Homework 4

MPCS 51100

due: Fri Dec. 9, 2016

Please package your assignment in a directory called hw4 with subdirectories p1, p2, ... p6. At the top level, under directory hw4, include a makefile with targets p1,p2, ..., p6 ,all, and clean. Include also a separate text document with short answers to the questions and timing results. where requested.

In problems 1-3, consider the shortest path problem for a directed, weighted graph $G(E, V)$. We learned that Dijkstra's algorithm finds the distance between any source node S and all other nodes in the graph in $O(|V|^2)$ operations. A similar problem, the all-source shortest path problem, finds the minimum distance between EACH vertex v in a graph and all other vertices.

1. Write a code that solves the all source shortest path problem by simplying applying Dijkstra $n = |V|$ times, i.e. once for each vertex in G. Note that this algorithm has complexity n^3. This code will be p1.c. Include a driver that tests the code on the two test graphs in the homework dropbox. Report the timings.

2. Next write a shared memory parallel version of all source Dijkstra using OpenMP. Your strategy will be straightforward – simply assign groups of different source vertices to each core. Assume $n > p$, where p is the number of processors (i.e. cores) on a multicore machine. What is the complexity of the parallel algorithm in terms of n and p? Compare the performance of the serial and parallel approach using the provided sample graph. Comment briefly on the observed timings. This code will be p2.c.

3. Finally, consider how you might use a shared memory parallel processor to accelerate Dijskstra *within* a single source computation? There are several possible approaches, and developing a reasonable approach yourself is part of the exercise. In this case, to guide your thinking you may assume that n is much larger than p – i.e there are many more vertices than cores. Consider ways to partition the graph among processors to minimize contention and load balance – ie to keep all cores busy but require minimum synchronization. Comment briefly on observed timings. This code will be p3.c.

4. Write an efficent function that takes a Markov matrix as input and calculates its *period* (as a reminder *aperiodic* is equivalent to a period of 1). Apply this function to the Markov graph provided in the homework directory. What is the period of this graph? This code will be named p4.c.

5. Write a function that takes a Markov matrix as input and calculates whether or not it is *irreducible* (as a reminder *irreducible* means strongly connected for a directed graph). Apply this function the the Markov graph provided in the homework directory. What is the result? This code will be named p5.c.

6. Write a code that takes a Markov matrix as input and uses randomization to execute realizations of Markov chain to calculate the stationary probabilities. Apply this function to the Markov graph provided in the homework directory. What are the resulting stationary probabilities? What do the answer to the previous two questions tell you about the values that you calculated? This code will be named p6.c.