OpenMP Programming Assignment

Organization: CS286, Computer Science, SIUE
Author: Mark McKenney

Overview:

Write a multithreaded C++ program using open MP threads. I will provide a text file containing numbers that must be read into a two dimensional array. The first line of the file will have 2 integers, the number of rows in the array followed by the by the number of columns in the array. The rest of the file will contain integers that the array must be initialized to. Your program must also utilize a random number generator. I will provide source code.

Notes on Compiling:

This program uses the C++ high resolution timer. It was introduced in the C++ 11 standard, so you may need to explicitly tell the compiler to use the C++ 11 standard. I have included the appropriate option in the example runs below.

If you do not wish to compete in the fast time competition, you do not need to include the time information in your output.

Description:

Your program will indicate the cell address (the row and column) of the cell with the highest neighborhood average. If there is a tie, your program must report only a single cell that ties the maximum value. I will post a solution consisting of all cells that tie the maximum value so you can be sure your solution is correct. The neighborhood of a cell is all cells that immediately border the cell, including the cell itself. For each cell, you must compute the average of numbers in the neighborhood. For example, in the following array:
unsigned int M[10000][10000];
the neighborhood of cell M[2][8] consists of the following cells:

```
M[1][7] M[1][8] M[1][9]
M[2][7] M[2][8] M[2][9]
M[3][7] M[3][8] M[3][9]
```

Because the array will be large, you will need to use dynamic memory allocation (keyword new) on the heap.

Be careful not to go out of bounds on the array when computing neighborhoods

Your program should take a command line argument indicating the number of threads that will be used.

The group that has the fastest program when run with the number of threads equal to the number of available cores on a chosen computer will win a prize (of non-monetary value)! So don't give away your speedy secrets.

The openMP wiki has a nice intro to using openMP. http://en.wikipedia.org/wiki/OpenMP

Note that you must have a GCC compiler version 4.3 or later to use openMP. The departrment's home.cs.siue.edu server has a sufficient GCC version. You can download the newest version of GCC for mac and linux. You may have to install cygwin to do this with windows. Microsoft's compiler has it too.

Sample input files are attached

Your program will be graded on home.cs.siue.edu, so make sure it compiles and runs correctly there. Your group must make exactly 1 submission, and your group member names should be in a comment in the first line of your file. Also, you MUST submit a MAKEFILE. A sample makefile is provided, you can use this one, or edit it for your needs. Your submission should be zipped and submitted as single file.

What to Turn In:

You must turn in your source code, a Makefile that will compile your source code on home.cs.siue.edu, and a README file that contains the names of your group members, and any additional information about your implementation that you think I may need to consider while grading.

Turn in a single zip or tar.gz file.

The README file should be a PLAIN TEXT FILE! The README must contain the group member names.

If you do not wish to try for the fastest time, then you do not need to include time information in your output.

Do not include any other information in your output other than what is shown in the example runs! For example, if you print out the entire array in your submission, you will receive a grade of 0 becuase we do not want to spend time parsing through a ton of unneccessary output.

Example Input and Expected Output:

Here are some sample runs of the program so you can see what the output looks like. Your output should look EXACTLY like mine, except that you will only show a single cell address with the largest average. The input file small3.txt contains two cells tying for the largest average

The following shows the expected output with the timing output. You output should match these. These runs were made on an previous instance of the home server, the new home server is substantially faster, so you should be able to beat them.

