Assignment 3: OpenMP Producer/Consumer

Please read these instructions carefully. Your grade is based upon meeting all requirements stated in this assignment.

Background and Problem Statement

We now have a working pthreads parallel producer/consumer program. Congratulations! We also have them benchmarked for runtime so that we can compare their relative performance with other parallel implementations.

A design objective for OpenMP was to take in-tact serial programs and inject compiler directives which would easily parallelize them. We have seen in class how this is possible in some cases. However, this may or may not be practicalin the general case, nor with your serial consumer/producer program.

Program Requirements

- Start with the skeleton program provided (prod_consumer_orig.c) and then re-architect it as needed to add support for OpenMP based parallelism.
- Your final source code MUST be named prod_consumer_omp.c
- You may read the input from either a serial or a parallel region within your program. All input should be read from stdin as in assignmen-2.
- Your program should support having **multiple** "producer" threads which will populate a shared data structure like a stack or a queue.
  - You are free to choose any other data structure you prefer. The data structure can be either declared globally or locally to the main function.
  - The simplest way to implement this would be with a stack with one pointer as below.
- You program should support having **multiple** "consumer" threads, which will take work from the shared data structure, perform the dummy computations.
  - The consumers MUST consume ALL the data the producers put in the shared data structure. Nothing should be left in the shared data structure once all the consumers have exited. If consumers do not read all the data, you will lose points.
  - We use a usleep as dummy computation.
    - The skeleton code calls usleep(10 * number) after each extraction
    - However, this must be changed to usleep(100000 * number) and re-tested with this change before submission. Failure to do this will be considered a sub-optimal implementation, and you will lose points.
  - Output should be printed on stdout.
  - The order of output NEED NOT be the same as the order of the input file.
- There are two helper functions provided that are used to print to stdout.
  - print_insertion takes the value being inserted and the insertion index/location
  - print_extraction takes the consumer number, the value being extracted, and the extraction index/location
  - You may perform your output from either a serial or a parallel region within your program, so long as the output retains the intended format (i.e. the output is not "garbled")

- The order of output NEED NOT be the same as the order of the input file.
- Please use these helper functions inside buffer_insert and buffer_extract. These will be used by the scripts to validate your solution's functionality.

- You should try to minimize the number of changes needed.
  - In the simplest case, with a data structure of stack and one location variable as given in the skeleton program, you only need 40 lines of change from the skeleton program to implement your OMP-based parallel version of the producer consumer problem. 11 of those lines are OMP pragmas.
  - The best solution only needs 3 different types of OMP pragmas.
    - private, parallel, shared, atomic, etc are considered as pragmas
  - There will be extra credits if you can complete your changes with less than 45 lines of change and less than 4 OMP pragmas.

Input Files Provided

The following 4 files have been provided to you. All files are available on Owens in the /fs/ess/PAS2661/Lab2-Assignment directory.

- prod_consumer_orig.c – The skeleton file
- Input files – longlist and shortlist
- run_script.sh – A script to run the final program and collate the output
- validate.py – A helper script to help validate the output of your program

Testing Instructions

- Use the GNU compiler ("gcc") only.
  - run_script.sh should already do this.
  - Your program should compile without warnings.
  - Points will be deducted for any warnings
- Logging in to Owens using SSH

  ssh -X -C <username>@owens.osc.edu
- When compiling and making short (less than a couple minutes) test runs, you can use the login nodes.
- For final benchmark runs or when running with the larger value of usleep, please do it on an interactive allocation as described below.
- Getting an interactive allocation

  salloc -N <number_nodes> -A PAS2661 --time=<time_in_minutes>

  e.g. To request for one full compute node from the serial partition for 30 minutes, you can use the following command. I am just using 30 as an example. Please use higher values as appropriate.

  -bash-4.2$ salloc -N 1 -A PAS2661 -p serial --time=30

  salloc: Pending job allocation 17868816

  salloc: job 17868816 queued and waiting for resources

  salloc: job 17868816 has been allocated resources

  salloc: Granted job allocation 17868816

  salloc: Waiting for resource configuration

  salloc: Nodes o0646 are ready for job <- That is the node which was allocated for you

```
bash-4.2$ hostname
owens-login03.hpc.osc.edu  <- You're still on the head node when you get the control back
bash-4.2$ ssh o0646  <- This will move you to the node that was allocated
<….snip….>
-bash-4.2$ hostname Now you're on the node that was allocated.
o0646.ten.osc.edu
```

Report Requirements

Provide a 1-2 page report which includes the following (note that accuracy and brevity are prized, needlessly lengthy reports may lower your score).

