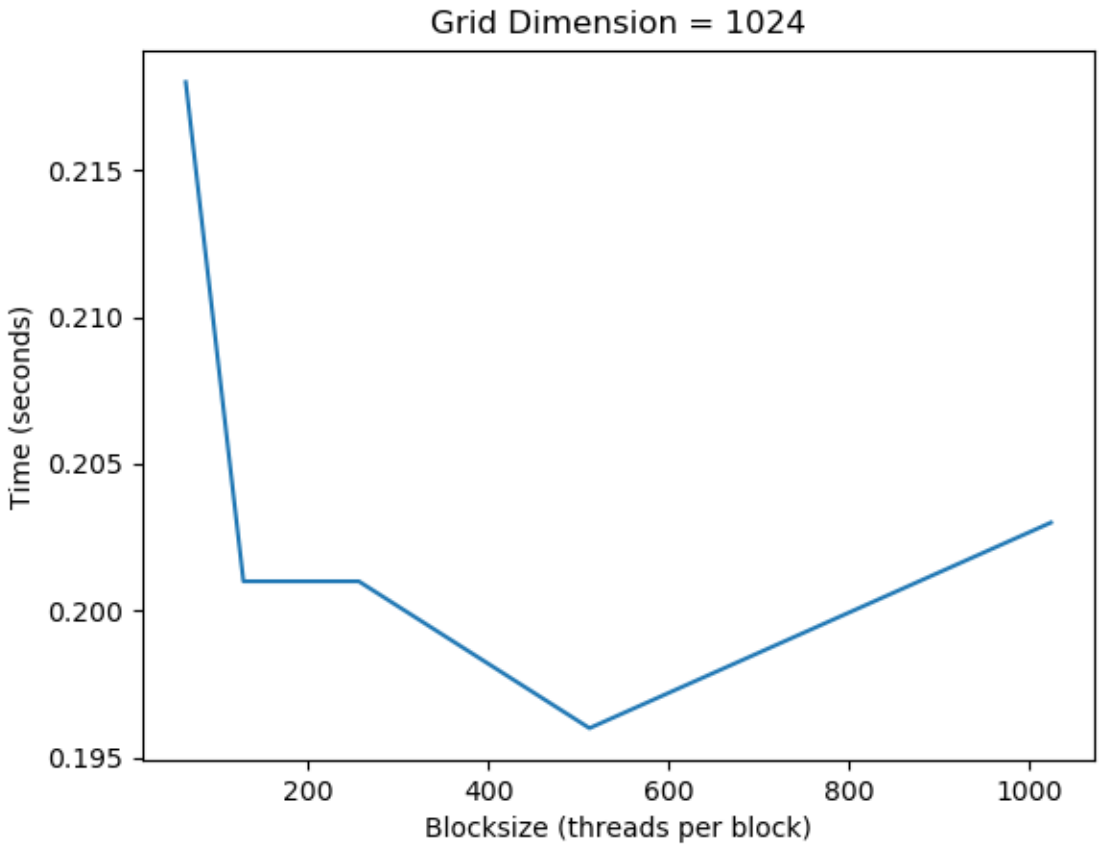
John Torborg

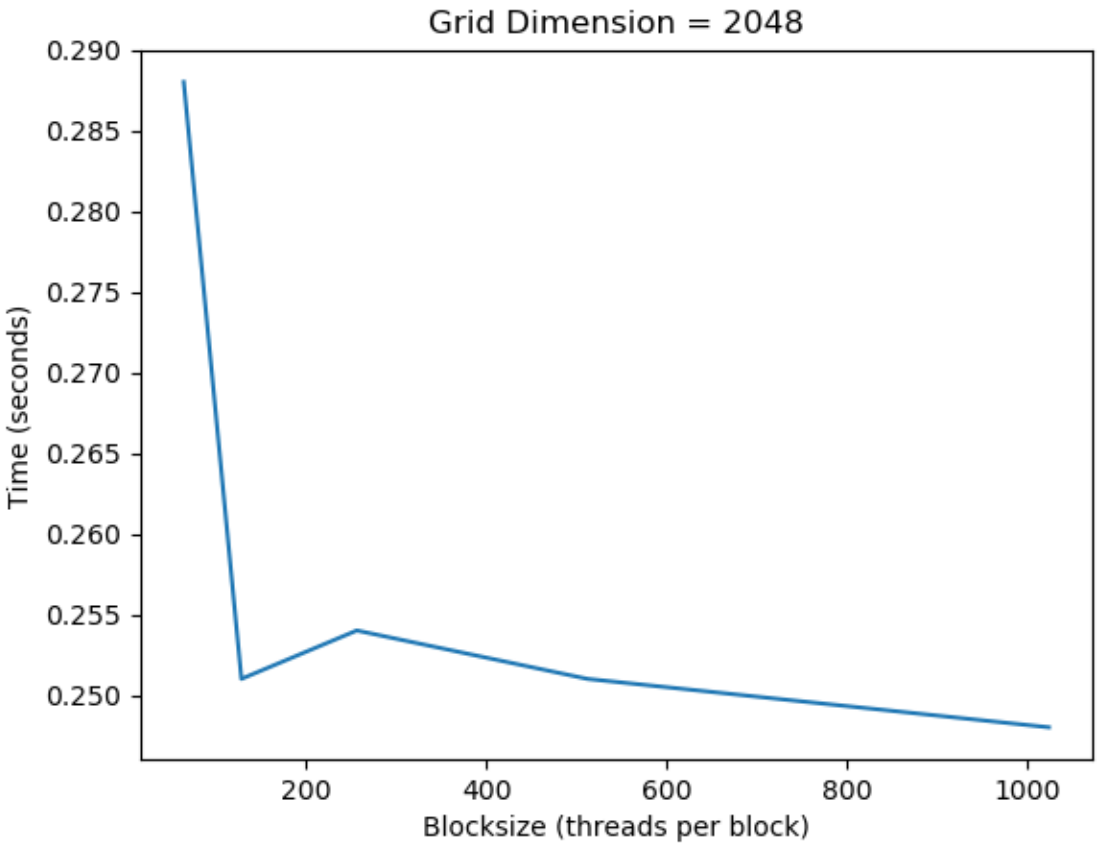
Parallel Programming

