MONASH University

MESSAGE PASSING INTERFACE (MPI) II

OBJECTIVES

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

MARKS

- The lab session is worth **6 marks** of the unit's final mark.

LAB INSTRUCTIONS

1. Preparation is recommended and required for this Lab session. Please **DO NOT** plan to complete the Lab without any preparation/understanding.
2. Students must submit their answers and codes via Moodle in the required format (report in PDF and codes in C/C++ files) before the end of the allocated class. A **late penalty** (5% per day) will apply if you do not submit the required materials on time!
3. The purpose of these biweekly Lab sessions is for students to get an opportunity to test their understanding of the lecture/pre-reading material with a low cost in lost marks if they have not understood the material, rather than not understanding the material.
4. Students should start working on the Lab tasks prior to starting the class.
5. Students' (or Team's) codes and answers will be reviewed and assessed in a coding presentation, peer review, or question-and-answer format.
6. For **team/group presentations**, marks will be allocated for:
    - The correctness of the codes or results
    - The quality and clarity of the presentation
    - The correctness of the explanation during the presentation

   The presentation content can include slides or a format that is suitable. Optionally, marks could be awarded to students capable of asking quality and insightful questions about their peers' presentations. Students can also opt to demonstrate the code, results and observations using a document in lieu of a presentation slide. Nevertheless, the quality/correctness also applies to a demonstrated work, submitted code and

documentation containing the results, analysis and observations.

7. For **individual interviews/Q&A sessions**, marks will be allocated for:
    - The correctness of the verbal answer
    - The quality of the verbal answer

   Try to focus on the most important and critical idea in the answer. Marks will be awarded on the student's ability to explain their ideas with reference to the submitted code or documents in a concise, focused and accurate manner. Answers with errors, inaccurate or lengthy explanations will lose marks. Showing little to no understanding of the code, associated libraries and submitted results/analysis will also result in losing marks.
8. Always make sure to read the marking rubric.
9. Marks will not be awarded if you skip the class, or do not make any submissions, or do not make any contributions in the working group/team, or make an empty submission to Moodle (unless a special consideration extension has been approved).
10. You are allowed to use Generative-AI to search for information and resources during the pre-class and in-class preparation period. However, you must declare in your report and upload all the prompt records (in a PDF file).
11. AI tools are not allowed during the presentation period or any oral/coding interview sessions.
12. Certain tasks or activities may be performed in a group/team format. However, students must still submit all the tasks (required files) individually in Moodle.
13. All submitted files should include students' names, ID and Monash email addresses.

LAB ACTIVITIES

Students are required to form teams as advised by the teaching staff. At the Clayton campus, each team should consist of approximately 4 to 6 students, while at the Malaysia campus, each team should consist of 2 students. Malaysia campus students who wish to work in larger teams can inform their respective lab tutor.

- Preparation period: You will have a **maximum of 40 minutes** (per team) to work on the Tasks during the session. You should work with your team members to complete the tasks before the start of the session, in particular any tasks labelled with [Pre-class activities]. Private coding repositories (e.g. GitHub) are recommended to share and work with your team members.
- Presentation Period: For each team, you are required to present or demonstrate your code, results and observations during the lab session. Additionally, documents or slides containing an explanation of the code design, results and analysis should be submitted as part of the presentation or lab demonstration. If a team is not comfortable presenting the work in person during the lab, the team can opt to record and submit video file(s) to Moodle prior to the lab session. The lab tutor will play the video in the class and review it during the lab session. However, all team members should be present in the lab session for the Q&A period with the lab tutor.
- Q&A Period: Each team member will be asked one to two questions based on the submitted and presented work.
- The overall session per team should be **between 10 and 15 minutes** long, depending on the size of the team.

Submission Checklist

- Core Task - All coding and supporting files.
- Extended Task - All coding and supporting files.
- Documentation describing the code design, results and analysis or observations (where applicable).
    - The documentation can be in the format of slides, docx, PDF or an equivalent format, but choose only one format.
    - Although the marking rubric requires the submitted content to show a deep understanding of the topics with materials, please aim to keep the length of the

slides or documentation to within a 10 to 15-minute presentation duration.
    - Focus on emphasising the design, results and observations that would demonstrate a good understanding of the lab tasks.
- A separate AI declaration (in PDF format), **if not** already declared in the submitted documentation as aforementioned.

