Code Overview and Notes

Here I have used peer to peer communications for optimal processor scaling, using a class based structure to store all the functions and data regarding to a single processor. Some key features include:

* Adaptive domain partitions which divide a rectangular input domain amongst any number of processors such that the internal boundaries are smallest (i.e. smallest total messages per iteration) with any remainders distributed amongst the sub-domains to distribute workload
* Use of an MPI datatype for row and column of each sub- domain to allow for messaging between processors without any additional copying of data
* Use of a \_config.txt file to store the input dimensions, number of partitions and time taken for the run for post processing and analysis
* Use of command line inputs for easy looping on HPC, the exact inputs are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| width (int) | height (int) | Iterations (int) | periodic (bool) | results directory (string) |
| (no input validation so be cautious) | | | | |

* Results directory is where all data is saved. These are labelled “iteration-processor.csv” so no meta needs be stored in the file itself. All saves are written though a stringstream to reduce time writing directly to a file.
* Periodic is implemented though use of self-messaging with a destination ID to copy data across without extra complication in the code (note MPI never sends a message so this process is not particularly slow), hence the only additional code is changing the “find\_neighbours” function
* #define statements are used to control code functionality
  + to\_print and to\_print\_all controls the amount of printing output
  + synch uses MPI barrier to synch every processor at each iteration and setup stage
  + CX1 adds writing run-time to a times.csv file and disables non-essential functions that are not compatible with the gcc compiler
* Post processing is done in python by first compiling each iteration to a single csv file, then animating that and saving as an embedded html file

Validation and Animations

To validate the code on both non-periodic and periodic domains the use of gliders initialised in each domain to see the cross all required boundaries on non-regular grids is used. These can be seen in the animations of the post processed output files. The following html files within the result directory (and the raw results in their respective directories in the same locations) show this:

* test\_nonsq\_periodic
* test\_nonsquare
* test\_periodic

Run-time analysis

Runtime results are collected with up to 128 processors across 4 CPUs on the CX2 HPC using square grids ranging from 100-400 wide to ensure relatively short run-times on low processor number comparisons. To ensure that grid distribution does not play a factor only square sub-domains are used by only selecting a square number of processors 22– 122. Each All runs are repeated 50 times and averaged.

The times are measured from after MPI has initialised to when processor zero has finished, it includes the initialisation of GOL\_grid and saving of each iteration. As this implementation uses peer to peer communications, hence the only serial part of the code is on processor zero for timing and creating the \_config.txt which is virtually negligible to the run-time, so we expect good scaling with number of processors.