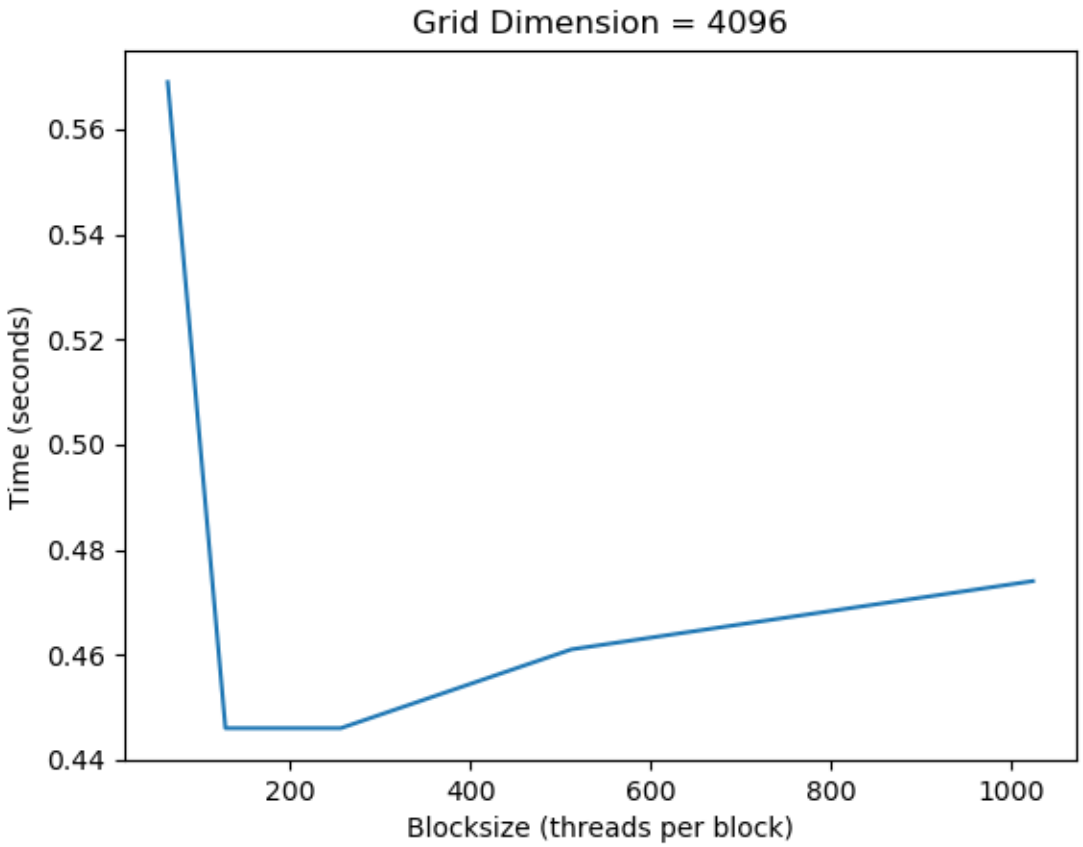
Spring 2020

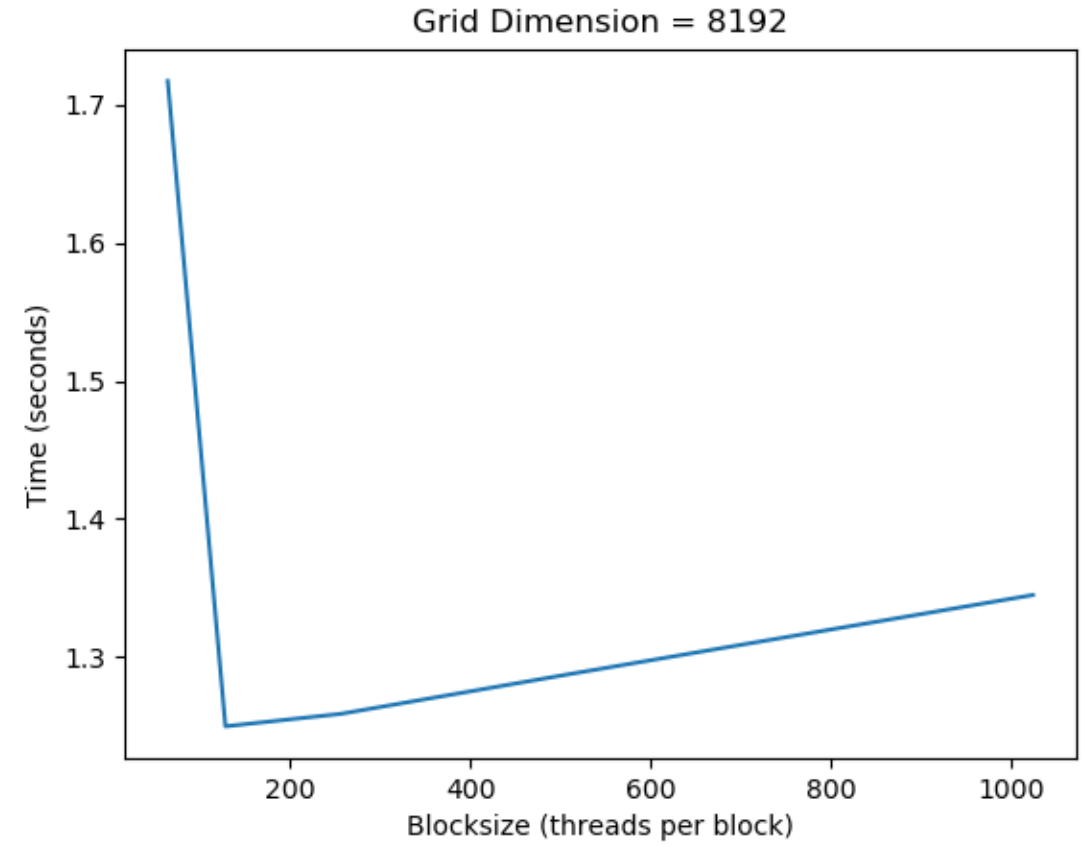
Assignment 2 Performance Analysis

