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-2
Самсонов И. А.

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 сортировки методом пузырька массива a

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 array, размер массива, а так же время выполнения планарного массива определены глобально:

```
const int SIZE = 1000;
public int[,] array = new int[SIZE, SIZE];

static long fullParallelTime = 0;
```

В методе Start() запускаются вычисления. Сначала в методе выполняется сортировка массива последовательным алгоритмом, затем та же задача решается с помощью параллельных вычислений. Рассмотрим этот метод сортировки. Выполняется инициализация структур данных:

```
protected override void Start()
{
    base.Start();
    // Add service specific initialization here.


    int[,] arrayCopy = new int[SIZE, SIZE];
    Random r = new Random();

    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = 0; j < SIZE; ++j)
        {
            array[i, j] = r.Next(100);
            arrayCopy[i, j] = array[i, j];
        }
    }
```

2.2. Последовательный алгоритм сортировки методом пузырька

```
System.Diagnostics.Stopwatch sp = new System.Diagnostics.Stopwatch();

sp.Start();

for (int k = 0; k < SIZE; ++k)
{
    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = i; j < SIZE; ++j)
        {
            if (arrayCopy[k, i] > arrayCopy[k, j])
            {
                int a = arrayCopy[k, i];
                arrayCopy[k, i] = arrayCopy[k, j];
                arrayCopy[k, j] = a;
            }
        }
    }
}

sp.Stop();
string planeTime = sp.ElapsedMilliseconds.ToString();
```

2.3. Параллельный алгоритм сортировки мутодом пузырька

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 int[] row;

Поля row класса хранят сортируемую строчку двумерного массива. Поле рассчитываются с помощью экземпляра вычислительного метода.

```
InputData[] data = new InputData[SIZE];

for (int i = 0; i < SIZE; ++i)
{
    data[i] = new InputData();
```

Далее, задаются исходные данные для каждого экземпляра вычислительного метода:

```
  data[i].row = new int[SIZE];

  for (int j = 0; j < SIZE; ++j)
  {
      data[i].row[j] = array[i, j];
  }
}
```

Создаётся диспетчер с пулом из 4х потоков и описывается порт, в который каждый экземпляр метода Sort() отправляет сообщение после завершения вычислений:

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

Port<int> port = new Port<int>();
```

Метод Arbiter.Activate помещает в очередь диспетчера 4 задачи (4 экземпляра метода Sort):

```
for (int i = 0; i < SIZE; i++)
    Arbiter.Activate(dq, new Task<InputData, Port<int>>(data[i], port, Sort));
```

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, port, SIZE, delegate(int[] array)
{
    Console.WriteLine("Вычисления завершены");
    Console.WriteLine("Parallel sorting time: {0}", fullParallelTime.ToString());
    Console.WriteLine("Linear sorting time: {0}ms", planeTime);
}));
```

Приёмник используется для определения момента окончания вычислений. Он сработает только после того, как в порт p придёт 4 сообщения. В делегат, описанный в приёмнике, включим действия, которые будут выполнены после завершения процесса сортировки в каждом

из потоков. Такими действиями является вывод результата, а также подсчет потраченного времени.

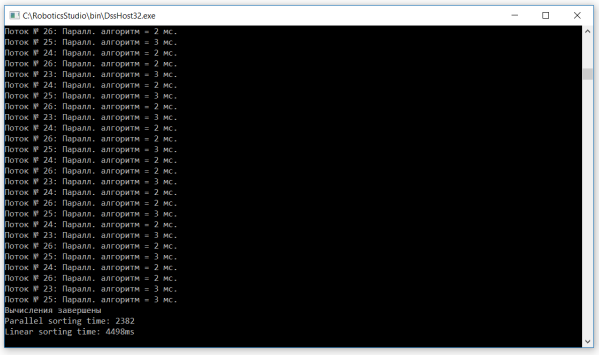
Метод Sort() выполняет сортировку строки матрицы:

```
public static void Sort(InputData data, Port<int> resp)
{
    Stopwatch sWatch = new Stopwatch();
    sWatch.Start();

    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = i; j < SIZE; ++j)
        {
            if (data.row[i] > data.row[j])
            {
                int a = data.row[i];
                data.row[i] = data.row[j];
                data.row[j] = a;
            }
        }
    }

    sWatch.Stop();
    Console.WriteLine("Поток № {0}: Паралл. алгоритм = {1} мс.", System.Threading.Thread.CurrentThread.ManagedThr
    fullParallelTime += sWatch.ElapsedMilliseconds;
    resp.Post(1);
}
```

Метод Sort() имеет два параметра: 1) индекс, хранящий значение элемента массива, который определяет параметры, передаваемые на вход метода; 2) порт завершения, в который отправляется экземпляр класса int после завершения вычислений. После завершения вычислений метод Sort отправляет в порт p значение экземпляра класса int, который необходим определения времени завершения вычислений. 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 сортировки массива методом пузырька. 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.