на рисунке 3.

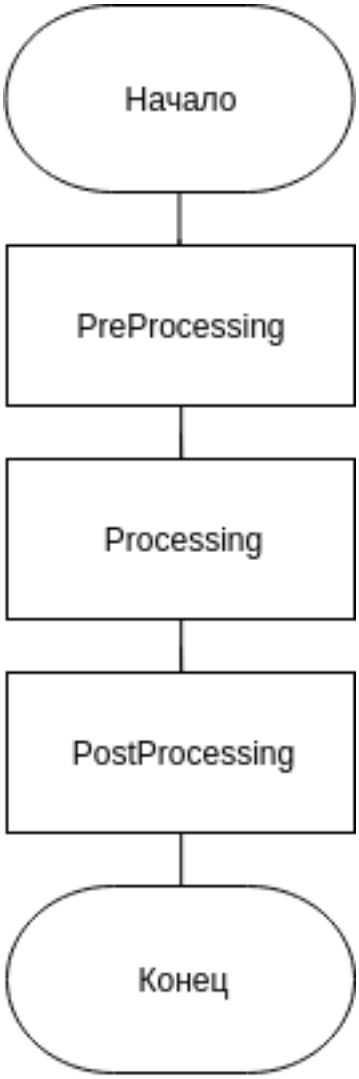

Рис. 2: Алгоритм обработки объектов класса MyObject

Рис. 3: Алгоритм обработки объектов на ленте PreProcessing

Принцип обработки объектов на ленте Processing изображен на рисунке 4.

Принцип обработки объектов на ленте PostProcessing изображен на рисунке 5.

Требования к программе

Требования к вводу:
Программа не имеет входных данных.

Требования к программе:
На выходе программа предоставляет лог-файл, где для каждого обработанного на конвейете объекта записано:

1. время начала обработки на первой ленте и время конца обработки на первой ленте;

2. время начала обработки на первой ленте и время конца обработки на второй ленте;

3. время начала обработки на первой ленте и время конца обработки на третьей ленте.

Кроме этого, в лог-файл долнжо быть записано общее время работы конвейера.

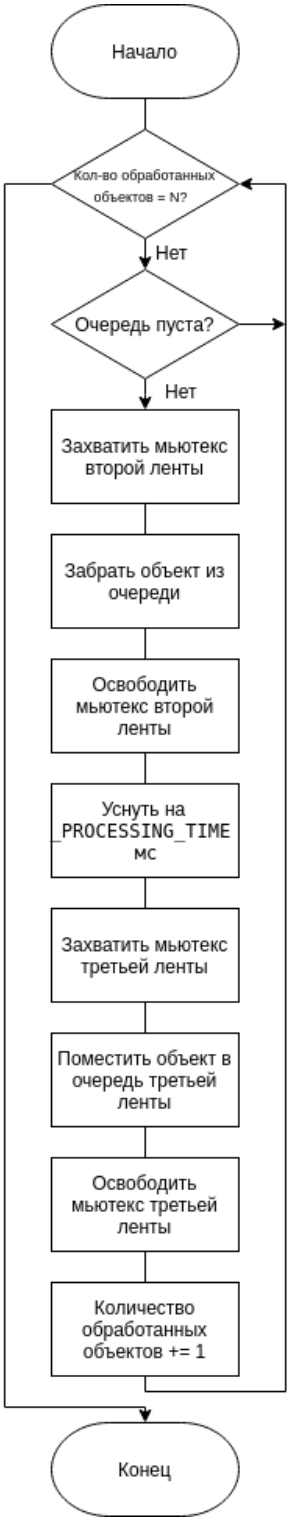

Рис. 4: Алгоритм обработки объектов на ленте Processing

Рис. 5: Алгоритм обработки объектов на ленте PostProcessing

Технологическая часть

В данном разделе будут определены средства реализации и приведен листинг кода.

Выбор языка программирования

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++.

