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

_____ *Пономарёв Е. В.*

“____” _____ _____

2015

Подп. и дата	
Инв. дубл.	
Взам. инв.	
Подп. и дата	
Инв. подл.	

СОДЕРЖАНИЕ

1 ЦЕЛЬ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Цель работы: получить представления о возможностях используемой библиотеки Concurrent 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 18: Разработать алгоритм вычисления значения определённого интеграла с использованием метода Монте-Карло

3.2 Особенности реализации

В ходе выполнения лабораторной работы возникли некоторые сложности: получалось так, что подынтегральная функция задается в исходном коде программистом, а пользователь программы будет иметь возможность при каждом запуске программы считать лишь один и тот же интеграл. Мной было принято решение устранить эту проблему. Для этого я решил воспользоваться возможностями скриптового языка программирования Python, с помощью которого можно с легкостью реализовать ввод подынтегральной функции пользователем как строки, а затем преобразования введенной строки в исполняемый исходный код языка Python с помощью функции eval(). В разработанном модуле на языке Python содержатся две функции, реализующие ввод значений пользователем param_enter() и само интегрирование методом Монте-Карло monte_carlo() , а так же код, тестирующий работу программы, но выполняющийся только когда модуль запущен как отдельный файл, а не вызван сторонним приложением. После разработки и отладки модуля на Python, возник вопрос об осуществлении связи между модулем, написанным на C#, и разработанным модулем на Python. Решением этого вопроса стала установка нескольких дополнительных пакетов с помощью менеджера NuGet, а именно:

а) DynamicLanguageRuntime.1.1.0 - для создания и использования динамических объектов;

б) IronPython.2.7.5 - для взаимодействия со скриптами языка Python;

в) IronPython.StdLib.2.7.5 - для подключения стандартной библиотеки (нужна для использования random.py);

Подробное описание механизма взаимодействия модулей C# и Python дается в комментариях в соотвествующих местах листинга разработанных модулей.

3.3 Листинг программы

3.3.1 Модуль C#

```
  using    System;
using Microsoft.Ccr.Core;
using System.Threading;
using IronPython;
using IronPython.Hosting;
using Microsoft.Scripting;
using Microsoft.Scripting.Hosting;

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

    class Program
    {
        static int nc=2;    //количество ядер


        static void SequentialIntegration(dynamic monte_carlo, dynamic
            fun, dynamic Low, dynamic Up, dynamic step, dynamic N)
        {
```

```
        //взятие определенного интеграла по всему диапазону

        System.Diagnostics.Stopwatch sWatch = new System.
            Diagnostics.Stopwatch();
        sWatch.Start();

        dynamic result = monte_carlo(fun, Low, Up, N);

        sWatch.Stop();
        Console.WriteLine("Результат: " + result);
        Console.WriteLine("Последовательный алгоритм = {0} мс.",
        sWatch.ElapsedMilliseconds.ToString());

    }

    static void ParallelIntegration(dynamic step, dynamic Low,
        dynamic Up, ScriptScope scope)
    {
        // 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:
        // заполняем массив параметров
        dynamic Low_temp = Low;
        for (int i = 0; i < nc; i++)
        {
            ClArr[i].start = Low_temp;
            if (i + 1 == nc)
                ClArr[i].stop = Up;
            else
                ClArr[i].stop = Low_temp + step;

            Low_temp = Low_temp + 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 Int
```

```
            ()
        //отправляет сообщение после завершения вычислений:
        Port<int> p = new Port<int>();
        //Метод Arbiter.Activate помещает в очередь диспетчера две
            задачи(два
        //экземпляра метода Mul):

        System.Diagnostics.Stopwatch ssWatch = new System.
            Diagnostics.Stopwatch();
        ssWatch.Start();

        for (int i = 0; i < nc; i++)
            Arbiter.Activate(dq, new Task<InputData, Port<int>,
                ScriptScope>(ClArr[i], p, scope, Int));
        //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)
        {
            Console.WriteLine("Вычисления завершены");
            ssWatch.Stop();
            Console.WriteLine("Полное время работы {0} мс",
                ssWatch.ElapsedMilliseconds.ToString());
            Console.ReadKey(true);
            Environment.Exit(0);
        }));



        Console.ReadKey(true);
        Environment.Exit(0);
    }
```

```
static void Int(InputData data, Port<int> resp, ScriptScope
    scope)
{
    //достаем функцию для интегрирования
    dynamic monte_carlo = scope.GetVariable("monte_carlo");
    //достаем необходимые переменные
    dynamic fun = scope.GetVariable("fun");
    dynamic N = scope.GetVariable("N");
    dynamic result;

    System.Diagnostics.Stopwatch sWatch = new System.
        Diagnostics.Stopwatch();
    sWatch.Start();

    result = monte_carlo(fun, data.start, data.stop, N);

    sWatch.Stop();

    Console.WriteLine("Поток      {0}: Паралл. алгоритм = {1} мс
        ,. Результат: {2}",
    Thread.CurrentThread.ManagedThreadId,
    sWatch.ElapsedMilliseconds.ToString(), result);
    resp.Post(1);
}


static void Main(string[] args)
{

    //Класс ScriptEngine применяется для создания движка, выпо
        лняющего скрипт.
    //Объект ScriptScope позволяет взаимодействовать со скрипт
        ом, получая или устанавливая его переменные, получая сс
        ылки на функции.


    ScriptEngine engine = Python.CreateEngine();
    ScriptScope scope = engine.CreateScope();

    //В вычислительном модуле python используется модуль
        random.py
```

```
            //он находится в IronPython.StdLib
            //для работы программы необходимо подключить папку с
               StdLib:
            var paths = engine.GetSearchPaths();
            //путь к папке
            paths.Add(@"D:\IronPython.StdLib.2.7.5\content\Lib");
            engine.SetSearchPaths(paths);

            //непосредственный запуск модуля
            engine.ExecuteFile("D://monte-carlo.py", scope);

            //теперь можно "разобрать" запущенный скрипт на части, выт
               аскивая из него необходимые функции и переменные
            //вытаскиваем две функции
            dynamic param_enter = scope.GetVariable("param_enter");
            dynamic monte_carlo = scope.GetVariable("monte_carlo");

            //запускаем одну из них
            param_enter();

            //вытаскиваем введенные пользователем данные
            dynamic fun = scope.GetVariable("fun");
            dynamic Low = scope.GetVariable("Low");
            dynamic Up = scope.GetVariable("Up");
            dynamic step = scope.GetVariable("step");
            dynamic N = scope.GetVariable("N");

            //интегрируем целостно
            SequentialIntegration(monte_carlo,fun,Low,Up,step,N);

            //интегрируем параллельно
            ParallelIntegration(step,Low,Up,scope);
        }
    }
}
```

