**Blue Waters Petascale Semester Curriculum v1.0**

**Unit 4: OpenMP**

**Lesson 6: When Should You Use OpenMP?**

**Exercise Instructions for Students**

*Developed by Colleen Heinemann for the Shodor Education Foundation, Inc.*



*Except where otherwise noted, this work by The Shodor Education Foundation, Inc. is licensed under CC BY-SA 4.0. To view a copy of this license, visit*[*https://creativecommons.org/licenses/by-sa/4.0*](https://creativecommons.org/licenses/by-sa/4.0)

*Browse and search the full curriculum at*[*http://shodor.org/petascale/materials/semester-curriculum*](http://shodor.org/petascale/materials/semester-curriculum)

*We welcome your improvements! You can submit your proposed changes to this material and the rest of the curriculum in our GitHub repository at*[*https://github.com/shodor-education/petascale-semester-curriculum*](https://github.com/shodor-education/petascale-semester-curriculum)

*We want to hear from you! Please let us know your experiences using this material by sending email to* [*petascale@shodor.org*](mailto:petascale@shodor.org)

**Running Different Programs with OpenMP**

This is going to be an in-depth exercise looking at how 3 very different programs perform with OpenMP. The idea is to look at how running on an increasing number of threads can either increase or impede performance. The exercises described here are going to look at increasing the number of OpenMP threads that the programs are run on in addition to combining this with different pairings of parameters that the programs require.

There are 3 different programs that you will run with OpenMP: (1) Conway’s Game of Life; (2) Pandemic/Infectious Disease; and (3) Sieve of Eratosthenes. A brief description of each program will be provided in these exercise instructions, but additional detailed documents will be provided in the References and Further Reading.

**Goals**

* Learn and understand the following concepts:
  + How to compile and run a program with OpenMP threading
  + Run OpenMP programs with different numbers of OpenMP threads combined with different pairings of parameters that the programs require
  + Analyze what information the timing results from running OpenMP programs are telling you
  + Critically think about the best combination of parameters and number of OpenMP threads for different types of programs
* Practice:
  + Connecting to a supercomputer
  + Downloading code from GitHub
  + Compiling OpenMP code on a supercomputer
  + Running different types of problems on a supercomputer with different numbers of processing elements (PEs)
  + Gathering timing information
  + Analyze results from running programs

**Conway’s Game of Life**

The goal of Conway’s Game of Life is to look at the scientific tool **cellular automaton**. This type of tool can be used to model a variety of natural phenomena. There are numerous types of natural phenomena that can be modeled, such as hydrodynamics, cellular growth, etc. Specifically in this program, the natural phenomena that is being modeled is population characteristics. The Game of Life is run on a two-dimensional grid of square cells, size determined by input parameters into the program. Each cell has two possible states—ALIVE or DEAD. Every cell interacts with its 8 neighbors and at every time step, also a parameter for the program, each of the rules are applied to every cell and the population is updated:

1. Any live cell with < 2 live neighbors dies
2. Any live cell with 2 or 3 live neighbors lives
3. Any live cell with > 3 live neighbors dies
4. Any dead cell with 3 live neighbors becomes a live cell again

The program takes in 3 parameters:

**r:** number of rows of the grid

**c**: number of columns of the grid

**t**: number of time steps to run

Additional details about how the program can be found in the Further Reading section of this lesson in the Conway’s Game of Life Module Document.

**Pandemic/Infectious Disease**

The goal of Pandemic/Infectious Disease is to model the epidemiological concept of transmitting diseases from person to person, depending on how susceptible each individual in the experiment is and how many individuals there are. A detailed explanation of how the algorithm works is provided in the Further Reading section in the Infectious Disease Module Document.

The program takes in several parameters:

**n**: number of people in the model

**i**: number of people infected at the start

**w**: width of the environment

**h**: height of the environment

**t**: number of days model runs

**T**: how long the disease can last (in days)

**c**: how contagious the disease is

**d**: infection radius of the disease

**D**: how deadly the disease is

Using all of these parameters can make the model extremely accurate, but for the sake of this exercise, we will only use a few of the parameters. The rest will be set to defaults in the program. The ones we will modify are:

**n**: number of people in the model

**w**: width of the environment

**h**: height of the environment

**Sieve of Eratosthenes**

The goal of Sieve of Eratosthenes is to find all of the prime numbers below a given integer, which is an input parameter to the program. A detailed explanation of the algorithm is provided in the Further Reading section in the Sieve of Eratosthenes Module Document.

