Parallel Programming: Worksheet 7

Exercise 1: Parallel Efficiency of a Master/Slave program

In this task I wish you to investigate the parallel efficiency of a program. A good one to test would be the second version of the Worksheet 2 Workshop Exercise (the reason I suggest the second version is that it will run on a single core as well). Alternatively, you can test any of the programs that you have written, though ideally choose a problem where you can vary both the size of the problem and the number of processes used (and which will run on a single core so that you can estimate the serial cost without writing a dedicated serial code).

You will need to add some timing code to you program. To be fairer in your analysis your code should ideally not write too many couts within the portion of the code that you are timing (or any at all – In fact cout should generally be kept to a minimum in parallel codes unless debugging due to the communications involved).

As the number of cores on your laptops will be quite limited, test this efficiency using the HPC system. Note how the efficiency changes as you go from using a single node to using more than one node.

Use the timings to calculate the parallel efficiency and the speedup ratio as a function of the number of cores. How does this change with the size of the problem that you are testing?

Note that because you are using random numbers as your input the time taken will change from run to run even for the same size of problem and number of cores. There are two ways around this problem – Either run the code multiple times and use the average time taken or you can give a set number rather than the time as the seed for the random number generator so that it always generates the same set of random numbers.

Workshop 7: Improving the Efficiency of a Program

The relative importance of communication vs computation will have a strong bearing on the efficiency of a program. This will depend on the algorithm used, its implementation and the hardware available.

Test how the efficiency of the code changes with the size of the chunks sent out to each process. Use the efficiency to optimise your code.

You will probably find that large chunks work better for this particular problem. This is because applying the GCD algorithm to these lists of data will take similar amounts of time. Can you think of an algorithm where the amount of time take might vary far more strongly with the size of the problem? A category of problems that will take dramatically different amounts of time depending on the size of the problem are those known as NP hard and which include things like the travelling salesman problem.

Try this optimisation with a problem that varies more strongly with the size of the problem than GCD and test it with values that vary over a smaller range (you won't be able to solve most NP hard problems using brute force for problems much beyond ten items in size in a sensible amount of time as the computational cost increases factorially).