3.3.2 Модуль Python

```
# -*- coding: utf-8 -*-
import random as R
from math import *
```

```
def monte_carlo(fun, Low, Up, N):
    s=0
    for i in range(1,N):
        x=Low+(Up-Low)*R.random()
        s=s+eval(fun)
    return ((Up-Low)*s)/N


def param_enter():
    global fun
    global Low,Up,step,N

    print("Интеграл берется по dx")

    fun = input('''\nВНИМАНИЕ! Математическая нотация Python\n
    Для возведения в степень вместо ^ используется **\n
    Введите подынтегральную функцию, заключенную в кавычки\n
    Например, "x**2":       ''')
    Low = int(input("Нижний предел интегрирования = "))
    Up = int(input("Верхний предел интегрирования = "))
    N = int(input("Точность (кол-во бросаемых точек, например, 10000)
       = "))

    #отрезок для подсчета одним клиентом
    step = (Up-Low)//2



if __name__ == '__main__':
    #код этого блока будет выполнен только если этот модуль
    #будет запущен как отдельный файл
    print("Модуль запущен как отдельный файл!\nКавычки при вводе не ну
       жны!")
    param_enter()

    I = monte_carlo(fun, Low, Up, N)

    print(I)
```

3.4 Результат работы программы

Скриншот работы программы представлен на Рис. 1.

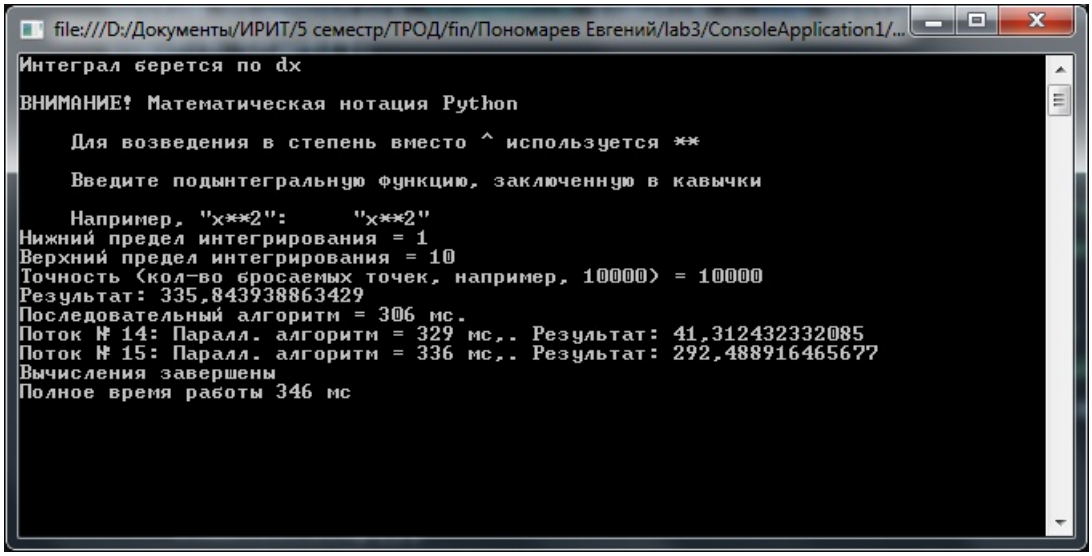

Рисунок 1

4 ВЫВОД

В результате выполнения лабораторной работы я получил представление о возможностях библиотеки Concurrent and Coordination Runtime для организации параллельных вычислений. Анализируя результаты работы программы, можно выделить положительные и отрицательные стороны. К положительным сторонам можно отнести предоставление пользователю возможности самостоятельно задавать подынтегральную функцию любой сложности. Отрицательной особенностью выполненной реализации является скорость её работы: она примерно одинакова для каждого исполняемого вычислительного потока и практически не зависит от сложности вычислений, осуществляемых в потоке. Такое поведение обусловлено тем, что все расчеты осуществляются вызовом функции из модуля Python, что приводит к необходимости каждый раз обращаться к сеансу интерпретатора, работающего в реальном времени, то есть имеющего динамическую структуру исполнения. Предположительно, эти недостатки возникли из-за рассогласования используемых средств: по заданию к лабораторной работе, обязательным было условие использования библиотеки CCR для распараллеливания алгоритма. Однако, в Python так же имеются средства для реализации параллельных вычислений, и использование одного из этих подходов целостно, а не слияние двух подходов, возможно, могло бы дать выйгрыш в производительности.