**CS 590 – PARALLEL AND DISTRIBUTED COMPUTING**

**HOMEWORK 4**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** October 2, 2018

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**SOURCE CODE:** 8](#_Toc446970374)

[**CONCLUSION:**](#_Toc446970375) 23

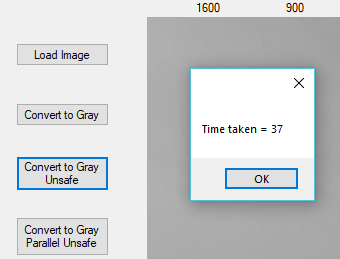
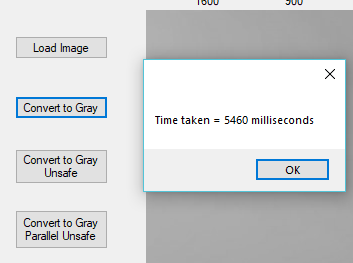
**INTRODUCTION**

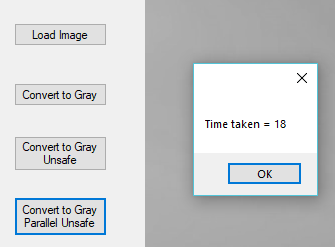
The porpuse of this assignment is to keep practicing different methods and attributes from the Task Parallel Library. Completing the different examples I am going to be able to see the difference in time of a sequential and parallel operations.

**SCREEN SHOTS:**

**Image Procesing:**

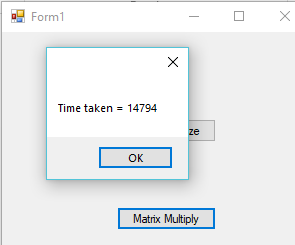
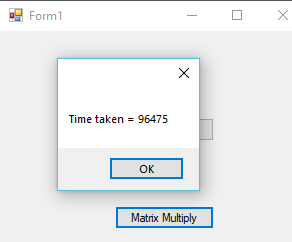
This example consists in load an image and convert or the bits to grey. In order to do that we have different methods: regular for loop, using unsafe coding (using pointers) and using unsafe code and Parallel.For. As we can see on the outputs the fastest way is using unsafe coding and Parallel.For

****

****

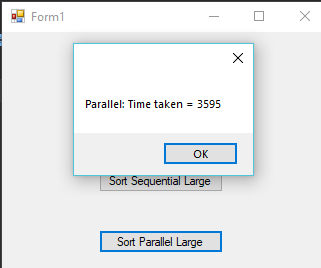
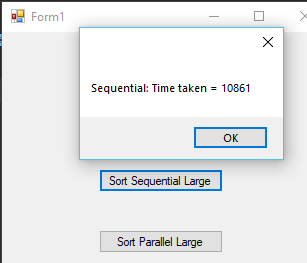
**Matrix Multiply:**

In the next example we are multiplying matrixes using two different ways: using sequential methods and using Parallel.For. The process will speed up depending on the number of cores of the laptop but generally, Parallel.For runs faster.

****

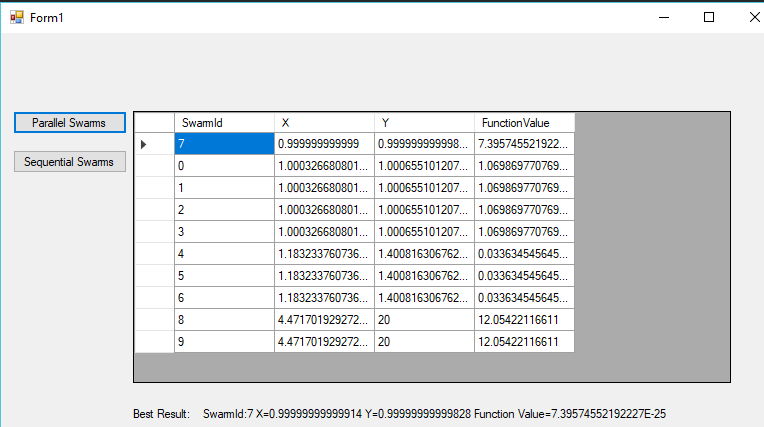
**Parallel QuickSort:**

Once again, we are checking the difference of timae taken by a sequential operation and a Parallel.For. This time we are checking it using the QuickSort algorithm.

