МГТУ им. Н.Э.Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Конвейерная обработка данных

Работу выполнил: Луговой Дмитрий, ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2019

Оглавление

Введение

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Цель работы: Получить навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Задачи работы

Задачами данной лабораторной являются:

1) Выбрать и описать методы обработки данных, которые будут сопоставлены методам конвейера;

2) Описать архитектуру программы, а именно какие функции имеет главный поток, принципы и алгоритмы обмена данными между потоками;

3) Реализовать конвейерную систему, а также сформировать лог событий с указанием времени их происхождения, описать реализацию;

4) Интерпретировать сформированный лог.

1 | Аналитическая часть

В данном разделе содержится описание работы конвейера.

1.1 Идея конвееризации

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд.

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 выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

Выполнение каждой команды складывается из ряда последовательных этапов (шагов стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется конвейерный принцип обработки информации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него

1.2 Задача конвееризации

Конвейер будет состоять из трех уровней. Генератор подает на вход конвейера (первый уровень) некоторые числа. 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 лен-

ты будет использована очередь задач, которые должны обработаться на этой ленте. Таким образом, лента будет работать, пока в её очереди есть задачи, но 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 результирующий пул. После завершения работы конвейера, то есть после завершения работы третьей ленты, проверяется равенство количества поданных на вход конвейера объектов количеству объектов в результирующем пуле. Если равенство верное, то конвейер отработал корректно, иначе где-то произошла ошибка и какие-то данные были обработаны неправильно, либо вообще были потеряны.

На каждом уровне конвейера алгоритм обработки информации заменяется задержкой выполняемой программы по времени для создания видимости работы ленты.

1.3 Вывод

Таким образом, конвейеризация алгоритмов, на вход которых поступают однообразные данные, позволяет ускорить процесс обработки этих данных.

2 | Конструкторская часть

В этом разделе содержится описание реализации работы конвейера, схемы работы конвейера и его лент.

2.1 Описание реализации конвеера

Каждой ленте конвейера выделен отдельный поток. В главном потоке запускаются все три рабочих потока - ленты конвейера. 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 список 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 список 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. На Рис. 2.1 представлена общая схема работы конвейера.

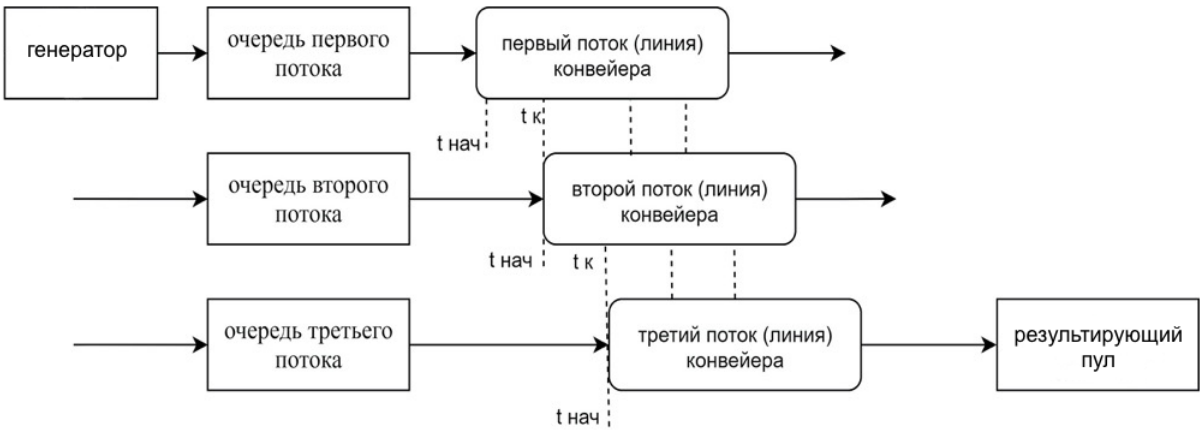

Рис. 2.1: Схема работы конвеера

На рисунках 2.2 - 2.3 будет представлены схемы алгоритмов работы основного потока и первой ленты (остальные ленты реализованы аналогично).

