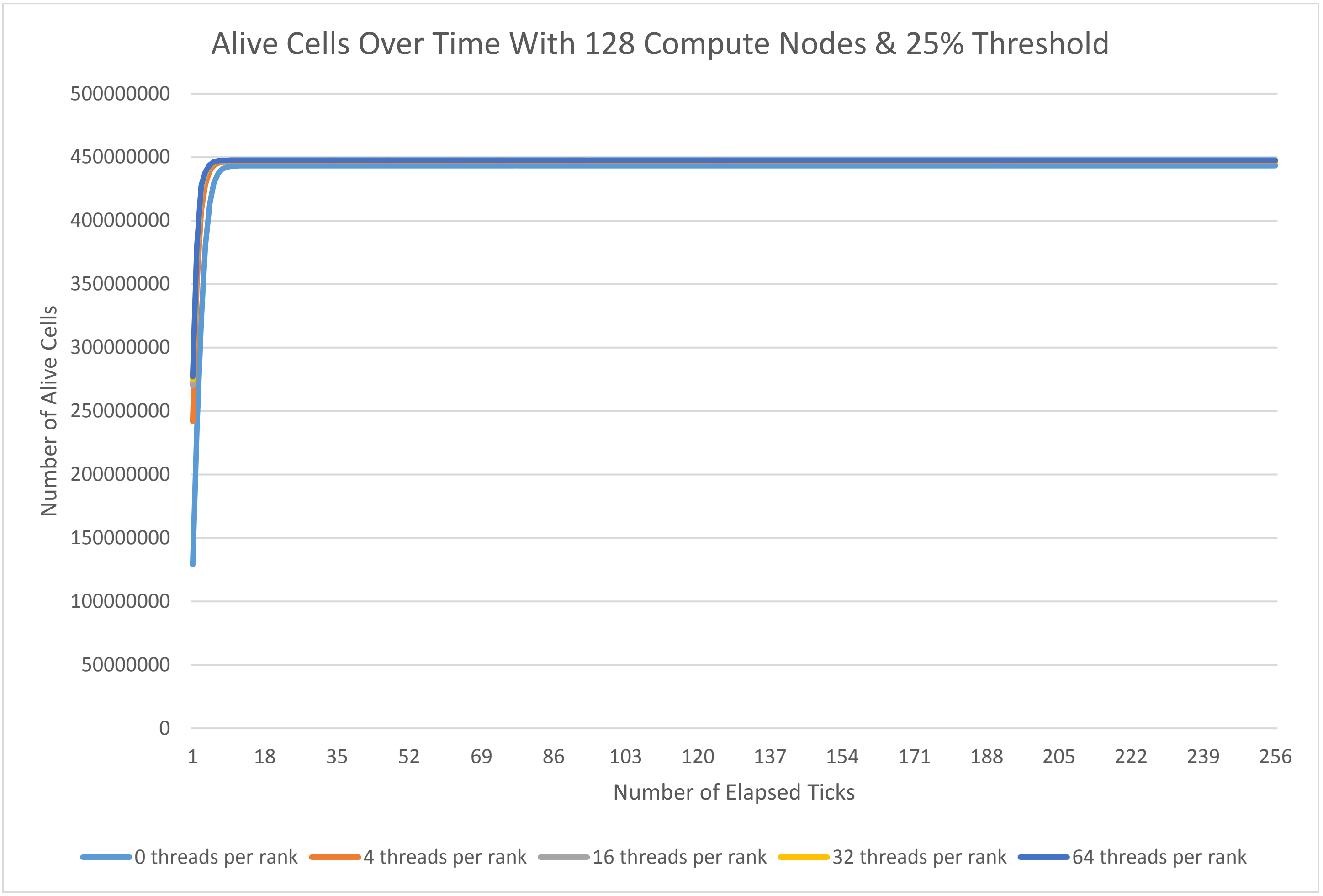
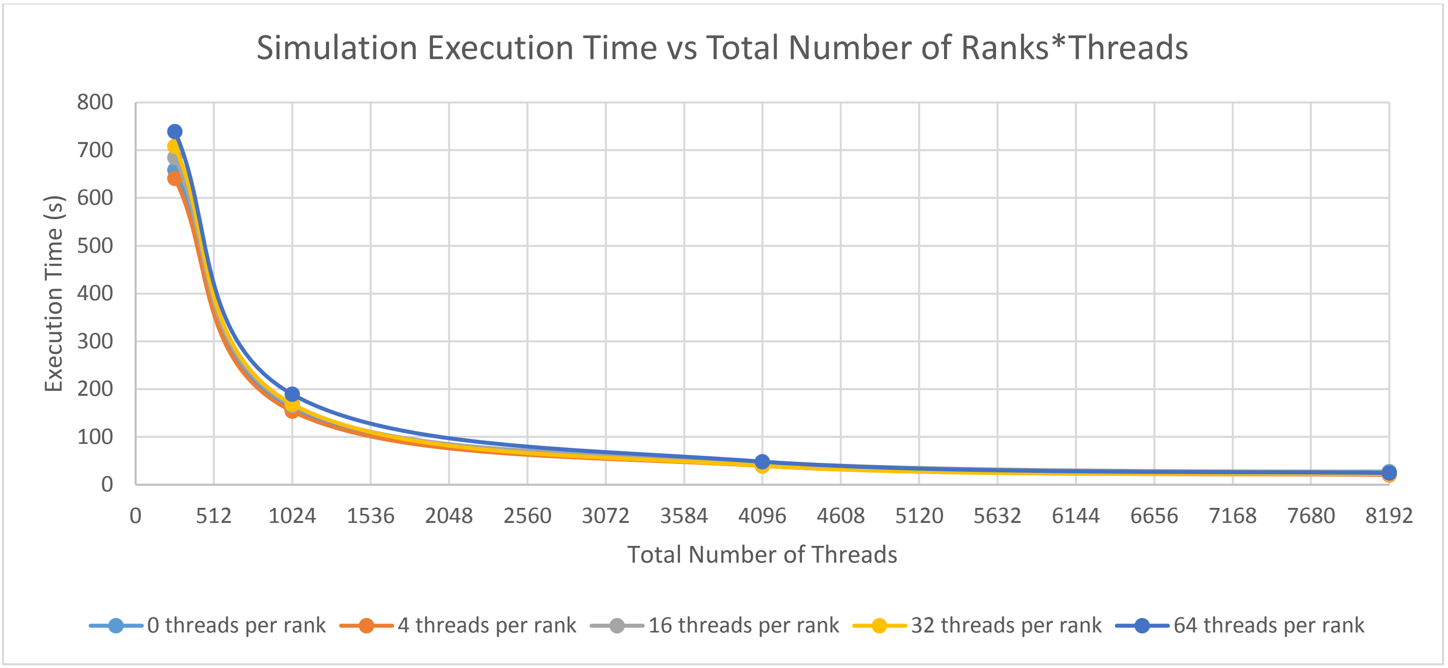
