ГБОУ ВПО Нижегородский государственный технический университет им. Р. Е. Алексеева
Институт радиоэлектроники и информационных технологий, кафедра "Вычислительные системы и технологии"

СОГЛАСОВАНО

Доцент каф. ВСТ

_____ *Гай В. Е.*

“____” _____ _____

ТЕХНОЛОГИИ РАСПРЕДЕЛЁННОЙ ОБРАБОТКИ ДАННЫХ

Отчет к лабораторной работе №3

РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМА С ПОМОЩЬЮ БИБЛИОТЕКИ CCR

Студент гр. 13-В-2

_____ *Смирнова С. В.*

“____” _____ _____

2015

СОДЕРЖАНИЕ

1 ЦЕЛЬ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Цель работы: получить представления о возможности библиотеки Concurren and Coordination Runtime для организации параллельных вычислений.

Порядок выполнения работы:

а) Разработка последовательного алгоритма, решающего одну из приведённых задач в соответствии с выданным вариантом задания;

б) Разработка параллельного алгоритма, соответствующий варианту последовательного алгоритма;

в) Выполнение сравнения времени выполнения последовательного и параллельного алгоритмов обработки данных при различных размерностях исходных данных.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 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:

а) Сообщение – экземпляр любого типа данных;

б) Порт – очередь сообщений типа FIFO (First-In-First-Out), сообщение остаётся в порте пока не будут извлечено из очереди порта получателем. Определение порта:

Port<int> p = new Port<int>();

Отправка сообщения в порт:

p.Post(1);

в) получатель – структура, которая выполняет обработку сообщений. Данная структура объединяет:

- один или несколько портов, в которые отправляются сообщения;
- метод (или методы), которые используются для обработки сообщений (такой метод называется задачей);
- логическое условие, определяющее ситуации, в которых активизируется тот или иной получатель.

Делегат, входящий в получатель, выполнится, когда в порт intPort придёт сообщение. Получатели сообщений бывают двух типов: временные и постоянные (в примере получатель – временный). Временный получатель, обработав сообщение (или несколько сообщений), удаляется из списка получателей сообщений данного порта.

г) процессом запуска задач управляет диспетчер. После выполнения условий активации задачи (одним из условий активации может быть получение портом сообщения) диспетчер назначает задаче поток из пула потоков, в котором она будет выполняться. Описание диспетчера с двумя потоками в пуле:

Dispatcher d = new Dispatcher(2, "MyPool");

Описание очереди диспетчера, в которую задачи ставятся на выполнение:

DispatcherQueue dq = new DispatcherQueue("MyQueue d);

2.2 Создание проекта

Нужно выполнить следующие действия:

а) Установить библиотеку CCR (CCR входит в состав Microsoft Robotics Developer Studio);

б) Создать проект консольного приложения и добавьте к проекту библиотеку Microsoft.Ccr.Core.dll.

2.3 Оценка времени выполнения

Время выполнения вычислений будем определять с помощью класса Stopwatch:

```
Stopwatch sWatch = new Stopwatch();
sWatch.Start();
<выполняемый код>
sWatch.Stop();
Console.WriteLine(sWatch.ElapsedMilliseconds.ToString());
```

3 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1 Вариант задания

Вариант 13:

