

# Project 2

**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 create a write-up after you finish coding. 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 in a later section). 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 cluster of up to 24 nodes with 12 cores each.

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 phenomena

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 partner, you may choose to do 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 *libomp* installed. You can try the example code attached to this assignment ([openmp.tar.gz](#)), as well as looking at the following informational resources:

You can compile and run MPI programs on any Linux machine that has *openmpi* installed (Note: `O_mpi != OpenMP_`). Although MPI is normally used for performing computations across different network-connected machines, it will also run on a single machine. This setup can be used for developing and testing your project locally. You can try running the example code attached to this assignment ([mpi.tar.gz](#)), as well as looking at the following informational resources:

### Part 2: Develop OpenMP Barriers

Implement **two spin barriers using OpenMP**. You may choose any two spin 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! However you can optionally use it as a third barrier in your experiments for baseline/control purposes, if you choose.

### Part 3: Develop MPI Barriers

Implement **two spin barriers using MPI**. At least one of these implementations must be a *tree-based barrier* (if you choose to do both as tree-based barriers, that's okay too). You may also opt for *one* of these implementations to use the same algorithm you chose 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

when using MPI vs. using OpenMP). Obviously, your barrier implementations cannot use the built-in MPI barrier! However you can optionally use it as a third barrier in your experiments, 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 also want to be sure to preserve your original code for the two barriers so that you can still run experiments on them separately. 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 do 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 were conducted.

You will measure your *\_OpenMP* barriers on a *\_single cluster node*, and **scale the number of threads from 2 to 8**.

You will measure your *MPI barriers on multiple cluster nodes*. You should **scale from 2 to 12 MPI processes**, one process per node.

You will measure your *MPI-OpenMP combined barrier on multiple cluster nodes*, **scaling from 2 to 8 MPI processes running 2 to 12 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 phenomena 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 every 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 an operation that completes too fast for your measurement tool (i.e., if your tool is not precise enough), you can run that operation several times in a loop, measure the time to run the entire loop, and then 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 several times and take the average).


## Part 6: Write-Up

The last part is to create the write-up. This should be 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 explanation 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 algorithm X from the paper and refer the reader to the MCS paper for details.
- An explanation 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 on which you experimented). Explain any trends or interesting phenomena. 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. There should be at least a couple of interesting points per experiment. The key is not to explain only the *\_what\_* of your results, but the *how* and *why* as well.
- A conclusion.

## Resources

You will have access to the *pace-ice* PACE cluster for use with this project. Please read the following instructions carefully to familiarize yourself with the PACE cluster computing environment from [this](#)  presentation.

**Note:** You will have to be on the Georgia Tech VPN in order to login to these clusters.

You must use the queue named “**pace-ice**” to run your programs on the cluster. Please refer to the following document on how to create a batch script to run your program job:

[http://docs.pace.gatech.edu/scheduler/job\\_submission/](http://docs.pace.gatech.edu/scheduler/job_submission/)(Links to an external site.)

**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. After you have developed working implementations locally, you may use the cluster to run these implementations and gather results for your write-up.

We will provide example PBS batch configuration scripts that you can use to more easily submit/run your job on the *coc-ice* cluster within a week after this assignment is released, but you should still familiarize yourself with basic usage of the cluster so you can interpret and adapt these scripts for your use.

## Submission Instructions

Please submit following results in a single zip file via Canvas:

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

- Your directory structure should be like this:

Firstname\_Lastname\_p2/

|- MPI/

|- Makefile

|- .c and .h files

|- OpenMP/

|- Makefile

|- .c and .h files

|- Combined/

|- Makefile

|- .c and .h files

|- README

|- Write up (PDF)

|- Data/

|- all experimental data

Only one team member needs to submit the actual project files to Canvas, but be sure that both team members' names are on the write-up! The other team member (who is not submitting the project files) should simply turn in a text file indicating both team members' names and the name of the team member who submitted the project files.

In addition, please note that the write-up will be an important part of the project grade. As such, it is a good idea to make sure you create a good write-up. The assignment description on Canvas has some guidelines for the content of the write-up, but be thorough. Treating the guidelines like a set of checkmarks and doing the minimum to meet those checkmarks will not earn you full points. I expect students in a graduate-level course to do more than following a checkmark list. For an example of what a *good* write-up looks like, see the MCS paper itself. Although your paper does not need to be as long as the MCS paper (since you are only implementing a subset of the algorithms), it would still be a good idea to emulate its style and content to ensure success.

## Examples and Instructions for Cluster setup

[project2-common-master.zip](#)