МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Конвейер

Работу выполнил: студент группы ИУ7-53Б
Наместник Анастасия

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

Москва, 2020

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Поиск максимума/минимаума 3
1.2 Параллельное программирование 3
1.3 Конвейерная обработка данных 4
1.4 Вывод 4

2 Конструкторская часть **5**
2.1 Схема главного потока 5
2.2 Схема рабочего потока 7
2.3 Вывод 7

3 Технологическая часть **8**
3.1 Выбор языка программирования 8
3.2 Сведения о модулях программы 8
3.3 Вывод 12

4 Исследовательская часть **13**
4.1 Характеристики ЭВМ 13
4.2 Временные характеристики 13
4.3 Вывод 14

Заключение **16**

Список литературы **16**

Введение

Конвейер - это 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 [6]. В рамках этой лабораторной работы будут рассмотрены три операции на массиве, которые будут последовательно обрабатываться на 3х конвейерных лентах. Каждая задача будет последовательно проходить три этапа обработки. Организация многопоточного режима программирования позволит достигнуть максимальной эффективности выполнения программы.

Целью данной лабораторной работы является изучение и реализация конвейера для выполнения элементарных операций на массиве, а также сравнительный анализ затрачиваемых временных ресурсов при параллельной и однопоточной реализаций одного и того же алгоритма.

В данной лабораторной работе требуется решить четыре задачи:

1. изучить основы конвейерной обработки данных;
2. получить практические навыки конвейерной обработки данных;
3. программно реализовать три алгоритма;
4. сделать сравнительный анализ по затрачиваемым ресурсам (времени) компьютера.

1 Аналитическая часть

В данной лабораторной работе метод конвейерной обработки данных будет применен к следующим трем стандартным операциям на матрице:

1. поиск минимума;
2. поиск максимума;
3. подсчет количества элементов.

Ниже будут представлены теоретические сведения, необходимые для программной реализации этой задачи.

1.1 Поиск максимума/минимаума

В лабораторной работе будет рассмотрен простейший и быстрый алгоритм поиска максимального и минимального элементов массива. Следует пояснить, что под быстротой алгоритма будет подразумеваться его асимптотическая сложность, а не физическое время выполнения. В действительности, единственный способ точно найти самое большое или самое маленькое число в случайном массиве будет перебор каждого числа в поисках максимума или минимума. Поэтому сложность такого алгоритма – O(N). Они называются линейными [4].

1.2 Параллельное программирование

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных

компьютеров и является подмножеством более широкого понятия многопоточности. Таким образом, параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно [3].

1.3 Конвейерная обработка данных

Представителем временного параллелизма является конвейерное выполнение программы, которое предполагает, что каждая команда программы выполняется не на отдельном устройстве за один такт, как на традиционном компьютере, а на устройстве, состоящем из нескольких последовательно соединенных устройств (ступеней), называемом конвейером. Таким образом, выполнение команды разделяется на несколько стадий, каждая из которых выполняется на соответствующей ступени конвейера [6].

1.4 Вывод

В данном разделе были рассмотрены теоретические сведения о параллельном программировании, а также организации конвейерной обработки данных.

2 | Конструкторская часть

В данном разделе будут представлены схемы главного потока и дочерних потоков, выделяемых для решения задачи поиска максимума, минимума и количества элементов массива.

2.1 Схема главного потока

На рисунке 2.1 представлена схема главного потока.