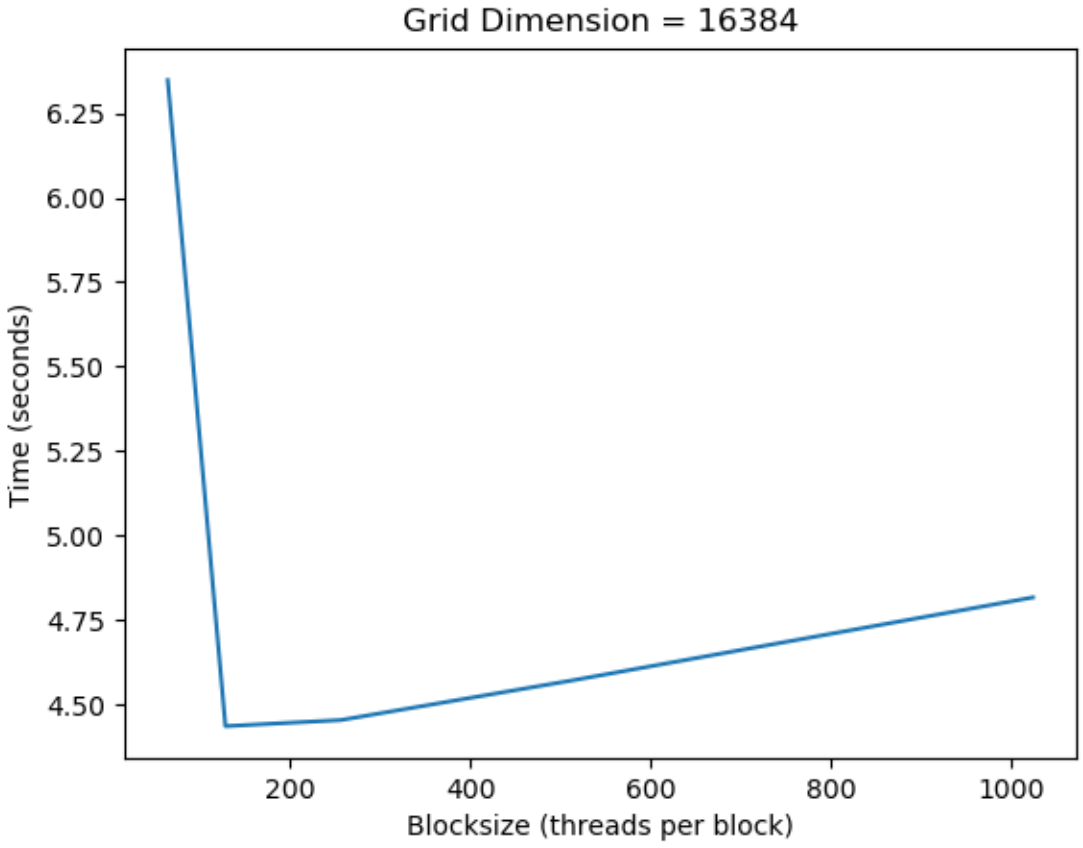
Using the specifications given in the assignment, runtime analysis was performed on my parallel Game of Life (GOL) implementation resulting in the following data, with each sample being run for 1024 iterations:

