# Parallel Assignment 4/5

Eric Johnson, Harrison Lee

05 April 2019

In figure 1, we can see that the total number of living cells quickly converged to a consistent number, around 420 million, within around 3 ticks. The first generation at tick one ranged from 130 million to 280 million, quickly going to 370-380 million in tick two, before converging to the aforementioned 420 million. This pattern was consistent across the 5 sets of rank to thread ratios in our set of 128 compute node tests, so our simulation results weren't affected by the hardware configurations.

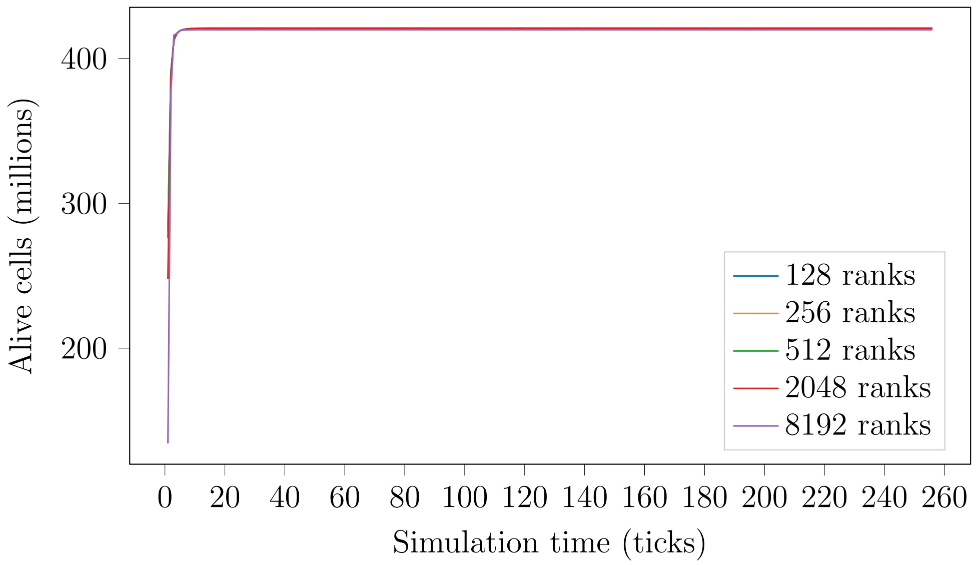

Figure 1: The number of live cells over time with a threshold of 25%.

When comparing execution time against the total ranks and threads for each run, we got the expected one-to-one speed ups. As the number of resources doubled or quadrupled, run times halved or quartered. This is reflected in figure 2, which shows straight lines on a log-log scale. Grouping the run times by the ratios of threads to MPI ranks shows a pattern of the 4 threads and 16 rank configurations being consistently the fastest when the run times were largely consistent with each other. This is likely a consequence of native support by Blue Gene/Q's architecture, where each compute chip has 16 user processors that are each 4-way multi-threaded.

The largest absolute speed up compared to the 256 rank 4 compute nodes experiment was a factor of 32.82 in the 2048 ranks (16 ranks to 4 threads) 128 compute nodes experiment.

