MPI Exercises

J.H.Davenport

15 September 2020

Notation 1 *A number in bold, e.g.* **1***, means the process whose rank is that number, e.g. process 1.* $\forall$ *means all processors.* N *is the number of processors in* `MPI_COMM_WORLD`*, so* **last** *is the process whose rank is* $N-1$*.*

Some exercises are optional, depending what skills you already have/want to learn.

1 Point-to-point

1.1 Ping Pong

1. Write a program in which two processes *repeatedly* pass a message back and forth, i.e. loop over the following steps.

   (a) **0** sends some data (i.e. an array of floating-point numbers or integers) to **1**.

   (b) **1** then sends the same data back to **0**.

   Use print statements to check that the data is being transferred in the correct way and when you are happy, comment them out.

   It is easy to get confused if you don't start each print statement with the rank.

2. Time how long this takes (you will need quite a large repeat count to get a meaningful value — I don't trust any time under 0.1sec.

3. What happens to the time taken if you increase the amount of data sent?

Opt. Plot a graph of amount of data versus time taken.

1.2 Ring

We will think of our processes as being connected in a ring, with **i** connected to $\mathbf{i+1}$ and $\mathbf{i-1}$ only.

4. **0** sends a timestamp to **1** [i.e. an array of one time value]. **1** then send an array of two timestamps (the one it received atndits own) to **2**, and so on to **last**, which adds its own timestamp and send an array of N timestamps to **0**. Print the timestamps out. Run the job more than once — how consistent are they?

5. Keep the original program, and make a modified version that does **i** to **i'** where $i' = i + 5 \pmod N$ (the remainder when you divide 5 by N: N had better not be a multiple of 5).

6. Now merge the programs so that the passing from question 4 happens first, then that of question 5. Not surprisingly, this takes roughly twice as long (at least for large N) as the individual parts.

This one is probably much more difficult. I suggest you work in pairs or groups trying to design a solution. What often works is that one person proposes a piece of design, and the other(s) try to shoot holes in it.

A "let's use `MPI_Waitany`".

B "What happens if two messages arrive simultaneously?"

7. Instead of doing question 5 after 4, overlap the two. This will require non-blocking calls.

8. Assume $N = 2^k$. For every odd nmber l between 1 and $2^k - 1$, assume we have a comunication task as in question 5. Overlap all these as in the previous question.

2 Collective

11. Compute $\sum_{i=1}^{N} i$ by having processor **i** return $i+1$ [remember ranks start at 0] in a call to `MPI_Reduce`, and **0** collects these and prints the sum.

12. Compute $\sum_{i=1}^{N} i^j$ for $j = 1 \ldots N$ by having processor **i** return a vector of N numbers $(i+1), (i+1)^2, \ldots, (i+1)^N$ in a call to `MPI_Allreduce`, and then processor j prints the sum $\sum_{i=1}^{N} i^j$ (saying which processor is printing it!).

13. Write a program to count the number of prime numbers and to find the highest prime number within a specified range. It would be sensible to initially specify a range for which the results can easily be verified.

    (a) Input the lower and upper values of the range of numbers in which to perform the search. If necessary, broadcast these numbers to all processes.

    (b) This interval must be divided fairly amongst all the processes.

(c) Each process must search through its allocated values, testing them for being prime. It must count the number of prime numbers it finds and keep a record of the highest one.

(d) Using two reduction operations, find the total number of prime numbers in the specified interval, as well as the largest one. Output these values.

How well load-balanced is your algorithm? Insert timing calls to check this and, if necessary, improve the load-balancing.