****

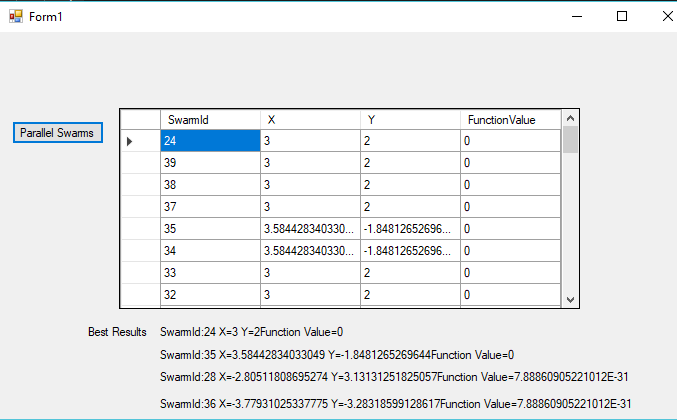
**Intelligence Swarm:**

In this example we are using the Particle Swarm Optimization in order to find the best solution (optimized solution) of a function. In order to do that, we have created 10 different swarms to find the optimized solution using Parallel work.



**Exercise 2 (Himmelblau):**

This exercise does the same as the Rosemborg’s one however, this time we are using Himmelblau’s equations which has 4 solutions. In order to display the 4 solutions, I implemented the Interface IEqualityComparer in order to override the Distinct() method so the List is sorted and gets unique solutions, no duplicates on the list. That means that the first 4 Swarms in the List are the optimized solutions



**SOURCE CODE:**

**Image Procesing:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ImageProcessing

{

public partial class Form1 : Form

{

Bitmap bmpOrig;

Bitmap bmpProc;

public Form1()

{

InitializeComponent();

}

private void btnLoadImage\_Click(object sender, EventArgs e)

{

OpenFileDialog ofd = new OpenFileDialog();

ofd.InitialDirectory = "C:\\Users\\ivans\_000\\Desktop\\Fall2018\\PARALLEL\\CS590\_SANGINES\_HW4";

if (ofd.ShowDialog() == DialogResult.OK)

{

bmpOrig = new Bitmap(ofd.FileName);

pic1.Image = bmpOrig;

lblWidth.Text = bmpOrig.Width.ToString();

lblHeight.Text = bmpOrig.Height.ToString();

}

}

private void btnConvertToGray\_Click(object sender, EventArgs e)

{

bmpProc = new Bitmap(bmpOrig.Width, bmpOrig.Height);

object olock = new object();

Stopwatch sw = new Stopwatch();

sw.Start();

int width = bmpOrig.Width;

int height = bmpOrig.Height;

Parallel.For(0, width, (x) =>

//for (int x = 0; x < width; x++)

{

for (int y = 0; y < height; y++)

{

Color currentPixel;

lock (olock)

{

currentPixel = bmpOrig.GetPixel(x, y);

}

int red = currentPixel.R;

int green = currentPixel.G;

int blue = currentPixel.B;

int gray = (int)(0.299 \* red + 0.587 \* green + 0.114 \* blue);

lock (olock)

{

bmpProc.SetPixel(x, y, Color.FromArgb(gray, gray, gray));

}

}

//}

});

sw.Stop();

pic1.Image = bmpProc;

MessageBox.Show("Time taken = " + sw.ElapsedMilliseconds.ToString() +

" milliseconds");

}

private void btnConvertToGrayUnsafe\_Click(object sender, EventArgs e)

{

bmpProc = new Bitmap(bmpOrig);

