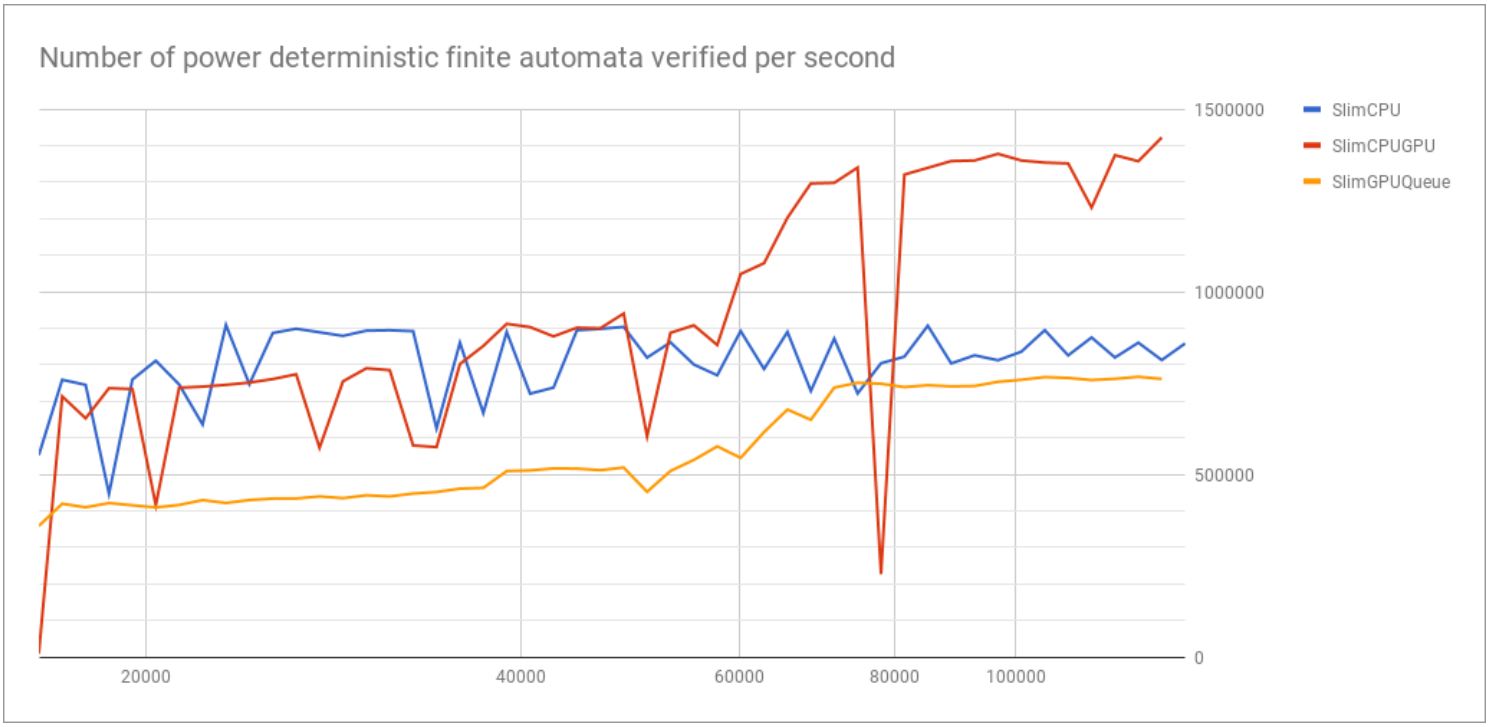

Ultimately the CPU algorithm wins because all graphs are partitioned into groups and accessed by all threads.

The threads allocate the least amount of memory possible and allocate it pretty rarely.

Both winning algorithms were improved by limiting LINQ usage and reverting to the good old arrays and indices (thus limiting unnecessary copy operations).

All in all the best GPU algorithm turns out to be a scaled up and parallelized CPU algorithm (smart BFS queue)

Previous generations of CPU algorithms were poor at being computed in parallel. PLINQ was used, along with AsOrdered command which dramatically slowed down execution.

Now each virtual processor accesses its own part of the whole array (actually it accesses the original large array only within the appropriate range).

This technique allows for full processor utilization thus outperforming the GPU.

Initial poor GPU performance is the effect of lazy PTX instructions compiling (JIT compiling)

The computations were performed on an Intel i7 8-th generation processor and the GPU part was done on an NVIDIA Pascal architecture GeForce mobile GPU (mx150).

Overall laptop hardware performace results:

CPU performance ranges between 700k up to 900k automata per second

GPU's performance usually settles down to ca. 770k automata per second

Their combined perforance usually reaches up to 1300-1400k automata per second! (the problems are split equally among both computing units)