– Разработать алгоритм сортировки массива чисел методом пузырька

3.2 Листинг программы

```
  using    System;
using Microsoft.Ccr.Core;
using System.Threading;

namespace ConsoleApplication3
{
    public class InputData
    {
        public int start; // начало диапазона
        public int stop;  // конец диапазона
        //public int i;
    }

    class Program
    {
        static int[] a;
        static int[] b;
        static int[] mem; //для "честности" запомним сгнерированный ма
           ссив случайных чисел, чтобы оба алгоритма сортировали одина
           ковые данные
        static int n;
        static int nc;
```

```
static void Mul(InputData data, Port<int> resp)
 {
     int i, j;
     System.Diagnostics.Stopwatch sWatch = new System.
        Diagnostics.Stopwatch();
     sWatch.Start();


    for (i = data.start; i <= data.stop-1; i++)
     {
         for (j = i+1; j <= data.stop; j++)
         {
              if (a[j] < a[i]) //сортировка пузырьком
            {
                var temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
     }
    //внимательно присмотревшись к этой функции и к условиям,
       в которых она вызывается, можно сделать вывод
    //что она вернет массив, состоящий из двух сортированных п
       оловинок, но не сортированный целиком
    //так оно и есть, потому что параллельный пузырек – необыч
       ная задумка
    //в данной лабораторной работе эта проблема решается в дел
       егате, описанном в приемнике, штатными средствами C#
     sWatch.Stop();
     resp.Post(1);
     Console.WriteLine("Поток      {0}: Параллельный алгоритм =
         {1} мс.", Thread.CurrentThread.ManagedThreadId,
        sWatch.ElapsedMilliseconds.ToString());
}

public static void arrDisplay() //функция вывода массива
{
    int i;
    Console.WriteLine("Показать текущее содержимое массива? (Y
       /N)\n");
```

```
var press = Console.ReadKey(true).Key;

if (press == ConsoleKey.Y)
{
    Console.WriteLine("Параметры вывода:\n\tДля вывода мас
       сива вертикально с нумерацией нажмите V\n\tДля выво
       да массива подряд без нумерации горизантально нажми
       те H\n\tКомпактный режим: Для вывода первых и после
       дних 25 элементов нажмите C\n");
    var pressnext = Console.ReadKey(true).Key;
    if (pressnext == ConsoleKey.V)
    {
        for (i = 0; i < n; i++)
            Console.WriteLine("{0}: {1}", i + 1, a[i].
               ToString());
    }
    else if (pressnext == ConsoleKey.H)
    {
        for (i = 0; i < n; i++)
            Console.Write("{0}\t", a[i].ToString());
    }
    else if (pressnext == ConsoleKey.C)
    {
        for (i = 0; i < 25; i++)
            Console.WriteLine("{0}: {1}", i + 1, a[i].
               ToString());
        Console.WriteLine("...");
        for (i = n - 25; i < n; i++)
            Console.WriteLine("{0}: {1}", i + 1, a[i].
               ToString());
    }
    else
    {
        Console.WriteLine("Некорректная клавиша. Вывод мас
           сива отклонен...\n");
    }
}
else if (press == ConsoleKey.N)
{
    Console.WriteLine("Вывод массива отклонен...\n");
}
```

```
        else
        {
            Console.WriteLine("Некорректная клавиша. Вывод массива
                отклонен...\n");
        }

    }
    static void Main(string[] args)
    {
        int i;
        nc = 2;
        n = 50000;

        Console.WriteLine("\nМассив включает в себя {0} элементов\
            n", n);

        a = new int[n];
        mem = new int[n];
        b = new int[nc];

        Random r = new Random();
        for (int j = 0; j < n; j++)
            a[j] = r.Next(10000);
        a.CopyTo(mem,0); //запомнили полученный массив
        Console.WriteLine("Исходный массив успешно заполнен случай
            ными значениями!\n");

        arrDisplay();

        System.Diagnostics.Stopwatch sWatch = new System.
            Diagnostics.Stopwatch();
        Console.WriteLine("Начата последовательная сортировка масс
            ива...\n");
        sWatch.Start();
        for (i = 0; i <= n - 1; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                if (a[j] < a[i])
                {
                    var temp = a[i];
```

```
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    sWatch.Stop();
    Console.WriteLine("Массив отсортирован последовательным ал
        горитмом!\n");
    Console.WriteLine("Последовательный алгоритм = {0} мс.",
        sWatch.ElapsedMilliseconds.ToString());

    arrDisplay(); //показали отсортированный массив

    mem.CopyTo(a, 0); ; // восстановили массив со случайными ч
        ислами

    // создание массива объектов для хранения параметров
    InputData[] tempArray = new InputData[nc];
    i = 0;
    while (i < nc)
        {tempArray[i] = new InputData(); i++;}
    // делим количество элементов  в массиве на пс частей
    int step = (Int32)(n / nc);
    // заполняем массив параметров
    int c = -1;
    for (i = 0; i < nc; i++)
    {
        tempArray[i].start = c + 1;
        tempArray[i].stop = c + step;
        c = c + step;
    }
    Dispatcher d = new Dispatcher(nc, "Test Pool");
    DispatcherQueue dq = new DispatcherQueue("Test Queue", d);
    Port<int> p = new Port<int>();


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

    Console.WriteLine("Начата параллельная сортировка массива
```

```
                ...\n");
            Arbiter.Activate(dq, Arbiter.MultipleItemReceive(true, p,
                nc, delegate(int[] array)
        {
            Console.WriteLine("Массив отсортирован параллельным алгоритмо
                м!\n");
            System.Diagnostics.Stopwatch newWatch = new System.
                Diagnostics.Stopwatch();
            Console.WriteLine("Начата последовательная сортировка массива
                ...\n");
            newWatch.Start();
            Array.Sort(a); //тот самый делегат в приемнике, с помощью кот
                орого шлейфуется результат параллельной сортировки пузырьк
                ом
            newWatch.Stop();
            Console.WriteLine("Окончательная сортировка средствами C#: {0}
                мс.\n", newWatch.ElapsedMilliseconds.ToString());
            arrDisplay();
            Console.WriteLine("Вычисления завершены");
            Console.ReadKey(true);
            Environment.Exit(0);
        }));

            }
        }
}
```

