Parallel Programming: Worksheet 2

Exercise 1: Non-blocking communications

Write a program in which each process must send data to two other processes and receive data from however many they are assigned to receive from. Each process should randomly decide the two processes that they are going to communicate with (remember to make sure that these are different from one another and not the current process.

Next, have a set of communications between every process that allows the processes to tell the other processes why they will be communicating with (you can send a single boolean variable from every process to every other process.

Once the processes know who they will be talking to do 10 rounds of communication between these processes (each process should be sending data to 2 process, but each will be receiving data from different numbers of processes, determined in the previous step). You can decide what data to send, but write something to stdout to show the communications that are occurring. Remember to increment the tag between communication rounds to prevent any race conditions.

Exercise 2: Doing work while waiting for communications

Rewrite the example that I gave earlier for communicating between all processes (i.e. every process sends data to every other process), but this time send an array of 10,000 doubles so that both setting up the data and the communication takes a lot longer.

Write a function that does some calculations (you can decide what this important extra work is!) and repeatedly call this function until all the communications for that process have been completed. Write to the screen how many cycles of this task were completed by each process.

Exercise 3: Non-blocking communications with neighbours

Divide your *p* processes into a grid of size *m* x *n* (try and ensure that *m* and *n* are integers that are as close to one another as possible. E.g. if *p* is 9, *m* and *n* should both be 3, while if *p* is 12, one should be 3 and the other 4). On this grid calculate an *i* and *j* index for each process such that $id = i + m\,j$.

Each process should communicate with the processes that next to it vertically, horizontally and diagonally (i.e. processes in the middle of the grid will communicate with 8 neighbours, those on edges with 5 neighbours and those in the corners with 3 neighbours). Send the neighbours the source's *id* as well as their *i* and *j* coordinates and display these

Communication patterns similar to this are also quite commonly used. You might encounter it if you are doing peer to peer communication for a domain decomposition problem (more on this later).

Workshop Exercise 2

Rewrite the workshop exercise from the last worksheet to use non-blocking sends and receives. As some processes might take longer than others to complete their tasks, instead of dividing all the tasks upfront, send only a portion of the data out to each process, with the other processes send these results back at the end. A good cycle for zero node is to use MPI_test to check if each process has sent back data. If they have they can be sent more data to process and if no processes have sent back data the zero node can do a couple of the calculations itself while it waits. Send an empty list of data to a process to let it know that it can finish and exit.

These type of communications are quite common in master-slave type applications for solving independent parallel tasks. Later in this course we will use communications like this to solve real problems.