Project 2: Barrier Synchronization

Read this assignment description carefully before you begin. **Start early**, because you will be running performance experiments. You will need time to do the experiments and write a write-up after you finish coding, and most of the points for this assignment will come from the experiments and write-up so you'll want enough time to do a good job. Also, there are limited resources for running experiments and if everyone waits until the last week then there will be a lot of contention for these resources. (*You* are solely responsible for finishing on time - too much contention for experimental resources is not an excuse for lateness, so start early!)

Overview

The goal of this assignment is to introduce OpenMP, MPI, and barrier synchronization concepts. You will implement several barriers using OpenMP and MPI, and synchronize between multiple threads and machines. You may work in groups of 2, and will document the individual contributions of each team member in your project write-up. (You may use Piazza to help you find a partner.)

OpenMP allows you to run parallel algorithms on shared-memory multiprocessor/multicore machines. For this assignment you will implement two spin barriers using OpenMP. MPI allows you to run parallel algorithms on distributed memory systems, such as compute clusters or other distributed systems. You will implement two spin barriers using MPI. Finally, you will choose one of your OpenMP barrier implementations and one of your MPI barrier implementations, and combine the two in an MPI-OpenMP combined program in order to synchronize between multiple cluster nodes that are each running multiple threads.

You will run experiments to evaluate the performance of your barrier implementations. (Information about compute resources for running experiments is below.) You will run your OpenMP barriers on an 8-way SMP (symmetric multi-processor) system, and your MPI and MPI-OpenMP combined experiments on a 12-node twelve-core cluster (12 nodes, and each node has two six-core processors).

Finally, you will create a write-up that explains what you did, presents your experimental results, and most importantly, analyzes your results to explain the trends and phenomona you see. (Some hints for analysis are given below.)

Detailed Instructions

These instructions are presented in a sequential order. However, depending on how you decide to divide the work with your project parter, you may choose do so some of these things in parallel. That is okay, so long as everything gets done, and you say who did what in your write-up.

Part 1: Learn about OpenMP and MPI

The first thing you want to do is learn how to program, compile, and run OpenMP and MPI programs.

You can compile and run OpenMP programs on any linux machine that has gomp installed. All the CoC Red Hat Enterprise Linux systems should have it (I believe). You can try the example code attached to this assignment (openmp.tar.gz), as well as looking at the following informational resources:

- OpenMP Website
- OpenMP Specification
- Introduction to OpenMP

You can compile and run MPI programs on any linux machine that has openmpi installed. (mpich is another MPI implementation, but we'll be using openmpi for this project. Note: *openmpi != OpenMP* !!) Unfortunately, most CoC systems do *not* have either openmpi or mpich installed. That is why we have allocated some cluster nodes for you. Details about using the cluster nodes are below. You can try running the example code attached to this assignment (mpi.tar.gz) on any of the development & testing nodes, as well as looking at the following informational resources:

- OpenMPI Website
- MPI v2 Specification

Part 2: Develop OpenMP Barriers

Implement **two spin barriers using OpenMP**. You may choose any two sping barriers you like. (For example, you could use ones from the MCS paper, anything covered in lecture, or any variation on these you think of.) Obviously, your barrier implementations cannot use the built-in OpenMP barrier! Although you can optionally use it as a third barrier in your experiments, as a baseline/control, if you choose.

Part 3: Develop MPI Barriers

Implement **two spin barriers using MPI**. At least one of these must be a *tree-based barrier* (if you choose to do both as tree-based barriers, that's okay too). If you choose, *one* of these may be the same algorithm you used for one of your OpenMP barriers, but the other one must be different. (However, even if you choose the same algorithm for one of them, you may find that you must implement it very differently using MPI than using OpenMP.) Obviously, your barrier implementations cannot use the built-in MPI barrier! Although you can optionally use it as a third barrier in your experiements, as a baseline/control, if you choose.

Part 4: Develop MPI-OpenMP Combined Barrier

Now choose one of the OpenMP barriers you implemented, and one of the MPI barriers you implemented. Combine them to create a barrier that synchronizes between multiple nodes that are each running multiple threads. (You'll want to be sure to preserve your original code for the two barriers too, so you can still run experiments on them seperately.) You can compare the performance of the combined barrier to your standalone MPI barrier. (Note that you will need to run more than one MPI process per node in the standalone configuration to make a comparable configuration to one multithreaded MPI process per node in the combined configuration, so that total number of threads is the same when you compare.)

Part 5: Run Experiments

The next step is to perform a performance evaluation of your barriers. You need to write a test harness that runs some OpenMP threads or MPI processes and synchronizes the threads/processes using your barrier implementation. Then your test harness should measure the performance of your barriers in a manner similar to the MCS paper. You should look at the experiments in that paper again and think about how they did them.
Measure your *OpenMP barriers on a fourcore node in Jinx cluster*, and **scale the number of threads from 2 to 8**. (1 thread is not really interesting for barriers, and more than 8 threads will show skewed results since the forecore nodes only have eight cores (two 4-core processors) with which to provide true concurrency.)
Measure your *MPI barriers on the sixcore nodes in Jinx cluster*. You should **scale from 2 to 12 MPI processes**, one process per sixcore node.

