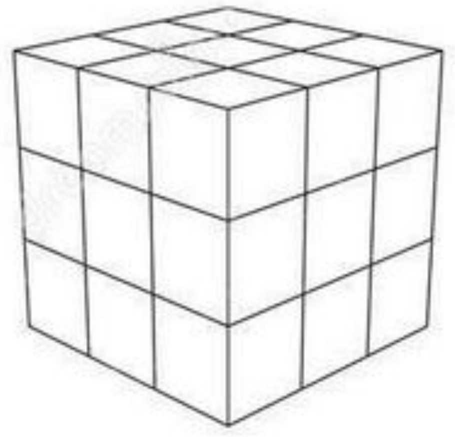
3D Conway's Game of Life

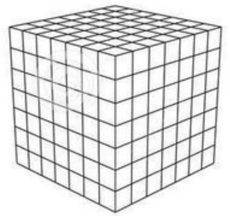
Implemented Conway's game of life with 3d (2n+1)x(2n+1) sized cube as "neighborhood" for a cell (centered on the cell).

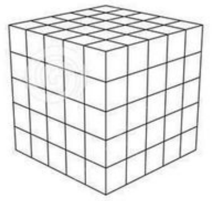
Tested n=1,2,3

Rules:

- Cell needs 4 neighbors to survive
- Cell dies if it has more than 15 neighbors
- Cell is born if it has 8 or more neighbors





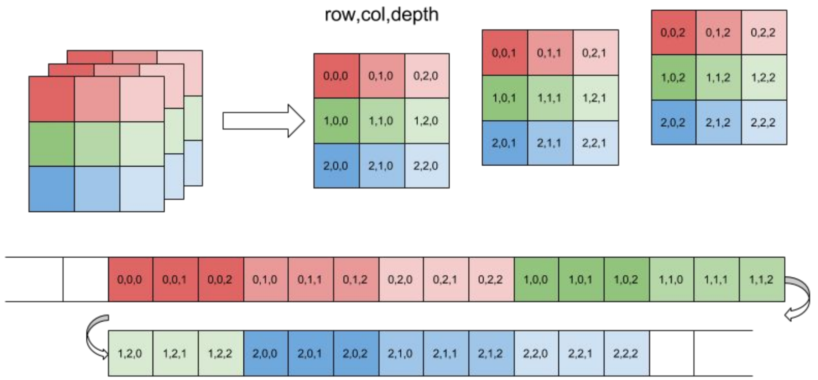


Single Threaded Implementation

Pad region with n ghost cells on each side (so the overall dimensions are (x+2n) by (y+2n) by (z+2n)) which will always be set to 0, this lets me get rid of having if statements checking for boundaries in the inner loop which speeds it up tremendously.

Store region as a giant contiguous array, though this makes data locality tricky to do right.

Slow: 176s to do one cycle on a 1024x1024x1024 region (n=2)

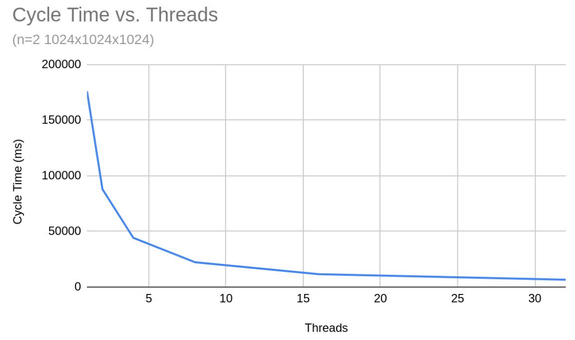


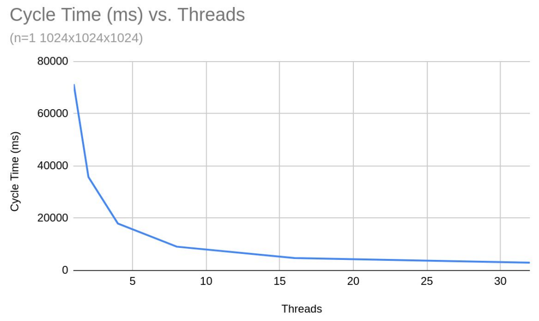
Set cube of 56M (400x350x400) cells to living near the center of the region.

Track cells changed and living cells per epoch, use this to validate other implementations.