Stopwatch sw = new Stopwatch();

sw.Start();

unsafe // unsafe block allows direct memory access

{ // like C/C++ pointers

// Project has to be compiled with unsafe option

BitmapData bitmapData = bmpProc.LockBits(new Rectangle(0, 0, bmpProc.Width, bmpProc.Height),

ImageLockMode.ReadWrite, bmpProc.PixelFormat);

int bytesPerPixel = System.Drawing.Bitmap.GetPixelFormatSize(bmpProc.PixelFormat) / 8;

int height = bmpProc.Height;

int widthinBytes = bitmapData.Width \* bytesPerPixel;

byte\* ptr = (byte\*)bitmapData.Scan0; // point to the first pixel

for (int y = 0; y < height; y++)

{

byte\* currentLine = ptr + (y \* bitmapData.Stride);

for (int x = 0; x < widthinBytes; x = x + bytesPerPixel)

{

int blue = currentLine[x];

int green = currentLine[x + 1];

int red = currentLine[x + 2];

int gray = (int)(0.299 \* red + 0.587 \* green + 0.114 \* blue);

currentLine[x] = (byte)gray;

currentLine[x + 1] = (byte)gray;

currentLine[x + 2] = (byte)gray;

}

}

bmpProc.UnlockBits(bitmapData);

} // end of unsafe

sw.Stop();

pic1.Image = bmpProc;

MessageBox.Show("Time taken = " + sw.ElapsedMilliseconds.ToString());

}

private void btnConvertToGrayParalle\_Click(object sender, EventArgs e)

{

bmpProc = new Bitmap(bmpOrig);

Stopwatch sw = new Stopwatch();

sw.Start();

unsafe // unsafe block allows direct memory access

{ // like C/C++ pointers

// Project has to be compiled with unsafe option

BitmapData bitmapData =

bmpProc.LockBits(new Rectangle(0, 0, bmpProc.Width,

bmpProc.Height),

ImageLockMode.ReadWrite, bmpProc.PixelFormat);

int bytesPerPixel = System.Drawing.Bitmap.GetPixelFormatSize(bmpProc.PixelFormat) / 8;

int height = bmpProc.Height;

int widthinBytes = bitmapData.Width \* bytesPerPixel;

byte\* ptr = (byte\*)bitmapData.Scan0; // point to the first pixel

Parallel.For(0, height, (y) =>

//for (int y = 0; y < height; y++)

{

byte\* currentLine = ptr + (y \* bitmapData.Stride);

for (int x = 0; x < widthinBytes; x = x + bytesPerPixel)

{

int blue = currentLine[x];

int green = currentLine[x + 1];

int red = currentLine[x + 2];

int gray = (int)(0.299 \* red + 0.587 \* green + 0.114 \* blue);

currentLine[x] = (byte)gray;

currentLine[x + 1] = (byte)gray;

currentLine[x + 2] = (byte)gray;

}

});

bmpProc.UnlockBits(bitmapData);

} // end of unsafe

sw.Stop();

pic1.Image = bmpProc;

MessageBox.Show("Time taken = " + sw.ElapsedMilliseconds.ToString());

}

}

}

**Matrix Multiply:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MatrixMultiply

{

public partial class Form1 : Form

{

double[,] A = null;

double[,] B = null;

double[,] C = null;

Random rand = new Random();

public Form1()

{

InitializeComponent();

}

private void btnInitializeMatrix\_Click(object sender, EventArgs e)

{

int size = 1000;

A = new double[size, size];

B = new double[size, size];

C = new double[size, size];

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

for (int j = 0; j < size; j++)

{

A[i, j] = rand.NextDouble() \* 100;

B[i, j] = rand.NextDouble() \* 50;

}

}

public double[,] MatrixMultiply(double[,] X, double[,] Y)

{

double[,] Res = new double[X.GetLength(0), X.GetLength(0)];

int size = X.GetLength(0);

//for (int i = 0; i < size; i++)

Parallel.For(0, size, (i) =>

{

for (int k = 0; k < size; k++)

//for (int j = 0; j < size; j++)

{

for (int j = 0; j < size; j++)

//for (int k = 0; k < size; k++)