Сведения о модулях программы

Программа состоит из следующих файлов:

- myobject.h, myobject.cpp - заголовочный файл и файл, в котором расположена реализация обрабатываемых объектов;
- main.cpp - главный файл программы, в котором расположен запуск конвейера;
- preprocessing.h, preprocessing.cpp - файл и заголовочный файл, в котором расположена реализация первой ленты конвейера;

- processing.h, processing.cpp - файл и заголовочный файл, в котором расположена реализация второй ленты конвейера;
- postprocessing.h, postprocessing.cpp - файл и заголовочный файл, в котором расположена реализация третьей ленты конвейера.

Листинги кода алгоритмов

Листинг 1: Запуск конвейера

```
1  const int COUNT = 10;
2
3  const string LOG_FILE = "/home/syzygy/lab5_aa/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 2: Объявление класса MyObject

```
1  class MyObject
2  {
3  public:
4      MyObject(int id);
5      void setTime(int time);
6      void printTimes();
7      void timesToFile(ofstream &fout);
8  private:
9      int _id;
10     vector<int> _times;
11 };
```

Листинг 3: 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 4: Объявление класса PreProcessing

```
1  class PreProcessing
2  {
3  public:
4      PreProcessing(int count, QTime *timer, Processing *p, QMutex *
           mutex2);
5      void addToQueue(MyObject obj);
6      void process();
7  private:
8      queue<MyObject> _queue;
9      QTime *_timer;
10     Processing *_proc;
11     QMutex *_mutex2;
12     int _count;
13     int preprocessed = 0;
14     const unsigned int _PROCESSING_TIME = 70;
15 };
```

Листинг 5: Реализация класса PreProcessing

```
1  PreProcessing::PreProcessing(int count, QTime *timer, Processing *p,
       QMutex *mutex2)
2  {
3      this->_count = count;
4      this->_timer = timer;
5      this->_proc = p;
6      this->_mutex2 = mutex2;
7  }
8
9  void PreProcessing::addToQueue(MyObject obj)
10 {
11     _queue.push(obj);
12 }
13
14 void PreProcessing::process()
15 {
16     while (preprocessed != _count)
17     {
18         if (_queue.size() != 0)
19         {
20             // cout << "PreProcessing" << _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              _queue.pop();
27              _mutex2->lock();
28              _proc->addToQueue(obj);
29              _mutex2->unlock();
30              preprocessed++;
31          }
32      }
33  }
```

Листинг 6: Объявление класса Processing

```
1  class Processing
2  {
3  public:
4      Processing(int count, QTime *timer, PostProcessing *p, QMutex *
           mutex2, QMutex *mutex3);
5      void addToQueue(MyObject obj);
6      void process();
7  private:
8      queue<MyObject> _queue;
9      QTime *_timer;
10     PostProcessing *_proc;
11     QMutex *_mutex2;
12     QMutex *_mutex3;
13     int _count;
14     int processed = 0;
15     const unsigned int _PROCESSING_TIME = 100;
16 };
```

Листинг 7: 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 8: Объявление класса PostProcessing

```
1  class PostProcessing
2  {
3  public:
4      PostProcessing(int count, QTime *timer, vector<MyObject> *dump,
           QMutex *mutex3);
5      void addToQueue(MyObject obj);
6      void process();
7  private:
8      queue<MyObject> _queue;
9      QTime *_timer;
10     vector<MyObject> *_dump;
11     QMutex *_mutex3;
12     int _count;
13     int postprocessed = 0;
14     const unsigned int _PROCESSING_TIME = 50;
15 };
```

Листинг 9: 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 }
```

Тесты

Тестирование ПО проводилось вручную на основании данных лог-файла. Проверялась работа и корректное завершение программы для разного количества объектов: для 1 - 5 объектов, от 10 до 100 с шагом 10. Все тесты были пройдены.

Экспериментальная часть

В данном разделе будет сравнительный анализ реализаций конвейера при разной нагруженности компонентов конвейера.

Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Измерение времени работы конвейера в три потока при его загруженности от 0 до 50 элементами с шагом 10.
2. Измерение времени работы конвейера в один поток (последовательно) при его загруженности от 0 до 50 элементами с шагом 10.
3. Измерение времени работы конвейера при различном времени работы его лент.

