ACSE 6.3 – Conway's Game of Life (Parallel with MPI)

Richard Boyne

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 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 subdomains to distribute the 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 is stored in the file itself. All saves are written through a string stream to reduce time writing directly to a file.
- Periodic is implemented through the 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 runtime to a times.csv file and disables non-essential functions that are not compatible with the gcc compiler

Post-Processing

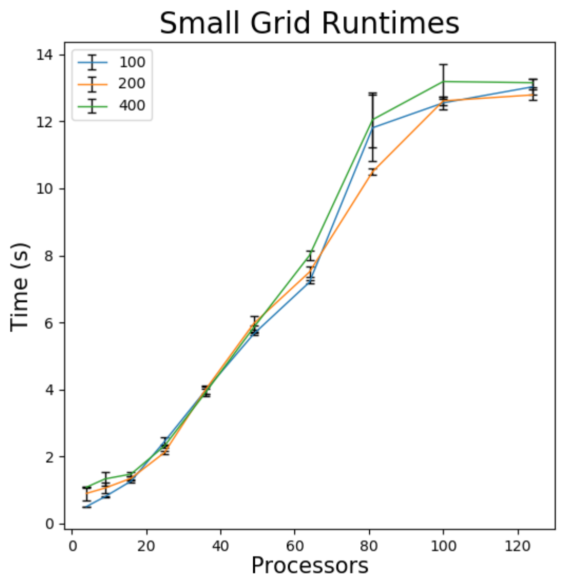
Post-processing is done in python by first compiling each iteration to a single csv file, then animating that and saving as an embedded html file. An example of running this code is shown post_processing.ipynb within the results directory and an example of some results post-processed are in results/local_random/.

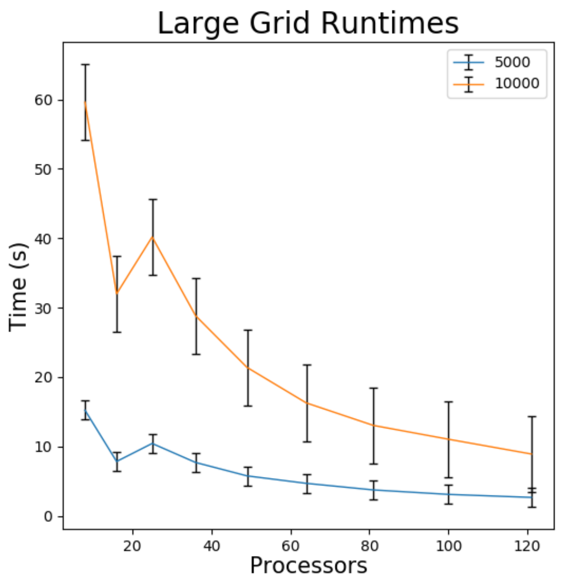
Validation

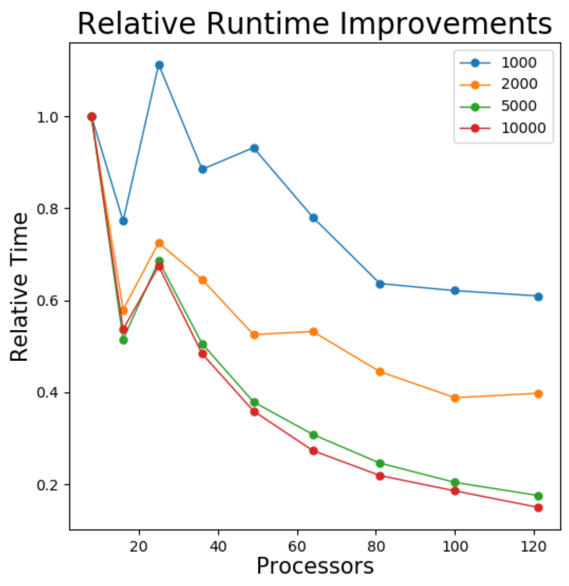
To validate the game of life simulation on both non-periodic and periodic domains the use of gliders initialised in each domain to see the cross all required boundaries is used (though stationary and oscillatory structures were also used to validate single processor simulations). Provided the gliders survive crossing the boundaries we can safely assume the code is implemented correctly. The following html files within the result/figures directory show this, as well as some other simulations.

Runtime Analysis

Runtime results are collected with up to 128 processors across 4 CPUs on the CX2 HPC using square grids ranging from 100-5000 wide with only 50 iterations to ensure reasonable run-time. To ensure that grid distribution does not play a factor only square subdomains are used by only selecting a square number of processors 2^2 to 11^2. Each run is repeated several times (shorter runs are repeated more) and averaged with the standard error calculated as an error estimate. Times are measured on processor zero from after MPI has initialised to just before calling MPI_finalize, it includes the initialisation of GOL_grid and saving of each iteration.







Upper plots are the average runtimes for small and large grids sizes on the CX2 with standard error used as an error estimate.

Below is the normalised run time for different width grids

Looking for small grids we see that increasing the number of processors slows the code's runtime, this is because the communication time is dominating the runtime. This is shown further by how increasing the simulation size on this scale does not increase the runtime significantly.

Looking at the larger system sizes there is a runtime improvement with nodes sizes that is beginning to level off at ~120 nodes. There is a clear spike in runtime at 25 processors, which is possibly due to there being only 16 CPUs per node in the CX2 cluster meaning an increase on communication time above this number of processors. The trend is reinforced by looking at the relative improvements, one always sees a slight increase in runtime after 16 nodes. This is odd however as the settings I chose requested 32 CPUs per node, but the exact chips I was given may not have matched what I asked for (PBS Is mysterious in its ways perhaps). On the other hand, runs with >16 processors show a very smooth exponential decay, which is likely due to peer to peer communications scaling better on large machines.

Looking again at the relative runtimes reinforces the idea improvements in runtime are most significant for larger systems, with the level off being at a better percentage for larger grids. One could try a further analysis with non-square grids to see if this trend directly follows from the ratio of subdomain perimeter to subdomain area (i.e. communication time to processor working time) as these results suggest.