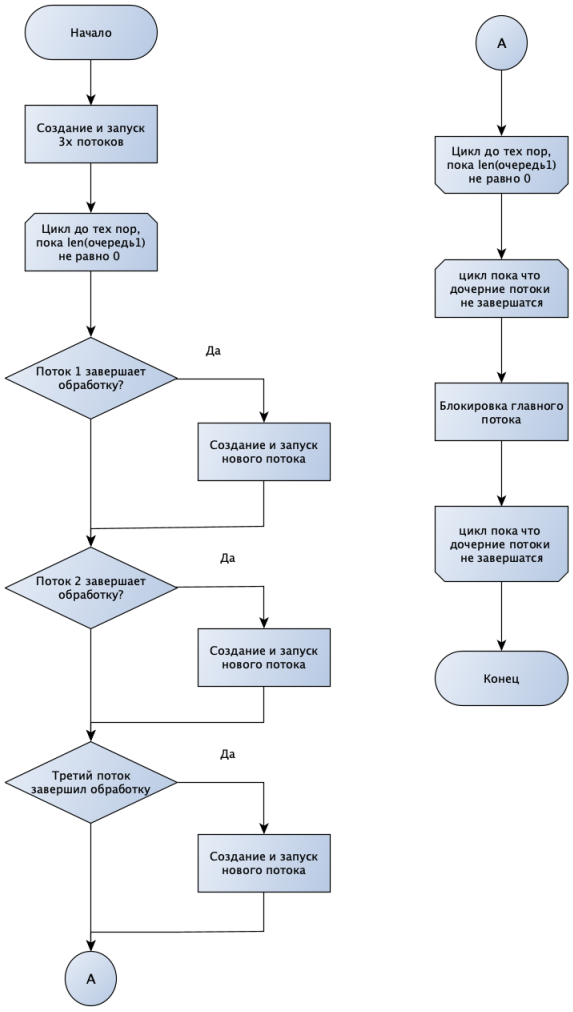

Рис 2.1: Схема главного потока

2.2 Схема рабочего потока

На рисунке 2.2 представлена схема рабочего потока

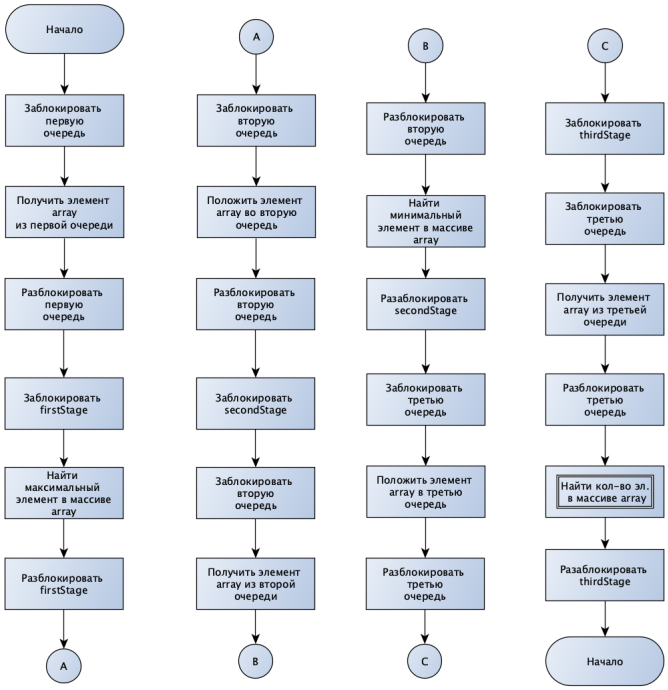

Рис 2.2: Схема рабочего потока

2.3 Вывод

В данном разделе были рассмотрены 2 схемы: схема главного и рабочего потоков реализации конвейерной обработки данных для решения задачи поиска максимума, минимума и подсчета элементов массива.

3 | Технологическая часть

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - C# [2], так как данный язык программирования поддерживает параллельное программирования, он является нативным. В качестве интегрированной среды разработки использовалась Visual studio [1]. Для генерации массива использовался метод Next класса Random [5].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- Program.cs - основная программа.

На листинге 3.1 представлена подпрограмма главного потока.

Листинг 3.1: Метод создания и запуска потоков

```
1  public static void MainTread(Queue<intPtr> queue)
2  {
3    ThreadArgs args = new ThreadArgs(queue);
4
5    Thread firstThread = new Thread(new
       ParameterizedThreadStart(Conveyor));
6    firstThread.Name = "Thread 1";
7
8    Thread secondThread = new Thread(new
       ParameterizedThreadStart(Conveyor));
9    secondThread.Name = "Thread 2";
10
```

```
11    Thread thirdThread = new Thread(new
          ParameterizedThreadStart(Conveyor));
12    thirdThread.Name = "Thread 3";
13
14    firstThread.Start(args);
15    secondThread.Start(args);
16    thirdThread.Start(args);
17
18
19    while (args.firstQueue.Count != 0)
20    {
21      if (!firstThread.IsAlive)
22      {
23        firstThread = new Thread(new ParameterizedThreadStart
              (Conveyor));
24        firstThread.Name = "Thread 1";
25        firstThread.Start(args);
26      }
27
28      if (!secondThread.IsAlive)
29      {
30        secondThread = new Thread(new
              ParameterizedThreadStart(Conveyor));
31        secondThread.Name = "Thread 2";
32        secondThread.Start(args);
33      }
34
35      if (!thirdThread.IsAlive)
36      {
37        thirdThread = new Thread(new ParameterizedThreadStart
              (Conveyor));
38        thirdThread.Name = "Thread 3";
39        thirdThread.Start(args);
40      }
41    }
42  }
```

На листинге 3.2 представлена подпрограмма конвейера.