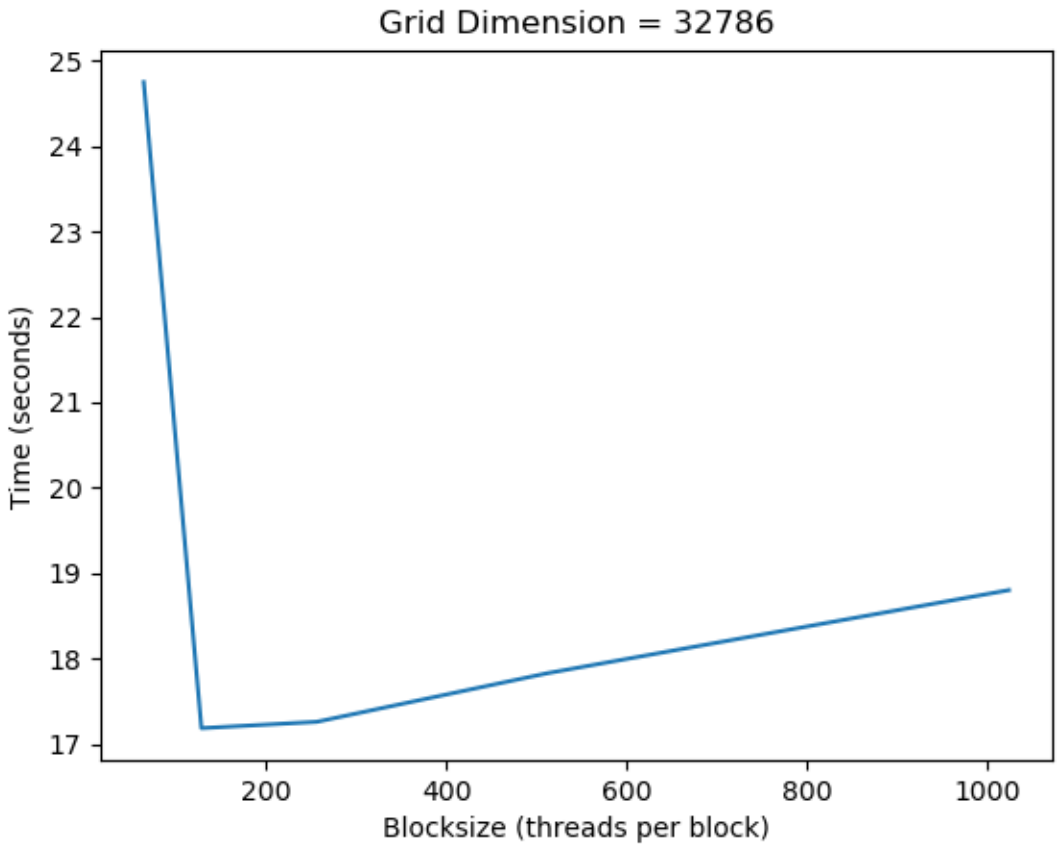



Grid Dimension = 2048
0.290
0.285
0.280
0.275
0.270
0.265
0.260
0.255
0.250
Time (seconds)
200
400
600
800
1000
Blocksize (threads per block)


Grid Dimension = 4096
0.56
0.54
0.52
0.50
0.48
0.46
0.44
Time (seconds)
200
400
600
800
1000
Blocksize (threads per block)


Grid Dimension = 8192
Time (seconds)
1.7
1.6
1.5
1.4
1.3
200
400
600
800
1000
Blocksize (threads per block)


Grid Dimension = 16384
Time (seconds)
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
200
400
600
800
1000
Blocksize (threads per block)



For nearly all grid dimension sizes the fastest thread configuration was with a blocksize of 128 threads, though there were a few notable exceptions. In the 1024 grid, performance improvements were seen from increased thread counts until 512 threads (which was the fastest in the 1024 case), while in the 4096 grid both the 128 and 256 thread cases had virtually identical runtimes. Another interesting pattern observed from the resulting data was the consistent performance slowdown as thread count rose past a certain point for all grid sizes, though this effect was less apparent on smaller grids. On the smaller grids, both the 1024 and 2048 grids the runtimes were so low that it is likely variability in thread overhead and thread initialization times contributed significantly to changes or inaccuracies in runtime measurement. The reason for a critical thread count for optimized runtime across grid sizes is also likely due to thread overhead. At a certain point the cost to maintain and read computation results of large amounts of threads becomes greater than the speed improvement computing on those same parallel threads affords.

To prove that this slowdown is caused by thread overhead, we can analyze thread behavior at runtime with the results provided by the CUDA profiler:

```
==112914== Profiling application: ./gol 0 32786 1024 256 0
==112914== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  18.2541s      1024  17.826ms  17.170ms  685.34ms  gol_kernel(unsigned char*, unsigned char*,
unsigned int, unsigned int)
      API calls:   94.87%  17.5512s         1  17.5512s  17.5512s  17.5512s  cudaDeviceSynchronize
                    3.79%  701.38ms      1024  684.94us  10.164us  668.12ms  cudaLaunchKernel
                    1.19%  219.89ms         2  109.94ms  71.079us  219.82ms  cudaMallocManaged
                    0.14%  26.216ms         2  13.108ms  11.828ms  14.388ms  cudaFree
                    0.01%  1.0882ms         1  1.0882ms  1.0882ms  1.0882ms  cuDeviceTotalMem
                    0.00%  835.23us        97  8.6100us     296ns  318.52us  cuDeviceGetAttribute
                    0.00%  72.660us         1  72.660us  72.660us  72.660us  cuDeviceGetName
                    0.00%  4.7540us         1  4.7540us  4.7540us  4.7540us  cuDeviceGetPCIBusId
                    0.00%  3.2890us         3  1.0960us     600ns  1.9250us  cuDeviceGetCount
                    0.00%  1.0510us         2     525ns     479ns     572ns  cuDeviceGet
                    0.00%     557ns         1     557ns     557ns     557ns  cuDeviceGetUuid

==112914== Unified Memory profiling result:
Device "Tesla V100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       2  64.000KB  64.000KB  64.000KB  128.0000KB  11.39200us  Host To Device
    5186         -         -         -           -  669.1261ms  Gpu page fault groups
Total CPU Page faults: 2
[PCP9trbr@dcs232 ~]$
```