{

Res[i, j] = Res[i, j] + X[i, k] \* Y[k, j];

}

}

//}

});

return Res;

}

private void btnMatrixMultiply\_Click(object sender, EventArgs e)

{

Stopwatch sw = new Stopwatch();

sw.Start();

C = MatrixMultiply(A, B);

sw.Stop();

MessageBox.Show("Time taken = " + sw.ElapsedMilliseconds.ToString());

}

}

}

**Parallel QuickSort:**

**CLASS:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ParallelQuickSort

{

class QuickSortAlgo

{

public static void Quicksort<T>(T[] data, int left, int right)

where T : IComparable<T>

{

if (right > left)

{

int pivot = Partition(data, left, right);

Quicksort(data, left, pivot - 1);

Quicksort(data, pivot + 1, right);

}

}

public static void QuickSortParallel<T>(T[] data, int left, int right)

where T : IComparable<T>

{

const int SEQUENTIAL\_THRESHOLD = 1000;

if (right > left)

{

if ((right - left) < SEQUENTIAL\_THRESHOLD)

Quicksort(data, left, right); // do sequential sort

else

{

int pivot = Partition(data, left, right);

Parallel.Invoke(

() => QuickSortParallel(data, left, pivot - 1),

() => QuickSortParallel(data, pivot + 1, right)

);

}

}

}

private static int Partition<T>(T[] data, int low, int high)

where T : IComparable<T>

{

int left, right;

T pivotItem;

pivotItem = data[low];

int pivot = left = low;

right = high;

while (left < right)

{

while (data[left].CompareTo(pivotItem) <= 0)

{

if (left < data.Length - 1)

left++;

else

break;

}

while (data[right].CompareTo(pivotItem) > 0)

{

if (right > 0)

right--;

else

break;

}

if (left < right) Swap(data, left, right);

}

data[low] = data[right];

data[right] = pivotItem;

return right;

}

private static void Swap<T>(T[] data, int i, int j)

{

T temp = data[i];

data[i] = data[j];

data[j] = temp;

}

}

}

**FORM:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ParallelQuickSort

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

double[] data = { 3, 9, 15, 7, 8, 4, 11 };

QuickSortAlgo.Quicksort<double>(data, 0, data.Length - 1);

string out1 = "";

foreach (int n in data)

out1 += n.ToString() + " " + "\n";

MessageBox.Show(out1);

}

void InitData(double[] data)

{

Random rand = new Random();

for (int i = 0; i < data.Length; i++)

data[i] = rand.NextDouble() \* 1000 + 5;

}

private void button2\_Click(object sender, EventArgs e)

{

int size = 10000000;

double[] data = new double[size];

InitData(data);

Stopwatch sw = new Stopwatch();

sw.Start();

QuickSortAlgo.Quicksort<double>(data, 0, data.Length - 1);

sw.Stop();

MessageBox.Show("Sequential: Time taken = " +

sw.ElapsedMilliseconds.ToString());

}

private void button3\_Click(object sender, EventArgs e)

{

int size = 10000000;

double[] data = new double[size];

InitData(data);

Stopwatch sw = new Stopwatch();

sw.Start();

QuickSortAlgo.QuickSortParallel<double>(data, 0, data.Length - 1);

sw.Stop();

MessageBox.Show("Parallel: Time taken = " + sw.ElapsedMilliseconds.ToString());

}

}

}

**Intelligence Swarm (Rosemborg):**

**FORM:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ParallelSwarmInteligence

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void ParallelSwarm\_Click(object sender, EventArgs e)

{

int numSwarmsToLaunch = 10;

Task<SwarmResult>[] TaskArr = new Task<SwarmResult>[numSwarmsToLaunch];

for (int i = 0; i < TaskArr.Length; i++)

{

TaskArr[i] = Task.Factory.StartNew<SwarmResult>(

(obj) =>

{

SwarmSystem ss = new SwarmSystem((int)obj);

ss.Initialize();

SwarmResult sr = ss.DoPSO();

return sr;

}, i);

