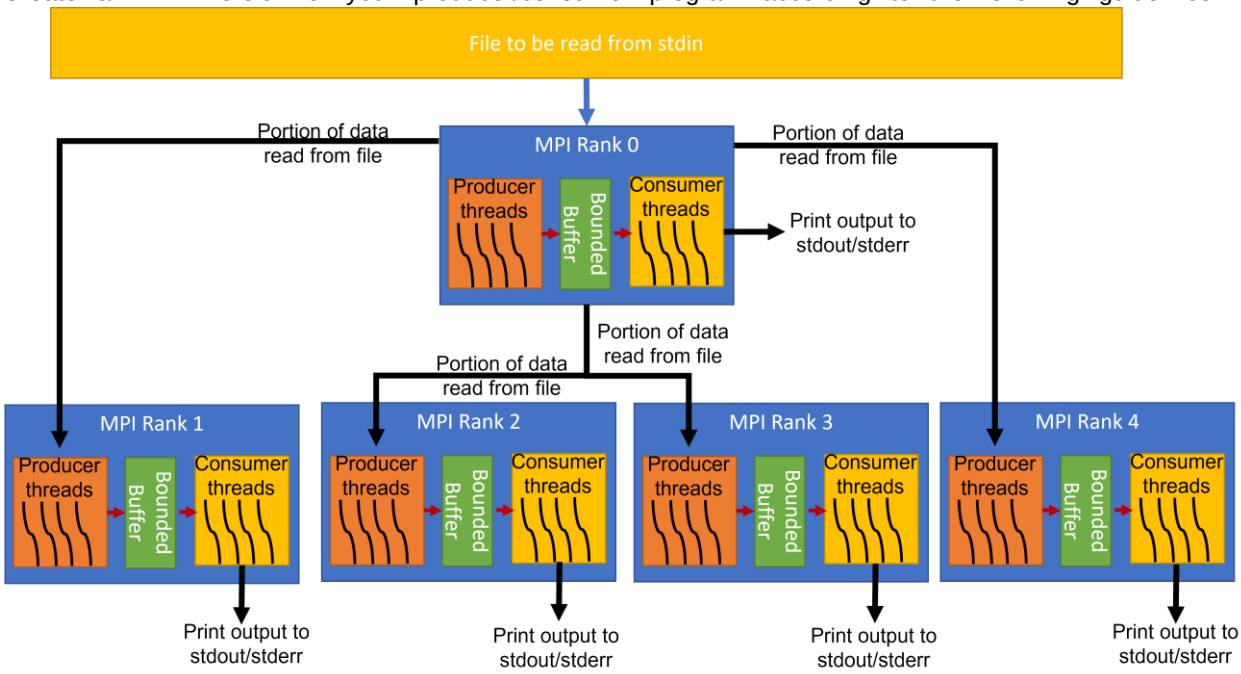
Assignment 4: Hierarchical Parallelism

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

Background and Problem Statement

Distributed computing both opens the possibility of an entire network of computers collaborating simultaneously on a single application while also enabling multiple levels of concurrent parallelism. Using MVAPICH2, an open-source standard MPI implementation, combined with global shared memory threading techniques we will implement a hierarchical parallel version of producer/consumer.

Create an MPI version of your producer/consumer program according to the following guidelines:



- Use a total of 5 MPI processes. One MPI process per compute node.
- Follow the guidelines from previous labs, where applicable, unless otherwise noted in this assignment.
- Your rank 0 MPI process will be your master MPI process. This process will:
  - Read values from the input file
  - Distribute the read values to other processes through MPI operations.
- All ranks (0 – 4) will perform parallel transform computations
  - Using OpenMP, create producer and consumer threads based on command line input.
    - The number of producers and consumers will be the same among all processes.
  - Producers will insert received values into a data structure.
  - Consumers will extract values from the data structure and do some “work.”
  - Report the MPI rank and thread id of the producer/consumer which performed the insertion/extraction.
- Essentially the objective is very similar to the OpenMP assignment, except now there are multiple processes, and the work must be split between these processes.

