AM 250 Final Project Report - Game of Life

1.

The first approach to partitioning made it clear that the grid should be divided into squares in order to reduce perimeter and corners. The program only has 1 piece of computation, so not much thought has to be put toward computational decomposition, other than the fact that communication with neighbors is required. The smallest grain possible is 1 cell to a processor. All the communication required for the problem comes down to the boundaries. Local communication makes the most sense because very little computation is required, and each timestep is sequential. Agglomerating the cells into squares increases the area (number of cells) faster than the perimeter (number of neighbors) increases. When it comes to mapping, the rules of Game of Life are easily computable but require heavy communication, thus the computation should be done locally. Locality is much more important than concurrency, again emphasizing the need to increase area vs perimeter. A complex implementation could be implemented to perfectly load balance any MxN grid with any P processors. This poses much more complexity to the program. The current implementation sticks to simplicity: the grid must be NxN, and not all processors are used so that the program runs with a number of processors that both divides N and is a perfect square. Because all the loads are equally balanced in this implementation and run approximately concurrently, there is no need for task scheduling.

2.

README.txt:
“On Hummingbird, do the following to compile and run my code:

To change the grid size, edit mpiGOL.f90 and change bigN
To change number of processors, edit slurm.cmd and change 1234

mpif90 mpiGOL.f90 -o mpiGOL.exe
sbatch slurm.cmd or mpirun –np 4 mpiGOL.exe
The output data will be in test.out
To look at the data, type more test.out”

The list of files is mpiGOL.f90, mpiGOL.exe, slurm.cmd, test.err, test.out
The slurm file is using Hummingbirds partition 128x24.

3.
Exact output:
“ numproc = 4 sqrtNP = 2 N = 10
running with 0
running with 1
running with 2
running with 3
Grid at t = 0
00100000000000000000
10100000000000000000
01100000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
Grid at t = 20
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000001000000000000
00000101000000000000
00000011000000000000
00000000000000000000
00000000000000000000
00000000000000000000

00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
 Grid at t =   40
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000010000000
00000000001010000000
00000000000110000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
 time elapsed =   2.03939100000000002E-003
 Grid at t =   80
00100000000000000000
10100000000000000000
01100000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000

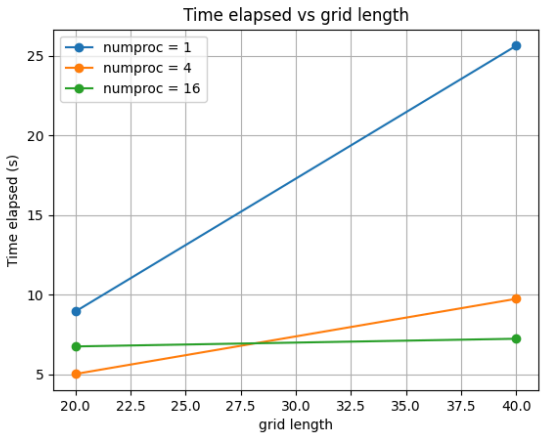
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000 ”

4.

According to my calculations
On Hummingbird, t_startup = 7.75e-4, t_cost/word = 4.57e-11

2D decomposition

numproc	sqrtNP	N	T	Time Elapsed (s)
1	1	20	1000000	8.9524
4	2	10	1000000	5.0330
16	4	5	1000000	6.7624
1	1	40	1000000	25.6376
4	2	20	1000000	9.7520
16	4	10	1000000	7.2461



When 1 numproc doubles N, the t_w also doubles but the computational cost quadruples: 2 times the perimeter, 4 times the area. That is why the total time more than doubles from 8.9524 to 25.6376.

When the program goes from numproc=4 to numproc=16 on a 20x20 grid, the total time increases! On 40x40 it decreases. Computational cost is the same but because its parallelized, we expect numproc=4 to perform 4 times slower than numproc=16. Communication size costs doubles (quadruple sends, but half perimeter per send) and startup cost quadruples (quadruple sends). At 20x20, the computational speed isn't very large, so the time spent sending outweighs the computational cost. At 40x40, we can see the orange point is above the green, indicating numproc=16 is faster than numproc=4, likely due to the fact that communication costs outweigh computation.

N doubles -> computation quadruples, words doubles, startup same
numproc quadruples -> computation decreases by factor of 4, words doubles, startup quadruples