# PH-M31 Assignment Question 5

**Tom Bourton**

E-mail: `701329@swansea.ac.uk`

## 1. Question 5 - Run the parallel code on more than two cores. Perform a weak and a strong scaling test. Explain your results.

(See Code_Scripts directory in ZIP for scripts used to run scaling tests)
(See Graphs directory in ZIP for scaling results graphs)

We can perform scaling tests which will give an indication as to how efficient the code is as and how well it lends itself to being run on many processors.

The two main types of scaling tests are 1)Strong Scaling, whereby we keep the global size of the problem constant and vary the number of processors, 2)Weak Scaling where we vary the global size of the problem, but ensure that the local size of the problem per processors remains constant.

In the strong scaling test we expect to see computation time decrease with the number of processors, until a minimum is reached, from there the time is then expected to increase as latency begins to dominate and the time required to send messages becomes larger than the time required for the processor to do it's computation. In the Weak scaling we expect to see an approximately flat, constant line as we vary the number of processors.

Strong scaling tests have been performed on both the Linux Lab Cluster and the Glamorgan Tier 1 Supercomputer from HPCWales. In the strong scaling test we keep the lattice size constant and vary the number of processors and measure the time taken to run the code. As in the code there is a test which checks whether or not the Lattice size is divisible by the number of processors, therefore a choice of lattice size must be chosen such that it is a multiple of many processes numbers, but must also be small enough that the tests can performed in a sensible amount of time, in this case 480 was chosen as it has many multiples in them region between {2...240} (because in the Linux Lab we have $\approx$ 30 PC's, each with 4 processors + hyperthreading = 8 effective processors).

The code was modified to compute the computation time that is taken for the program to run by recording the start time and finish time using the clock functions within C, as this is the variable we are trying to measure.

To realise the strong scaling test a script was created that would take a list of number of processors and then run the same code with the same lattice size at each number of processors and the output was sent to an individual out file.

The results for computation time were plotted against the number of processes for both the Linux lab cluster and the Glamorgan Supercomputer. The Linux lab results show a steady decrease in computation time as the number of processes increases until a minimum is reached at around 16 processors which appears to be a "sweet spot" on this configuration. The general trend is then an increase to an approximately constant value as the communication time between processors begins to dominate over the computation time. It is odd that there occurs a peak at approximately 40 processors, this may either be because in the Linux lab cluster it is hard to ever be 100% sure that the PC's are dedicated to only the specific job being run, it is possible they may have been accessed by others which could cause an extended computation time for that particular number of processors, this could be verified and removed by running the strong scaling test many times and comparing the runs. It could also be that this number of processors is an "off resonance" configuration, just as the opposite appears to occur at 16 processors.

The results for the strong scaling test on the Glamorgan Supercomputer was also plotted, we see a exponential shaped decrease in computation time, however unlike the Linux Lab cluster we don't see a minimum reached, this is most likely because the Glamorgan Supercomputer uses 40Gbps infiniband interconnect as opposed to the Linux labs which use standard 1Gbps Ethernet connections, therefore latency is not as big a problem on the Supercomputer so we do not see a minimum reached. However the most efficient use of processors is at approximately 50 processors for this configuration, as after this we only see a small decrease in time for a large increase in the number of processors.

In The Weak Scaling, the code was edited so the line which is sizenp=size/nprocessors, this effectively sets the number of columns (the length) that we will have in the lattice, so the size part of this was fixed to a constant number, therefore we get the same number of total length our array each run, therefore we are then free to linearly scale up the number of rows with the increase in the number of of processors while keeping the number of messages sent per processors and the size of the messages the same for any run. To do this a script was created similar to the strong scaling which will increase the number of processes, but this time it will also a set a new lattice height size defined as L=nproc*const.

The script was run on both the Linux cluster and the HPCWales Glamorgan Supercomputer, both results were plotted for computation time against the number of processors, it was expected that the time should stay approximately constant with the number of processors. However the actual observed fit is approximately linear, which is obviously not what is expected to occur. One possible reason for this is possibly that the statistical analysis is run serially, so the workload on one processor is increasing as the lattice height does, however this does not explain such a large increase in computation time as taking the averages of a large set of data is not as huge amount of extra processing time.