Output that does match the format shown below will recieve a grade of 0.

```
vm-02$ cat Makefile



all:
    g++ -fopenmp -ggdb -std=c++11 matAverager.cpp -o matavg

vm-02$ make
g++ -fopenmp -ggdb -std=c++11 matAverager.cpp -o matavg
vm-02$ ./matavg
 usage: exe [input data file] [num of threads to use]
 or usage: exe rand [num of threads to use] [num rows] [num cols] [seed value]
vm-02$ ./matavg small3.txt 10
largest average: 7.66667
found at cells: (0,1) (0,2)
elapsed time: 0.033653
vm-02$ ./matavg rand 10 5 5 0
largest average: 6780.75
found at cells: (4,4)
elapsed time: 0.0271368
vm-02$ ./matavg rand 10 100 100 0
largest average: 8296.33
found at cells: (99,2)
elapsed time: 0.0396051
vm-02$ ./matavg rand 10 1000 1000 0
largest average: 9042.44
found at cells: (966,225)
elapsed time: 0.08917
vm-02$ ./matavg rand 10 1000 2000 0
largest average: 9339.83
found at cells: (999,1504)
elapsed time: 0.080462
vm-02$ ./matavg rand 1 1000 2000 0
largest average: 9339.83
found at cells: (999,1504)
elapsed time: 0.162964
vm-02$ ./matavg rand 4 1000 2000 3
largest average: 9370.22
found at cells: (874,617)
elapsed time: 0.077152
vm-02$ ./matavg rand 1 1000 2000 3
largest average: 9370.22
found at cells: (874,617)
elapsed time: 0.155171
vm-02$ ./matavg rand 10 1000 2000 3
largest average: 9370.22
found at cells: (874,617)
elapsed time: 0.0878711
vm-02$ ./matavg rand 10 10000 2000 0
largest average: 9240.89
```

```
found at cells: (8524,739)
elapsed time: 0.336246
vm-02$ ./matavg rand 1 10000 2000 0
largest average: 9240.89
found at cells: (8524,739)
elapsed time: 2.0535
vm-02$ ./matavg rand 10 10000 20000 0
largest average: 9461.78
found at cells: (618,2726)
elapsed time: 2.48621
vm-02$
```

Additional Files for this Project:

```
#include <thread>
#include <chrono>
#include <fstream>
#include <iostream>
#include <omp.h>
#include <sstream>
#include <string>

using namespace std;
// a class to get more accurate time

class stopwatch{

private:
        std::chrono::high_resolution_clock::time_point t1;
        std::chrono::high_resolution_clock::time_point t2;
        bool timing;

public:
        stopwatch( ): timing( false ) {
                t1 = std::chrono::high_resolution_clock::time_point::min();
                t2 = std::chrono::high_resolution_clock::time_point::min();
        }

        void start( ) {
                if( !timing ) {
                        timing = true;
                        t1 = std::chrono::high_resolution_clock::now();
                }
        }

        void stop( ) {
                if( timing ) {
                        t2 = std::chrono::high_resolution_clock::now();
                        timing = false;
                }
        }

        void reset( ) {
                t1 = std::chrono::high_resolution_clock::time_point::min();
                t2 = std::chrono::high_resolution_clock::time_point::min();;
                timing = false;
        }

        // will return the elapsed time in seconds as a double
        double getTime( ) {
                std::chrono::duration<double> elapsed =
std::chrono::duration_cast<std::chrono::duration<double>>(t2-t1);
                return elapsed.count();
        }
};



// function takes an array pointer, and the number of rows and cols in the array, and
// allocates and intializes the two dimensional array to a bunch of random numbers

void makeRandArray( unsigned int **& data, unsigned int rows, unsigned int cols, unsigned int
seed )
{
        // allocate the array
        data = new unsigned int*[ rows ];
        for( unsigned int i = 0; i < rows; i++ )
        {
                data[i] = new unsigned int[ cols ];
        }

        // seed the number generator
        // you should change the seed to get different values
        srand( seed );

        // populate the array

        for( unsigned int i = 0; i < rows; i++ )
                for( unsigned int j = 0; j < cols; j++ )
                {
                        data[i][j] = rand() % 10000 + 1; // number between 1 and 10000
                }

}

void getDataFromFile( unsigned int **& data, char fileName[], unsigned int &rows, unsigned
int &cols )
{
        ifstream in;
        in.open( fileName );
        if( !in )
        {
                cerr << "error opening file: " << fileName << endl;
                exit( -1 );
        }

        in >> rows >> cols;
        data = new unsigned int*[ rows ];
        for( unsigned int i = 0; i < rows; i++ )
        {
                data[i] = new unsigned int[ cols ];
        }

        // now read in the data

        for( unsigned int i = 0; i < rows; i++ )
                for( unsigned int j = 0; j < cols; j++ )
                {
                        in >> data[i][j];
                }

}


int main( int argc, char* argv[] )
{
        if( argc < 3 )
        {
                cerr<<"Usage: " << argv[0] << " [input data file] [num of threads to use] "
<< endl;

                cerr<<"or" << endl << "Usage: "<< argv[0] << " rand [num of threads to use]
[num rows] [num cols] [seed value]" << endl;
                exit( 0 );
        }

        // read in the file
        unsigned int rows, cols, seed;
        unsigned int numThreads;
        unsigned int ** data;
        // convert numThreads to int
        {
                stringstream ss1;
                ss1 << argv[2];
                ss1 >> numThreads;
        }

        string fName( argv[1] );
        if( fName == "rand" )
        {
                {
                        stringstream ss1;
                        ss1 << argv[3];
                        ss1 >> rows;
                }
                {
                        stringstream ss1;
                        ss1 << argv[4];
                        ss1 >> cols;
                }
                {
                        stringstream ss1;
                        ss1 << argv[5];
                        ss1 >> seed;
                }
                makeRandArray( data, rows, cols, seed );
        }
        else
        {
                getDataFromFile( data,  argv[1], rows, cols );
        }

/*      //UNCOMMENT if you want to print the data array
 *      cerr << "data: " << endl;
 *      for( unsigned int i = 0; i < rows; i++ )
 *      {
 *              for( unsigned int j = 0; j < cols; j++ )
 *              {
 *                      cerr << "i,j,data " << i << ", " << j << ", ";
 *                      cerr << data[i][j] << " ";
 *              }
 *              cerr << endl;
 *      }
 *      cerr<< endl;
*/
        // tell omp how many threads to use
        omp_set_num_threads( numThreads );


        stopwatch S1;
        S1.start();

        //////////////////////////////////////////////////////////////////////
        ///////////////////////  YOUR CODE HERE       ///////////////////////
        ///////////////         Make it parallel!        ////////////////////
        //////////////////////////////////////////////////////////////////////



        S1.stop();

        // print out the max value here
        cerr << "elapsed time: " << S1.getTime( ) << endl;
}
```

```
all:
	g++ -std=c++11 -fopenmp -ggdb  matAverager.cpp -o matavg
```