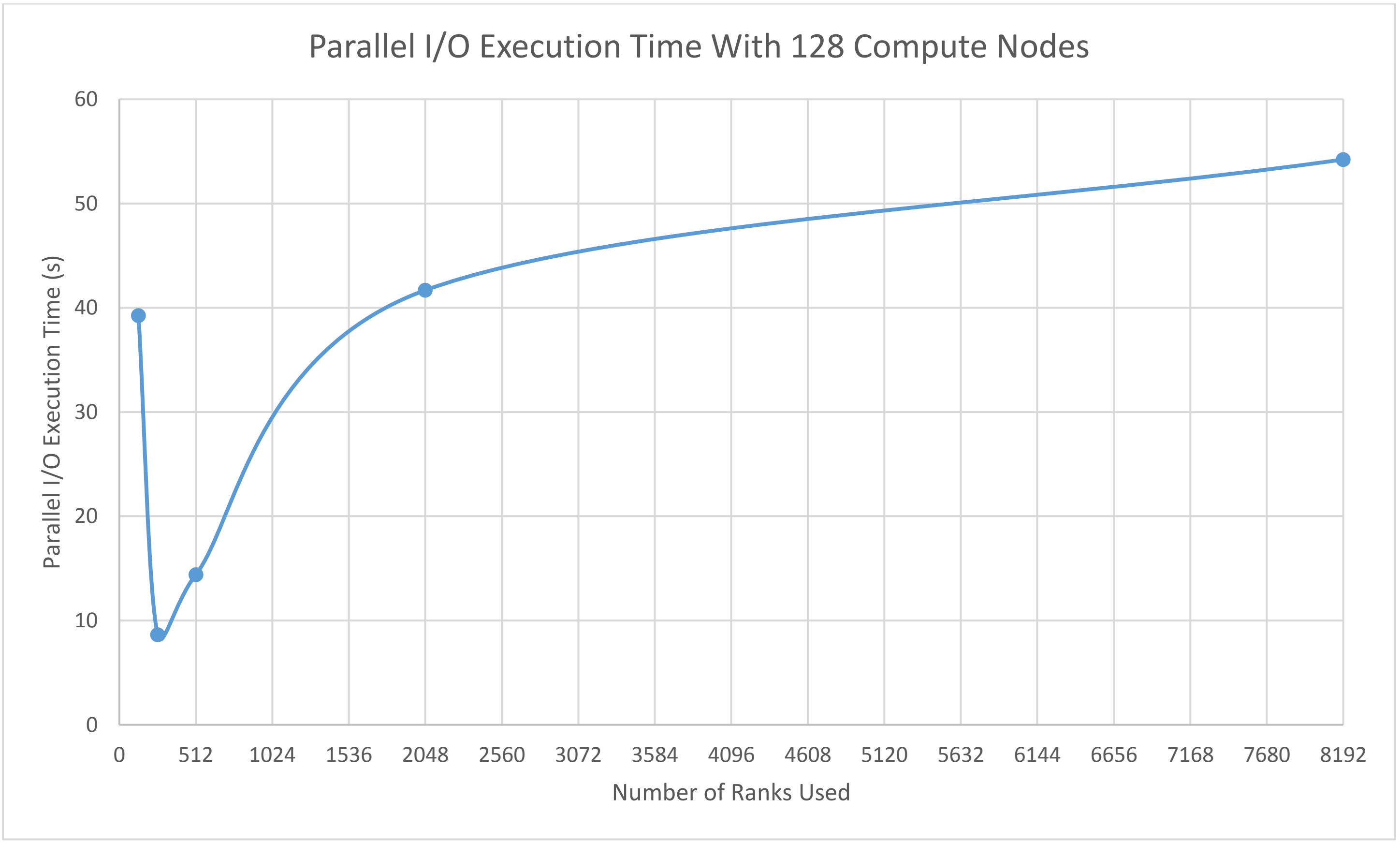
Ryan Stillings, Peter Wood
Parallel Programming and Computing

