Lab Exercise 3: MPI Coding

COMP3499 Operating Systems for Engineers
Department of Electrical and Computer Engineering
Wentworth Institute of Technology

Objectives:

- To create a parallel program using MPI.
- To develop and conduct an experiment to measure the speedup of an MPI parallel program running on up to 40 virtual processors.

Note: This lab can be done in pairs or individually. Be prepared to demo the working code to your Instructor if requested.

Instructions:
Refer to the MPI lecture slides and MPI_Document.pdf on Blackboard.

Implementing Monte Carlo Integration Using MPI
In Monte Carlo methods, numerical solutions to mathematical problems are produced by using a large sample of random variables. Monte Carlo Integration therefore approximates integration by using a large sample of random numbers in the following way:

Consider the following integral

$$\boldsymbol{F = \int_a^b f(x)dx}$$

The numerical solution $\boldsymbol{h(x)}$ can be approximated by generating a very large sample of random variables $\boldsymbol{X}$ uniformly distributed over $[\boldsymbol{a, b}]$ and using the formula

$$\boldsymbol{h(x) = \frac{(b-a)}{N}\sum_{i=1}^{N} f(X_i)}$$

The estimate $\boldsymbol{h(x)}$ will converge to the correct solution as the number of samples $\boldsymbol{N}$ grows.

For further information on Monte Carlo estimation and integration: http://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration

Since each sample is independent, the calculation can be easily parallelized.

Write an MPI program that uses Monte Carlo Integration to calculate:

$$\boldsymbol{\int_a^b \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}$$

Guidelines:

- All Worker nodes will each generate its own set of random numbers and produce a value to send back to the Master; therefore, each Worker node would also need to use its own seed.
- Each node should return a single value to the Master node which would then be combined into the final integral.
- Use only the six basic MPI functions for the main functionality: `MPI_Init`, `MPI_Finalize`, `MPI_Comm_rank`, `MPI_Comm_size`, `MPI_Send`, and `MPI_Recv`.
- Use the command-line to pass in the integral bounds ***a*** and ***b*** and number of samples ***N*** – remember only rank 0 can use `stdin`.
- Use `MPI_Wtime` and/or the time reported by Torque to calculate the execution time.
- Your code should print ***a***, ***b***, ***N***, and the calculated integral value.
- Using a single process on a single node, vary the number of samples ***N*** (100, 1,000, 100,000, etc.). The solution to your integral should converge. From this experiment, pick a value of ***N*** to use for the remainder of the experiment –remember ***N*** should be very large. Submit the values obtained for the integrals, and your choice of ***N***.
- The following steps will need to be run first using the `mpirun` command directly from the command line and then using Torque to do batch queuing
  - Using the value of ***N*** found above, plot a graph of the speedup obtained when you run the program with 1, 2, 3, …, 40 processes. Submit your graph and speedup values.
  - Keep increasing the number of processes beyond 40. Does the speedup improve? Why or why not?
  - A note on using batch queuing. If you create 40 processes and run the program from the command line directly the OS decides how many processors your program runs on. If no other user is using `turing` at the time it will likely schedule one process per processor and run all 40 processes in parallel. If there is another user, the 40 processes will still run but on the available processor where processing will be interleaved.

    If you use batch queuing (Torque, in our case), you create 40 processes and ask for 40 processors, the scheduler will wait until all 40 processors are available before your program is run. This way you are guaranteed use of the resources without having to share with another user. The trade-off is that your program may have to wait it's turn to begin executing.

The Lab Exercise Report

The Lab Exercise Report must be submitted as a **<u>single PDF</u>** on Blackboard. It must include:

- Student Information: Name, WIT ID, Course code and name, date, Lab Exercise number.
- Academic Honest pledge as shown below.
  By submitting this assessment, I hereby declare that I have neither given nor received any unauthorized help or used any unauthorized materials during this assessment, and that I have adhered to any additional policies regarding Academic Honesty set by Wentworth Institute of Technology.

  Please sign below to acknowledge this or your assessment will not be graded.

  Student Signature: Date:

- **Part 1 –** Develop an experiment that executes your parallel code for Monte Carlo Integration on the cluster and measures speedup while varying the number of processes **[8mks]**.
    - This must be written as an experiment with aim/objective, methodology i.e. steps you plan to follow.
    - State any assumptions made.
    - The experiment developed will need to pick the value of ***N*** as described above, as well as, measure execution time to calculate speedup as you vary the number of processes and processors without using batch queuing and then using Torque batch queuing.
- **Part 2 –** Conduct the experiment and obtain any values needed for analysis **[8mks]**.
    - Conduct the experiment following your detailed steps from Part 1 to collect data.
    - You need to report ***a***, ***b***, ***N***, and the calculated integral value.
    - Any data collected related to execution times and speedup should be presented in tables along with the necessary graphs.
- **Part 3 –** Analyze and interpret the data **[8mks]**.
    - Analyze and discuss the data.
    - Your discussion should include things such as the following: does the speedup level off? Is the speedup linear? Compare your calculated vs predicted speedup. Why doesn't the predicted speedup match the calculated speedup? Discuss the overheads that will prevent the predicted speedup, including those related to the OS. Does speedup improve above 40 processes? Why or why not? Why can you increase the number of **processes** past 40 even though you cannot increase the number of processors past 40? Compare the results you obtained without using batch queuing to those obtained when you used batch queuing (Torque). Are the results the same? Are they expected to be the same? Why or why not?
- **Part 4 –** Draw a conclusion based on your analysis and interpretation **[4mks]**.
    - Your conclusion should include the optimal value of ***N*** and number of processes.

Upload your final .c file and Torque script on Blackboard as well.