CSCI 370/CIS 570 Homework 2

Threads Homework: Calculating Average Temperature using Parallel Programming

Problem Definition:

A small grid of plates looks as follows:

	30	30	30	
15				72
15				72
15				72
	75	75	75	

The plates with numbers indicate a heating unit ensures that the temperature of that plate is set to the indicated value. Plates without numbers will have a temperature equal to the average of the temperature of their neighboring plates. Thus, the plate in the top right corner will have an average temperature of (72+30)/2, and the plate in the bottom left corner will have an average temperature of (15+75)/2. The plates in the middle must also be calculated as an average of 4 sides (and recalculated). The problem is to solve a much larger square grid in a short time. We will implement this with a single thread and multiple threads, to determine the speedup of multiple threads on a multiprocessor configuration.

The goal of this project is to process a very large matrix in the shortest time possible. The grid may always be square of a hard-coded size. To achieve such efficiency, consider how to minimize the use of memory with a large matrix, and how your code can be sped up using multiple threads and minimal OS calls.

To Solve:

To calculate the average value for all plates will require multiple iterations of calculating the average of each plate. In other words, the temperature for any plate is equal to the average of its 4 sides. This is an iterative process since temperatures will change with each iteration.

To speed up execution, the grid can be divided into 4 parts and a thread assigned to each part. Each thread should process only its section in an iterative way.

In this program you continue to recalculate the grid temperature until the error between two iterations is small. The error for any plate is calculated as the absolute value of |the newly calculated average minus the previous average|. The error for the grid is the total error for all plates, for one iteration of averaging all plates. We will consider that we are done when the total grid error is less than 5.0.

At the end of the program, print the following statistics: the total grid error, the grid average temperature, and the number of iterations run per thread. The grid's average temperature is the average temperature of all plates in the grid, and should be close to the average of (32+15+72+75/4).

To Program:

You may want to implement the program in four stages:

1) Program the matrix solution with a single thread.
2) Implement a 4-thread solution (without barrier).
3) Implement a barrier for a multi-threaded synchronized solution.
4) Implement one other algorithm that may speed up this implementation.

You can write this in Java or C# - or another language with prior approval from me.

Step 1: Single Thread solution: Code the description of the solution of the matrix, as described above. You will program this as a normal, single-threaded program.

You will eventually want to compare accuracy and execution times. For accuracy, calculate an average, in addition to the error, for each solution. The average is the sum of all plate average temperatures divided by the number of plates, while the error is the sum of differences between the last two iterations. This is most easily calculated by returning the total error and average value for each iteration. It can be returned as return values or saved (per thread) within the matrix object.

Your output should print:

Type: Single thread; Size=NNNN; Average Grid Value=NN.N; Total Error=NN.NN

To accurately record the execution times, you may want to create a Unix command file which prints the time, executes the program, then prints the time again (without any input prompts):

```
date
java YourProgam
date
```

You then execute this command file at a Unix prompt:

```
% ./CommandFile
```

Alternatively you can embed a time print function in the beginning and end of your code execution.

Step 2: Multiple Threads: With multiple threads, the grid must be in one shared object that all threads have access to. The size of the grid should be flexibly designed as a parameter at the top of the program: it will eventually become very large. You will want to run this with one thread, or four threads running simultaneously sharing the work. When you use multiple threads, divide the work so that each thread solves one quarter of the grid.

With a multithreaded program, you should be concerned about shared versus non-shared memory. A class's instance private variables can be easily shared. Local variables stored on a stack (such as variables defined in a method) are 'reentrant', since each thread has their own stack and thus local memory. Multiple threads will need to share the single matrix and their iterations' final error, while errors being accumulated should be local (not-shared).

In order to ensure an efficient program, you are advised to:

1) Make each matrix element a single element: a double value indicating the temperature. Errors should not be stored in the matrix, since it will easily double (or quadruple) the size of the matrix, slowing down your program.

2) Do not destroy and recreate threads per iteration. Implement a multi-iteration mechanism within each thread's run() routine. Have your iterative loop within run(), not outside of it.