Assignment 4-5 Performance Report

Performance Graphs




Simulation Execution Time vs Total Number of Ranks*Threads
Execution Time (s)
800
700
600
500
400
300
200
100
0
0
512
1024
1536
2048
2560
3072
3584
4096
4608
5120
5632
6144
6656
7168
7680
8192
Total Number of Threads
0 threads per rank
4 threads per rank
16 threads per rank
32 threads per rank
64 threads per rank


Parallel I/O Execution Time With 128 Compute Nodes
Parallel I/O Execution Time (s)
60
50
40
30
20
10
0
0
512
1024
1536
2048
2560
3072
3584
4096
4608
5120
5632
6144
6656
7168
7680
8192
Number of Ranks Used

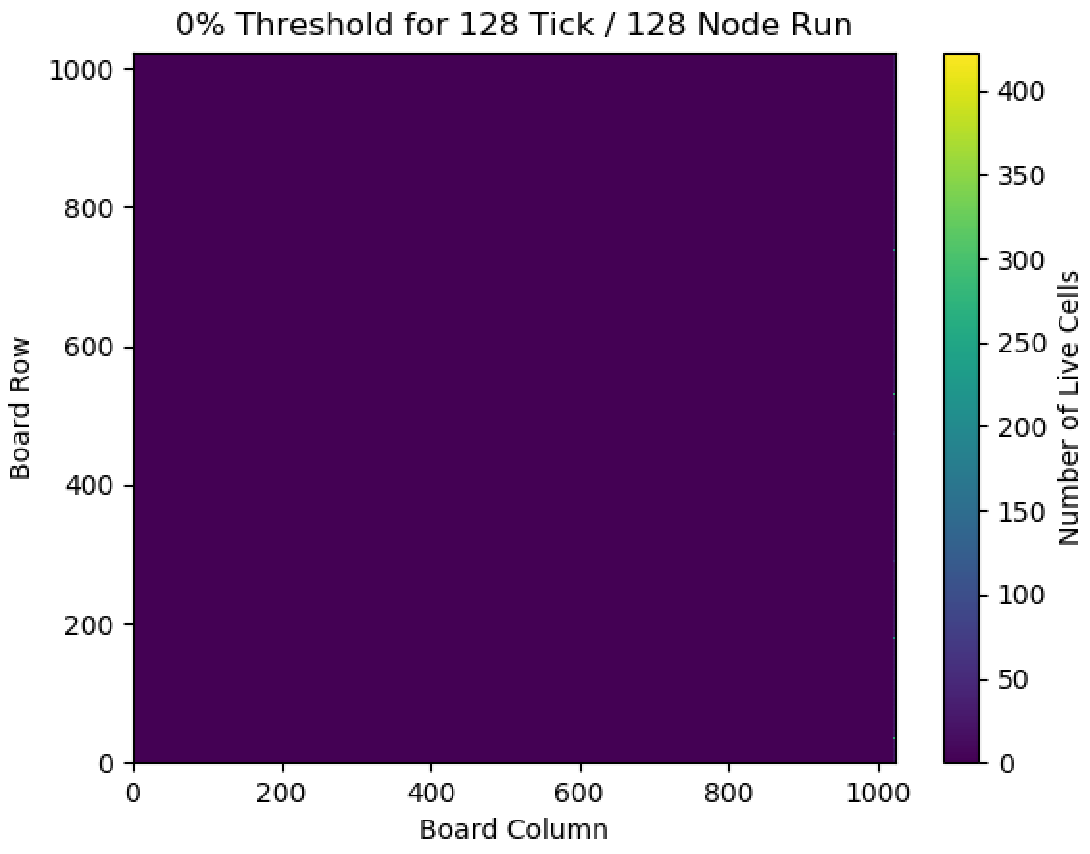
Performance Analysis

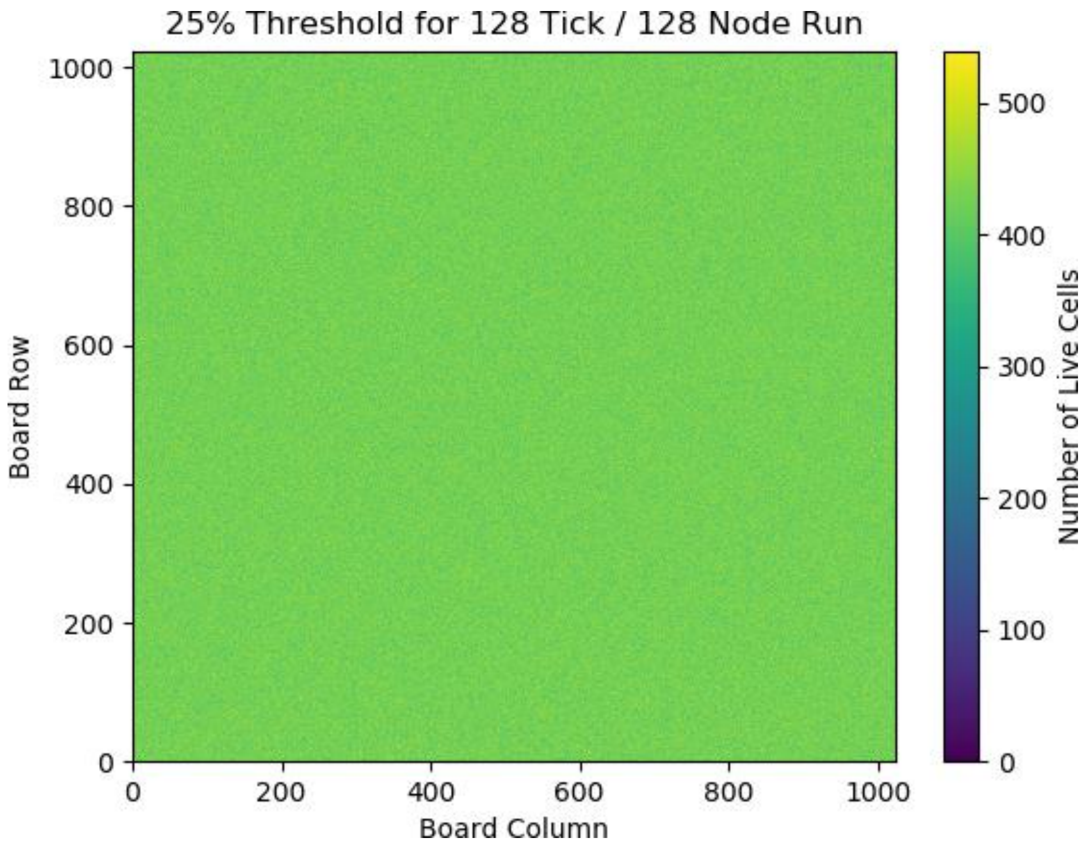
The first performance graph measures the number of alive cells as a function of the tick number and the number of ranks/threads. Generally speaking, all five 128 node configurations saw a similar trend at the 25% threshold. In all cases the first tick resolved with a total number of live cells in the low hundred millions, with the number of live cells rapidly increasing over the next 10 or so ticks until they reached a value of roughly 450 million, at which point the number largely leveled out. This is due to the fact that a large portion of the population dies during the first tick due to overpopulation, and increases/stabilizes over the following ticks due to the game of life rules mixed with the random threshold. However, there were two significant differences in alive cells observed across the 5 configurations. First, the number of alive cells after the completion of the first tick varied considerably across configurations, increasing greatly as the number of threads was increased and the number of ranks was decreased. Second, the maximum number of alive cells approached after 10 or so ticks increased moderately as the number of threads was increased and the number of ranks was decreased. Both of these observations are likely due to the decreasing impact of ghost rows as the number of ranks is reduced.