OpenMP Version

- Parallelize outermost of the triple nested loop
- Got 2.5x speedup by matching the storage order with the loop order.
- Initially I processed the loops x->y->z, but stored z->y->x (so the x axis was the contiguous block).
- After switching to storing x->y->z, OMP 32 threads went from ~7 seconds per cycle to 3 seconds per cycle (1024x1024x1024 n=1).
- Very good return on thread count

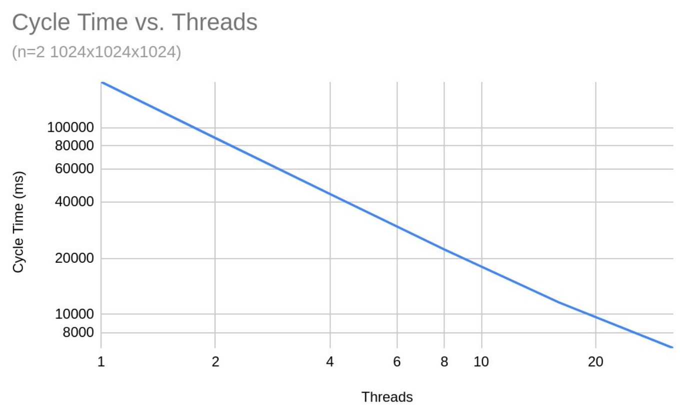


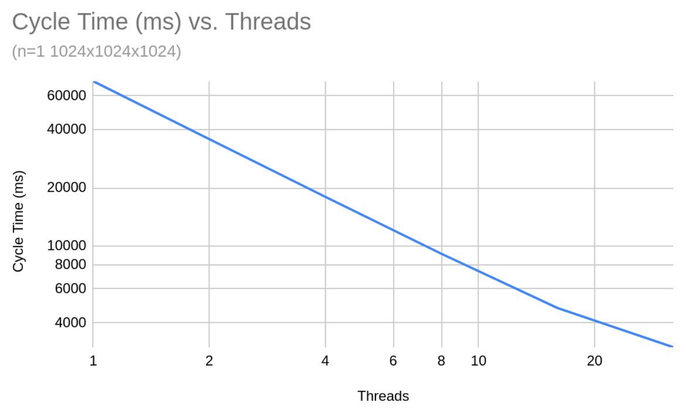


Interesting Performance Results

- 5x5x5 cube only took about 2.5 times as long as 3x3x3 cube despite being 4.6x larger
  - Benefit from caching more when local area is larger?
- Parallel performance improvement almost perfectly linear up until 32 threads
- Minimal NUMA penalty when moving beyond 8 cores (prior tests I've done on AXIS machines suffer horribly from NUMA penalties)

(Same charts as before but with log scales)



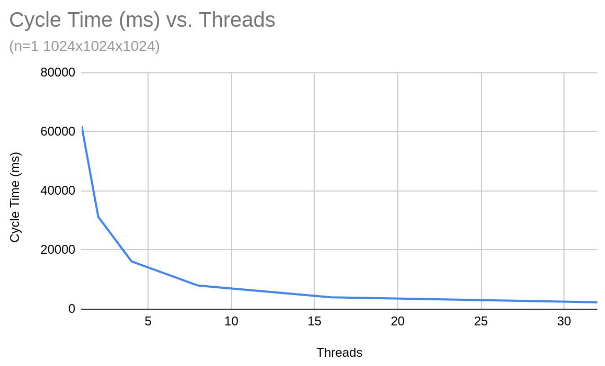


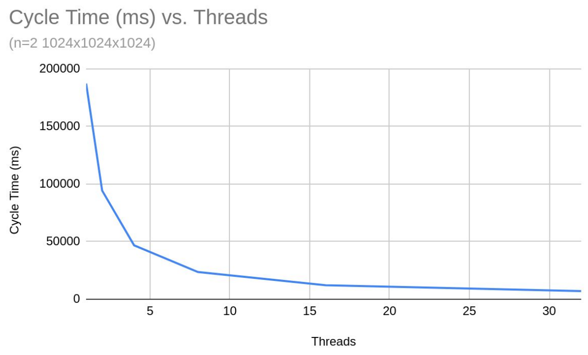
Lessons Learned About OpenMP

- Shared memory synchronization between threads is a mess, needed to use reduction pragmas to make counting the number alive and changed cells work correctly
    - Spent a solid 2-3 hours trying to figure out why the simulation wasn't working when it was really just the cell counts being reported wrong
- You still need #pragma omp barrier even if you are doing #pragma omp parallel for
- __restrict__ pointers had almost 0 impact on performance for this
- -march=native is very powerful if the compiler can spot a good opportunity to use SIMD, absolutely useless for the AXIS CPUs otherwise (not OpenMP specific)
- Loop unrolling is pretty good in inner loops (specifically the neighborhood test)

