15-418: Assignment 4:
Wandering Salesman Problem in MPI

Due: Mon Mar 26, 11:59PM

1 Overview

This assignment deals with parallel programming using the message passing model. In assignment 3 you solved the Wandering Salesman Problem (WSP) using the shared memory programming model (in OpenMP). Now you will implement a second solution to the WSP using the message-passing programming model (you will write code that uses the Message Passing Interface: MPI). In addition to using MPI to construct a parallel solution to the WSP, this assignment aims to help you understand trade-offs involved when writing message passing programs and encourages you to compare message passing with shared memory progrmming.

You are allowed to work in groups of two people (one hand-in per group).

2 Environment Setup

Setup is the same as for assignment 3.

For this assignment you will be using "Blacklight", a 4096-core supercomputer hosted at the Pittsburgh Supercomputing Center (PSC). There are two ways to access Blacklight to develop, compile, and test your code. The first is through the XSEDE web portal. You will need to login into your XSEDE account at `http://portal.xsede.org`. Once logged in, navigate to "MY XSEDE", then "SSH Terminal" and select "Blacklight" in the dropdown menu to initiate an SSH session. The second is via ssh to `blacklight.psc.teragrid.org` (we expect that the latter option will be far more convenient). To login via ssh you may first need to change your Blacklight password by following the instructions found here: `http://www.psc.edu/machines/sgi/uv/blacklight.php#password`

To get started:

- Download the Assignment 4 starter code and documentation from the course directory: `/afs/cs.cmu.edu/academic/class/15418-s12/assignments/asst4.tgz`.
- For more details about the Blacklight, please see: `http://www.psc.edu/machines/sgi/uv/blacklight.php`
- For more information on using Blacklight, please see: `tutorials/blacklight_tutorial.pdf`

3 Assignment

What you need to do:

1. (35 points) Implement a parallel Branch-and-Bound program for the WSP, using MPI on Blacklight. The objective is to obtain the best speedup possible. Your program should accept the command-line

arguments `-p` and `-i` to specify the number of processors used and the input file respectively. It should also use the same input file format described in the previous assignment. (File loading and command-line parsing are already implemented for you in the starter code).

2. (15 points) Produce a brief report describing and analyzing your solution. Your report should include the following items:

   - A brief description of how your program works. Describe the general program flow and all significant data structures. How is the problem decomposed? How is the work assigned to threads? Where is the communication and synchronization?
   - Give your solution to the problem given in `input/distances`. This file contains a 17 city problem. (For debugging purposes, you may want to use some of the smaller input files included in the same directory.)
   - Execution time and speedup (both total and computation) for 1, 2, 4, 8, 16, 32, 64, 128 and 256 processors on Blacklight. See Section 4 for definitions of total and computation time.
   - Discuss the results you expected and explain the reasons for any non-ideal behavior you observe. In particular, if your program does not achieve perfect speedup, explain why. Is it due to work imbalance? Communication/synchronization overhead? Performing redundant (or unnecessary) work? Is it possible to achieve better than perfect speedup? Provide measurements to back up your explanations.
   - Compare the performance of your message-passing version of WSP on Blacklight with your shared-memory version of the application on the same machine. (Include graphs to illustrate the difference in performance.) Discuss any differences in performance with possible reasons for such a behavior.

4 Measuring Performance

While it may be helpful to compile with the `-g` flag when you are debugging your code, **please be sure to use the `-O3` flag to generate any programs that you will be timing!** There can be significant differences in performance between different levels of compiler optimization, and we are only interested in the speedup of optimized code. The provided Makefile generates two executables `wsp` and `wsp_debug` (corresponding to an optimized and debug build of the code).

To evaluate the performance of your parallel programs, measure the following times using the provided `gettime()` function which returns the current time in seconds:

1. *Initialization Time:* the time required to do all the sundry initialization, read the command line arguments, and create the separate processes. Start timing when the program starts, and end just before the main computation starts.
2. *Computation Time:* this is strictly the time to compute the answer. Start timing when the main computation starts (after all the processes have been created), and finish when the answer has been calculated.

Note that: *Total Time* = *Initialization Time* + *Computation Time*. *Speedup* is calculated as $\frac{T_1}{T_p}$, where T_1 is the time for one processor, and T_p is the time for P processors. *Computation Speedup* uses only Computation Time, and *Total Speedup* uses the Total Time.

5 Performance Analysis

The goal of this assignment is for you to think carefully about how real-world effects in a parallel machine can limit your speedup, and how you can improve your program to get better performance. If your performance is disappointing, then it is likely that you can restructure your code to make things better. We are especially interested in hearing about the thought process that went into designing your program, and how it evolved over time based on your experiments.

6 Using MPI

A small introduction and tutorial is being handed out with this assignment. This tutorial gives only a very rudimentary overview to all the things that MPI allows you to do. To learn more about MPI, please consult one of the following websites:

`http://www.citutor.org`
: This NCSA online course "Introduction to MPI" is free. Register online at the above URL, with any easy-to-remember login and password. Chapters 2-8 cover everything you will need for this assignment. You can log into your account any number of times, so the course material can also be used as an online reference.

`https://computing.llnl.gov/tutorials/mpi/`
: Another tutorial on MPI. (If you take the NCSA course, this one is optional.)

7 Hand-in

Hand-in directories have been created at:
`/afs/cs.cmu.edu/academic/class/15418-s12/handin/asst4/<ANDREWID>/`.

Copy your `asst4/` directory into your hand-in directory. Include the `asst4/` directory itself (don't copy only the contents) and don't submit a tarball. If working in a group, copy your `asst4/` directory in either group member's hand-in directory. The written answers should be in `asst4/writeup.pdf`. **Your code should compile and run on Blacklight without any modifications!** This means that we should be able to `make` and execute your programs without manual intervention.