Parallel Sudoku Solver Milestone

Members:
Kevin Zhu (kzhu2)
Hao Zhang (haozhan2)

Summarizing Current Work Completed

We have been following our proposed schedule quite closely and we are on track. Starting from the beginning, we researched our topic and prior works regarding previous Sudoku solvers. Aside from a few select mentions of solving human-solving strategies or paradigms like (constraint problem analysis), there is not much prior work into solving this problem.

Currently, we have an easily-extendable framework built up. It supports reading from files to get Sudoku grids, Sudoku correctness validation, data structures and a few naive algorithms that solve the board. The algorithms implemented are a naive backsolve sequential algorithm, naive BFS algorithm, and a naive parallel BFS algorithm written in OpenMP.

Deliverables and Goals for Poster Session

Current Deliverables

We currently do not have deliverables to show at this time. While we do have a naive version of algorithms at this time, we feel like the results are not indicative of what is to come on the final product and will be changed at a later time with improvements. This is mainly due to measuring performance and speedup with some grain of ‘fairness’. For example, the naive backsolve uses 9^(N*N) guesses, which takes a considerable amount of time to complete. It would not be fair to compare speedup to such a case. The BFS algorithm also uses guessing but runs a N^2 checker that greatly reduces the exponent, and completes in seconds as opposed to hours, yet parts of the BFS algorithm could also be improved. As we are continuously developing the algorithms, we will gauge and draw the measuring line of where to time the performance of each algorithm as not to be unfair in measurements.

List of Goals and Plan for Poster Session and Schedule

For the poster session, we will show the overall performance improvement of our implementations due to parallelization efforts. This will mostly be done with graphs. For the various approaches that we do try, we will talk about the bottlenecks and various design decisions made. If we have time, a nice to have would be a GUI showing the solution and what it looks like in parallel for the demo for visualization.

Detailed Project Schedule

Deadline Date	
April 19	Naive OpenMP version for global work queue (Hao) Project Milestone Report (Kevin)
April 22	OpenMP Global Work Queue Optimizations (private variables, reducing barriers, etc) (Hao) Start Distributed Task Queue (Hao) Nice to Have: Compare against Sudoku File (Hao) Add few Constraints Problem Sudoku Tactics (Kevin) Speedup Sequential Solvers (Both BFS and Backsolve) (Kevin) Explore Parallel BFS using Aggregate Strategy (Kevin)
April 26	Timing Code and Comparisons (Hao) Extra datasets if 9x9 not good (Hao) Complete Distributed Task Queue and any optimizations(Hao) Finish Parallel BFS Aggregate Strategy (Kevin) Some additional CP Sudoku Tactics (Kevin) If Parallel BFS Aggregate Strategy works well, try DFS, if not help Hao with Distributed (Kevin)
April 29	Finish Distributed Queue Version (Hao) More datasets to cover obscure cases (Hao) Nice to have: DFS Version (Kevin) Study Performance for different CP Problems (Both) Start Final Report, Summary, Presentation, any loose ends (Both)

	Nice to have: GUI (Both)
May 1	Final report, summary and presentation (Both) Nice to have : MPI Version (Both)

As previously mentioned, we are on track with our current goals and deliverables in our proposal. Going at our current pace, deliverables should still be possible. We do have quite a few nice to haves: the most interesting being using an aggregate strategy to try and increase parallelism for higher number of core counts since there are only 81 squares in sudoku. We are not sure if this approach will work, but regardless, it's an interesting point to discuss for our report and may also scale and be a possible implementation for a DFS strategy. If this fails and is slow, then we will not pursue DFS and will both work on trying to improve performance of a distributed work queue with work stealing. Of the other nice to haves, more datasets could always help us better gauge our performance. While we did write MPI as a nice to have, unless we have an excessive amount of time, it is unlikely that it will have high performance.

List of Concerns

Work balance
Solving Sudoku could have many possible solutions, but generally so far in our problem, we have reduced the scope to solve for a single solution. When the solution is found, the problem is solved and we measure the time. As a result, when comparing timing, it's obvious that comparing the time for DFS and BFS solution would not be fair, so we are writing multiple versions of these algorithms which is taking a lot of time.

One particular concern we have is the benefit of work-stealing. It’s quite hard to gauge what types of problems would require work-stealing and would directly benefit from it. Finding datasets that show the performance improvements for the tuning that we do is hard to justify.

Performance and Limited Parallelism
Especially when moving to higher core counts, it's much harder to achieve high performance scaling. The grid count for a standard Sudoku is 9x9, which if parallelized by squares would only support 81x speedup max. However, it’s fewer because some of the squares (generally at least 17) are filled in for a one solution Sudoku. From what we learned in class, we plan to scale the amount of work to 16x16 to encourage more parallelism, and also try an aggregate method mentioned earlier to try and have all threads participating for better scaling. We are open to other strategies as well as designing algorithms that scale well.