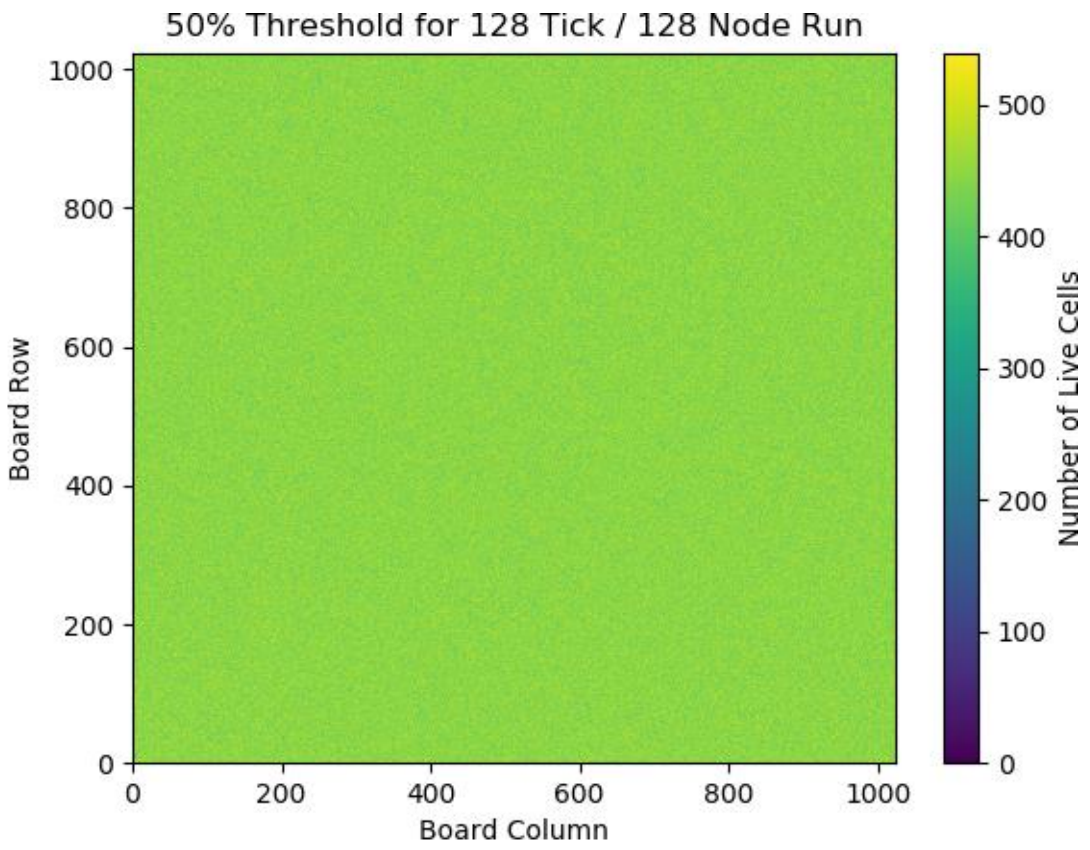
The second performance graph measures the total execution time of each configuration vs the total number of threads (that is, the number of MPI ranks times the number of pthreads in each rank). The overall shape of the graph represents an exponential decay in execution time as more ranks*threads are introduced, with greatly diminishing returns after around 1024 total threads. A consistent trend is present demonstrating faster execution time within configurations with a greater ratio of ranks to threads. However, the fastest performing configuration was not 0 threads and 64 ranks, which would be consistent with that trend, but rather 4 threads and 16 ranks. This is likely due to the fact that threads in our implementation do not have the messaging and synchronization overhead that ranks have, making shifting a small amount of the burden from ranks to threads more efficient, despite threads performing worse in larger quantities. The maximum speedup relative to the 256 rank 4 compute node case was very substantial at all configurations, with all configurations reducing their execution time from 600-800 seconds at 4 compute nodes to just 20-30 seconds at 128 compute nodes. That said, among the 4 compute node configurations the 256 rank configuration was actually the second fastest, slower only than the 4 threads / 16 ranks per node configuration. Regarding parallel efficiency, while all configurations saw a massive reduction in execution time, the 4 threads 16 ranks configuration once again saw the greatest reduction. This configuration completed the simulation in the fastest time at each number of compute nodes, from 641.1 seconds at 4 compute nodes to 20.4 seconds at 128 compute nodes.