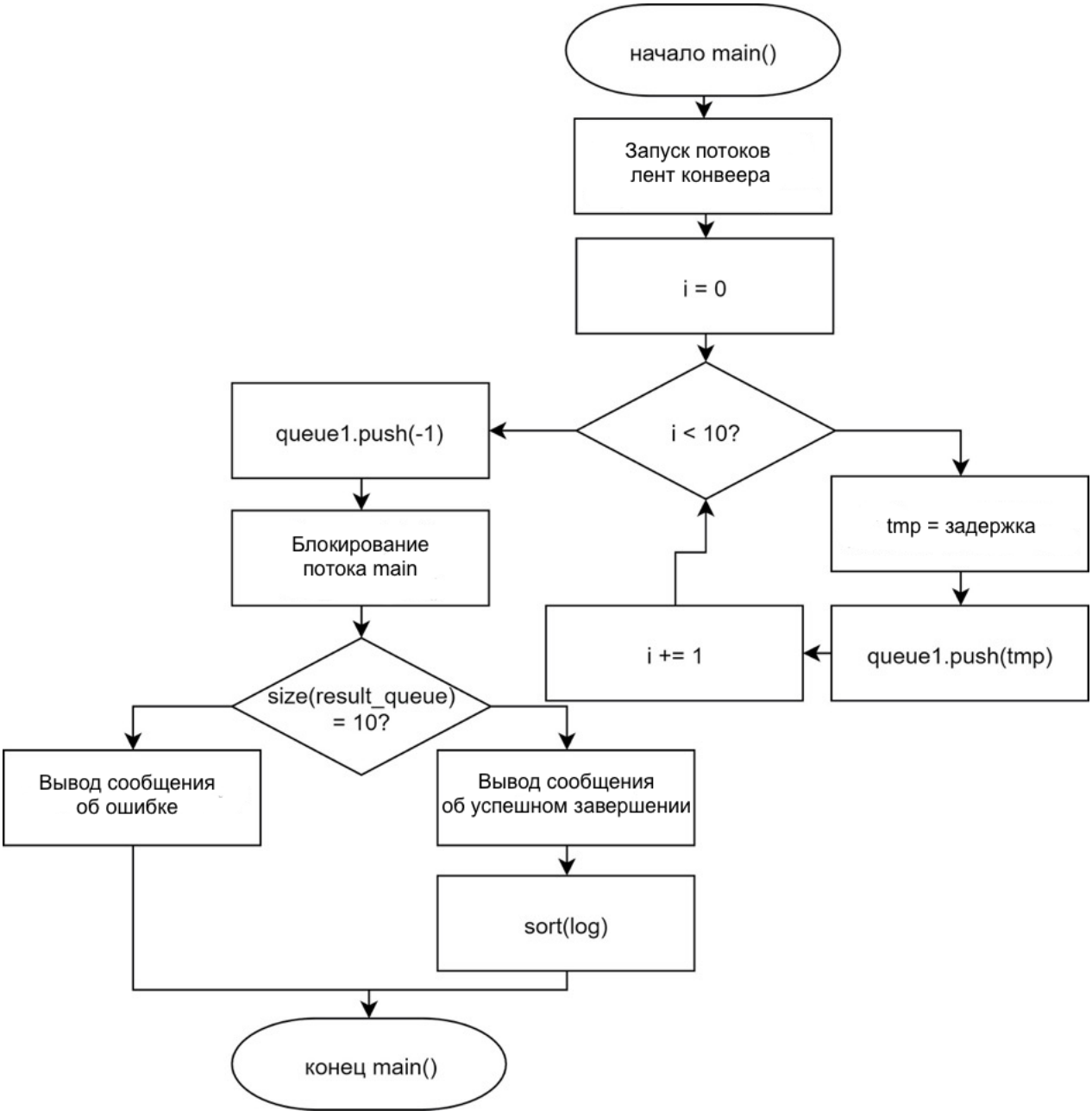

Рис. 2.2: Схема работы основного потока

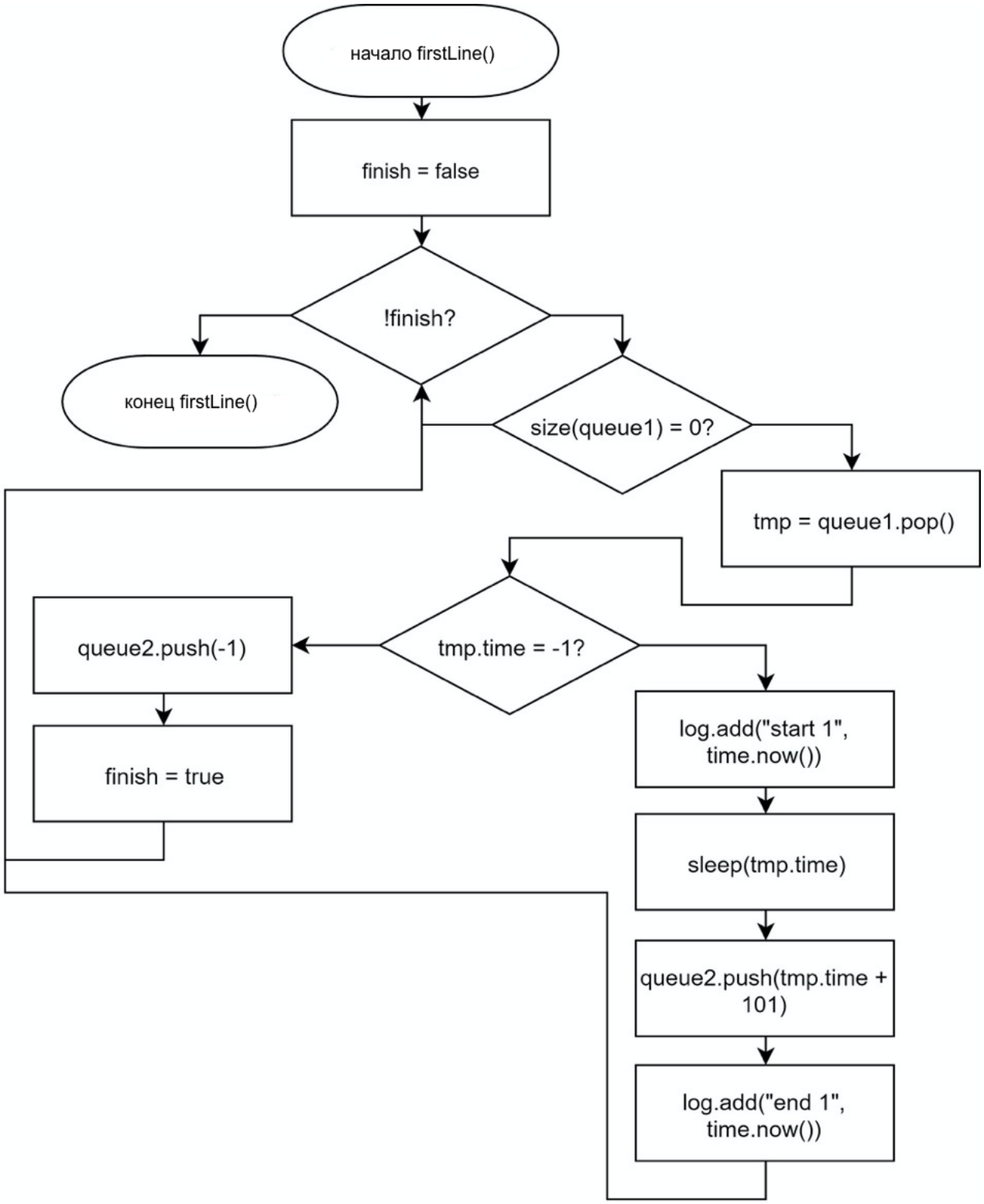

Рис. 2.3: Схема работы первой ленты

2.2 Вывод

Была разработаны структура алгоритма, использующего конвейерную обработку данных.

3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

На вход поступает значение задержки при обработке данных на линиях.

3.2 Средства реализации

Для реализации представленных алгоритмов был выбран язык C++. Время работы алгоритмов было замерено с помощью функции system_clock() из библиотеки chrono. Для тестирования использовался компьютер на базе процессора Intel Core i5 (4 физических ядра, 8 логических).

3.3 Листинги кода

В Листинге 3.1 показана реализация главного потока.

Листинг 3.1: Функция главного потока

```
1  int main()
2  {
3      thread lineOne(&firstLine);
4      thread lineTwo(&secondLine);
5      thread lineThree(&thirdLine);
6      for(int i = 0; i < nItems; ++i)
7      {
8          int inTime = 1000;
9          cout << "Item: " << i << ", value: " << inTime << " was added."
               << endl;
10         inMtx.lock();
11       i nQue.push({i, inTime});
12       i nMtx.unlock();
13         std::this_thread::sleep_for(std::chrono::milliseconds(50));
14     }
15     inMtx.lock();
16     inQue.push({-1, -1});
17     inMtx.unlock();
```

```
18
19      lineOne.join();
20      lineTwo.join();
21      lineThree.join();
22      if (outQue.size() == nItems)
23      {
24          cout << "Conveyor finished his work successfully." << endl;
25          sortLog();
26          createLog("log.txt");
27          cout << "Conveyor took: " << logMessages[logMessages.size() - 1].first -
                logMessages[0].first << " seconds." << endl;
28      }
29      else
30      {
31          cout << "Conveyor missed " << nItems - outQue.size() << " items.\n"
                << endl;
32      }
33      return 0;
34  }
```

В Листингах 3.2, 3.3, 3.4 показаны реализации функций для линий конвейера.

