Problem 1.2

Problem 1.2

When executing a parallel program utilizing P P P processors to output the message "Hello from Processor X" to the recipient (who receives these P P P greetings sequentially), the sequence in which processors deliver their messages (i.e., the order of the first, second, and subsequent greetings) exhibits non-deterministic behavior across multiple executions of the same program on an identical computing system. For instance, when executed on a system with P=10 P = 10 P=10, the following output may be observed:

Hello from Processor 3
Hello from Processor 9
Hello from Processor 8
...
Hello from Processor 0

Upon a subsequent execution of the same program on the same system, a different sequence may emerge:

Hello from Processor 0
Hello from Processor 8
Hello from Processor 7
...
Hello from Processor 4

This non-deterministic characteristic can be undesirable and is, however, rectifiable. Should a specific order be expected, it is feasible to implement a parallel program to ensure consistent output. For example, a program can be designed to always produce the following sequence:

Hello from Processor 0
Hello from Processor 1
Hello from Processor 2
...
Hello from Processor 9

Design a program to enable such order and test it on a system with a minimum of 16 processors.

This exercise serves as an example of revealing differences between parallel and sequential programming paradigms.

Project description

This project aims to solve the problem of nondeterministic output in parallel programs. When many processes try to print at the same time, the order of the messages can change each run, which makes the results hard to reproduce. To address this, I wrote an MPI program that makes each process wait its turn before printing. This way, the output always appears in the same order, starting from rank 0 and continuing up to the last process.

Source Code

Github Repository

Algorithm/Pseudo-code

Initialize MPI

Obtain rank and size

If rank == 0:

  Print message

  Send token to rank 1

Else:

  Receive token from rank-1

  Print message

  If not the last process:

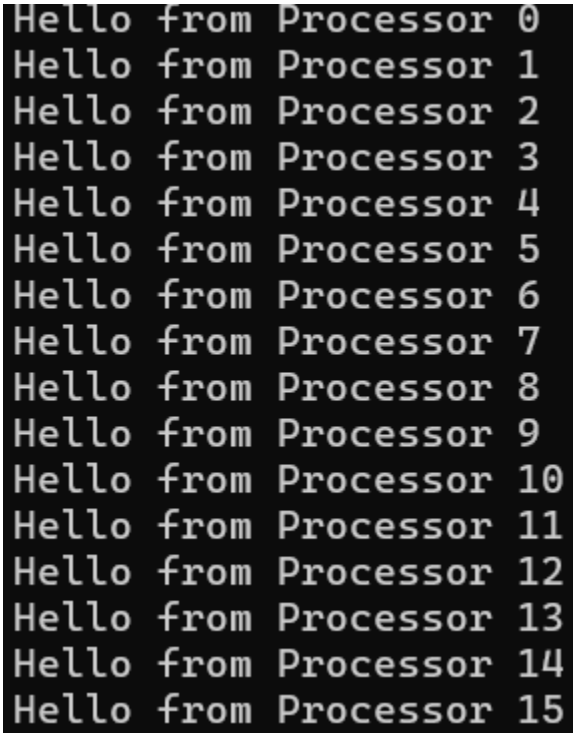
    Send token to rank+1

Finalize MPI

Results

Run on 16 processors with: `mpiexec -n 16 python ordered_hello.py`

Output:



Analysis

The program works by passing a simple token between processes so that each one waits its turn before printing. This guarantees that the output always appears in the correct order, from rank 0 up to the last rank, instead of being jumbled differently on each run. The extra work added by the token passing is very small because each process only sends or receives one lightweight message. As the number of processes increases, the time grows linearly, but since the messages are so minimal, the method stays efficient even for larger runs.