- You will submit the PDF report on Carmen.
- In your PDF report, you should address the following points
  - Provide a brief explanation describing how your changes ensure mutual exclusion.
    - Describe the data structure that is shared among the workers.
      - Explain all the logistics of how the data structure is used.
      - Saying you used a “shared buffer” is not descriptive enough.
    - List the methods of synchronization
    - List *all* variables protected by the synchronization.
      - e.g. do not say “I used the mutex to protect my stack”
      - A good example: “Variables *i*, *j*, and array *my_array* can only be read/written when *my_mutex* is acquired.”
  - Provide two graphs in your report
    - One graph shows the latency of each solution
    - Another graph shows the speedup over the single producer single consumer case
    - The X-axis should be number of consumers
    - The graph will have one line each for number of producer
    - Graph datapoints for 1, 2, 4, 8, and 16 consumers
    - Also include lines for 1, 2, 4, 8, and 16 producers
    - Use log scaling on the X-axis
    - Deviation from this format will result in lost points
  - Discuss how the performance changes as you vary the parameters. Is there a point where the solution doesn’t scale as effectively or is lacking efficiency?
  - If you saw any unusual or unexpected results, describe them.

Code Submission Instructions

- The run_script.sh file will compile your program and run it for different combinations of producers and

consumers with different input files.

- On successful completion, the run_script.sh file will create a sub-directory named "cse5441-omp-lab" and place all necessary files for submission there.
- Submit your cse5441-omp-lab directory from an Owens login node using the OSC submit system
  - More details about how to submit your assignment from OSC is available at the following website: https://www.osc.edu/resources/getting_started/howto/howto_submit_homework_to_repository_at_osc
- Submit this directory to "assignment-3"
  - You can use the following command to do the submission

    /fs/ess/PAS2661/CSE-5441-SP24/submit
  - Note that the system will automatically close at the specified deadline for submission. You will not be able to submit your files after that.
  - Here is a sample set of steps to submit.

./run_script.sh

Program may be using….

<…snip…>

-bash-4.2$ ll

total 72
-rwxr-xr-x 1 osc0577 PAS2661 14344 Mar 15 09:38 a.out
drwxr-xr-x 2 osc0577 PAS2661 4096 Mar 15 09:42 cse5441-omp-lab
-rw-r--r-- 1 osc0577 PAS2661 9170 Mar 15 09:42 cse5441-omp-lab.tgz
-rw-r--r-- 1 osc0577 PAS2661 410 Mar 15 09:31 longlist
-rw-r--r-- 1 osc0577 PAS2661 9404 Mar 15 09:31 prod_consumer_omp.c
-rw-r--r-- 1 osc0577 PAS2661 8468 Mar 15 09:31 prod_consumer_orig.c
-rwxr-xr-x 1 osc0577 PAS2661 5137 Mar 15 09:38 run_script.sh
-rw-r--r-- 1 osc0577 PAS2661 78 Mar 15 09:31 shortlist

-bash-4.2$ /fs/ess/PAS2661/CSE-5441-SP24/submit

****

Hello. This script will submit your assignment assigned by Hari Subramoni to OSC

Note:

Before you run this script, please create one directory which includes all the files you want to submit

The previous submission of the same assignment from you will be replaced by the new submission

Enter the name of assignment and press [ENTER]

>>> Lab2-Assignment

Enter the absolute path of the directory which includes all the files of your assignment Lab2-Assignment and press [ENTER]

You will submit the current directory if you simply press [ENTER]

>>>/users/PAS2661/osc0577/CSE-5441/OpenMP/Homework/cse5441-omp-lab

Your assignment Lab2-Assignment is submitted successfully. Thank you!

Rubrics

Note to students: I have attempted (to the best of my abilities) to cover all possible cases and create a comprehensive rubric. I am and have been updating these based on feedback/questions I have received over the duration of this course. Even after this, it could still not cover all the questions you may have.

The student should have submitted one program – prod_consumer_omp.c.

1. 70 points for the correct program without any data validation errors
    a. For a correct program, the points will be divided as follows
        i. 50 points for the correctness of the implementation
        ii. 20 points for the quality of the implementation
            - 10 points will be deducted if the code does not pass a linting test using cpplint
            - Students are encouraged to use cpplint on their submission code before submitting
            - See https://github.com/cpplint/cpplint for how to setup and use cpplint
    b. If you have validation errors, you will lose a minimum of 25 points.
    c. If the program does not run for the grader, a minimum of 25 points will be deducted.
        i. Questions of the type “the program ran fine for me, but not for the grader”, “the program did not hang for me, but it hung for the grader” etc may be considered on a case-by-case basis.
    d. If you forget to submit your program(s) because you did not follow the instructions listed above, you will lose all 70 points that are meant for the program. No excuses.
2. 15 points for getting approximately linear scaling/improvements in performance with the input longlist on the Owens batch partition.
    a. The program should approximately show linear scaling/improvements in compute time as reported by the run_script.sh.
    b. If the improvement is less than 80% of linear scaling, the student will lose 15 points for the program.
3. 10 points for the report and the clarity with which the student has described the changes
4. 5 points if the program cleans up all resources it allocates
    a. The program should cleanup all resources it allocated.
    b. If it does not – i.e., if there are memory leaks, then the student will lose points based on the percentage of resources they failed to clear up.

5. Late submissions will lose points
    a. The due date will not be extended
    b. 5 points will be deducted for every 24 hours of delay for a maximum allowable delay of 48 hours.
    c. Beyond 48 hours of delay, the student will receive no points for the assignment.

Academic Integrity

Collaborating or completing the assignment with help from others or as a group is NOT permitted

Copying or reusing previous work done by others is not permitted. You may reuse the work you did as part of this class