Листинг 3.2: Функция первой линии конвеера

```
1  void firstLine()
2  {
3      bool finish = false;
4      while (!finish)
5      {
6          inMtx.lock();
7          if (!inQue.empty())
8          {
9              pair<int, int> curItem = inQue.front();
10             inQue.pop();
11             int index = curItem.first;
12             int t0 = curItem.second;
13             if (index != -1)
14             {
15                 logMtx1.lock();
16                 string tmp = "Line 1: Starting processing item " + to_string(
                       index) + ", value: " + to_string(t0) + " milliseconds";
17                 logMessages.push_back({currentTime(), tmp});
18                 logMtx1.unlock();
19                 std::this_thread::sleep_for(std::chrono::milliseconds(t0));
20                 tmpMtx1.lock();
21                 tmpQue1.push({index, t0});
22                 tmpMtx1.unlock();
23
24                 logMtx1.lock();
25                 tmp = "Line 1: Finished processing item " + to_string(index);
```

```
26                  logMessages.push_back({currentTime(), tmp});
27                  logMtx1.unlock();
28              }
29              else
30              {
31                  tmpMtx1.lock();
32                  tmpQue1.push({-1, -1});
33                  tmpMtx1.unlock();
34                  finish = true;
35              }
36          }
37          inMtx.unlock();
38      }
39  }
```

Листинг 3.3: Функция второй линии конвеера

```
void secondLine()
{
    bool finish = false;
    while(!finish)
    {
        tmpMtx1.lock();
        if (!tmpQue1.empty())
        {
            pair<int, int> curItem = tmpQue1.front();
            int index = curItem.first;
            int t0 = curItem.second;
            tmpQue1.pop();
            if (t0 != -1)
            {
                logMtx1.lock();
                string tmp = "Line 2: Starting processing item " + to_string(
                    index) + ", value: " + to_string(t0) + " milliseconds";
                logMessages.push_back({currentTime(), tmp});
                logMtx1.unlock();

                std::this_thread::sleep_for(std::chrono::milliseconds(t0));

                tmpMtx2.lock();
                tmpQue2.push({index, t0});
                tmpMtx2.unlock();

                logMtx1.lock();
                tmp = "Line 2: Finished processing item " + to_string(index);
                logMessages.push_back({currentTime(), tmp});
                logMtx1.unlock();
            }
            else
            {
                tmpMtx2.lock();
```

```
                tmpQue2.push({-1, -1});
                tmpMtx2.unlock();
                finish = true;
            }
        }
        tmpMtx1.unlock();
    }
}
```

Листинг 3.4: Функция третьей линии конвеера

```
void thirdLine()
{
    bool finish = false;
    while(!finish)
    {
        tmpMtx2.lock();
        if (!tmpQue2.empty())
        {
            pair<int, int> curItem = tmpQue2.front();
            int index = curItem.first;
            int t0 = curItem.second;
            tmpQue2.pop();
            if (t0 != -1)
            {
                logMtx1.lock();
                string tmp = "Line 3: Starting " + to_string(index) + " item,
                    value: " + to_string(t0) + " milliseconds";
                logMessages.push_back({currentTime(), tmp});
                logMtx1.unlock();
                std::this_thread::sleep_for(std::chrono::milliseconds(t0));
                outMtx.lock();
                outQue.push({index, t0});
                outMtx.unlock();
                logMtx1.lock();
                tmp = "Line 3: Finished processing item " + to_string(index
                    );
                logMessages.push_back({currentTime(), tmp});
                logMtx1.unlock();
            }
            else
            {
                finish = true;
            }
        }
        tmpMtx2.unlock();
    }
}
```

В Листинге 3.5 показана реализация функции сортировки лога.