3) Java has a synchronized mechanism to ensure that only one thread can operate at any point in time. The more time a thread spends in a synchronized method, the less time multiple threads can run simultaneously – and your multithreaded solution may become slower than your single-threaded solution. So avoid synchronized methods unless data cannot be safely shared between threads.

At the time that your error is sufficiently low, use the interrupt() capability at the group level:

```
Thread.currentThread ().getThreadGroup ().interrupt ();
```

Step 3: Barriers: In step three, you will work with a barrier (or Synch Bar) to synchronize threads. The answer you calculated in the previous solution may not have been precise in spite of multiple iterations. It is possible that one thread ran more often than other threads, making one quarter of the grid more accurate than other quarters. With a Synch Bar, we ensure that all threads do one iteration and stop before starting their next iteration: every thread gets one iteration and cannot start the next iteration until each other thread has completed its iteration. An example of a java barrier is shown in the voting lab we did in class.

Step 4: Your Own Test: In the last aspect of this assignment you will enhance your program to compare the efficiency of different models and algorithms. Please select another test to compare performance.

- Does a computer with additional processors speed things up? (Must compare apples with apples – just changing the processor for a single test does not indicate anything…)
- Does this program run faster on Unix versus Windows?
- Can you further improve the algorithm to speed up the multithreaded version?
- What is the impact of making the grid a two-dimensional array of doubles versus class objects?

- Does the size or efficiency of the matrix make a difference in your answer? Where does the algorithm really begin to bog down?

Stage 2: Analyzing and Preparing a Report

Analyzing Your Data

There are many questions that could arise from this experiment. Here are some mandatory questions you shall answer:

- Compare Accuracy: Which answer is more correct, the single thread, the synchronized multi-thread, or the unsynchronized multithread or your fourth method? Why? How much more correct is one solution over the other? Show graphs of results for all methods.

- Compare Time: Will the Synch Bar produce a better answer faster than the other multithreaded solution? Are the multithreaded solutions faster than the single threaded solution? Show times to prove your point.

- Which would you recommend from a balanced speed and accuracy perspective?

To prove your results, prepare graph(s) showing the accuracy and performance of the different techniques.

You will find it useful to observe the processor utilization of each processor for your official tests. In MS Windows, do a Control-Alt-Delete simultaneously to obtain access to system functions, then select Task Manager, and the Performance and Networking tabs. For Linux systems, use Applications-> System Tools-> Monitor System to learn about processor and network utilization. Are both processors used equally?

Writing a Report:

Finally you write a lab report to compare different scenarios:

1) Single-threaded
2) Multi-threaded
3) Multi-threaded with Synch Bar (Barrier)
4) Another comparison of your choice (further improved algorithm, different computer or OS type, many different sizes of grids, etc.)

Report Format:

The report will use the same structure as some of the in-class labs that we performed. The lab report structure includes:

- Introduction: What is the purpose of this lab?
- Method: What specific machines, operating systems, grid sizes, and algorithms did you use in testing? Describe what you did so that we can fully understand the experiment.

- Results: Provide a table, and if possible, graph of all numeric results to contrast the different methods you tested. Show differences in time and accuracy. (Graphs are highly encouraged and will be rewarded with higher grades.)
- Analysis: Why do you think you got the results that you did? Did the multiprocessor perform as expected, and if not, why? Why was one method more accurate or speedier than other methods? Can you prove it or convince me of your arguments?
- Conclusion: What did you learn from this experiment, concerning parallel programming? What would you recommend other people do with similar problems?

The report will be graded on 1) proper English and report format, 2) result table and graph showing results for all 4 algorithms, and 3) the quality of the Analysis section in explaining the differences observed between the 4 algorithms. I am expecting that you can write like a college graduate – please use MS Word grammar-check and spell-check to achieve this. For the proper report format, ensure that your report is clear, descriptive, and follows the outline well. For the Analysis section, show me that you have thought about and understand the reasons your program performed as it did. Convince me of your reasoning. You will also be graded separately for your program.

Turning in the Homework

Please submit your code file (.java) and report (.doc, .docx, or .pdf) via Canvas.