Warm-Up Task – Getting used to manual pages

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!

Core Task – Prime Search using Message Passing Interface [Maximum D for overall marks (Demonstration and Q&A)]

Let us revisit the prime search problem, which you previously worked on in Week 4 (Lab #2).

In week 4:

> "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 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 are: 2, 3, 5, 7 (sorted in ascending order). If the value of n is large (e.g., n = 1,000,000), then the list of prime numbers should be printed to a text file.
>
> In week 4, you are also required to implement a parallel version of your serial code in C utilising **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 from Week 4. You are required to implement a parallel version of your serial code in C using **Open Message Passing Interface** (Open MPI).

Note: You will need to 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 to the application.

Before writing the parallel code, you should include a theoretical speed-up analysis of the prime search algorithm. You may select either Amdahl, Gustafson or an equivalent method, but you need to justify your selection of the theoretical speed-up method. Additionally, you should include the fabric time in your theoretical speed-up calculation.

Coding requirements:

- The root process reads the integer n value (e.g., *n* = 10,000,000 or higher) as a command-line argument.
- 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 the workload to compute the prime number. You will need to design your own workload distribution and *explore/experiment with different workload distribution options* to find the best approach. You should design and implement an efficient and balanced workload distribution algorithm.
- After the result is computed by all the processes, the root process will print the final result(s) to a text file (must be in correct sorting order).

After you implement your program:

★ Make sure you execute and test your compiled program with *different numbers of MPI processes and different workload distributions* 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 4), and compute the actual speed-up of your parallel implementation. Make sure you also include information on the number of processors (cores) available in your machine and the number of MPI processes you have created. Will increasing the number of MPI processes always yield higher speed-ups? Tabulate your results (or plot a chart) with different numbers of MPI processes to analyse the parallelisation performance and the speed-ups. Compare your actual speed against the theoretical speed up. Will a different workload distribution improve the speed-up? Will different machines in your team change the speed up drastically? Write down all your answers/tables/graphs and observations in your presentation materials (notes/document/slides).

b) You will also need to compare your results and speed-ups against the POSIX thread or the OpenMP version (in Week 4). 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 analyse the parallelisation performance and speed-ups. Write down all your answers/tables/graphs and observations in your presentation materials (notes/document/slides).

Some questions to guide your presentation/demonstration:

- How does the actual speed up compare against the theoretical speed up?
- Will more MPI processes always increase the speedup?
- How would the workload distribution affect the speed up?
- Will the speed-up results (for the above two questions) be the same for every team member's machine?
- Does the POSIX/OpenMP version provide a better speed-up over MPI or vice versa?

Extended Task – CAAS [Required for HD level for overall marks (Demonstration and Q&A)]

Make sure you are familiar with the CAAS materials/resources in Week 5 of the Moodle page before attempting the task. You can also find more details about CAAS and a tutorial video in this section of Moodle. Please ensure that you have tried out using CAAS before the lab.

In this task, you are required to recompile and test the following using the CAAS platform:

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

You are required to measure the time taken to complete these tasks with a varying number of threads [for POSIX thread/Open MP versions] and MPI processes [for Open MPI version].

Experimental procedures:

Using the *n* value from the preceding task, 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 OpenMP 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 2*x* the 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.

Tabulate your results (or plot a chart) with different numbers of threads/processes (i.e., *x*) to analyse the parallelisation performance and the speed-ups. Will CAAS be running faster on the same number of threads/processes against your machine? Will CAAS give you a better speedup (against your machine) when the number of threads/processes becomes large? Why? Make sure you write down all your answers/tables/graphs in your presentation materials (notes/document/slides).

Reminder: Since you are submitting jobs in a non-interactive environment (the cluster environment), you will need to 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 to the application.

IMPORTANT: The CAAS platform is a job-based system. If many students were to submit jobs simultaneously during a particular time, the submitted jobs may be in pending mode longer. Consequently, you would need to wait longer before the job can be scheduled for execution on the cluster. Therefore, it is important that you complete the **Extended Task** before your actual lab session. Additionally, please do not increase the default *#SBATCH --time=00:10:00* in the job file. You can set a lower time, but once more, please do not increase it.