OpenMPI Version

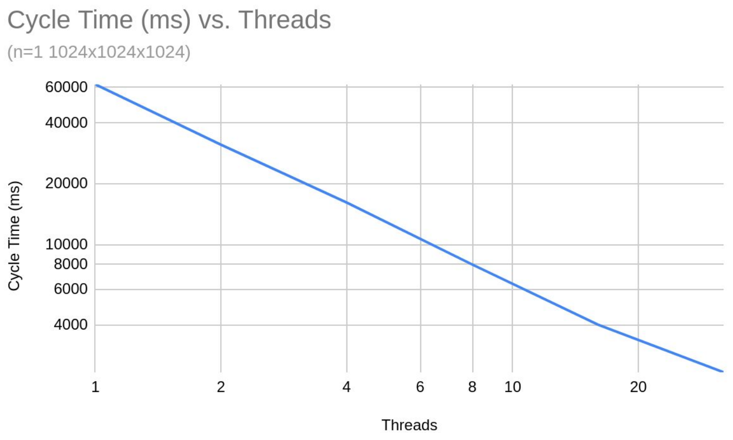
- Split along the X axis by slicing the YZ-plane such that there was one region per process.
  - NOT the most efficient, doing cubes would have been much better
  - Slices send Distance*2*Height*Depth*Threads cells per epoch, cubes send Distance*2*root3(Threads)*((Height*Depth) + (Height*Width) + (Depth+Width)) per epoch.
  - MUCH easier to program
- Ghost cells at boundary of each region are updated with values from neighbor before each epoch
- MPI calls are done asynchronously which sped it up a bit and made it much easier to program.

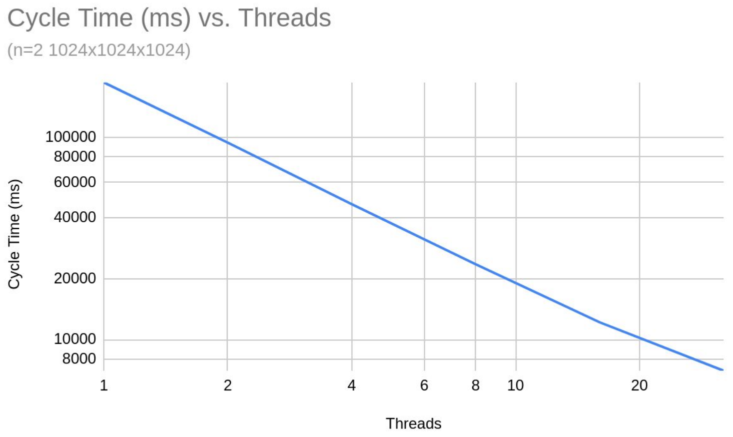




Interesting Performance Results

- Performance overhead not as bad as I expected for doing slices instead of cubes
  - At 32 threads, each slice is 32 cells wide, and we are sending two n slice wide chunks, which means we are sending n*6% of our cells each epoch, but that seems to have a relatively negligible performance impact
- FASTER than OpenMP version for many threads
  - Much better memory locality???
  - Smaller workspaces means you can fit more of it in the cache
  - Would cubic work areas improve this further?
- AXIS cluster was very busy while I was working on this and slurm wasn't cooperating, so I didn't push it to multiple machines. Communication overhead would likely be much more brutal across multiple machines.





Things I Would Like To Test

- Multiple nodes with MPI (ideally with different networking speeds/protocols [IP vs RDMA])
- Rather than using a big contiguous array, store things in many smaller arrays (may improve locality)
    - Could try exotic memory layout where the region is sliced into a bunch of cubes where all the data can fit into a couple cache lines so locality is improved
- Packing cells in to bitfields
    - Currently cells are represented as uint8_t’s, really only need one bit per cell
    - Would reduce required memory bandwidth and fit in to the cache much better
    - Would also require extra ops to mask and shift on reads and writes
- Only updating around live cells (this would probably offer the single biggest speedup, tricky to track locations of live cells)
- Hand done SIMD in neighborhood search