15-418/618, Fall 2020
Assignment 4
WSP: Message Passing Edition

Assigned:	Fri. Oct 16th
Due:	Wed. Oct 28th, 11:59 pm
Last day to handin:	Sat. Oct 31st

1 Overview

Now that you have solved the Wandering Salesman Problem (WSP) using the shared memory programming model, you will now solve the same problem using the message-passing programming model. In particular, you will write your program using the Message Passing Interface (MPI).

As before, you are permitted to work in groups of 2 people to solve the problems for this assignment (hand in one assignment per group.) You will be submitting to both Autolab and Gradescope.

Before you begin, please take the time to review the course policy on academic integrity at:

http://www.cs.cmu.edu/418/academicintegrity.html

2 The Assignment

Refer back to the Assignment 3 handout for the problem specification. The aim of this assignment is to understand the trade-offs involved in the message-passing model, and compare them with what we have learned about the shared memory model for writing parallel programs. The objective, as always, is to obtain the best speedup possible.

```
git clone https://github.com/cmu15418-f20/asst4-f20.git
```

Again, there is no starter code, and you are responsible for creating the following:

- wsp-mpi.c – contains your MPI implementation for WSP
- Makefile – compiles wsp-mpi.c to wsp-mpi
- writeup.pdf – your report to the assignment

Your WSP MPI program should run as such:

```
mpirun -np [numProcesses] ./wsp-mpi [inputFile]
```

Where numProcesses is the number of MPI processes and inputFile is reference to a dist file containing city distances. Your program must output the optimal path as well as execution time, although you are allowed to output anything else you think may help you in producing your report.

Your report should include the following items:

1. A brief (roughly one or two pages) description of how your program works. Describe the general program flow and all significant data structures. Compare and contrast with the design you used for the shared-memory version.
2. The 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. The city locations corresponding the distance files are provided in the 'city' files.)
3. Execution time and speedup (both total and computation) for 1, 2, 4, 8, 16, 24, and 32 processors on both `pople` and `blacklight` on Latedays.
4. Discuss the results you expected and explain the reasons for any non-ideal behavior you observe. In particular, if you don't get perfect speedup, explain why.
5. Compare the performance of your message-passing version of WSP on `blacklight` with your shared-memory version of the application on the Latedays. (Include graphs to illustrate the difference in performance.) Discuss any differences in performance with possible reasons for such a behavior.

If the execution time for your program takes more than a few minutes, double-check your program and algorithm. Make sure your programs run on a uniprocessor before trying to run them in parallel. Also, debug your programs using smaller numbers of cities (perhaps `dist4`) and small numbers of processors before trying larger runs.

3 MPI

We are using the OpenMPI implementation of the MPI programming model. To get started with MPI, here are some websites you may find useful:

- https://mpitutorial.com/ contains a brief introduction to MPI.
- https://www.open-mpi.org/doc/v1.6/ contains the documentation for the MPI API.
- https://www.citutor.org/ contains online courses for MPI and various other topics. Specifically, the `Introduction to MPI` is relevant. Registration is free.

The handout also provides an example MPI program sqrt3 that approximates sqrt(3).

4 Hand-in

You will submit your code via Autolab and your report via Gradescope. The relevant websites are:

https://autolab.andrew.cmu.edu/courses/15418-f20/

https://www.gradescope.com/courses/156025

- **If you are working with a partner, form a group on Autolab.** Do this before submitting your assignment. One submission per group is sufficient.
- Make sure all of your code is compilable and runnable. We should be able to simply run make, then execute your programs on Latedays without manual intervention.
- Run the command `tar -czvf handin.tar wsp-mpi.c Makefile` to create a handin.tar for you to upload to Autolab.
- Upload your report as file `report.pdf` to Gradescope, one submission per team, and select the appropriate pages for each part of the assignment. After submitting, you will be able to add your teammate using the *add group members* button on the top right of your submission.

5 Running on Latedays

As with Assignment 3, you will again be using the `submitjob.py` script to run on the latedays cluster. **Note the following changes to the script:**

- The new `-p PROCS` argument specifies the number of MPI processes to spawn.
- Do **not** supply the number of processes to the `-a ARGS` argument, only the input file name and any other optional arguments your program accepts.

You would now use the script like so:

```
./submitjob.py -s test -p 16 -a "input/distances"
```

Good luck!