Parallel Programming: Worksheet 3

Exercise 1: Gather and Scatter Operations

Have every process calculate a single random number between 0 and 100. Later on we wish to communicate based on the order of these numbers (you don't need to implement this communication). You now need to ensure that every process knows what this order is. In other words, the process with the lowest random number should be first in the list, the one with the second lowest second etc. (if they have the same number you should order them according to the process id).

One tactic to do this is to gather all of the random numbers onto a single process, sort the list while keeping track of the id corresponding to each number, using any sort algorithm you choose (bubble sort is very easy to implement, while quick sort is much quicker, but a bit trickier to do. You could alternatively just rely on the sort algorithm from std, though you will need think about getting the indexes in order). Scatter the list of the id order back to all the processes (i.e. send each process their location in the sorted order).

Exercise 2: MPI_Reduce

In simulations it is quite common for all processes to calculate a number, but then require that every process know, for instance , the smallest or largest number calculated. An example might be calculating the time step based on the element size and fluid velocity in a finite element simulation, where the maximum time step calculated on each process is different, but the actual time step used must be the smallest one calculated on all processes.

Have each process calculate a random "time step" and then use a combination of MPI_Reduce and MPI_Bcast to ensure that every process knows the same smallest time step.

Exercise 3: MPI_Allreduce

Use MPI_Allreduce to combine the MPI_Reduce and MPI_Bcast in the previous example into a single operation.

Exercise 4: MPI_Iallreduce

Modify the code from the previous exercise to use a non-blocking MPI_Iallreduce rather than the blocking MPI_Allreduce. In this example there is no real advantage to using a non-blocking collective communication, but in other situations it may be worth doing so in order to allow other calculations to be carried out while waiting for the communications to complete (or to have multiple reduce operations occur at the same time).

Workshop Exercise 1 – Parallel Dartboard Algorithm for finding π

There are many problems which can be solved using a Monte-Carlo algorithm. A Monte-Carlo algorithm is one in which a sequence of random numbers are used in calculations, with the average and/or distribution of the results converging on the correct solution as the number of iterations increase.

While it is not an efficient way of doing so, we can use a dartboard algorithm to calculate the value of π. To do so choose a random *x* and *y* value each within the range of -1 to 1. Calculate the distance of this point from the origin. If we do this a large number of times the fraction of points that are within a distance 1 of the origin will be π/4.

To speed up the convergence have all the processes carry out these calculations and after a sufficient number of evaluations use a reduction operation to pull the results calculated on each process back to process zero and report the calculated value of π.