

ECE 459: Programming for Performance

Assignment 1

Jack Erickson

January 27, 2021

1 Sudoku Solver

The sudoku solver uses a recursive brute force method to solve input puzzles. Every cell of the puzzle is iterated through until an empty cell is found. Each value between one and nine inclusive is tested to see if that value will fit in that square based on the current known values in its row, column, and square. For each of these possible values, the value is inserted into the puzzle and the function calls itself again. If no value can be found for a certain cell, it is assumed there is an issue with a value guessed earlier so false is propagated back up the call chain until a layer is able to try a different value. Eventually a solution will be found where there are no more empty spaces, and the function returns true back up the recursion.

This solution is not very optimized and could be improved. For example, if we take an initial pass at the puzzle, we can store possible values for each empty cell. Then we can start by guessing squares with the least number of possible values with the same recursive method. This can drastically reduce the number of iterations required. If we order the possible choices randomly as well, the likelihood the number is correct is increased with smaller possible choices, this means we have the highest probability of guessing correctly sooner, and these clues can help reduce possible values for other squares as well.

2 Nonblocking I/O

The conversion to the multi library resulted in a noticeable improvement to the time to run the verifier. (See Table 1)

Table 1: Benchmark results for Sudoku verifier on 100 puzzles.

Concurrent Connections	Runtime Average (s)	Time to verify 1 puzzle (ms)
1	26.203	262
3	6.581	199
4	4.877	195
16	1.451	232
32	1.026	328

Increasing the Number of available connections to N connections should speed up the runtime by a factor of N. The results of the benchmark reflect more improvement on performance, than expected. Which suggests that by preventing any one connection from blocking indefinitely the verifier can continue working on other connections while another finishes. Furthermore, the time taken to verify a puzzle seems to decrease up until 4 concurrent connections, at which point it starts increasing again indicating diminishing returns for multi threading. I can't say for certain why this is the case, but I believe this may be due to the bottleneck shifting from network latency to the overhead in curl multi. Although multi is non-blocking, it isn't multi threaded. From my testing it would seem beyond 4 connections the internal overhead is non-trivial and to improve performance linearly, should be converted to a multi-threaded model.