}

//Task.WaitAll(TaskArr); // wait for all tasks to finish

List<SwarmResult> RList = new List<SwarmResult>();

Task tskFinal = Task.Factory.ContinueWhenAll(TaskArr,

(tsks) =>

{

Console.WriteLine(tsks.Length.ToString() + " tasks");

for (int i = 0; i < tsks.Length; i++)

RList.Add(tsks[i].Result);

});

tskFinal.Wait();

RList.Sort();

dg1.DataSource = RList;

dg1.Refresh();

lblResult.Text = RList[0].ToString();

}

private void button1\_Click(object sender, EventArgs e)

{

SwarmSystem ss = new SwarmSystem(0);

ss.Initialize();

SwarmResult sr = ss.DoPSO();

lblResult.Text = sr.ToString();

}

}

}

**PARTICLE CLASS**

using System;

namespace ParallelSwarmInteligence

{

class Particle

{

public double W { get; set; } // inertia or weight

public double C1 { get; set; } // cognitive social const

public double C2 { get; set; }

Random ran = new Random();

// we will be solving a 2-D problem in x and y

// so there will be two components to velocity and position

public double Xx { get; set; } // poistion in x

public double Xy { get; set; } // position in y

public double Vx { get; set; } // velocity in x

public double Vy { get; set; } // velocity in y

public void UpdateVelocity(double Px, double Py,

double Gx, double Gy)

{

// P is the current best position of any particle in the swarm

// G is the global best position discovered so far

Vx = W \* Vx + C1 \* ran.NextDouble() \* (Px - Xx) +

C2 \* ran.NextDouble() \* (Gx - Xx);

if (Vx > 5)

Vx = 5;

if (Vx < -5)

Vx = -5;

Vy = W \* Vy + C1 \* ran.NextDouble() \* (Py - Xy) +

C2 \* ran.NextDouble() \* (Gy - Xy);

if (Vy > 5)

Vy = 5;

if (Vy < -5)

Vy = -5;

}

public void UpdatePosition()

{

Xx = Xx + Vx;

// we need to put some bounds on the position

if (Xx > 20)

Xx = 20;

if (Xx < -20)

Xx = -20;

Xy = Xy + Vy;

if (Xy > 20)

Xy = 20;

if (Xy < -20)

Xy = -20;

}

}

}

**SYSTEM CLASS:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ParallelSwarmInteligence

{

class SwarmSystem

{

public SwarmSystem(int snum)

{

this.swarmNum = snum;

}

int swarmNum;

public int SwarmNum

{

get { return swarmNum; }

}

public List<Particle> PList = new List<Particle>();

public double Px { get; set; }

public double Py { get; set; }

public double Gx { get; set; }

public double Gy { get; set; }

public void Initialize()

{

Random ran = new Random();

for (int i = 0; i < 50; i++) // 50 particles in swarm

{

Particle p = new Particle();

p.W = 0.73;

p.C1 = 1.4;

p.C2 = 1.5;

p.Xx = ran.NextDouble() \* 20;

p.Xy = ran.NextDouble() \* 20;

double num = ran.NextDouble();

if (num > 0.5)

{

p.Xx = -1 \* p.Xx;

p.Xy = -1 \* p.Xy;

}

p.Vx = ran.NextDouble() \* 5;

p.Vy = ran.NextDouble() \* 5;

num = ran.NextDouble();

if (num > 0.5)

{

p.Vx = -1 \* p.Vx;

p.Vy = -1 \* p.Vy;

}

PList.Add(p);

}

}

public double FunctionToSolve(double x, double y)

{

// Rosenbrock function

double res = (1 - x) \* (1 - x) + 100 \* (y - (x \* x)) \* (y - (x \* x));

return res;

}

public SwarmResult DoPSO() // Particle movement to achieve

{ // for particle swarm optimization

Gx = PList[0].Xx;

Gy = PList[0].Xy;

for (int i = 0; i < 1000; i++) // ietrations

{

// find best position in the swarm

Px = PList[0].Xx;

