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-1

_____ *Мишкорудный А. И.*

"____" _____ _____

2016

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 9:

Разработать алгоритм умножения матрицы a ($m \times n$ элементов) на вектор b (n элементов) по следующей формуле: $c_i = \sum_{j=1}^{n} a_{ij} b_j, 1 \le i \le m$.

Рисунок 1

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; //хранение матрицы
        static int[] B;  //хранение вектор–столбца для умножения
        static int[] C;  //хранение результата
        static int m;    //количество строк матрицы
```

```
static int n;     //количество столбцов матрицы
static int nc;    //количество ядер

static void Test()
{
    nc = 2;
    ConsoleKey press;

    Console.WriteLine("Желаете задать размер матрицы самостоят
       ельно? [Y/N]\n");


        press = Console.ReadKey(true).Key;
    if (press != ConsoleKey.N & press != ConsoleKey.Y)
    {
        Console.WriteLine("Некорректная клавиша");
        press = ConsoleKey.N;
    }

    if (press == ConsoleKey.Y)
    {
        Console.Write("Введите количество строк матрицы: ");
        m = Convert.ToInt32(Console.ReadLine());
        Console.Write("Введите количество столбцов матрицы: ")
           ;
        n = Convert.ToInt32(Console.ReadLine());
    }
    else if (press == ConsoleKey.N)
    {


        m = 15000;
        n = 15000;
        Console.WriteLine("\nРазмеры матрицы заданы автоматиче
           ски и составляют {0} x {1}\n",m,n);
    }




    A = new int[m, n];
    B = new int[n];
```

```
C = new int[m];

Console.WriteLine("\n\nЖелаете заполнить матрицу и вектор-
   столбец самостоятельно? [Y/N]\n");
press = Console.ReadKey(true).Key;

if (press != ConsoleKey.N & press != ConsoleKey.Y)
{
    Console.WriteLine("Некорректная клавиша");
    press = ConsoleKey.N;
}

if (press == ConsoleKey.Y)
{
    Console.WriteLine("Ввод матрицы\n");
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Console.Write("A[{0},{1}] = ", i+1, j+1);
            A[i, j] = Convert.ToInt32(Console.ReadLine());
        }

    }

    Console.WriteLine("\n\nВвод вектор-столбца\n");
    for (int j = 0; j < n; j++)
    {
        Console.Write("B[{0}] = ", j+1);
        B[j] = Convert.ToInt32(Console.ReadLine());
    }
}

else if (press == ConsoleKey.N)
{
    Console.WriteLine("Заполнение случайными значениями
       ...\n");
    Random r = new Random();
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
```

```
                A[i, j] = r.Next(100);
            }
            for (int j = 0; j < n; j++)
                B[j] = r.Next(100);

            Console.WriteLine("Исходная матрица и вектор–столбец у
                спешно заполнены случайными значениями!\n");
        }
    }

    static void SequentialMul()
    {
        System.Diagnostics.Stopwatch sWatch = new System.
            Diagnostics.Stopwatch();
        sWatch.Start();
        for (int i = 0; i < m; i++)
        {
            C[i] = 0;
            //Console.WriteLine("\n");  //эта строка тоже относитс
                я к блоку проверки правильности
            for (int j = 0; j < n; j++)
            {
                C[i] += A[i, j] * B[j];
                //Внимание!
                //ниже закоментирован блок отображения проверки пр
                    авильности умножения
                //Если включить данный блок в исполняемый код,
                //то для корректного вывода проверки
                //рекомендуется брать матрицы небольшого размера
                //например, 3x3
                //
                //Console.Write(" {0} * {1} ", A[i,j], B[j]);
                //if (j+1 != n)
                //{
                //    Console.Write("+");
                //}
                //else
                //{
                //    Console.Write("= {0}", C[i]);
                //}
            }
```

```
        }
        //Console.WriteLine("\n");    //эта строка тоже относится к
            блоку проверки правильности
        sWatch.Stop();
        Console.WriteLine("Последовательный алгоритм = {0} мс.",
        sWatch.ElapsedMilliseconds.ToString());

    }

    static 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)(m / 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:
        Dispatcher d = new Dispatcher(nc, "Test Pool");
        DispatcherQueue dq = new DispatcherQueue("Test Queue", d);
        //Описывается порт, в который каждый экземпляр метода Mul
           ()
        //отправляет сообщение после завершения вычислений:
        Port<int> p = new Port<int>();
        //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<int>>(
               ClArr[i], p, Mul));
```

```
        //Первый параметр метода Arbiter.Activate      очередь дисп
            етчера,
        //который будет управлять выполнением задачи, второй парам
            етр
        //запускаемая задача.