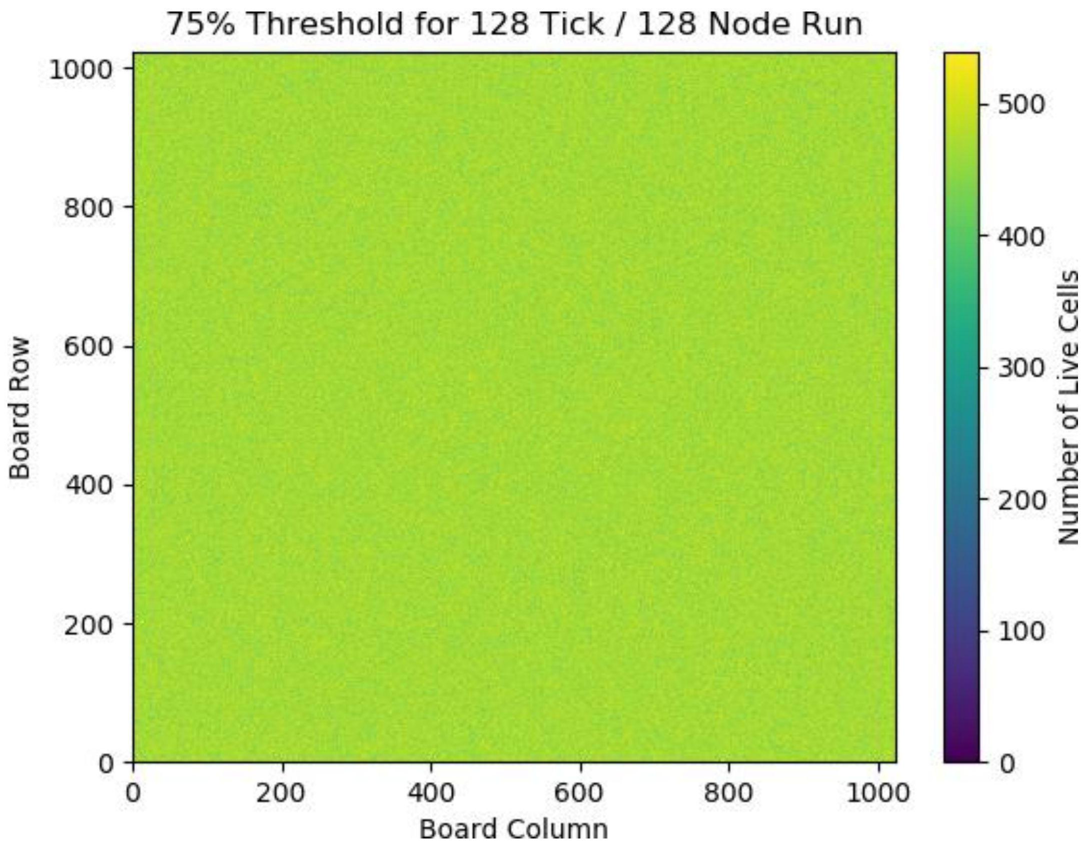
The third performance graph measures the execution time of writing out the 32768x32768 cell universe via MPI parallel I/O with 128 compute nodes and a varying number of MPI ranks. The results of this plot were surprising, with the parallel I/O time dropping sharply from 128 ranks to 256 ranks, then increasing considerably as additional ranks were added. This trend is most likely indicative of the marginal benefit vs marginal cost of adding additional ranks to parallel I/O. The jump from 128 ranks (1 rank per compute node) to 256 ranks (2 ranks per compute node) benefits heavily from the greatly increased rank parallelization, explaining the drop from 39 seconds to just 8 seconds. Further increases to 512, 2048, and 8192 ranks have relatively less room to benefit from the increased parallelization, while simultaneously suffering increasing slowdown due to synchronizing write calls across so many ranks, with MPI_File_write_at being a blocking operation.

