ФГОУ ВО НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Р. Е. АЛЕКСЕЕВА

Лабораторная работа № 3

Технлогии распределенной обработки данных

Тема: Распараллеливание алгоритма с помощью библиотеки Concurrent and Coordination Runtime

Вариант № 3

Проверил:
Гай В. Е.
Выполнил:
Студент гр. 14-В-1
Кузнецова П.В.

Нижний Новгород 2016

1. Выполнение лабораторной работы

1.1. Цель и вариант задания на лабораторную работу

Целью данной лабораторной работы является получение представления о возможностях библиотеки Corrent and Coordination Runtime для организации параллельных вычислений.

Вариант индивидуального задания:

Разработать алгоритм поиска максимального и минимального значения массива a:

$$mn = \min_{i \in [1;N]} a_i \,,\; mx = \max_{i \in [1;N]} a_i$$

1.2. Библиотека Concurrent and Coordination Runtime

Библиотека Concurrent and Coordination Runtime (CCR) предназначена для организации обработки данных с помощью параллельно и асинхронно выполняющихся методов. Взаимодействие между такими методами организуется на основе сообщений. Рассылка сообщений основана на использовании портов.

Основные понятия CCR:

1) сообщение – экземпляр любого типа данных;

2) порт – очередь сообщений типа FIFO, сообщение остаётся в порте пока не будут извлечено из очереди порта получателем. Отправка сообщения в порт: p.Post(1);

3) получатель – структура, которая выполняет обработку сообщений.

Данная структура объединяет:

а) один или несколько портов, в которые отправляются сообщения;

б) методы, которые используются для обработки сообщений;

в) логическое условие, определяющее ситуации, в которых активизируется тот или иной получатель.

Получатели сообщений бывают двух типов: временные и постоянные. Временный получатель, обработав сообщение, удаляется из списка получателей сообщений данного порта.

4) процессом запуска задач управляет диспетчер. После выполнения условий активации задачи (получение портом сообщения) диспетчер назначает задаче поток из пула потоков, в котором она будет выполняться.

2. Выполнение лабораторной работы

2.1. Объявление структур данных

Массив A, переменные для хранения минимума и максимума в последовательном алгоритме, а так же для хранения кол-ва элементов в массиве определим глобально:

```
int nc;
int n = 16000000;
int[] A;
int min = 900001;
int max = -1;
```

В методе Start() запускаются вычисления. Сначала в методе выполняется поиск минимума и максимума в массиве с помощью последовательного алгоритма, затем та же задача решается с помощью параллельных вычислений. Рассмотрим этот метод. Выполняется инициализация структур данных:

```
protected override void Start()
{
    base.Start();
    nc = 2;
    Random r = new Random();

    A = new int[n];
    for (int i = 0; i < n; i++)
    {
        A[i] = r.Next(900000);
        // Console.WriteLine("A[{0}]={1}",i ,A[i]);
    }

    SequentialMul();
    ParallelMul();

}
```

2.2. Последовательный алгоритм поиска минимального и максимального элементов массива

Рассмотрим метод SequentialMul:

```
void SequentialMul()
{
    // оценка времени выполнения
    Stopwatch sWatch = new Stopwatch();
    sWatch.Start();
    // нахождение минимального и максимального элементов массива A[] с помощью последовательного алгоритма
    for (int i = 0; i < n; i++)
    {
        if (A[i] < min)
            min = A[i];
        if (A[i] > max)
            max = A[i];
    }
    sWatch.Stop();
    Console.WriteLine("Последовательный алгоритм = {0} мс.", sWatch.ElapsedMilliseconds.ToString());
    Console.WriteLine("Min = {0}", min);
    Console.WriteLine("Max = {0}", max);
}
```

2.3. Параллельный алгоритм поиска минимального и максимального элементов массива

Параллельная обработка выполняется с помощью запуска нескольких копий вычислительного метода. Каждая копия метода выполняет обработку определённой части исходных данных. Для описания задания для каждого метода используется класс InputData:

public class InputData

public int start; // начало диапазона

public int stop; // начало диапазона

Поля start / stop класса хранят номер начальной / конечной строки вектора C. Поля рассчитываются с помощью экземпляра вычислительного метода. Рассмотрим метод ParallelMul:

```
void ParallelMul()
  {
     // создание массива объектов для хранения параметров
     InputData[] ClArr = new InputData[nc];
     for (int i = 0; i < nc; i++)
     ClArr[i] = new InputData();
```