Листинг 3.2: Конвейер

```
1  public static void Conveyor(object obj)
2  {
3    ThreadArgs args = (ThreadArgs)obj;
4    int max, min, count;
5    intPtr array;
6
7    lock (args.firstQueue)
8    {
9      // Get array from the first queue.
10     array = args.firstQueue.Dequeue();
11   }
12
13   lock (firstStage)
14   {
15     // The first tape is running.
16     max = FindMax(array);
17   }
18
19   lock (args.secondQueue)
20   {
21     // Added element to the second queue.
22     args.secondQueue.Enqueue(array);
23   }
24
25   lock (secondStage)
26   {
27     // The second tape is running.
28     lock (args.secondQueue)
29     {
30       // Get array from the second queue.
31       array = args.secondQueue.Dequeue();
32     }
33     min = FindMin(array);
34   }
35
36   lock (args.thirdQueue)
37   {
38     args.thirdQueue.Enqueue(array);
39   }
40
```

```
41    lock (thirdStage)
42    {
43      lock (args.thirdQueue)
44      {
45        array = args.thirdQueue.Dequeue();
46      }
47      count = FindCount(array);
48    }
49  }
```

На листинге 3.3 представлена подпрограмма поиска максимума массива.

Листинг 3.3: Подпрограмма поиска максимума массива

```
1  public static int FindMax(intPtr array)
2  {
3    int max = array[0];
4
5    for (int i = 0; i < operationsCount; i++)
6    foreach (var elem in array)
7    if (elem > max)
8      max = elem;
9
10   return max;
11 }
```

На листинге 3.4 представлена подпрограмма поиска минимума массива.

Листинг 3.4: Подпрограмма поиска минимума массива

```
1  public static int FindMin(intPtr array)
2  {
3    int min = array[0];
4
5    for (int i = 0; i < operationsCount; i++)
6    foreach (var elem in array)
7    if (elem < min)
8      min = elem;
9
10   return min;
11 }
```

На листинге 3.5 представлена подпрограмма поиска количества элементов массива.

Листинг 3.5: Подпрограмма поиска количества элементов массива

```
public static int FindCount(intPtr array, int num)
{
  int count = 0;

  for (int i = 0; i < operationsCount; i++)
  foreach (var elem in array)
  if (elem < num)
    count++;

  return count;
}
```

3.3 Вывод

В технологической части были представлены модули программы, листинги кода, а также обусловлен выбор языка программирования и приведены использовавшиеся в ходе работы инструменты.

4 | Исследовательская часть

В этом разделе будет проведено сравнение последовательной реализации трех алгоритмов и конвейера с использованием параллельного режима программирования.

4.1 Характеристики ЭВМ

- MacBook Pro (Retina, 15-inch, Mid 2014).
- 2,5 GHz Intel Core i7.
- Число логических ядер: 8.

4.2 Временные характеристики

Для проведения анализа затрачиваемых временных ресурсов были использованы 10 массивов размерностью [5, 10, 25, 50, 100, 250, 1000]. Время выполнения алгоритмов поиска максимума, минимума, а также подсчета количества элементов массива довольно мало, поэтому, чтобы оценить его, следует взять среднее арифметическое от времени, за которое процедура отработает установленные количество раз.

Результат сравнения последовательной реализации трех алгоритмов при конвейерной обработке данных с использованием параллельной реализации того же алгоритма представлен на рис. 4.1.

Результаты анализа времени свидетельствуют о том, что время, затрачиваемое на параллельную конвейерную реализацию, значительно меньше затрат на последовательную реализацию тех же алгоритмов, что становится очевидным при увеличении количества элементов массива. Выигрыш последовательной обработки на массивах относительно

