# Introduction to Parallel Programming (CS462)

### Assignment 2: OpenMP

### Due: Wednesday September 27, 2023 @ 11:59 PM

The purpose of this programming assignment is to gain experience in parallel programming with OpenMP. You will first create a working serial program that models an ocean's temperature, analyze its performance, and then add OpenMP directives to create a parallel program.

Your version of myocean will simulate water temperatures using a large grid of integer values over a fixed number of time steps. At each time step, the value of a given grid location will be averaged with the values of its immediate north, south, east, and west neighbors to determine the value of that grid location in the next time step (total of five grid locations averaged to produce the next value for a given location).

The grid points in the myocean grid can be separated into two independent red and black subsets as discussed in class. Instead of calculating a new value for each grid location at every time step, the values for the grid points in the red subset are updated on even time steps and the values for the grid points in the black subset are updated on odd time steps.

The edges of the grid shown i(the boundary conditions) do not participate in the averaging process (they contribute a value, but their value does not change). Thus, myocean will converge (given sufficient runtime) to a gradient of the water temperatures on the perimeter of the grid.

The goal is to be systematic in figuring out how to parallelize this program. You can use one of three things: manually inserted timers, or gprof, to figure out what parts of the program take the most time. From there you should examine the loops in the most important subroutines and figure out how to add OpenMP directives.

**Part 1 (sequential code):**

Write a single-threaded (sequential) version of myocean as described above. Your version of myocean must take three arguments: the x-dimension of the grid, the y-dimension of the grid, and the number of time steps. You may assume for simplicity that all grid sizes will be powers of two plus two (i.e. $2^n+2$); therefore the area of the grid that will be modified will be sized to powers of two (+2 takes care of the edges/boundary conditions that are not modified).

You are required to make an argument/demonstrate that your implementation of myocean is correct. A good way to do this is to initialize the grid to a special-case starting condition, and then show that after a number of time steps the state of the grid exhibits symmetry or some other expected property. I will provide myocean.in and myocean.in.short files with initial conditions provided for the grid.

Your final sequential version of myocean should initialize a grid of the requested size, potentially using the input files, then perform simulation for the specified number of time steps. I will test your code using one or both of these input files.

**Part 2 (parallel implementation):**

For this problem, you will use OpenMP directives to parallelize your sequential program. Please be sure to explicitly label all appropriate variables as either shared or private. Make an argument for the correctness of your implementation (it is acceptable to use the same argument as part 1, provided it is still applicable).

The program should take an additional command-line argument: the number of threads to use in parallel sections. It is only required that you parallelize the main portion of the simulation, but parallelizing the initialization phase of myocean is also worthwhile. You will not be penalized if you choose not to parallelize the initialization phase.

For simplicity, you may assume that the dimensions of the grid are powers of two plus two as before, and that only N=[1,2,4,8] will be passed as the number of threads.

**Using OpenMP**

To compile OpenMP we should use gcc (e.g., on a node of hydra or tesla), which has OpenMP support built in. In general, you can compile this assignment with:

```
gcc -fopenmp -o myocean myocean.c
```

The -fopenmp tells the compiler to recognize OpenMP directives.

The environment variable **OMP_NUM_THREADS** sets the number of threads (and presumably cores) that will run the program. Set the value of this environment variable in the script you submit the job from.

**Running the program**

Myocean should read its input from standard input, and produces its output on standard output. Myocean should generated an output message periodically (e.g., every 30 of its simulation time steps), so you can tell if it is making progress.

Since myocean runs for a while on the input myocean.in dataset for a small numbers of threads, myocean.in.short is another input file that runs for much less time (you can use this for testing).

**What to Submit**

You must submit the following files and no other files:

- A report that describes what you did, and identifies the code regions that consume the most time.
- In same report, add the times to run it on the input file myocean,in (for 1, 2, 4, and 8 threads).
- myocean-omp.c: modified myocean.c file with OpenMP directives.
- A Makefile to compile myocean.c and myocean-omp.c

You should put the code, Makefile and report in a single directory (named `LastName-FirstName-assign2`), and create a tar or zip file with the source code.

NOTE: If the output file or timing output is not as described above, you will not get any points. To time your program, use omp_get_wtime() by placing one call at the beginning of main (next to the omp_get_max_threads call) and another one toward the end of main (before the "Done. Terminating the simulation" print statement). Sample timing code below:

```
double start, end;

start = omp_get_wtime();
... work to be timed ...
end = omp_get_wtime();

printf("TIME %.5f s\n", end - start);
```

**Grading**

The project will be graded as follows:

| Component | Percentage |
|---|---|
| **Runs correctly on 2 threads** | 20 |
| **Runs correctly on 8 threads** | 40 |
| **Performance with 2 threads** | 20 |
| **Performance with 8 threads** | 10 |
| **Writeup** | 10 |

NOTE: If your program does not run correctly, you do NOT get any points for performance/ speedup.