SIENAcollege
Computer Science

Computer Science 335
Parallel Processing and HPC
Siena College
Fall 2024

Lab 10: OpenMP Practice

Due: 4:00 PM, Monday, December 2, 2024

In this lab, you will get a little practice parallelizing programs with OpenMP.

You may form groups of 2 if you wish. You should write your programs in your own (or your own group's) repository and answer the questions on this lab in your own document, as we work together as a class through the tasks.

Learning goals:

1. To practice using OpenMP to parallelize programs.

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `openmp-lab-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

You may choose to answer the lab questions in the `README.md` file in the top-level directory of your repository, or upload a document with your responses to your repository, or add a link to a shared document containing your responses to the `README.md` file.

More Pi

No, we're not going to do Jacobi iteration again, but we will revisit a familiar problem.

Practice Program: Write an OpenMP version of the Monte Carlo approximation of π. Call your program `openmp_pi.c` in the `pi` directory of your repository. See below for some tips. (20 points)

- Your program should take one command-line parameter: the number of random points to be generated by each thread. Let OpenMP determine the number of threads based on the `OMP_NUM_THREADS` environment variable.
- Each thread in the `parallel` block will compute and report its own estimate of π.
- Use a `reduction` directive to combine the number of points in the circle to get your total for the final approximation.

Practice Program: Add appropriate timers to each of your Monte Carlo π approximation programs (MPI, pthreads, OpenMP) to report the time taken to compute the entire approximation. Include all three programs in the `timepi` directory of your repository. (5 points)

Question 1: Run each of your Monte Carlo π approximation programs (MPI, pthreads, OpenMP) with 16, 32, and 64 processes/threads on a Stampede3 production node (can be either interactive or batch, whatever you prefer) and 1 billion points per process/thread. Report the times taken in tabular form. (10 points)

Question 2: Discuss what these timings tell you about the relative efficiencies of the three versions of the program. (5 points)

Another Closest Pairs Variant

This section's program will be graded as a practice program and is worth 25 points.

In the `cp` directory of your repository, you will find a copy of the OpenMP closest pairs code from the class example. Your task is to remove the shared variable access by adding an array of structs where each thread can put its solution, and the main thread finds and reports the overall winner. This mode of operation of this program will be selected by passing the string `noshared` in `argv[1]`. The following steps will guide you.

1. Add the new mode to the parameter check near the start of `main`.
2. Add a third case to the condition under the "`// do it`" comment in `main`. Call a function (which you will write next) `tmg_closest_pair_omp_noshared` with the same parameter list as the others.
3. Define a `struct` with three fields to hold the two indices into the array of vertices and the distance between them that will hold the information about a "leading" pair.
4. Make a copy of the `tmg_closest_pair_omp_coarse` function and name it `tmg_closest_pair_omp_noshared`.
5. Remove the variables `v1`, `v2`, and `distance` from the `shared` clause on the `parallel` directive near the start of the function.
6. Move the declaration of the `num_threads` variable before the `parallel` directive, since we will need its value after the parallel block. Also, add `num_threads` to the `shared` clause of the `parallel` directive.
7. Move the two function calls that get the number of threads and thread number to the start of the `parallel` block. Don't forget that you no longer need to declare `num_threads` since it's declared outside of the `parallel` block and that variable will be shared by all of the threads.
8. Outside the `parallel` block, declare a variable that points to an array of the `struct` you defined above. If it's called `leader`, your declaration will look like:

```
leader *leaders;
```

This will point to an array of these `structs`, one for each thread. Note that we cannot construct that array yet because we can't find out how many threads there will be until we get into the `parallel` block.

9. Add the `leader` variable you declared in the previous step to the `shared` clause.
10. Now we can allocate space for the array, but we want to make sure it gets created just once. Place the appropriate `malloc` below a `single` directive to ensure that only one thread executes it. The statement should allocate an array of the `structs`, one per thread.
11. The `leader` array will be accessed by the threads, but each will access only the one at the index corresponding to its thread number. But we need to make sure that the one thread that is chosen to create it has completed its work before any thread tries to access it. So follow the statement with a `barrier` directive.
12. Replace the `local_v1`, `local_v2`, and `local_distance` variables with references to `leaders[thread_num].v1`, `leaders[thread_num].v2`, and `leaders[thread_num].distance`, respectively.
13. Remove the `critical` directive and its `if` statement at the end of the `parallel` block.
14. Now, outside of the `parallel` block, add a loop over the `leaders` array to find the one with the smallest distance, and store those vertex indices and the distance in `v1`, `v2`, and `distance`.
15. `free` the memory you allocated for the `leaders` array.
16. Test your code.

Submission

Commit and push!

Grading

This assignment will be graded out of 65 points.

Feature	Value	Score
OpenMP pi	20	
Timers in pi programs	5	
Question 1: timings	10	
Question 2: timing analysis	5	
closest pairs variant	25	
Total	65	