Programming Assignment #2: Measuring Parallel Performance

Due: 03/05/2012 (11:59 pm)

In this assignment, you will write your first parallel programs using threads in Java. Follow the guidelines for creating concrete Runnable objects in Appendix C of Patterns for Parallel Programming. There will be few choices in the design of an appropriate parallel algorithm. The point of this exercise is to explore scaleup and speedup on multi-core processors.

The majority of your grade (80%) will be derived from your answers to the questions. These should be concise, thoughtful, and specific. Please turn in a PDF format document that answers the questions put forth. Please include your name, the date, and the assignment (in this case Programming Assignment #2) on your writeup. Items in **bold** represent additional deliverables. You will turn in source code as well.

Part 0: What You Need to Know

Amazon EC2

You will run your programs on Amazon EC2 using a High-CPU Extra Large Instance. These machines have 20 EC2 Compute Units on 8 virtual cores and will allow you to explore scaleup and speedup to that scale in multicore systems.

You should develop your code on your personal computer and only use EC2 for scalability testing.

To get started on AWS: You will need to sign for Amazon EC2. 1) Log into the AWS management console. (http://aws.amazon.com/console/ [http://aws.amazon.com/console/]) 2) Launch a new instance (Linux/Windows) and connect to your instance.The following link walks you through the process: http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/ [http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/] (Note:You should ideally start EBS backed instances for persistant storage of data. 3) You can then run your program on your instance as you would your local machine.

Submissions

You will submit a tarball that contains your source code and a PDF document that answers the specified questions. Your tar file should be named lastname-hw1.tar.gz. When extracting the tar file (using tar xfz), a single directory should be created called lastname-hw1. Your code should be in that directory. You also must include a README file indicating how to compile and test your code. Your design document must include details about your implementation including your design, any quirks or limitations as well as anything that is noteworthy. It must be in PDF format; do not turn in a MS Word document. If your code has limitations and you note them in the document, e.g. if it doesn't compile. Ideally, you will provide a make file using appropriate targets (all, clean, etc.). Finally, do not include object code or any other intermediate code in your submission.

In short, the contents of your tar file should have the following form:

- lastname/
  - README
  - Makefile
  - writeup.pdf

- source code…

The assignment should be submitted on blackboard.

Part 1: Parallel Coin Flipping

Write a java program that flips coins in parallel. The program can be invoked from the shell (command line) and should take two arguments: the number of threads to start and the number of coin flips. The program should output the number of heads (or tails), the number of total coin flips, and the elapsed (wall clock) time to run the program. For example, my implementation does the following:

```
<randal@rio:/Users/randal/class/420/P1.MergeSort> java CoinFlip
Usage: CoinFlip #threads #iterations
```

```
<randal@rio:/Users/randal/class/420/P1.MergeSort> java CoinFlip 1 10000000
5000728 heads in 10000000 coin tosses.
Elapsed time: 525ms
```

```
<randal@rio:/Users/randal/class/420/P1.MergeSort> java CoinFlip 2 10000000
5001027 heads in 10000000 coin tosses.
Elapsed time: 267ms
```

For this experiment you will need to use some java functions. Specifically, System.currentTimeMillis() and Random.generate() and Random.nextInt(). The SealedDES.java program I give you in part 2 has examples. You will also need join() all your child threads from the parent thread. This is the most basic form of synchronization.

Analysis

For reasonable parameters, measure the scale-up and speed-up of this program from 1 thread to twice as many threads as cores. Your experiment should take seconds (so that it's a significant number of trials) but not hours. For example, for 8 cores, you might run the program for 1,2,3,…,16 threads at 1000000000 coin tosses for speedup. (Don't worry about +/- rounding errors for number of flips per thread.) And, run the program for 1,2,3,…, 16 threads at 1000000000,2000000000, …,16000000000 for scaleup.

1. Scaleup and speedup
    I. Produce charts that show the scaleup and speedup of your program.
    II. Algorithm (true) speedup/scaleup measures the scaling performance of the algorithm as a function of processing elements. In this case, from 1..8. Characterize the algorithmic speedup/scaleup. If it is sub-linear, describe the potential sources of loss.
    III. Why does the speedup not continue to increase past the number of cores? Does it degrade? Why?

1. Design and run an experiment that measures the startup costs of this code.
    I. Describe your experiment. Why does it measure startup?
    II. Estimate startup cost. Justify your answer.
    III. Assuming that the startup costs are the serial portion of the code and the remaining time is the parallel portion of the code,what speedup would you expect to realize on 100 threads? 500 threads? 1000 threads? (Use Amdahl's law.)

Part 2: Brute Force a DES Key

In this step, you will implement a parallel/threaded Java program that attempts to brute force a DES key. Because we do not have reasonable computing power to attack 56 bit keys, we are going to attack smaller keys. I also wanted to keep it simple. So, rather than giving you an already encrypted message, we are going to have your program generate the ciphertext and then brute force the decryption.

The attack is a combination brute force + known plaintext attack. This is quite common. If you the attacker know part of the message, you can attempt to decrypt with all possible keys and check to see if the decrypted output matches the know plaintext.

The assignment starts with my serial implementation of a brute force attack. SealedDES.java [http://hssl.cs.jhu.edu/~randal/420/src/SealedDES.java]. Parallelize this program so that it will search for a specified number of keys using a specified number of threads. My implementation does the following:

```
<randal@rio:/Users/randal/class/420/P1.MergeSort> java BruteForceDES
Usage: BruteForceDES #threads key_size_in_bits
```

```
<randal@rio:/Users/randal/class/420/P1.MergeSort> java BruteForceDES 2 20
Generated secret key 798711
Thread 0 Searched key number 0 at 0 milliseconds.
Thread 1 Searched key number 600000 at 1803 milliseconds.
Thread 0 Searched key number 100000 at 2204 milliseconds.
Thread 1 Searched key number 700000 at 3384 milliseconds.
Thread 0 Searched key number 200000 at 3783 milliseconds.
Thread 1 found decrypt key 798711 producing message: Johns Hopkins afraid of the big bad wolf?
Thread 1 Searched key number 800000 at 4891 milliseconds.
Thread 0 Searched key number 300000 at 5292 milliseconds.
Thread 1 Searched key number 900000 at 6426 milliseconds.
Thread 0 Searched key number 400000 at 6801 milliseconds.
Thread 1 Searched key number 1000000 at 7925 milliseconds.
Thread 0 Searched key number 500000 at 8309 milliseconds.
Final elapsed time: 9045
```

You should make sure that your threads have no shared state, variables, or objects. For example, make sure to create a separate SealedDES object for decryption in each thread. Do not use a single encryption/decryption object shared among all threads.

Analysis for Part 2

1. For reasonable parameters and for however many cores you have on the system, measure the scaleup and speedup of this program on the cluster.
    I. Produce charts and interpret/describe the results. Is the speedup linear?
    II. Why do you think that your scaleup/speedup are less than linear? What are the causes for the loss of parallel efficiency?
    III. Extrapolating from your scaleup analysis, how long would it take to brute force a 56 bit DES key on a machine with 64 cores? Explain your answer.

randal/teach/cs420/mppjava.2012.txt · Last modified: 2012/03/03 07:21 by priya

Except where otherwise noted, content on this wiki is licensed under the following license:CC Attribution-Noncommercial-Share Alike 3.0 Unported [http://creativecommons.org/licenses/by-nc-sa/3.0/]