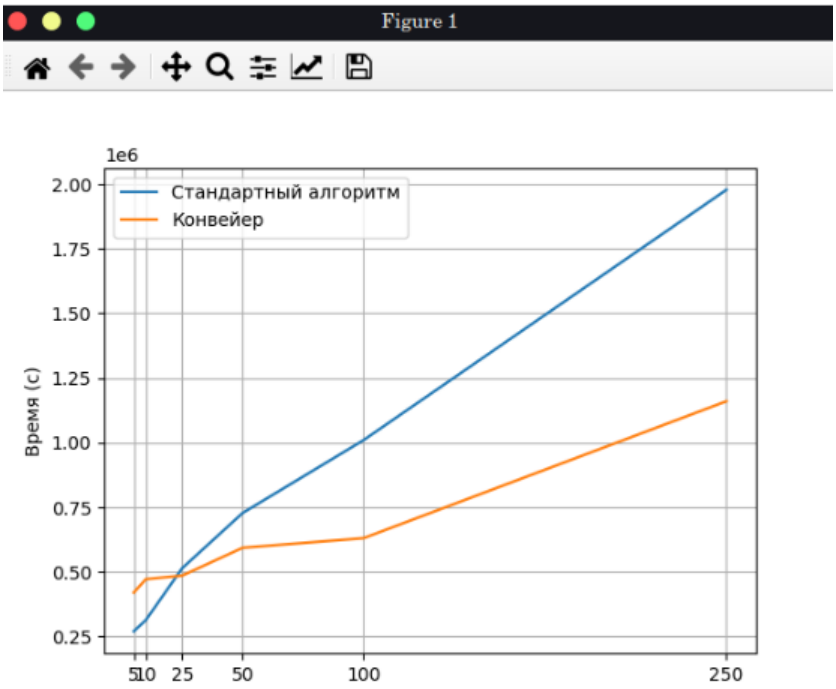

Рис. 4.1: Временные характеристики

небольшого размера показывает то, что при таком решении маленьких задач значительное время тратится на создание потоков и ожидание доступа к переменной, что снижает временную эффективность программы.

На рис. 4.2 приведена статистика.

4.3 Вывод

В данном разделе было произведено сравнение последовательной реализации трех алгоритмов и конвейера с использованием нескольких потоков. Из результатов исследования становится очевидно, что конвейерную обработку на нескольких потоках нецелесообразно применять на небольших задачах. Исследование показало, что конвейерная обработка работает правильно.

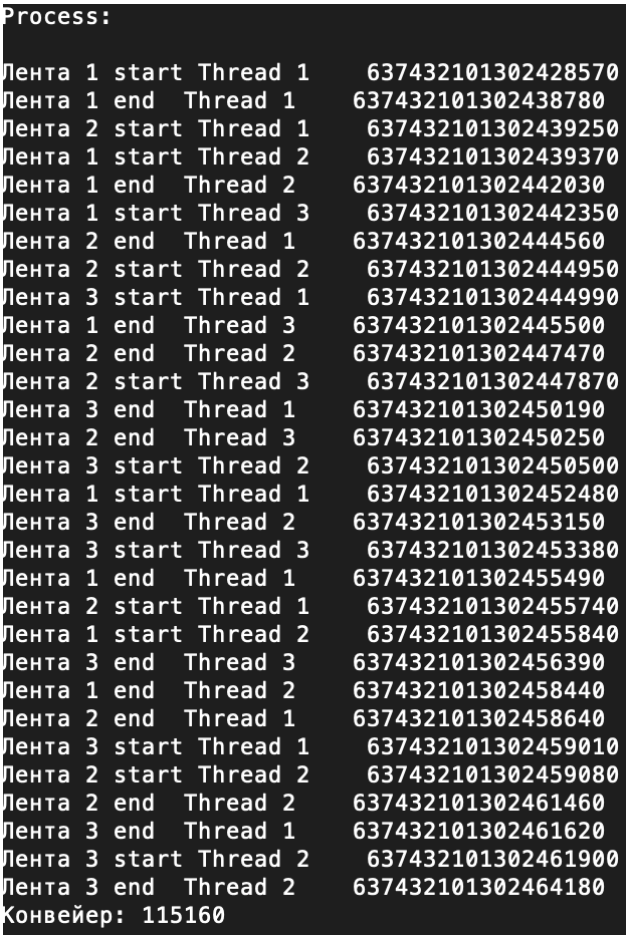

Рис. 4.2: Временные характеристики

Заключение

В ходе лабораторной работы был изучен и реализован конвейер для выполнения элементарных операций на массиве, а также проведен сравнительный анализ затрачиваемых временных ресурсов при параллельной и однопоточной реализаций одного и того же алгоритма.

В рамках выполнения работы решены следующие задачи:

1. изучены основы конвейерной обработки данных;
2. получены практические навыки конвейерной обработки данных;
3. программно реализованы три алгоритма: поиск максимума в массиве, поиск минимума в массиве, поиск количества элементов массива;
4. сделан сравнительный анализ по затрачиваемым ресурсам (времени) компьютера.

Литература

[1] Visual Studio [Электронный ресурс], режим доступа:https://visualstudio.microsoft.com/ru/ (дата обращения: 01.10.2020)

[2] Документация по C# [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 02.10.2020)

[3] Академия Microsoft: Основы параллельного программирования с использованием Visual Studio [Электронный ресурс], - режим доступа: https://intuit.ru/studies/courses/4807/1055/lecture/16369 (дата обращения: 02.10.2020)

[4] Алгоритм поиска максимума в массиве [Электронный ресурс], режим доступа:https://proglib.io/p/fastest-max-algorithm (дата обращения: 05.12.2020)

[5] Random Класс [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/api/system.random?view=netcore-3.1 (дата обращения: 02.12.2020)

[6] Конвейерная организация [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/api/system.random?view=netcore-3.1 (дата обращения: 01.12.2020)