Py = PList[0].Xy;

foreach (Particle pt in PList)

{

if (Math.Abs(FunctionToSolve(pt.Xx, pt.Xy)) <

Math.Abs(FunctionToSolve(Px, Py)))

{

Px = pt.Xx;

Py = pt.Xy;

}

}

if (Math.Abs(FunctionToSolve(Px, Py)) <

Math.Abs(FunctionToSolve(Gx, Gy)))

{

Gx = Px;

Gy = Py;

}

foreach (Particle pt in PList)

{

pt.UpdateVelocity(Px, Py, Gx, Gy);

pt.UpdatePosition();

}

}

SwarmResult sr = new SwarmResult

{

SwarmId = swarmNum,

X = Gx,

Y = Gy,

FunctionValue = FunctionToSolve(Gx, Gy)

};

return sr;

}

}

}

**RESULT CLASS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ParallelSwarmInteligence

{

class SwarmResult : IComparable<SwarmResult>

{

public int SwarmId { get; set; }

public double X { get; set; }

public double Y { get; set; }

public double FunctionValue { get; set; }

public override string ToString()

{

return "SwarmId:" + SwarmId.ToString() +

" X=" + X.ToString() +

" Y=" + Y.ToString() +

" Function Value=" + FunctionValue.ToString();

}

public int CompareTo(SwarmResult other)

{

return this.FunctionValue.CompareTo(other.FunctionValue);

}

}

}

**Exercise 2 (Himmelblau):**

The code for the SYSTEM and PARTICLE classes are the same. It just changes the RESULT class and the FORM:

**SWARM RESULT CLASS:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Himmelblau\_s

{

class SwarmResult : IComparable<SwarmResult>, IEqualityComparer<SwarmResult>

{

public int SwarmId { get; set; }

public double X { get; set; }

public double Y { get; set; }

public double FunctionValue { get; set; }

public override string ToString()

{

return "SwarmId:" + SwarmId.ToString() +

" X=" + X.ToString() +

" Y=" + Y.ToString() +

"Function Value=" + FunctionValue.ToString();

}

public int CompareTo(SwarmResult other)

{

return this.FunctionValue.CompareTo(other.FunctionValue);

}

public bool Equals(SwarmResult first, SwarmResult second)

{

//I rounded because I had cases where the last decimal place was different

return Math.Round(first.X,2)==Math.Round(second.X,2);

}

public int GetHashCode(SwarmResult obj)

{

return obj.X.GetHashCode();

}

}

}

**FORM:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Himmelblau\_s

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

int numSwarmsToLaunch = 50;

Task<SwarmResult>[] TaskArr = new Task<SwarmResult>[numSwarmsToLaunch];

for (int i = 0; i < TaskArr.Length; i++)

{

TaskArr[i] = Task.Factory.StartNew<SwarmResult>(

(obj) =>

{

SwarmSystem ss = new SwarmSystem((int)obj);

ss.Initialize();

SwarmResult sr = ss.DoPSO();

return sr;

}, i);

}

List<SwarmResult> RList = new List<SwarmResult>();

Task tskFinal = Task.Factory.ContinueWhenAll(TaskArr,(tsks) =>

{

Console.WriteLine(tsks.Length.ToString() + " tasks");

for (int i = 0; i < tsks.Length; i++)

RList.Add(tsks[i].Result);

});

tskFinal.Wait();

RList.Sort();

//List<SwarmResult> Res = new List<SwarmResult>();

//Res = RList.Distinct(new SwarmResult());

var Res= RList.Distinct(new SwarmResult()).ToList();

dg1.DataSource = RList;

dg1.Refresh();

lblResult1.Text = Res[0].ToString();

lblResult2.Text = Res[1].ToString();

lblResult3.Text = Res[2].ToString();

lblResult4.Text = Res[3].ToString();

}

}

}

**Conclusion:**

After completing this assignment I was able to understand better the Task Parallel Library. I was able to see the difference between a sequential and parallel operations. Also, I got to learn that c# has the unsafe coding which is similar as using pointer in C++.