Измерения проводились на компьютере HP Pavilion Notebook на базе Intel Core i5-7200U, 2.50 Гц с 6 Гб оперативной памяти и с 4 логическими ядрами под управлением операционной системы Linux Mint.

Сравнительный анализ на материале экспериментальных данных

Результаты замеров времени работы конвейера в три потока отображены в таблице 1. Результаты замеров времени работы конвейера в один поток отображены в таблице 2.

Таблица 1: Результаты замеров времени для трех потоков

Кол-во объектов	время, мс
0	0
10	773
20	1482
30	2174
40	2878
50	3579

Таблица 2: Результаты замеров времени для одного потока

Кол-во объектов	время, мс
0	0
10	1409
20	2813
30	4221
40	5629
50	7036

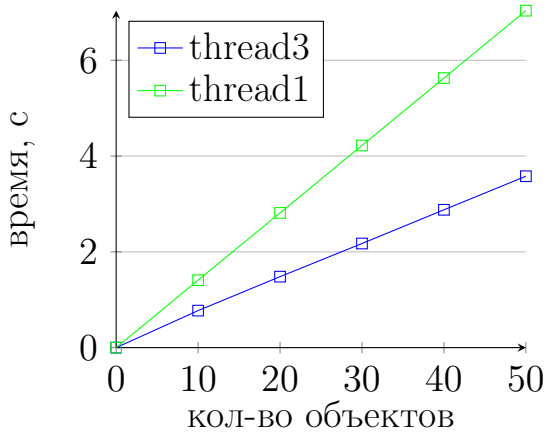

Рис. 6: Результаты замеров времени для конвейера в три потока и в один поток

Выводы

В результате проведенных экспериментов было установлено следующее.

1. Время работы конвейера в три потока практически линейно зависит от количества обрабатываемых объектов.

2. Время обработки объектов в один поток практически линейно зависит от количества обрабатываемых объектов.

3. Время обработки объектов в один поток растет быстрее времени работы конвейера в три потока при росте количества обрабатываемых объектов.

Лог-файл и его интерпретация

Ниже представлены данные из лог-файл, который создает программа.
Условные обозначения:

1. t11, t12 - время поступления объекта на первую ленту и время конца его обработки на первой ленте соответсвенно в мс;

2. t21, t22 - время поступления объекта на вторую ленту и время конца его обработки на первой ленте соответсвенно в мс;

3. t31, t32 - время поступления объекта на третью ленту и время конца его обработки на первой ленте соответсвенно в мс;

4. id объекта - его номер в очереди.

Таблица 3: Данные из лог-файла

ID объекта	t11, мс	t12, мс	t21, мс	t22, мс	t31, мс	t32, мс
0	5	55	55	125	125	145
1	55	105	125	195	195	215
2	105	155	195	265	265	285
3	155	205	265	335	335	356
4	205	255	335	406	406	426
5	255	306	406	476	476	496
6	306	356	476	546	546	566
7	356	406	546	616	616	636
8	406	456	616	686	686	706
9	456	506	686	756	756	777

Total time: 777
Из данных лог-файла можно сделать следующие выводы.

1. Конвейер работает корректно.

2. Общее время работы конвейера для 10 объектов составило 777 мс.

3. На второй легнте на протяжении всей ее работы наблюдаются простои.

4. На третьей легнте на протяжении всей ее работы наблюдаются простои.

5. Простоев нет только на первой ленте.

Заключение

В ходе данной лабораторной работы были получены навыки огранизации асинхронного взаимодействия между потоками на примере конвейерных вычислений. Было спроектировано ПО, реализующее конвейерную обработку. Был проведен сравнительный анализ времени работы конвейера при работе в один поток и в три. Были проанализированы данные из лог-файла.

Список литературы

1. Amos R. Omondi. Microarchitecture of Pipelined and Superscalar Computers. — Springer, 1999.

2. Корнеев В.В. Вычислительные системы.-М.:Гелиос АРВ, 2004.-512с., ил.