Program Requirements

- Start with the skeleton program provided and then re-architect it as needed.
- Your final source code MUST be named prod_cons_mpi_hybrid.c
  - What to do if vim gives you errors about the swap file being present?
  - When your vim crashes without closing things properly, it leaves behind a hidden .swp file. In your case, it is probably named. prod_cons_mpi_hybrid.c.swp.
  - Please note the dot in front of the file name. If you delete that swp file, you will not get the error.
- Please find the optimal value of producer and consumer OMP threads that yields the best performance.
- You program should support having multiple “producer” threads which will populate a shared data structure like a stack or a queue.
  - The simplest way to implement this would be with a stack with one pointer as below.
  - But 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.
    - int buffer[MAX_BUF_SIZE] = {0};
    - int location = 0;
- 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 to simulate this work.
    - 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.
- You may perform your output from either a serial or a parallel region within your program.
  - Output should be printed on stdout.
  - 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.
- Make sure the final files you submit has the best values for the number OpenMP threads hardcoded in the code.
  - The grader WILL NOT try out your program with different number of OpenMP threads.
  - They will run your program without any command line parameters and use the time it gives when they run it to grade your assignment and give you points.

- You should try to minimize the number of changes needed.
  - We will be using the “wc -l” command to count the number of lines in each program.

Input Files Provided

The following file have been provided to you. All files are available on Owens in the /fs/ess/PAS2661/Assignment-5 directory.

- prod_consumer_orig.c – The skeleton file
- 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
- Input files – longlist and shortlist

Testing Instructions

- Use the MPI C compiler (“mpicc”) only obtained after loading the default ICC-based mvapich2/2.3.3 module. You should see the following modules available by default intel/19.0.5 mvapich2/2.3.3
  - 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
- Note that the Owens HPC system only has a limited number of compute cores on each node. If you exceed this, you will see very poor performance as different threads will be competing for CPU resources. Please keep this in mind while choosing the number of OpenMP threads.
- 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> --ntasks-per-node=28 -A PAS2661 -p <partition> --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 5 --ntasks-per-node -A PAS2661 -p debug --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

- Note: you do not actually need to ssh into the allocated nodes to run ./run_script.sh. The MPI launcher will launch the processes on the correct nodes automatically.

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 (preferably bar graphs) in your report
  - One graph shows the execution time of your solution when processing the longlist
  - 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 series each for number of producers
  - Graph datapoints for 1, 2, 4, and 8 consumers
  - Also include lines for 1, 2, 4, and 8 producers
  - Use a log scale on the X-axis
    - i.e. you should have 1, 2, 4… on the X-axis, each marker evenly spaced
  - Use a linear scale on the Y-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-mpi-lab" and place all necessary files for submission there.
- Submit your cse5441-mpi-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 "Lab4-Assignment"
  - 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…>
-rwxr-xr-x 1 osc0577 PZS0622 155864 Apr 14 19:33 a.out
drwxr-xr-x 2 osc0577 PZS0622 4096 Apr 14 19:35 cse5441-mpi-lab
-rw-r--r-- 1 osc0577 PZS0622 6393 Apr 14 16:40 longlist
-rw-r--r-- 1 osc0577 PZS0622 11165 Apr 14 19:22 prod_cons_mpi_hybrid.c
-rw-r--r-- 1 osc0577 PZS0622 8051 Apr 14 21:44 prod_consumer_orig.c
-rwxr-xr-x 1 osc0577 PZS0622 4476 Apr 14 19:33 run_script.sh
-rw-r--r-- 1 osc0577 PZS0622 492 Apr 14 16:40 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]

>>> Lab4-Assignment

Enter the absolute path of the directory which includes all the files and press [ENTER]

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

>>>/users/PZS0622/osc0577/CSE-5441/MPI/Homework/cse5441-mpi-lab

Your assignment Lab4-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_cons_mpi_hybrid.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
            1. If you have validation errors, you will lose a minimum of 25 points.
            2. If the program does not run for the grader, a minimum of 25 points will be deducted.
                a. 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.
        ii. 20 points for the quality of the implementation
            1. 10 points will be deducted if the code does not pass a linting test using cpplint
            2. Students are encouraged to use cpplint on their submission code before submitting
            3. See https://github.com/cpplint/cpplint for how to setup and use cpplint
    b. If you forget to submit your program 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