CS474 Spring 2016 Lab #1 Due 4/12, 2016

What to do

Write a parallel program by following the instructions:

1. Install MS VS 2013/2015 if necessary
2. Start Microsoft Visual Studio 20XX and start a console application. Make sure you are using C# as the programming language
3. Declare a constant named SIZE and give it a value of 20
4. Declare an array of integers of SIZE number of elements.
5. Fill the array sequentially with random numbers.
6. Find the largest random number sequentially, print out the result, and the duration
7. Find the largest random number in parallel using a data parallel algorithm, and the duration
8. Print out the results of steps 5, 6, and 7.
9. Change the value for SIZE to 1,000, 1,000,000, 2,000,000, and 20,000,000 and record the time.
10. For 20,000,000, try the chunk seize of 1, 10, 100, 500, 1000, and 20,000,000/CC, where CC is you core count. There is an environment variable that gives you the core count.
11. Run the 20,000,000 using one of the instruction stations unless you have an i7 computer.

What to turn in

1. Create a file with your code and screen dup of outputs using Word because I may have to run your code.
2. Add a one-page report that analyzes the results with the calculation of maximum speedups for all problem sizes.
3. Drop your file to the Moodle folder.

What are the things I will check?

1.	Documentation of code	15%
2.	Logic structure of your code,	20%
3.	Correctness,	40%
4.	Efficiency, and	15%
5.	Analysis	10%

The estimated time is 2 ~ 5 hours after you have installed Visual 201X.

A Sample program

```
using System;
using System.Collections.Generic;
using System.Diagnostics;  // for the stop watch
using System.Threading.Tasks;    // for parallel for

namespace ParallelFor2
{
    class Program
    {
        #region Sequential_Loop
        static void MultiplyMatricesSequential(double[,] matA, double[,] matB,
                                               double[,] result)
        {
            int matACols = matA.GetLength(1);
            int matBCols = matB.GetLength(1);
            int matARows = matA.GetLength(0);

            for (int i = 0; i < matARows; i++)
            {
                for (int j = 0; j < matBCols; j++)
                {
                    for (int k = 0; k < matACols; k++)
                    {
                        result[i, j] += matA[i, k] * matB[k, j];
                    }
```

```
                }
            }
        }
        #endregion

        #region Parallel_Loop
        // no control of the task size
        static void MultiplyMatricesParallel(double[,] matA, double[,] matB, double[,]
result)
        {
            int matACols = matA.GetLength(1);
            int matBCols = matB.GetLength(1);
            int matARows = matA.GetLength(0);

            // A basic matrix multiplication.
            // Parallelize the outer loop to partition the source array by rows.
            Parallel.For(0, matARows, i =>
            {
                for (int j = 0; j < matBCols; j++)
                {
                    // Use a temporary to improve parallel performance.
                    double temp = 0;
                    for (int k = 0; k < matACols; k++)
                    {
                        temp += matA[i, k] * matB[k, j];
                    }
                    result[i, j] = temp;
                }

            }); // Parallel.For

        }

        // task size is controlled by the value of chunk, the high value the larger
the tasks
        static void MultiplyMatricesParallelChunk(double[,] matA, double[,] matB,
double[,] result, int chunk)
        {
            int matACols = matA.GetLength(1);
            int matBCols = matB.GetLength(1);
            int matARows = matA.GetLength(0);

            // A basic matrix multiplication.
            // Parallelize the outer loop to partition the source array by chunk of
rows.
            Parallel.For(0, matARows / chunk, ii =>
            {
                int nSIZE = (ii + 1) * chunk;
                for (int i = ii * chunk; i < nSIZE; i++)
                {
                    for (int j = 0; j < matBCols; j++)
                    {
                        // Use a temporary to improve parallel performance.
                        double temp = 0;
                        for (int k = 0; k < matACols; k++)
                        {
                            temp += matA[i, k] * matB[k, j];
                        }
                        result[i, j] = temp;
                    }
                }
            }); // Parallel.For
```

```
        }


        #endregion


        #region Main
        static void Main(string[] args)
        {
            // Set up matrices. Use small values to better view
            // result matrix. Increase the counts to see greater
            // speedup in the parallel loop vs. the sequential loop.
            int colCount = 1024;
            int rowCount = colCount;
            int colCount2 = colCount;
            double[,] m1 = InitializeMatrix(rowCount, colCount);
            double[,] m2 = InitializeMatrix(colCount, colCount2);
            double[,] result = new double[rowCount, colCount2];

            // First do the sequential version.
            Console.WriteLine("Executing sequential loop...");
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            MultiplyMatricesSequential(m1, m2, result);
            stopwatch.Stop();
            Console.WriteLine("Sequential loop time in milliseconds: {0}",
stopwatch.ElapsedMilliseconds);

            // For the skeptics.
            PrintCheckSum(rowCount, colCount2, result);

            // Reset timer and results matrix.
            stopwatch.Reset();
            result = new double[rowCount, colCount2];

            // Do the parallel loop.
            Console.WriteLine("Executing parallel loop...");
            stopwatch.Start();
            MultiplyMatricesParallel(m1, m2, result);
            stopwatch.Stop();
            Console.WriteLine("Parallel loop time in milliseconds: {0}",
stopwatch.ElapsedMilliseconds);
            PrintCheckSum(rowCount, colCount2, result);
            result = new double[rowCount, colCount2];

            for (int chunk = 2; chunk <= 512; chunk *= 2)
            {
                // Do the parallel loop with chunk
                Console.WriteLine("Executing parallel loop chunk == {0}...", chunk);
                stopwatch.Reset();
                stopwatch.Start();
                MultiplyMatricesParallelChunk(m1, m2, result, chunk);
                stopwatch.Stop();
                Console.WriteLine("Parallel loop time in milliseconds: {0}",
stopwatch.ElapsedMilliseconds);

                PrintCheckSum(rowCount, colCount2, result);
            }

             // Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit.");
            Console.ReadKey();
```

```
        }


        #endregion

        #region Helper_Methods

        static double[,] InitializeMatrix(int rows, int cols)
        {
            double[,] matrix = new double[rows, cols];

            Random r = new Random();
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    matrix[i, j] = r.Next(100);
                }
            }
            return matrix;
        }

        private static void PrintCheckSum(int rowCount, int colCount, double[,]
matrix)
        {
            double dSum = 0;
            Console.WindowWidth = 150;
            Console.WriteLine();
            for (int x = 0; x < rowCount; x++)
            {
                for (int y = 0; y < colCount; y++)
                {
                    dSum += (long)matrix[x, y];
                }
            }
            Console.WriteLine("{0:#.##} ", dSum);
        }
        #endregion
    }

}
```