

Computer Science 335
Parallel Processing and HPC
Siena University
Fall 2026

Lab 11: Life with MPI and OpenMP

Due: 10:00 AM, Wednesday, December 2, 2026

In this lab, you will see how to write, compile, and run programs that use a combination of MPI and OpenMP for parallelization.

We will work together through this lab, but everyone must submit their own responses to the lab questions.

Learning goals:

1. Gain experience with programming in a hybrid MPI/OpenMP model.

Getting Set Up

Since you will not be writing your own code for this lab, there is no lab-specific repository. However, you should clone the repositories for these examples:

- `mpiopenmp-hello`
- `mpiopenmp-life`

At the end, you will submit a PDF file or a link to a Google shared document that contains your responses to the questions below through the Canvas assignment corresponding to this lab.

Compiling and Running

Clone the `mpiopenmp-hello` repository on noreaster.

Before looking at the program, notice that the `Makefile` uses the `mpicc` compiler wrapper along with some extra compiler and linker flags related to OpenMP. There are some extra flags related to running on my Mac that don't seem to cause any problems on noreaster.

Compile and link using `make`.

Set `OMP_NUM_THREADS` to 4 and run with 2 MPI processes (`mpirun -np 2`).

Question 1: Copy and paste the output when you run the program this way. (3 points)

Question 2: How many threads were created overall? (1 point)

Question 3: How would you run with 4 MPI processes and 2 threads per MPI process? (1 point)

Question 4: Draw a memory diagram for a run with 2 MPI processes and 2 threads per MPI process, when all threads are executing the printout at line 30. (10 points)

Conway's Game of Life Example

Clone the `mpiopenmp-life` repository on noreaster.

This version of Conway's Game of Life is similar to the MPI and OpenMP versions we saw previously, bringing in the parallelization techniques from each.

Before we look at that, we take a look at the pseudorandom number generation (PRNG) used by this version. Since we would like the numbers generated to determine the initial state of the cells to be as pseudorandom as the case where we have a single process and single thread, separate PRNG seeds are computed for each thread, and each thread uses a separate state variable that gets passed to `next_random` to generate values. We do not need to worry about the math, just be aware that this is an issue in parallel computing in general.

Now looking at the parallelization.

Question 5: Suppose we run a 1024×1024 simulation. If there is no parallelization, the grids are allocated with 1026×1026 `int`s, which we can think of as rows 0 to 1025. Rows 0 and 1025 represent the boundary of the Life universe, while rows 1 to 1024 are the cells of the simulaton. If we run with 4 MPI processes, a distributed data structure is created. a. How large are the grids allocated on each process? b. On which process is each of the 1026 "global" rows allocated? c. For any row which is replicated, which process computes values in that row and which process has a remote copy? d. What messages need to be sent between each pair of iterations to update remote copies? (6 points)

OpenMP threads are used in multiple phases of the computation, each using a `for` directive.

Question 6: Within an MPI process, how does OpenMP assign work to threads for its parallel `for` blocks? (2 points)

Question 7: Describe how the `birth_count`, `live_count` and `death_count` variables go from their initialization on lines 222-224 to being printed out on line 267. (4 points)

Here are some example timing studies. The first two are for 10 iterations on a 40960×40960 universe.

Running 16 total threads on a MacBook Pro M5 with 18 Cores (6 Super and 12 Performance). The script to run is here.

MPI Processes	OpenMP Threads/Proc	Time
1	16	18.860615
2	8	18.532131
4	4	18.548327
8	2	17.802275
16	1	18.286892

And on noreaster, 20 cores:

MPI Processes	OpenMP Threads/Proc	Time
1	16	50.557521
2	8	47.161144
4	4	48.092721
8	2	45.999678
16	1	46.226815

Question 8: Discuss these results. Is any advantage to MPI processes *vs.*OpenMP threads apparent here? (5 points)

For multicore systems like these, we can run our computation with MPI processes, OpenMP threads, or any combination. But if we want to use cores in different physical computer systems (that do not share memory), such as multiple nodes of a supercomputer like Stampede3, we need to involve MPI for cross-node communication. But is it better to simply use as many MPI processes as there are cores, or use one MPI process per node and use OpenMP threads to take advantage of the cores on that node?

Two timing studies, submitted to 4 Stampede3 Skylake nodes using these Slurm files: Pure MPI and MPI/OpenMP Hybrid. Each was run with a 192000×192000 universe with a total of 192 threads.

Results:

- 192 MPI processes: 22.613497
- 4 MPI processes with 48 OpenMP threads each: 25.360132

Question 9: Discuss these timing results. What about this particular combination might cause this behavior? (8 points)

Tools and techniques exist, that are beyond what we discuss in this class, to measure how much time is spent in computation, thread management, message passing, etc.

Submission

Submit your responses to the lab questions through the Canvas assignment for this lab.

Grading

This assignment will be graded out of 40 points.

Feature	Value	Score
Question 1	3	
Question 2	1	
Question 3	1	
Question 4	10	
Question 5	6	
Question 6	2	
Question 7	4	
Question 8	5	
Question 9	8	
Total	40	