The program takes one parameter:

**n**: the program will find prime numbers less than this number

**Activity**

1. Login to the supercomputer or cluster that you are using
2. Copy the code to the supercomputer or cluster that you are using
3. Change to the code directory for **life**:

**cd life<ENTER>**

1. For this exercise, we need the serial and OpenMP versions of the code. Compile the code by typing the following two commands:

**make serial<ENTER>**

**make openmp<ENTER>**

1. Change directories from **life** to **pandemic**:

**cd ../pandemic<ENTER>**

1. For this example we need the serial and OpenMP versions of the code. Compile the code by typing the following two commands:

**make serial<ENTER>**

**make openmp<ENTER>**

1. Change directories from **pandemic** to **sieve**:

**cd ../sieve<ENTER>**

1. For this example, we need the serial and OpenMP versions of the code. Compile the code by typing the following two commands:

**make serial<ENTER>**

**make openmp<ENTER>**

1. Open the Excel file titled **OpenMP Results**. There are 3 sheets in this file, one for Game of Life, one for Pandemic, and one for Sieve of Eratosthenes. This is where you are going to put the timings for each of the tests that you run. A few points to make:
   1. Each experiment will be run 3 times. There is a box for the average time it takes for the 3 runs to complete. This is in case there is some type of fluke in the program while it’s running that makes it take longer than normal. Having 3 runs provides a more realistic result.
   2. For every set of parameters given in Column A in the spreadsheet, you will run the program in serial and with OpenMP on 1, 2, 4, 8, and 16 threads
   3. Do this for each of the three of the programs and record the timing results for each of the three programs in the correct locations.

Here we will walk through an example for each of the three programs using the first set of parameters to be run. After walking through each of these examples, you should be able to execute the rest on your own.

First, request an interactive job on the supercomputer or cluster that you are using. It is up to you how long you want to request for your job to run. Some of the serial runs, particularly for those with much larger values for parameters, may take a very long time.

When you built the serial and OpenMP versions of each program, it output executables with .serial and .openmp at the end. For example, for the life program, the executables output should be life.serial and life.openmp.

We will be using the **time** command to calculate how long the program took to run. There should be 3 different times output, such as:

**real 0m35.170s**

**user 1m16.892s**

**sys 0m8.021s**

You should record the **real** time in the spreadsheet.

Change directories into **life**

For the **serial** version, **life** program, and parameters **–r 100 –c 100 –t 100**:

**$time ./life.serial -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

To run the OpenMP version, we need to set the number of OpenMP threads before each run where the number of threads is different than the previous run. For example, we will set the number of OpenMP threads to 1, do 3 runs, change the number of OpenMP threads to 2, do 3 runs, etc.

For the **OpenMP** version, **life** program, and parameters **–r 100 –c 100 –t 100**:

**$export OMP\_NUM\_THREADS=1<ENTER>**

**$time ./life.openmp -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=2<ENTER>**

**$time ./life.openmp -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=4<ENTER>**

**$time ./life.openmp -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=8<ENTER>**

**$time ./life.openmp -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=16<ENTER>**

**$time ./life.openmp -r 100 -c 100 -t 100<ENTER>**

Record the **real** time that is reported and run 2 more times.

Change directories into **pandemic**

For the **serial** version, **pandemic** program, and parameters **–n 100000 –w 10000 –h 10000**:

**$time ./pandemic.serial -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

For the **OpenMP** version, **pandemic** program, and parameters **–n 100000 –w 10000 –h 10000**:

**$export OMP\_NUM\_THREADS=1<ENTER>**

**$time ./pandemic.openmp -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=2<ENTER>**

**$time ./pandemic.openmp -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=4<ENTER>**

**$time ./pandemic.openmp -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=8<ENTER>**

**$time ./pandemic.openmp -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=16<ENTER>**

**$time ./pandemic.openmp -n 100000 -w 10000 -h 10000<ENTER>**

Record the **real** time that is reported and run 2 more times.

Change directories to **sieve**

For the **serial** version, **sieve** program, and parameter **–n 1000000**:

**$time ./sieve.serial -n 1000000<ENTER>**

Record the **real** time that is reported and run 2 more times.

For the **OpenMP** version, **sieve** program, and parameter **–n 1000000**:

**$export OMP\_NUM\_THREADS=1<ENTER>**

**$time ./sieve.openmp -n 1000000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=2<ENTER>**

**$time ./sieve.openmp -n 1000000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=4<ENTER>**

**$time ./sieve.openmp -n 1000000<ENTER>**

Record the **real** time that is reported and run 2 more times.

**$export OMP\_NUM\_THREADS=8<ENTER>**

**$time ./sieve.openmp -n 1000000<ENTER>**

Record the **real** time that is reported and run 2 more times.

Continue running tests with the other combinations of parameters for each of the three programs. You should have your spreadsheet completely filled in by the end of this exercise.

**Analysis**

Now that you have run all of your tests for all 3 of the programs for both the serial and OpenMP versions, it is now time to take a look at the results and see if any conclusions can be drawn. Answer the following questions based on your results.

1. What was the best combination of parameters and number of threads for each of the programs, Game of Life, Pandemic/Infectious Disease, and Sieve of Eratosthenes? Why do you think this is? Is it because there is enough data to spread across numerous threads, the problem size was large enough, there were enough threads to maximize on performance, etc.?
2. Was there a decrease in performance at any point for any of the programs you ran? If so, when was it? Why do you think this happened?
3. What recommendations would you have for someone running these programs that would want to maximize on performance while running these three programs?