|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2500x2000 | Sequential | 2 Threads | 4 Threads | 8 Threads |
| 1 | 2.708215274 | 2.999114444 | 2.936745327 | 3.952196529 |
| 2 | 4.266670702 | 2.671903543 | 2.682263202 | 3.610384063 |
| 3 | 3.095809593 | 2.25767184 | 2.26639845 | 3.955513023 |
| 4 | 2.588002555 | 2.175530248 | 2.146155012 | 3.319244987 |
| 5 | 3.504280462 | 2.169075497 | 2.130389993 | 3.214597518 |
| 6 | 4.386558415 | 2.180994874 | 2.170709094 | 3.447860853 |
| 7 | 4.755956894 | 2.254749789 | 2.179635988 | 3.890460054 |
| 8 | 4.304692495 | 2.263939174 | 2.196344903 | 3.235343463 |
| 9 | 4.28940771 | 2.240331811 | 2.289128083 | 3.784613782 |
| 10 | 4.388106313 | 2.297984539 | 2.390287499 | 3.483732584 |
| 11 | 4.401470736 | 2.262345344 | 2.176345262 | 4.535791958 |
| Average | 3.880833741 | 2.343058282 | 2.3240366194 | 3.675430801 |

The table above is the performance times of the cellular automaton of Conway’s Game of Life with a parallel step function. This was run on a quad core cluster computer. The optimal solution with the fastest average runtime was the one with 4 threads. This is a 1.67 fold increase over the sequential step function. The grid was split into an appropriate number of rectangles and each rectangle was passed to a Thread to process concurrently. This reduced the runtime of the code significantly as evidenced by the table above. Most of the program is inherently sequential because it is a sequence of steps that display cellular automata. The only part of this program that can be parallelized was the step function because looping through a big array sequentially is expensive. However, the step function accounted for over 50% of the code, and as a result of Amdahl’s law, the predicted speedup is 1 / ((1-0.5)+(0.5/4)) = 1.6, which is close to our number. 4 threads are optimal over 8 because according to Amdahl's law, there is a certain maximum speedup that can occur for a given number of processors and a proportion of the program that can be made parallel. Eventually adding a new thread each time will not benefit and instead the overhead of creating a thread will impact the performance negatively. This is also shown in the table as the average run time for 8 threads is pretty close to the sequential time. Increasing the thread count even more would probably give us a time that is even slower than our sequential time.