Листинг 3.5: Функция сортировки лога

```
void sortLog()
{
    bool flag;
    for (size_t i = 0; i < logMessages.size(); i++)
    {
        flag = false;
        for (size_t j = 1; j < logMessages.size() - i; j ++)
        {
            if (logMessages[j].first < logMessages[j - 1].first)
            {
                pair<time_t, string> tmp = logMessages[j];
                logMessages[j] = logMessages[j - 1];
                logMessages[j - 1] = tmp;
                flag = true;
            }
        }
        if (!flag) return;
    }
}
```

3.4 Вывод

Была реализован алгоритм, использующий конвейерную обработку данных, и осуществлена запись результатов в лог.

4 | Экспериментальная часть

В данном разделе будут приведены примеры работы программы, а также будет приведена интерпретация сформированного программой лог-файла.

4.1 Примеры работы

На рисунках 4.1 - 4.3 приведены примеры работы программы. В консоль выводятся входные данные (данные для первой ленты конвейера) и результат прохождения этих данных через конвейер.

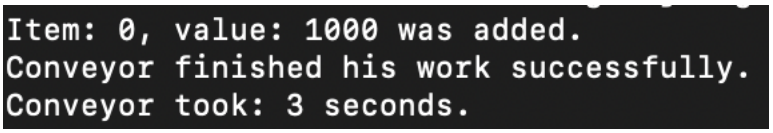

Рис. 4.1: Результат работы при 1 входном элементе

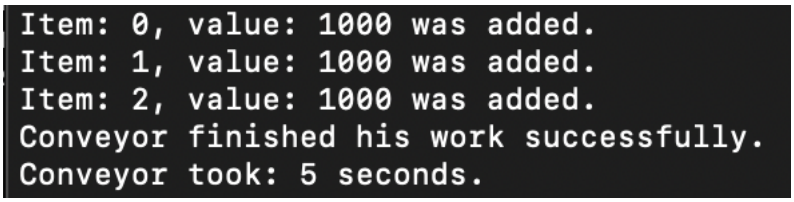

Рис. 4.2: Результат работы при 3 входных элементах

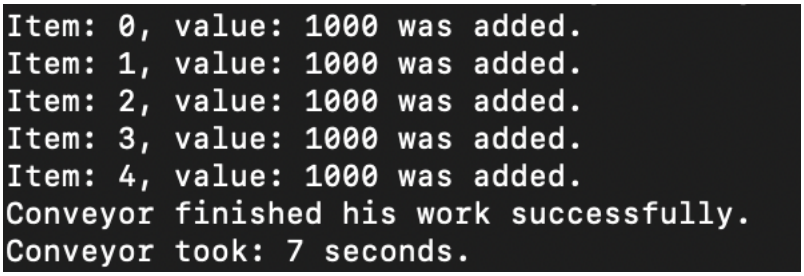

Рис. 4.3: Результат работы при 5 входных элементах

Как видно из приведенных результатов: один входной элемент, который должен быть обработан на всех трех лентах, обрабатывается последовательно и его суммарное время обработки равно 1 секунда * 3 ленты = 3 секунды.

При этом уже для трех входных объектах суммарное время конвейерной обработки становится гораздо меньше суммарного времени последовательной обработки: 5 секунд $<$ 9 секунд. Аналогичная ситуация наблюдается и для входной очереди из пяти элементов.

4.2 Анализ лога

Для того, чтобы отследить работу потоков, необходимо получить лог-файл, отражающий время начала работы ленты над очередным заданием и время окончания этой работы. Например, на рисунке 4.4 представлен лог-файл для ситуации, когда генератор подал на вход конвейеру пять элементов.

```
13:49:06 Line 1: Starting processing item 0, value: 1000 milliseconds
13:49:07 Line 1: Finished processing item 0
13:49:07 Line 2: Starting processing item 0, value: 1000 milliseconds
13:49:07 Line 1: Starting processing item 1, value: 1000 milliseconds
13:49:08 Line 2: Finished processing item 0
13:49:08 Line 3: Starting 0 item, value: 1000 milliseconds
13:49:08 Line 1: Finished processing item 1
13:49:08 Line 2: Starting processing item 1, value: 1000 milliseconds
13:49:08 Line 1: Starting processing item 2, value: 1000 milliseconds
13:49:09 Line 3: Finished processing item 0
13:49:09 Line 2: Finished processing item 1
13:49:09 Line 3: Starting 1 item, value: 1000 milliseconds
13:49:09 Line 1: Finished processing item 2
13:49:09 Line 2: Starting processing item 2, value: 1000 milliseconds
13:49:09 Line 1: Starting processing item 3, value: 1000 milliseconds
13:49:10 Line 3: Finished processing item 1
13:49:10 Line 2: Finished processing item 2
13:49:10 Line 3: Starting 2 item, value: 1000 milliseconds
13:49:10 Line 1: Finished processing item 3
13:49:10 Line 2: Starting processing item 3, value: 1000 milliseconds
13:49:10 Line 1: Starting processing item 4, value: 1000 milliseconds
13:49:11 Line 3: Finished processing item 2
13:49:11 Line 2: Finished processing item 3
13:49:11 Line 3: Starting 3 item, value: 1000 milliseconds
13:49:11 Line 1: Finished processing item 4
13:49:11 Line 2: Starting processing item 4, value: 1000 milliseconds
13:49:12 Line 3: Finished processing item 3
13:49:12 Line 2: Finished processing item 4
13:49:12 Line 3: Starting 4 item, value: 1000 milliseconds
13:49:13 Line 3: Finished processing item 4
```