small.txt **Page 1 of 1**

```
4 4
4 9 7 4
8 9 5 9
4 2 5 5
9 6 8 2
```

small2.txt **Page 1 of 1**

```
10 10
42876 80635 79090 55398 88954 60207 42109 78347 24948 83861
84807 83387 75906 77075 81374 78301 6499 46947 75625 41873
83385 40154 24212 89395 39855 91713 45647 21516 33935 87363
13206 28824 98607 52530 47248 2899 27395 6 70589 4602
65547 69509 56011 41134 87304 40822 80754 88828 55322 46952
24351 74639 4755 74920 52453 91664 18970 94612 71803 28076
93594 29326 67589 31367 55106 75102 26878 97867 30321 25078
21074 98872 65557 67195 17754 83799 67330 56550 94735 61454
93786 39378 73680 20705 85303 83209 78279 63671 54949 34690
49007 50233 64178 84813 53042 7961 7972 20015 28102 86594
```

small3.txt **Page 1 of 1**

```
4 4
4 9 7 4
8 9 9 8
4 2 2 4
9 6 6 9
```

numberGen.py **Page 1 of 1**

```
import sys
import random

if len( sys.argv ) >= 4:
        fileName = sys.argv[1]
        rows = int( sys.argv[2] )
        cols = int( sys.argv[3] )
else:
        print 'usage: exe [output filename] [num of rows] [num of cols]'
        exit()


file = open( fileName, 'w' )

random.seed( )

file.write( str( rows ) + ' ' )
file.write( str( cols ) + "\n" )

for i in range( 0,rows ):
        for j in range( 0, cols ):
                file.write( str( random.randrange(2,100000) ) )
                file.write( ' ' )
        file.write( "\n" )
```

timeDiff.bash **Page 1 of 1**

```
#!/bin/bash
START=$(date +%s)
# do something

# start your script work here
$1 $2 $3 $4 $5 $6
# your logic ends here

END=$(date +%s)
DIFF=$(( $END - $START ))
echo "It took $DIFF seconds"
```