MONASH University

FIT3143 Lab #4 (Week 7)

MESSAGE PASSING INTERFACE (MPI) II

OBJECTIVES

- Design and analyze distributed parallel algorithms with Message Passing Interface (MPI)

MARKS

- The lab is worthed 5 of the unit final mark.

INSTRUCTIONS

- Before class: Attempt all the tasks before class. Make sure you read the marking rubric and the lab pre-readings!
- During class:
  - You will be given 15 minutes to prepare your code for the demonstration period.
  - After you finish the preparation, you must submit all the files to Moodle.
  - During the demonstration period, the teaching staff will ask you questions on your code submission(s).
  - Late submission: 5% penalty (per day).
- Marks will not be awarded if you skip the class, do not make any submissions, or make an empty submission to Moodle (unless a special consideration extension has been approved).
- You are allowed to use search engines or AI tools to search for information and resources during pre-class preparation. However, search engines and AI tools are not allowed during the code presentation period.
- Always double check all your codes & answers.

ASSESSED TASKS

- Task 1 (70%)
- Task 2 (30%)

LAB ACTIVITIES

Task 0 – Getting used to manual pages (Not assessed)

Open the manual pages https://docs.open-mpi.org/en/v5.0.x/man-openmpi/index.html

Make sure it sits comfortably in your browser and you can quickly access the API descriptions as quickly as you can!

Task 1 – Prime Search using Message Passing Interface (70%)

Let's revisit the prime search problem which you previously worked on in Week 3 (Lab #2).

In week 3:

> "You are required to write a **serial** C program (Task 1 in Lab#2) to search for prime numbers which are strictly less than an integer n, provided by the user. The program should print to the standard output (screen) a list of all prime numbers found.
>
> Example: For instance, if the user inputs n as 10 on the terminal, the prime numbers being printed on the screen are: 2, 3, 5, 7.
>
> You are also required to implement a parallel version of your serial code in C utilizing **POSIX Threads** (Task 2 in Lab#2) and **OpenMP** (Task 3 in Lab#2), by designing parallel partitioning schemes to distribute the workload and implement in C."

In this week's lab activity, we will continue with the same old prime searching problem in Week 3. You are required to implement a parallel version of your serial code in C using **Open Message Passing Interface** (Open MPI).

Coding requirements:

- The root process is required to prompt the user for the integer n value (e.g., n = 1,000,000 or higher).
- After obtaining the value n from the user, the value is required to be disseminated to all the other MPI processes.
- Each process (including the root process) will be tasked with a share of workload to compute the prime number. You will need to design your own workload distribution.
- After the result is computed by all the processes, the root process will print the final results to the standard output (in sorting order).

After you implement your program:

★ Make sure you execute and test your compiled program with different numbers of MPI processes using your machine with Docker, a Linux virtual machine, or a native Linux machine.

a) You will need to measure the overall wall-clock time required to search for prime numbers strictly less than an integer n and compare your results against the serial version (in Week 3), and compute the speed-up by your parallel implementation. Make sure you also include information on the number of processors (cores) available in your machine and the number of processes you have created. Will increasing the number of MPI processes always yield larger speed-ups? Tabulate your results (or plot a chart) with different numbers of MPI processes to analyze the parallelization performance and the speed-ups. Write down all your answers/tables/graphs in your notes.

b) You will also need to compare your results and speed-ups against the POSIX thread or the OpenMP version (in Week 3). Will the same number of processes and threads produce the same speed-ups? Test with varying numbers of processes/threads, and tabulate your results (or plot a chart) with different numbers of MPI processes & threads to analyze the parallelization performance and speed-ups. Write down all your answers/tables/graphs in your notes.

c) In your notes, you need to provide <u>at least two reasons</u> on why the speed up may not be the same as the theoretical speed-ups. You need to also provide <u>at least two reasons</u> on why Open MPI is faster/slower than your POSIX thread/OpenMP implementation in Week 3.

Submission:

1. Please submit your Open MPI code in this task to Moodle in a file “task1_openmpi.c”.
2. Please submit your report/notes in this task to Moodle in a file “task1.pdf”.
3. If you have other codes you want to submit, please submit all these files with “task1_” prefix.

Task 2 – Prime Search using Message Passing Interface on CAAS (30%)

Make sure you are familiar with the CAAS materials/resources in Week 5 of the Moodle page before attempting the task. Make sure you also review the CAAS tutorial video and to try out the sample source code and job scripts too.

In this task, you are required to re-compile and test:

i) Your serial code on Prime Search;
ii) Your POSIX thread/OpenMP version on Prime Search, and;
iii) Your Open MPI version on Prime Search

in the CAAS platform. Again, you are required to measure the time taken to complete these tasks (with varying number of threads [for POSIX thread/Open MP versions] and MPI processes [for Open MPI version]).

Experimental procedures:

Start with x = 2,

a) Measure the time taken and speed up with x number of threads on a single compute node for your POSIX thread or Open MP version.
b) Measure the time taken and speed up with x number of MPI processes on a single compute node for your Open MPI version.
c) Measure the time taken and speed up with 2x number of MPI processes on two compute nodes (i.e., x number of MPI processes on each node/machine) for your Open MPI version.
d) Double x until 16, and re-test!

Again, tabulate your results (or plot a chart) with different numbers of threads/processes (i.e., x) to analyze the parallelization performance and the speed-ups. Will CAAS be running faster on the same number of threads/processes against your machine? Will CAAS giving you a better speedup (against your machine) when the number of threads/processes become large? Why? Make sure you write down all your answers/tables/graphs in your notes.

Note: Since you are submitting jobs in a non-interactive environment (the usual cluster environment), you will need to slightly modify the code to make sure the code no longer asks the user for the value of *n* at runtime. Instead, the value of *n* should now be passed as a command line argument into the application.

Task 3 – Introduction to Network Bandwidth & Latency (not assessed)

For this task, you are required to perform Ookla Speedtest (https://www.speedtest.net/) on your desktop/laptop (WiFi/LAN) and smartphone (with ether 3G/4G/5G mobile network).

Details:

- Measure the bandwidth and latency to 5 different servers in the Melbourne area and record the results.
- Perform this test at four different times covering different parts of a day (e.g., 11 AM, 2 PM, 5 PM and 8PM).
- Tabulate the results and compute the mean μ and standard deviation σ of the latency at each of the four different times.
- Compare the measured mean latencies between your desktop/laptop and your smartphone. Explain any observed differences in terms of the service and protocols employed.
- Estimate the worst case latencies between your desktop/laptop and your smartphone for a 95 percent confidence interval using the $\mu + 2\sigma$ approximation, and explain the observed differences;

Task 4 – Introduction to Ping Test (not assessed)

For this task, you are required to perform ping tests to different NTP servers (i.e., servers to tell you the time). Use ping command in your machine and send 10 ping requests (ping -c 10 <target>) to NTP servers across the world:

- Perform a ping test to NTP server in Australia: 0.au.pool.ntp.org
- Perform a ping test to NTP server in North America: 0.north-america.pool.ntp.org
- Perform a ping test to NTP server in Europe: 0.europe.pool.ntp.org
- Perform a ping test to NTP server in Asia: 0.asia.pool.ntp.org

Report the round-trip min, average, max, and standard deviation results.

If you are finding a server partner in Australia, North America, Europe, and Asia to host your distributed application serving clients in your location, which region would be the best according to the ping test? Why?

[Note: Sometimes server 0 may not be responsive (or not responding to ICMP requests), you can safely switch to server 1, 2, or 3 for the test, e.g., 2.au.pool.ntp.org]