        //С помощью метода Arbiter.MultipleItemReceive запускается
             задача
        //(приёмник), которая обрабатывает получение двух сообщени
            й портом p:
        Arbiter.Activate(dq, Arbiter.MultipleItemReceive(true, p,
            nc, delegate (int[] array)
        {
             dispResult();
             Console.WriteLine("Вычисления завершены");
             Console.ReadKey(true);
             Environment.Exit(0);
        }));
    }


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

        for (int i = data.start; i < data.stop; i++)
        {
             C[i] = 0;
             for (int j = 0; j < n; j++)
                 C[i] += A[i, j] * B[j];
        }
        sWatch.Stop();
        Console.WriteLine("Поток      {0}: Паралл. алгоритм = {1} м
            c.",
       Thread.CurrentThread.ManagedThreadId,
       sWatch.ElapsedMilliseconds.ToString());
        resp.Post(1);
    }
```

```
        static void dispResult()
        {

            int i;
            Console.WriteLine("Показать результат умножения? [Y/N]\n")
               ;

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

            if (press != ConsoleKey.N & press != ConsoleKey.Y)
            {
                Console.WriteLine("Некорректная клавиша");
                press = ConsoleKey.N;
            }

            if (press == ConsoleKey.Y)
            {
               for (i = 0; i < m; i++)
                  Console.WriteLine("C[{0}]: {1}", i + 1, C[i].
                     ToString());
            }

            if (press == ConsoleKey.N)
            {
                Console.WriteLine("Вывод результата отклонен.");
            }
        }

        static void Main(string[] args)
        {
            Test();
            SequentialMul();
            dispResult();
            ParallelMul();
        }
    }
}
```

3.3 Результат работы программы

Скриншот работы программы представлен на Рис. 2.

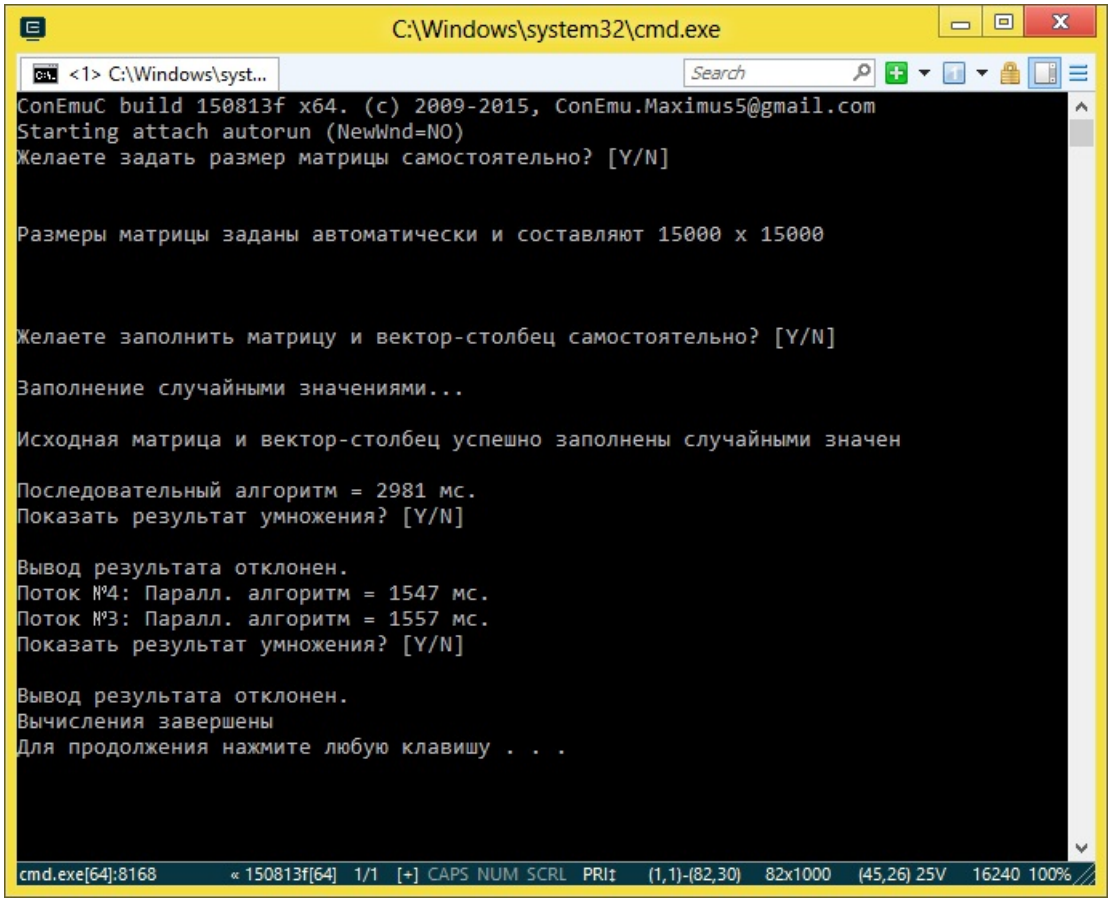

Рисунок 2

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.