3.3 Результат работы программы

Скриншот работы программы представлен на Рис.1. и Рис. 2.

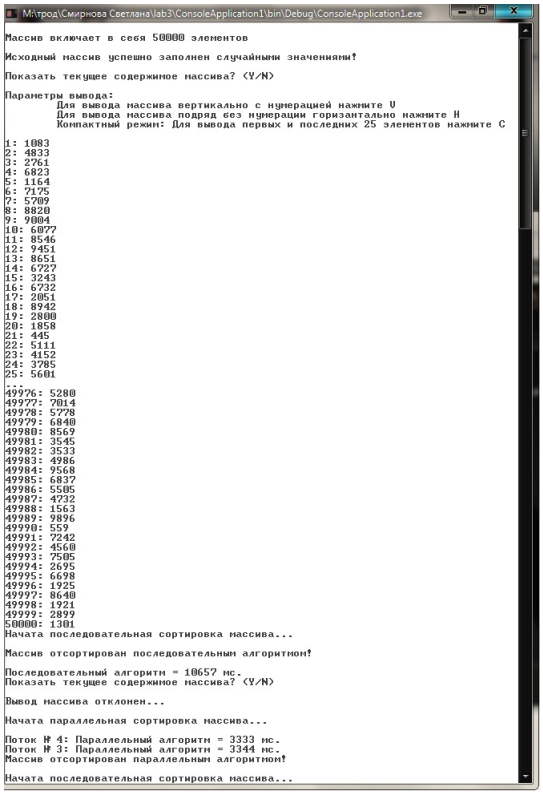

Рисунок 1

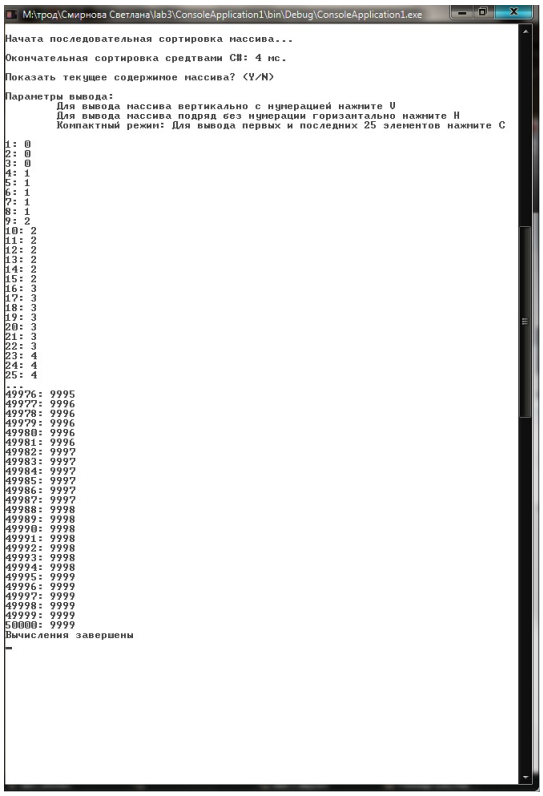

Рисунок 2

4 ВЫВОД

В результате выполнения лабораторной работы мы получили представление о возможности библиотеки Concurrent and Coordination Runtime для организации параллельных вычислений. Мы выяснили, что скорость работы параллельного алгоритма превосходит скорость работы последовательного алгоритма. Быстродействие параллельного алгоритма напрямую зависит от числа используемых ядер.