Measure your *MPI-OpenMP combined barrier on the sixcore nodes in Jinx cluster*, **scaling from 2 to 8 MPI processes running 2 to 8 OpenMP threads per process**.
Some things to think about in your experiments:

- When scaling from X to Y of something, you don't need to run every single number between X and Y. However, you should run one at X and one at Y, of course, and enough in between to see any interesting trends or phenomona that occur. You'll have to decide at exactly what values you need to run the experiment in order to accomplish this. (Although if you have time and want to, you may run ever single number.)
- You can use the gettimeofday() function to take timing measurements. See the man page for details about how to use it. You can also use some other method if you prefer, but explain in your write-up which measurement tool you used and why you chose it. Consider things like the accuracy of the measurement and the precision of the value returned.
- If you're trying to measure something that's too fast for your measurement tool (i.e.your tool is not precise enough), you can do it a bunch of times in a loop, measure the time to run the entire loop, and the divide by the number of iterations in the loop. This gives the average time for a single loop iteration. Think a moment about why that works, and how that increases the precision of your measurement.
- Finally, once you've chosen a measurement tool, think a bit about how you will take that measurement. You want to be sure you measure the right things, and exclude the wrong things from the measurement. You also want to do something to account for variation in the results (so, for example, you probably don't want to just measure once, but measure a bunch of times and take the average).

Part 6: Write-Up

The last part to create the write-up. This should be either a PDF file, and it should include a *minimum*of the following:

- The names of both team members

- An introduction that provides an overview of what you did. (Do not assume the reader has already read this assignment description.)
- An explaination of how the work was divided between the team members (i.e. who did what)
- A description of the barrier algorithms that you implemented. (You do not need to go into as much implementation detail, with pseudocode and so forth, as the MCS paper did. However, you should include a good high-level description of each algorithm. You should *not* simply say that you implement algorighm X from the paper and refer the reader to the MCS paper for details.)
- An explaination of the experiments, including what experiments you ran, your experimental set-up, and your experimental methodology. (Give thorough details. Do not assume the reader has already read this assignment description.)
- Your experimental results. DO present your data using graphs. DO NOT use tables of numbers when a graph would be better. (Hint: a graph is usually better.) DO NOT include all your raw data in the write-up. (If you want to submit your raw data, you may include it in a separate file in your submission.) Compare both your OpenMP barriers. Compare both your MPI barriers. Present the results for your MPI-OpenMP barrier.
- An analysis of your experimental results. You should explain why you got the results that you did (think about the algorithm details, and the architecture of the machine you exerimented on). Explain any trends or interesting phenomona. If you see anything in your results that you did not expect, explain what you did expect to see and why your actual results are different from that. There should be at least a couple intersting points per experiment - the key is not explain not only the *what* of your results, but the *how* and *why* as well.
- A conclusion.

Resources

You have access to Jinx cluster. Please read carefully instructions in the following website to perform experiments.
http://support.cc.gatech.edu/facilities/instructional-labs/how-to-access-the-jinx-cluster
You must use a queue name "cs6210" to run your program on the cluster. Plesae refer the following document and use "#PBS -q cs6210" instead of "#PBS -q class".
https://support.cc.gatech.edu/facilities/instructional-labs/how-to-run-jobs-on-the-jinx-cluster

Note that you must not use the cluster for development. Develop all implementations (OpenMP, MPI, OpenMP-MPI combination) using your local resource (e.g., a laptop) and make sure your code runs fine. Then perform experiments using Jinx cluster.

Submission Instructions

Please submit following results to T-square:

- All your code (barrier implementations, experiment test harness, etc.)
- Makefile
- Anything else we may need to compile and run all your barriers
- Experimental data
- Your write-up, that includes all the things listed above (PDF)

Only one team member has to submit the project to T-Square - be sure both team members' names are on the write-up! The other team member (who is not submitting the project) should simply turn in a file (e.g. text file) with the team members' name in it.

In addition, everyone please note that the write-up will be an important part of the project grade. It will be a good idea to make sure you create a good write-up. The assignment description on T-Square has some guidelines for the content of the write-up, but be through. Treating the guidelines like a set of checkmarks and doing the minimum to meet those checkmarks will not earn you full points. I expect graduate students to do better the following a checkmark list. For an example what what a _good_ write-up looks like, see the MCS paper itself. (Although yours doesn't have to be as many pages long as the MCS paper since you're only implementing a subset of the algorithms. Instead, consider the content and style of the paper.)