Heatmaps


0% Threshold for 128 Tick / 128 Node Run
Board Row
Board Column
Number of Live Cells
0
200
400
600
800
1000
0
50
100
150
200
250
300
350
400


25% Threshold for 128 Tick / 128 Node Run
Board Row
Board Column
Number of Live Cells
0
200
400
600
800
1000
0
100
200
300
400
500


50% Threshold for 128 Tick / 128 Node Run
Board Row
Board Column
Number of Live Cells
0
200
400
600
800
1000
100
300
500


75% Threshold for 128 Tick / 128 Node Run
Board Row
Board Column
Number of Live Cells
0
200
400
600
800
1000
100
300
500

Heatmap Analysis

Each of the four heatmaps shown above was generated during our simulation, exported to a file, and then converted to an image in Python. Each MPI rank maintained a 2d array of integers corresponding to its local chunk of heatmap values, and updated it either after each tick (when testing heatmap values over the course of the entire simulation) or after the simulation completed (when analyzing the final board state only). Upon completion of the simulation, the ranks took turns sending their heatmap chunks to rank 0, who then wrote out the heatmap values in blocks of 32x32. From there, a Python script imported the file, parsed the heatmap data, and converted it into an image file via the pcolormesh function in Python's matplotlib library. The resulting images were then cropped in gimp in order to fit neatly into our write up.

The first heatmap, showcasing the 0% threshold, unsurprisingly turned out to be an almost entirely dead universe. The only signs of life after 128 ticks at this threshold were a few pockets of live cells at the rightmost edge of the grid. This was just as we expected; with no consideration of RNG essentially the entire universe dies off due to overpopulation. The few pockets of life that are able to survive exist at the rank boundaries, due to the fact that our implementation of ghost rows does not benefit from changes that occur during the current simulation tick.

The second heatmap, showcasing the 25% threshold, is a very different story. This heatmap shows a sea of life, with a random, noisy scattering to it. The massive difference in population is easily explained by the presence of the threshold. While without the threshold nearly all cells die at the start due to overpopulation, with the threshold cells now have a 12.5% chance to be alive at any given time, allowing for the universe to avoid complete death from overpopulation and quickly regrow to an equilibrium at around 450,000,000 total live cells. The randomness by which that threshold is applied, coupled with the parallel nondeterminism of our implementation, also explains the noisiness of the heatmap, as there is no telling exactly how many live cells will end up in any particular 32x32 chunk.

The remaining two heatmaps, showcasing the 50% and 75% thresholds respectively, are very similar to the 25% threshold heatmap. All three heatmaps show a well-populated universe, with the precise population of each 32x32 chunk varying in a noisy, aimless fashion. As the threshold increases from 25% to 50%, and again from 50% to 75%, we see a moderate but still measurable increase in the average number of live cells across the entirety of the universe. When considering the live cell count numbers, this increase as the threshold is raised is quite logical. The 25% threshold simulation leveled out at around 450 million live cells, roughly 40% of the universe's total size of 1 billion cells, and random numbers which fall within the threshold result in a 50% chance for any given cell to be alive. The result is that increasing the threshold results in a greater chance for any given cell to be alive at any given tick.