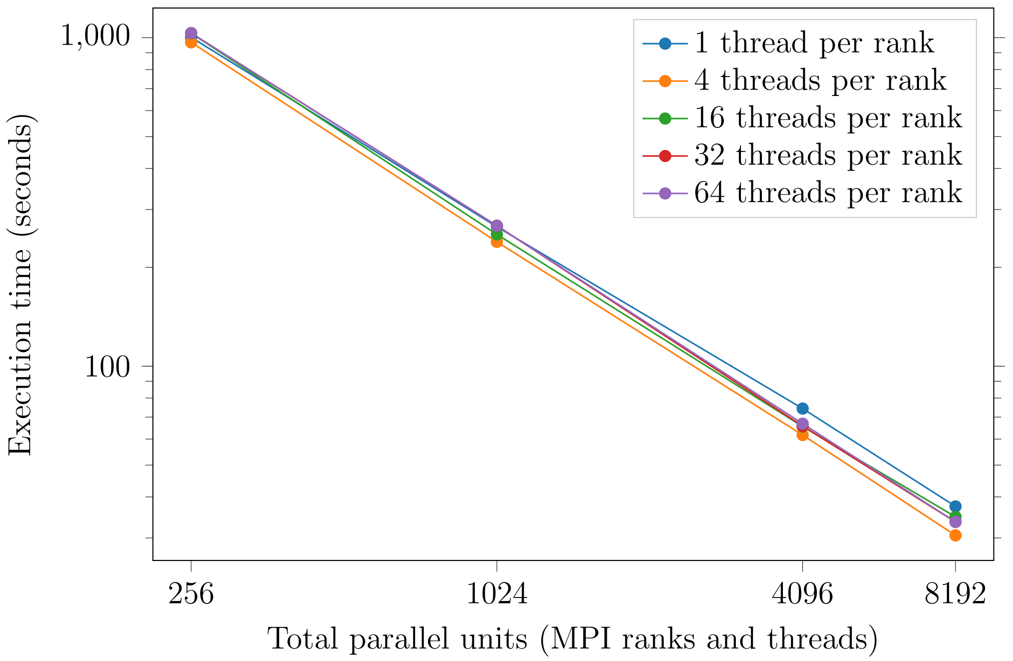

Figure 2: Execution time comparison

We calculated the parallel efficiency of each run by comparing the equivalent serial compute time used (run time multiplied by the total number of ranks and threads) against that of the 256 ranks, 4 compute nodes run. These results are shown in figure 3. Our highest parallel efficiencies for each set of compute nodes were also in the 16 ranks to 4 threads experiments, with the 16 compute node experiment having the highest efficiency by a slim margin, followed closely by the 4, 64, then 128 node runs.

In our tests of I/O performance we found a performance decrease as we increased the number of ranks, going from 0.17 seconds to 29.70 seconds between 128 and 8192 total ranks and threads. There was a strange pattern, however, with there being roughly three tiers of run times at around 0.2 seconds, 5 seconds, and 30 seconds, as shown in figure 4. In our I/O implementation, we provided `MPI_File_Write_at` with our boards in a single memory chunk, so it is likely the MPI implementation had internal optimizations that allowed it to more quickly write fewer larger chunks, as in our smaller rank counts, than they could many small chunks, as in our larger rank counts.

We constructed our heatmaps in the main program by having each MPI rank allocate a local 1K×1K heatmap array and use their simulated rows to fill in the corresponding heatmap blocks, with zeros in all the other values. We then performed a sum reduction over all the ranks, which left us with a complete heatmap at rank 0. Looking at figure 5, with a threshold of 0, the only alive cells are on the left and right edges of the map, with none in the middle. For non-zero thresholds, the heat maps are uniform over the entire area, with slight random texturing. As the threshold is increased, the average number of living cells per heatmap block also increases. With a threshold of 25%, the average is 401 cells/block (39% alive); 424 cells/block (41% alive) for a threshold of 50%, and 457 cells/block (45% alive) for a threshold of 75%.

