Нижегородский Государственный Технический Университет

Технологии распределенной обработки данных

Лабораторная работа №2

«Распараллеливание алгоритма с помощью библиотеки Concurrent

and Coordination Runtime»

Выполнил:

13ВМ ЗВФ

Вариант №6

Чернышев В.В.

Проверил:

Гай B.Е

г. Нижний Новгород

2015 г.

**Цель работы:**

получить представления о возможностях библиотеки

Concurrent and Coordination Runtime для организации параллельных

вычислений.

**Задание:**

Вариант №6: Разработать алгоритм вычисления значения определённого интеграла с использованием метода парабол (метод Симпсона). Разбив интервал интегрирования на 2⋅*N* равных частей, приближённое значение интеграла можно вычислить следующим образом: 

Интеграл: 

(Используемые обозначения: *a*, *b* – нижний и верхний пределы интегрирования, 2⋅*N* – число равных частей на которые делится отрезок [*a*; *b*], *h* – шаг)

**Выполнение:**

Для выполнения работы была использована Microsoft Visual Studio 2015 с подключенной сторонней библиотекой Concurrent and Coordination Runtime(Microsoft.Ccr.Core.dll), входящей в состав Microsoft Robotics Developer Studio.

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.

В данной работе было разработано консольное приложение, выполняющее вычисление определенного интеграла. По заданному числу точек программа сначала высчитывает интеграл в однопоточном режиме, а затем с помощью параллельного выполнения. В обоих случаях пользователю выводится время выполнения программы.

**Метод Симпсона:**

(для чётных)

По заданию: a=0; b=1; h=(b-a)/n;

**Листинг:**

Подключаем библиотеку CCR:

using Microsoft.Ccr.Core; // подключена библиотека для параллельных вычислений

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class Program

{

В методе Main происходит выполнение однопоточных и запуск параллельных вычислений:

static void Main(string[] args)

{

double a = 0; // нижний предел

double b = 1; // предел предел

for (int j = 0; j < 10; j++) // зацикливаем выполнение программы 10 раз

// для большей наглядности

{

Stopwatch sWatch = new Stopwatch(); // таймер выполнения программы в однопоточном режиме

sWatch.Start(); // начало таймера

Console.WriteLine("Число (чётное) разбиения отрезков n:"); // введите значение n

String m = Console.ReadLine(); // чтение введенного значения

int n = int.Parse(m); // string в int

double h = ((b - a) / n); // получем значение шаг h

double intgrl;

intgrl = (h / 3) \* (Program.sum(h, n, a, b)); // считает интеграл

Console.WriteLine("1 Результат вычислений: " + intgrl); // вывод значения интеграла

sWatch.Stop(); // конец таймера

Console.WriteLine("Время выполнения в однопоточном режиме:" +sWatch.Elapsed.ToString()); //вывод значения таймера

Program.parallel(n, h, a, b); // запуск метода параллельных вычислений

Console.WriteLine("Многопоточный режим:"); // чтобы

Console.WriteLine(" "); // окно

Console.ReadKey(); // не закрылось сразу же

}

}

Методы, использующиеся в методах Main и Mul, для вычисления интеграла:

static double Y(double p)

{

return ((p\*p\*p) / ((p\*p+1)\*(p \* p + 1))); // заданная функция

}

public static double sum(double h, int n, double a, double b) //считает сумму F(Xi)

{

double S = 0; // сумма F(Xi)

double Xi;

for (int i = 1; i < n; i++)

{

Xi = a + (h \* i);

S = S + 4 \* Y(Xi);

i = i + 1;

Xi = a + (h \* i);

S = S + 2 \* Y(Xi);

}

S = (S + Y(a) + Y(b));

return S;

}

}

}

Далее идут методы необходимые для параллельных вычислений:

public class InputData

{

public int start; // начало диапазона

public int stop; // конец диапазона

public double h; // шаг

}

Метод parallel, в котором создается диспетчер, описываются порты, помещаются задачи в очередь, выполняется получение общего результата вычислений:

public static void parallel(int n, double h)

{

int nc = 4; // число ядер - потоков

InputData[] ClArr = new InputData[nc];

for (int i = 0; i < nc; i++)

ClArr[i] = new InputData();

int step =(Int32) (n / nc);

// заполняем массив параметров

int c = 0;

for (int i = 0; i < nc; i++)

{

ClArr[i].start = c + 1;

ClArr[i].stop = c + step;

ClArr[i].h = h;

c = c + step;

} // создаем диспечер

Dispatcher d = new Dispatcher(nc, "Test Pool");

DispatcherQueue dq = new DispatcherQueue("Test Queue", d);

// описываем порт, в который отправляются сообщения после завершения вычислений

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

a = 0; // значение a

b = 1; // значение b

for (int i = 0; i < nc; i++)

// метод Arbiter помещает в очередь диспечера задачи

Arbiter.Activate(dq, new Task<InputData, Port<double>>(ClArr[i], p, Mul));

Arbiter.Activate(dq, Arbiter.MultipleItemReceive(true, p, nc,

delegate (double[] array)

{

double res = (h / 3) \* (Y(a) + Y(b)+array[0]+ array[1]+array[2]+array[3]);

// вычисление результата путем сложения значений полученных в каждом потоке Console.WriteLine("Вычисления завершены = " + res);

}));

}

Метод Mul, который используется потоком для вычисления своей части интеграла:

public static void Mul(InputData data, Port<double> resp)

// метод для параллельных вычислений

{

Stopwatch sWatch = new Stopwatch(); // таймер для наблюдения за временем выполнения потока

sWatch.Start();

double S = 0; // сумма F(Xi)

double Xi;

double a = 0;

double b = 1;

double h = data.h;

for (int i = data.start; i < data.stop; i++)

{

Xi = a + (h \* i);

S = S + 4 \* Y(Xi);

i = i + 1;

Xi = a + (h \* i);

S = S + 2 \* Y(Xi);

}

sWatch.Stop();

Console.WriteLine("Поток № {0}: Паралл. алгоритм = {1} мс.", Thread.CurrentThread.ManagedThreadId, sWatch.Elapsed.ToString());

resp.Post(S);

}

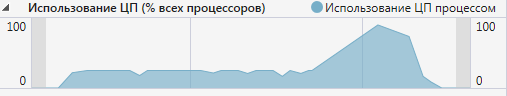
**Результаты работы программы:**

В результате работы программы был успешно вычислен заданный интеграл в однопоточном режиме и в режиме параллельных вычислений. Во втором случае время выполнения вычислений сократилось в несколько раз.

Пример работы программы:



Использование всех ядер ЦП программой:



Левая часть графика отображает использование ЦП во время однопоточного режима выполнения, правая часть графика показывает использование ЦП во время параллельных вычислений.

**Вывод:**

в ходе лабораторной работы было получено представление о возможностях библиотеки Concurrent and Coordination Runtime для организации параллельных вычислений. Была успешно создана программа, реализующая вычисления заданного интеграла в однопоточном режиме и при использовании параллельных вычислений. На практике были показаны преимущества параллельных вычислений с использованием библиотеки Concurrent and Coordination Runtime.