

Distributed May 23, 2005
Due June 2, 2005 in Lecture

Homework #4 – Parallel Computing Models, Costs, and A Large, Application Example

Homework Problems

Problem 4.1. (**Parallel Computing Models**) In class, you have heard about several different parallel computing paradigms which differ in how they communicate and express parallelism. Three very commonly used models are: Master-worker, Message-passing (or Process parallel), and Data/Object Parallel. For each of these, write a short definition (3 sentences) of the model in your own words. And then for each, give an example of an application which maps well into that model, but not the other two. Be sure to explain enough detail about the application so that you can support your arguments. Then, explain specifically why the application is suited to one and not the others.

Problem 4.2 (**Understanding Parallel computing costs**). In writing a number of programs for the lab exercises, you have doubtless noticed that cost of certain operations in parallel computing systems determines the “granularity of parallelism” that can be supported (the chunks of work that can be run in parallel and improve performance). In this problem, we will characterize the overhead of these operations, and then use them to determine the granularity of work required to achieve efficient parallel performance.

- a. Thread Creation Cost Write a program that creates and then destroys 1000 threads, and accurately measures the cost of the thread creation and thread destruction operation.
- b. Thread Suspend and Resume Cost Write a program that blocks (wait()) and then wakes (notify()) a thread 1000 times and accurately measures the cost of the thread suspension and resumption.
- c. Proactive Remote Object Invocation Write a program that creates objects on two nodes and measures the shortest time to do a round trip invocation with two numbers as arguments.
- d. Proactive Remote Object Invocation – Data Cost Write a program that creates objects on two nodes and measures the shortest time to do a round trip invocation with an array of 256K floating point numbers as arguments.
- e. Putting it all Together Write a simple program that runs on N nodes with 2N parallelism (2 threads/node) but starts with a single thread and uses a binary tree to spread activity to the other nodes as rapidly as possible. After reaching 2N-fold parallelism, your program will compute for K seconds, and then use a binary tree structure to detect completion of all 2N threads, and then exit.
- f. Run the program for varying numbers of nodes ($N = 2^X$ where $X=0, 1, \dots, 5$ i.e. you just do this for the powers of 2 nodes and $2*N$ processors since each node has a dual processor). For what values of K can you achieve reasonable parallel efficiency? (50% of the time spent on parallel work? 75%? 90%?)

Explain your results, using the measurements you have taken in the first five parts of this problem.

Problem 4.3. (**Building a Parallel Search Engine based on PageRank**). In this problem you will implement and optimize a PageRank computation which is the core of the producing relevant entries in response to a web search.

Step 1. (Familiarize yourself with the code) The basic code for this problem can be found in /home/cse160/hw4/. The files you need to look into are MakeSerial.java and PageRank.java. You have been provided with 16 data directories (csucsd, mit, yahoo, ucsb, intel, java, nba, ibm, berkeley, slashdot, rediff, hp, msft, stanford, uiuc, nyu) Each of these directories have several data files (*.data). The format of these files is straight forward – the first line gives the name of the URL and the remaining lines are outgoing links from the URL. The directories also have a file called filelist.txt that stores the number of files and filenames of the data files in that directory. The MakeSerial class takes the files in these directories and makes an array of Vectors out of them. The structure of each element in the array is a Vector that begins with the name of the URL and is followed by the outgoing link from that URL. **You don't need to use MakeSerial.** It is there just for your understanding and the serialized output can be found in a file called **myobject.ser**.

What you do need to understand is the class **PageRank**. This class reads the serialized object saved in the myobject.ser file in the directory and calculates the pagerank of the individual pages for a given number of iterations (given as a command line parameter). If you have any problems understanding the framework or getting it running contact Sagnik immediately.

The code provided in PageRank.java is supposed to provide basic guidance on how to solve the problem. It is not necessary that you use it. However, you CANNOT change the format of the data files or the serial object file.

- a. Run the provided application on the sample data set (the code provided works on a directory basis. So you have to modify it to accommodate links spanning across directories). Characterize how long it takes to run as a function of the number of nodes ($N=2^K$ where $K=0,\dots,5$), and the size of the data set (small = {csucsd, ucsb, intel, msft}, large = {all 16 directories}). Output the 50 most and least important links for each data set. The number of iterations you need to perform is 5000.
- b. Modify the application to improve its performance. Some ideas include adding thread parallelism, optimizing thread overhead, reducing inter-node communication, improving cache behavior, using optimized data structures etc.
- c. Run your improved application on the sample data sets. Characterize how long it takes to run as a function of the number of nodes (N), and the size of the data set (small, large). Break this down by the key phases of the application, and point out how you have improved performance, and what the key contributors were.

- d. PageRank can be run to terminate in a fixed number of iterations, or to meet a particular threshold of error. Thus far, you have been using a version of the PageRank computation which computes a fixed number of iterations. However, due to your improvements which speed up the computation, you could now increase the number of iterations. If you increase the number of iterations to bring the total runtime back to that of the original version, how much smaller is the error? (size of the last updates to the PageRank values)