Далее, задаются исходные данные для каждого экземпляра вычислительного метода:

```
// делим количество строк в матрице на nc частей
int step = (Int32)(n / nc);
// заполняем массив параметров
int c = -1;
for (int i = 0; i < nc; i++)
{
    ClArr[i].start = c + 1;
    ClArr[i].stop = c + step;
    c = c + step;
}
```

Создаётся диспетчер с пулом из двух потоков и описывается порт, в который каждый экземпляр метода Mul() отправляет сообщение после завершения вычислений:

```
Dispatcher d = new Dispatcher(nc, "Test Pool");
DispatcherQueue dq = new DispatcherQueue("Test Queue", d);

Port<MinMax> p = new Port<MinMax>();
```

Метод Arbiter.Activate помещает в очередь диспетчера две задачи (два экземпляра метода Mul):

```
for (int i = 0; i < nc; i++)
 {
     Arbiter.Activate(dq, new Task<InputData, Port<MinMax>>(ClArr[i], p, Mul));

 }
```

Первый параметр метода Arbiter.Activate – очередь диспетчера, который будет управлять выполнением задачи, второй параметр – запускаемая задача.

С помощью метода Arbiter.MultipleItemReceive запускается задача (приёмник), которая обрабатывает получение двух сообщений портом p:

```
Arbiter.Activate(Environment.TaskQueue, Arbiter.MultipleItemReceive(true, p, nc, delegate(MinMax[] array)
    {
        int minn = 900001;
        int maxx = -1;
        for (int i = 0; i < nc; i++)
        {
            if (array[i].min < minn)

                minn = array[i].min;

            if (array[i].max > maxx)

                maxx = array[i].max;
        }
        Console.WriteLine("Max={0}", maxx);
        Console.WriteLine("Min={0}", minn);
        Console.WriteLine("Вычисления завершены");
 }));
}
```

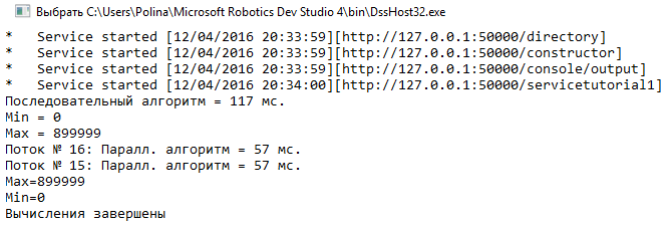
Приёмник используется для определения момента окончания вычислений. Он сработает только после того, как в порт p придёт два сообщения. В делегат, описанный в приёмнике, включим

действия, которые будут выполнены после завершения процесса поиска минимума и максимума в каждом из потоков. Такими действиями является поиск минимума и максимума из результатов, полученных от каждого потока.

Метод Mul() выполняет поиск минимума и максимума в части матрицы на вектор:

```
void Mul(InputData data, Port<MinMax> resp)
{
    int minn = 900001;
    int maxx = -1;

     Stopwatch sWatch = new Stopwatch();
    sWatch.Start();
            for (int i = data.start; i <= data.stop; i++)
            {

                if (A[i] < minn)

                    minn = A[i];

                if (A[i] > maxx)

                    maxx = A[i];

               // Console.WriteLine("Поток № {0}: = {1}.", Thread.CurrentThread.ManagedThreadId, i.ToString()); //Thread возвращает выполняющийся в данный момент поток
            }
            MinMax  Myarr = new MinMax();
            Myarr.max = maxx;
            Myarr.min = minn;
        sWatch.Stop();
        Console.WriteLine("Поток № {0}: Паралл. алгоритм = {1} мс.", Thread.CurrentThread.ManagedThreadId, sWatch.ElapsedMilliseconds.ToString()); //Thread возвращает
     //выполняющийся в данный момент поток
        resp.Post(Myarr);

}
```

Метод Mul() имеет два параметра: 1) индекс, хранящий значение элемента массива, который определяет параметры, передаваемые на вход метода; 2) порт завершения, в который отправляется экземпляр класса MinMax после завершения вычислений. После завершения вычислений метод Mul отправляет в порт p значение экземпляра класса MinMax, в котором хранятся значения минимума и максимума частей массива. Результат вычислений показан на рисунке:



Выводы

Таким образом, в ходе данной лабораторной работы были изучены и освоены на практике возможности библиотеки Corrent and Coordination Runtime для организации параллельных вычислений. Разработаны последовательный и параллельный алгоритмы вычисления минимума и максимума массива. Исходя из работы программы можно сделать вывод о том, что время выполнения последовательного алгоритма примерно в 2 раза больше, чем время выполнения параллельного алгоритма, также результаты при последовательном и параллельном алгоритмах одинаковые, то есть программа работает правильно.