Рис. 4.4: Результирующий лог-файл при 5 входных элементах

В первом столбце показано время записи, во втором лента конвейера, на которой производилась запись, далее указано начало или конец обработки и номер элемента, который обрабатывался. Если это начало обработки, то в конце указывается количество миллисекунд, которые должен обрабатываться этот элемент. Из представленного файла видно, что как только первый входной объект 0 item был обработан на первой ленте, его начинает обрабатывать вторая лента. В это время на первой ленте уже идет обработка следующего элемента. Как только на второй ленте закончилась обработка 0 item, начинает работать третья лента конвейера.

Время обработки текущего элемента на всех трех лентах конвейера одинаковое, поэтому ленты простаивают в течение работы программы, кроме си-

туации в самом начале, когда первый элемент еще не поступил в очередь на обработку ко второй и третьей лентам.

Рассмотрим другой пример, когда на вход подаются задачи, требующие на обработку одинакового количества времени, однако на второй ленте конвейера обработка будет занимать в 2 раза больше времени, чем на других лентах. На рисунке 4.5 представлен лог работы программы.

```
14:35:33 Line 1: Starting processing item 0, value: 1000 milliseconds
14:35:34 Line 1: Finished processing item 0
14:35:34 Line 2: Starting processing item 0, value: 2000 milliseconds
14:35:34 Line 1: Starting processing item 1, value: 1000 milliseconds
14:35:36 Line 2: Finished processing item 0
14:35:36 Line 3: Starting 0 item, value: 1000 milliseconds
14:35:36 Line 1: Finished processing item 1
14:35:36 Line 2: Starting processing item 1, value: 2000 milliseconds
14:35:36 Line 1: Starting processing item 2, value: 1000 milliseconds
14:35:37 Line 3: Finished processing item 0
14:35:38 Line 2: Finished processing item 1
14:35:38 Line 3: Starting 1 item, value: 1000 milliseconds
14:35:38 Line 1: Finished processing item 2
14:35:38 Line 2: Starting processing item 2, value: 2000 milliseconds
14:35:38 Line 1: Starting processing item 3, value: 1000 milliseconds
14:35:39 Line 3: Finished processing item 1
14:35:40 Line 2: Finished processing item 2
14:35:40 Line 3: Starting 2 item, value: 1000 milliseconds
14:35:40 Line 1: Finished processing item 3
14:35:40 Line 2: Starting processing item 3, value: 2000 milliseconds
14:35:40 Line 1: Starting processing item 4, value: 1000 milliseconds
14:35:41 Line 3: Finished processing item 2
14:35:42 Line 2: Finished processing item 3
14:35:42 Line 3: Starting 3 item, value: 1000 milliseconds
14:35:42 Line 1: Finished processing item 4
14:35:42 Line 2: Starting processing item 4, value: 2000 milliseconds
14:35:43 Line 3: Finished processing item 3
14:35:44 Line 2: Finished processing item 4
14:35:44 Line 3: Starting 4 item, value: 1000 milliseconds
14:35:45 Line 3: Finished processing item 4
```

Рис. 4.5: Результирующий лог-файл при увеличенном времени обработки на 2 ленте

Как видно из лог-файла, 3 лента конвейера простаивает 1 секунду на каждом элементе, кроме первого.

4.3 Вывод

При анализе лога было выяснено, что при разном времени обработки на разных лентах, некоторые из них могут простаивать.

Заключение

В ходе лабораторной работы был изучен и реализован алгоритма, использующий конвейерную обработку данных с использованием методов распараллеливания процессов. Были выявлены оптимальные для конвейерной обработки параметры входных задач - сбалансированное время обработки данных на всех лентах конвейера.