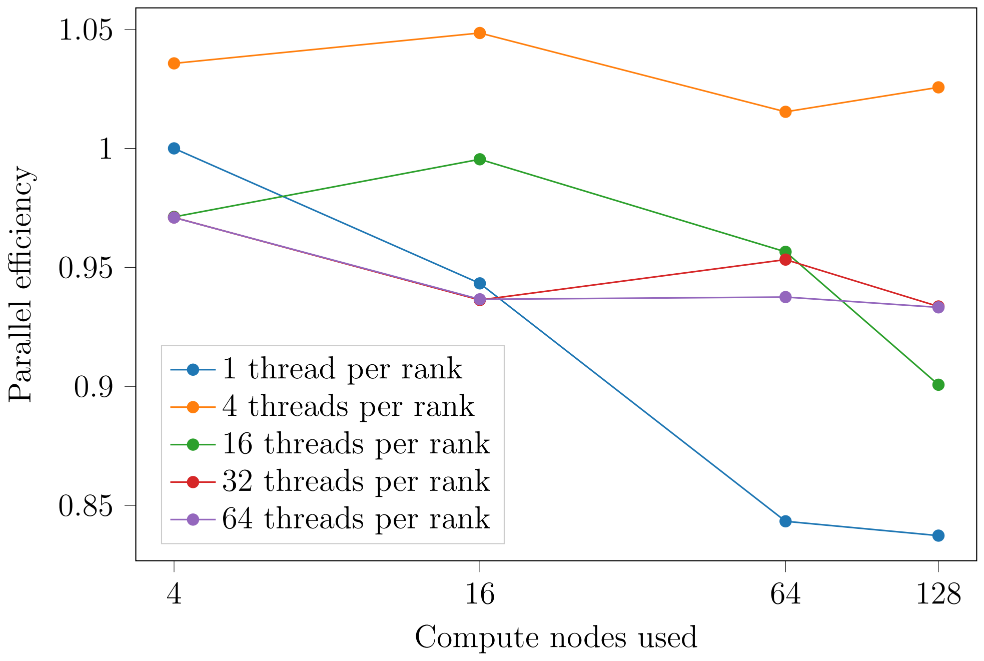

Figure 3: Parallel efficiency, as compared to the 4 node, 256 rank experiment

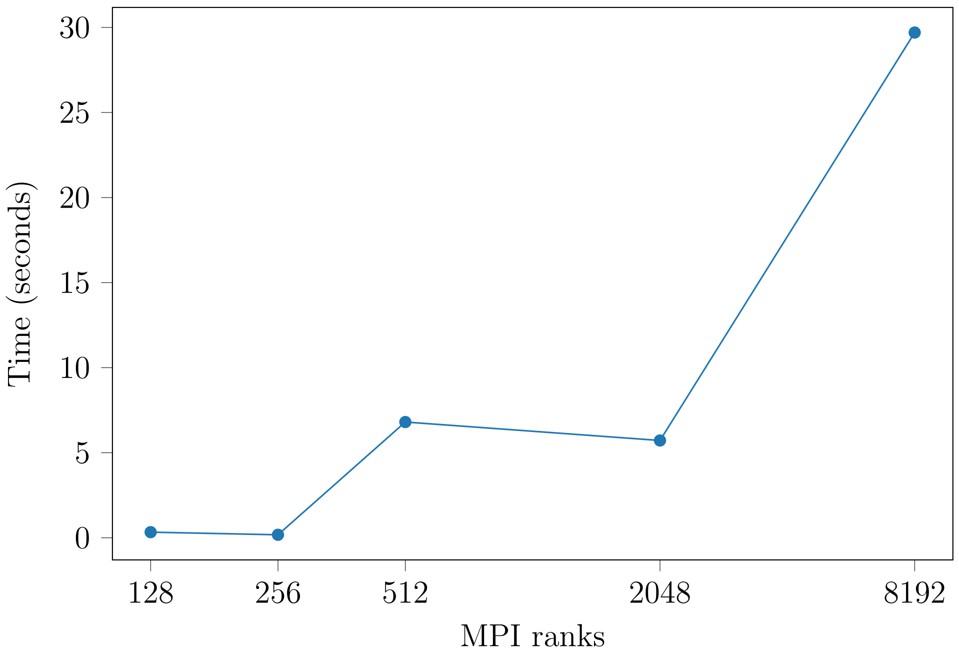