The above results present a more detailed look at the runtime of a parallel GOL over a 32786 grid with 256 threads and the standard 1024 iterations. The next data set is from a parallel GOL over the same grid and number of iterations but with 1024 threads:

```
==137656== NVPROF is profiling process 137656, command: ./gol 0 32786 1024 1024 0
==137656== Profiling application: ./gol 0 32786 1024 1024 0
==137656== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  19.5122s      1024  19.055ms  18.549ms  485.06ms  gol_kernel(unsigned char*,
 unsigned char*, unsigned int, unsigned int)
      API calls:   96.32%  19.0082s         1  19.0082s  19.0082s  19.0082s  cudaDeviceSynchronize
                    2.55%  502.92ms      1024  491.14us  9.6090us  474.04ms  cudaLaunchKernel
                    1.00%  198.10ms         2  99.051ms  70.374us  198.03ms  cudaMallocManaged
                    0.12%  23.057ms         2  11.528ms  11.234ms  11.823ms  cudaFree
                    0.01%  1.2745ms         1  1.2745ms  1.2745ms  1.2745ms  cuDeviceTotalMem
                    0.00%  832.17us        97  8.5790us     342ns  322.65us  cuDeviceGetAttribute
                    0.00%  69.956us         1  69.956us  69.956us  69.956us  cuDeviceGetName
                    0.00%  4.8180us         1  4.8180us  4.8180us  4.8180us  cuDeviceGetPCIBusId
                    0.00%  4.0880us         3  1.3620us     682ns  2.5040us  cuDeviceGetCount
                    0.00%  1.4080us         2     704ns     541ns     867ns  cuDeviceGet
                    0.00%     543ns         1     543ns     543ns     543ns  cuDeviceGetUuid

==137656== Unified Memory profiling result:
Device "Tesla V100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       2  64.000KB  64.000KB  64.000KB  128.0000KB  12.35200us  Host To Device
    4637         -         -         -           -  465.8997ms  Gpu page fault groups
Total CPU Page faults: 2
[PCP9trbr@dcs220 ~]$
```

The total GPU runtimes presented by both sets of data are comparable, but from the previous plot, we know that they have a relatively significant difference between their total runtimes. Therefore we can conclude that the slowdown is mostly taking place on the CPU side, meaning the additional thread overhead managed by the CPU is causing the most performance detriment.

The final table lists “cell updates per second” for each grid size and blocksize configuration:

Cell Updates per Second

Blocksize (threads per block)

Grid size	64	128	256	512	1024
1024	4.93E+09	5.34E+09	5.34E+09	5.48E+09	5.29E+09
2048	2.98E+10	3.42E+10	3.38E+10	3.42E+10	3.46E+10
4096	1.21E+11	1.54E+11	1.54E+11	1.49E+11	1.45E+11
8192	3.20E+11	4.40E+11	4.37E+11	4.27E+11	4.09E+11
16384	6.93E+11	9.91E+11	9.87E+11	9.62E+11	9.13E+11
32786	1.42E+12	2.05E+12	2.04E+12	1.98E+12	1.87E+12

Based on this data, the 32786 grid running on 128 threads has the fastest updates per second rate, though it is very comparable to the rate for 256 threads at this grid size.

As a final point, to run my specific implementation of the parallel GOL, the terminal initialization command follows this sequence:

Arg. 1 - Pattern

Arg. 2 - World size

Arg. 3 - Number of iterations

Arg. 4 - Number of threads (blocksize)

Arg. 5 - “0” for no terminal output

‘1” for terminal output