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
Носов А.В.

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 скалярного произведения n-мерных векторов

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

Развмерность умножаемых векторов, количество компонент векторов, обрабатываемых в одном потоке, вектора a и b и переменные для хранения результата определены глобально:

```
const int SIZE = 10000000;
const int SIZE_THREAD = 10000;

public double[] a;
public double[] b;

double result = 0;
double planeResult = 0;
```

В методе Start() запускаются вычисления. Сначала в методе выполняется вычисление скалярного скалярного произведения n-мерных векторов последовательным алгоритмом, затем та же задача решается с помощью параллельных вычислений. Рассмотрим этот метод: Выполняется инициализация структур данных:

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

    a = new double[SIZE];
    b = new double[SIZE];

    Random r = new Random();

    for (int i = 0; i < SIZE; ++i)
    {
        a[i] = r.Next() / 3.1415;
        b[i] = r.Next() / 3.1415;
    }
```

2.2. Последовательный алгоритм вычисления скалярного произведения n-мерных векторов

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

for (int k = 0; k < SIZE; ++k)
    planeResult += (a[k] * b[k]);

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

2.3. Параллельный алгоритм вычисления скалярного произведения n-мерных векторов

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 double[] a;
public double[] b;

Поле a класса хранит компоненты векторов. Поля рассчитываются с помощью экземпляра вычислительного метода.

```
int nc = 8;

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

Далее, задаются исходные данные для каждого экземпляра вычислительного метода:

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

    data[i].a = new double[SIZE_THREAD];
    for (int j = 0; j < SIZE_THREAD; ++j)
        data[i].a[j] = a[i * SIZE_THREAD + j];

    data[i].b = new double[SIZE_THREAD];
    for (int j = 0; j < SIZE_THREAD; ++j)
        data[i].b[j] = b[i * SIZE_THREAD + j];
}
```

Создаётся диспетчер с пулом из 8 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<double> port = new Port<double>();
```

Метод Arbiter.Activate помещает задачи в очередь диспетчера:

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

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 сообщения:

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

```
Arbiter.Activate(Environment.TaskQueue, Arbiter.MultipleItemReceive(true, port, SIZE/SIZE_THREAD, delegate(double[]
{
    for (int i = 0; i < array.GetLength(0); ++i)
        result += array[i];

    Console.WriteLine("Plane result: {0}", planeResult);
    Console.WriteLine("Parallel result: {0}", result);


    Console.WriteLine("Computation completed");
    Console.WriteLine("Parallel vector mul time: {0}ms", fullParallelTime.ToString());
    Console.WriteLine("Linear vector mul time: {0}ms", planeTime);
}));
```

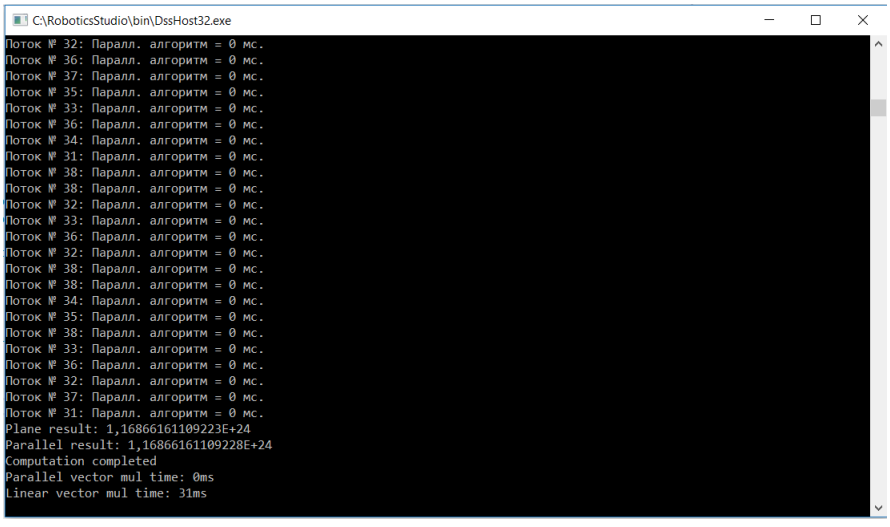
каждом из потоков. Такими действиями является вывод результата, а также подсчет результатов интегрирования подотрезков.

Метод Mul() выполняет скалярное произведение:

```
public static void Mul(InputData data, Port<double> resp)
{
    Stopwatch sWatch = new Stopwatch();
    sWatch.Start();
    double result = 0;
    for(int i = 0; i < SIZE_THREAD; ++i)
        result += data.a[i] * data.b[i];
    sWatch.Stop();

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

Метод Mul() имеет два параметра: 1) индекс, хранящий значение элемента массива, который определяет параметры, передаваемые на вход метода; 2) порт завершения, в который отправляется экземпляр класса double после завершения вычислений. После завершения вычислений метод Mul отправляет в порт p значение экземпляра класса double, 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 вычисления скалярного произведения n-мерных векторов. Исходя из работы программы можно сделать вывод о том, что время выполнения последовательного алгоритма примерно в несколько раз больше, чем время выполнения параллельного алгоритма, также результаты при последовательном и параллельном алгоритмах одинаковые, то есть программа работает правильно.