Lab 3: Travelling Salesman

Due: March 26th, 2009

March 18, 2009

1 Introduction

In this lab you will be exploring a shared memory approach to programming distributed systems. You will use the OpenMP API to write a distributed solution to the travelling salesman problem (TSP). TSP is known to be NP-complete, but that means only that it cannot be solved efficiently. The problem is still solvable through the application of sufficiently large quantities of hardware. You will be able to run your implementation on systems provided by the Pittsburgh Supercomputing Center (PSC). Once you have a working application, you will need to run it on various numbers of processors and write a report on your results.

2 Important Dates

Lab due on March 26th, 2009

3 IMPORTANT NOTE

As the machines we are using for this assignment are managed by the PSC, they may be busy at times and your jobs may not always be run quickly. You can run on the head nodes (the ones you log into) for debugging, but when using larger numbers of nodes or gathering data for your report, you should allow time for the runs to complete (it may take up to a day for the runs with many processors to be scheduled).

4 Lab Requirements

For this lab, you will need to implement a distributed solution to the travelling salesman problem (TSP) using OpenMP. In TSP, you have a list of cities and the distances between them. The goal of the problem is to find the shortest possible path along which the salesman visits each of the cities exactly once and then returns to his start city. For those of you in graph theory, the solution to the TSP is the shortest Hamiltonian cycle of the graph made up by the cities and the distances between them.

You should first write a serial solution, based on a brute force approach. Once the length of at least one cycle has been computed, you will want to prune any paths that are already as long as the best known cycle as they will not get shorter (you may assume non-negative edge lengths). Once you have a serial version, you will need to produce a parallel implementation using OpenMP.

4.1 Cities

A collection of example city graphs have been provided along with this lab. Your implementations should read the file format used for these cities. Each file first contains the number of cities. The rest of the file consists of pairs of cities (numbered from 1 to N-1) and the distances between them.

5 OpenMP Tutorials and Resources

- CMU 15-418 OpenMP handout
  `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15418-s08/public/asst/asst1/openmp_tutorial.pdf`
- NCSA online course on OpenMP
  `http://foxtrot.ncsa.uiuc.edu:8900/public/OPENMP/`
- Lawrence Livermore National Lab's concise OpenMP tutorial
  `http://www.llnl.gov/computing/tutorials/openMP/`
- Official OpenMP website
  `http://openmp.org/`

6 PSC Machines

For this assignment you will be using the Pople machine at the PSC. The Pople machine is an SGI Altrix shared-memory system with 384 Itanium2 dual-core processors. Additional information about the machine is available at: `http://www.psc.edu/machines/sgi/altix/pople.php`.

6.1 Connecting to pople

In order to connect to pople you must ssh to `pople.psc.edu` and login with your PSC username and password. PSC usernames and initial passwords will be handed out in class. When you receive them, you should change your initial password as it is set to expire soon after you receive it. If your password expires, you will be unable to login and we will need to have it reset.

6.2 Compiling on pople

There are both GNU and Intel C and C++ compilers available for use on pople, each of which require particular flags for building OpenMP jobs. You may use whichever compiler you would like for this system, although it is our understanding that the Intel compiler leads to better performance. Information on the compilers and which flags to use for OpenMP programs is available at: `http://www.psc.edu/machines/sgi/altix/pople.php#compilers`.

6.3 Submitting jobs with PBS

When you connect to pople, you will be logged into a frontend system. This system is suitable for compiling your program, but is not a good place to run it. When you are ready to run a job, you will need to submit it to the run queue using the Portable Batch System (PBS). To use PBS, you will need to create a script for running your process (such as the example here: `http://www.psc.edu/machines/sgi/altix/pople.php#batch`) and submit it to PBS with `qsub` (as explained here: `http://www.psc.edu/machines/sgi/altix/pople.php#qsub`).

7 Technical Requirements

- Your project must function properly on the PSC Popel system
- Your code must compile on the PSC Popel system
- Your project must include a Makefile that builds your project

8 Lab Write-up

As part of this lab you will need to produce a write-up explaining your approach to distributing the program across many nodes. You should include all of the major issues you ran into, and explain all of your results.

8.1 Scalability

As the lab is about distributing computation across many nodes, we require that you measure the performance of your solution on 1, 2, 4, 8, 16, 32, 64, and 128 processors. You must include plots with the actual runtimes for these different numbers of processors as well as the speedups your solution was able to obtain. Be sure that your charts include runs of the single processor version of the program as a base line in addition to the multi-processor version run on a single processor.

For these experiments, select a city that takes 2-5minutes to run with your erial implementation.

9 Grading

20% Code quality, style

20% Lab Write-up

60% Code functionality, robustness, scalability