Figure 4: Parallel I/O time to write out the 32K×32K universe

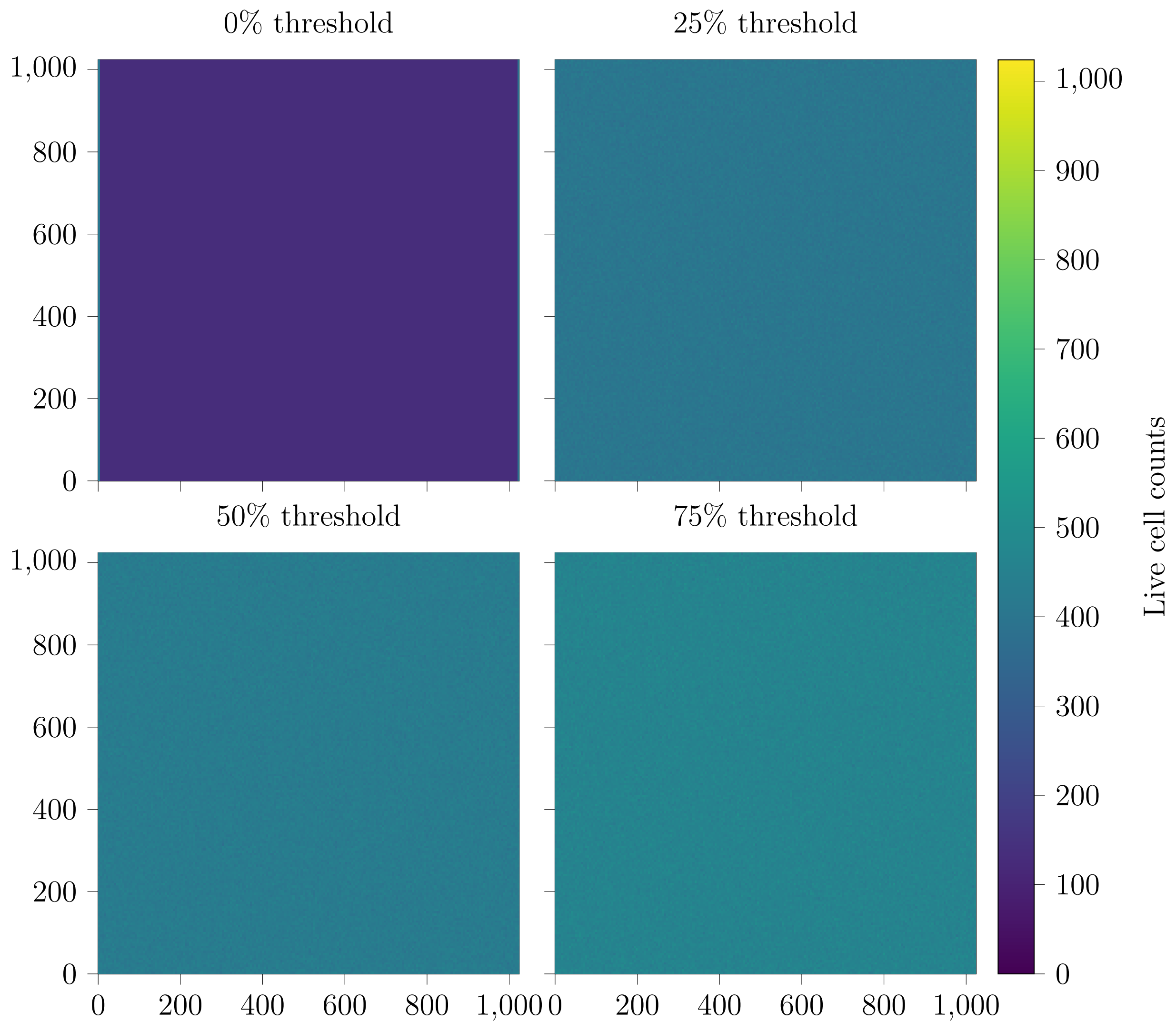

Figure 5: Heatmaps for several different thresholds, run with 128 nodes at 4 threads per rank