MPHYG002 Coursework 2

David Owen

September 2, 2016

Serial Solution

The serial solution uses a nested vector (`std::vector< std::vector<Cell> >`) to store the grid at each moment in time. The `Cell` class is essentially a wrapper around `bool`, for the sake of clarity. The `Grid` class contains the logic for iterating the game. Unit tests ensure that updates are as expected, that loading data works, etc. The individual iterations of the grid are simply saved as text files with ones or zeros, indicating live and dead cells respectively.

Remote Computation

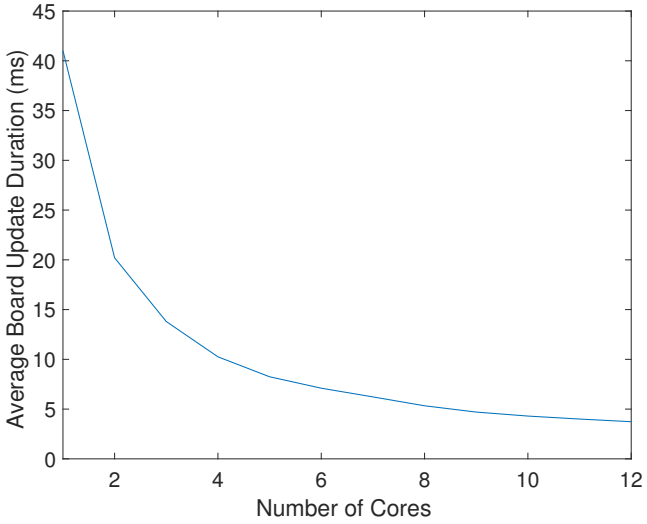
Bash scripts are used for the deployment and qsub submission. By default, any job submission's output directory is timestamped, allowing outputs to be distinguished. If this were to be developed further, it would be wise to also save a copy of the various program options to the output directory. A MATLAB script is used to generate a video from the results.

The attached example video shows a grid updating until equilibrium.

Shared Memory Parallelism

As documented in the README, the OpenMP implementation is in the `openmp` branch. Parallelisation using OpenMP targets the main for-loop, in which the grid is updated. This is easily parallelisable: each update is independent of every other update, so one need only give the threads the board's previous state. They then perform updates in parallel, with implicit blocking until the updated version has been updated by every thread. Preprocessor directives ensure the code remains valid without OpenMP, using `#if(_OPENMP)` to guard the `#pragma`s.

In this particular implementation, when examining larger grid sizes, the main bottlenecks were in file I/O and data transfer. Consequently, to examine performance versus core count, I used std::chrono to measure the time taken for individual iterations. This shows the pattern one would expect: as the number of cores increases, the update duration decreases. However, there are



diminishing returns due to the overhead in thread resynchronisation, such that anything over approximately eight cores makes negligible difference.

Distributed Memory Parallelism and Accelerators

Unfortunately, I ran out of time to implement MPI or an accelerated solution. I should note that a two-dimensional decomposition would would involve splitting the grid into N subgrids, ideally of close-to-equal size. Each subgrid would also need to retain a copy of its "halo" – the values at the edge, needed for updating edge cells. After updating, each thread would request to resynchronise with the master thread (i.e. reassemble the subgrids to make the whole grid).