Sudoku Solver

Team Members: Madeleine Hagar and Jennifer Chou

URL: https://github.com/jcchou12/Sudoku-Solver

SUMMARY: We are going to implement a parallel sudoku solver on the NVIDIA GPUs in two different ways. We will be starting off with a sequential algorithm and use tools such as OpenMP and MPI. We will be researching other libraries that may help convert a sequential algorithm to a parallel algorithm, as well.

BACKGROUND: Sudoku is a puzzle that consists of a NxN grid with a few numbers between 1 to N already filled in on the board. The entire grid is also divided into smaller squares that each consist of N unit squares. The goal of the game is to insert the rest of the numbers from 1 to N such that each number only appears once within each row, column, and subdivided square.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Example of a 9x9 Unsolved Sudoku Puzzle

As the value of N increases, the time it takes to solve the puzzle scales exponentially due to the large increase in possible filled in boards. Due to this issue, parallelism would be beneficial for eliminating possible boards or solving different sections of the puzzle separately. If parallelism is used in separate sections of the puzzle, it would be able to easily eliminate all possibilities where one section of the board is infeasible in a shorter amount of time.

THE CHALLENGE: As the size of the board increases, there are a lot more possibilities on how the board could be filled. Due to this, there would be a lot more communication if we use MPI, so we would need to figure out how to best minimize the communication between processes. We would need to think about how to split up the work between processes and which processes should be communicating with which. For OpenMP, we want to make sure that we are not using too much shared memory for all parts of the board, so as the board size increases, we need to figure out how to best split up the amount of memory being shared to make the solver as efficient as we can. Because of the amount of dependence between the

different sections of the grid, separating out the workload is a very challenging constraint. We need to figure out how to ensure that we do not share too much information all at once because this would slow performance, but still share enough so that the processes are able to find a solution. For this project, we hope to learn more about different ways to split up work between threads or processes. We want to gain more experience using these tools and to do more in-depth learning about their differences in performance.

Each row's unit squares depend on other columns and the subdivided squares that they intersect with. They need to be able to share memory or communicate with whatever processes are in charge of the unit squares that they depend on, so that processors reduce the amount of extra work they are doing. There would likely be a high communication to computation ratio because the processors need to basically communicate with the entire board since a row intersects with each column and we need to make sure there are not any conflicts within the unit square number assignments. There should be some divergent execution based on how we split the work on the sudoku board. Each process can individually assign numbers in their own section first to see what might work.

RESOURCES: We plan on researching different sequential algorithms to find which algorithm we might want to build off of and parallelize. We will be using OpenMP and MPI on the NVIDIA GPU. We will also research more on other libraries that may be helpful for parallelization of a sequential algorithm.

GOALS AND DELIVERABLES:

PLAN TO ACHIEVE

- Implement a sudoku solver using OpenMP
- Do an analysis on the performance of the OpenMP algorithm on different tests
- Implement a sudoku solver using MPI
- Do an analysis on the performance of the MPI algorithm on different tests
- Do a comparison analysis on the performance of both algorithms and the initial sequential algorithm

HOPE TO ACHIEVE

- Implement a third sudoku solver and complete an analysis on the algorithm's performance

DEMO

- Our demo at the poster session will demonstrate our sudoku solvers and will output the performance times of each algorithm on different tests with different sized boards and starter numbers. We also want to see if we can program some sort of visualization for the solver as it works.

PLATFORM CHOICE: We plan to use OpenMP for one algorithm because it hides the low-level details and allows us to just specify what needs to be parallel and will balance the workload for us. This would be helpful because the workload for each parallel section would be split more evenly and this could be

adaptable to different starter boards. We plan to use MPI for the other algorithm because with larger boards, shared memory may be too large and we might not be able to get the performance we want with OpenMP. MPI allows us to send messages between different processes, so if we had each process in charge of a row/col/square, we could communicate with other sections of the board, but avoid communicating with processes that don't conflict with the current process's section.

SCHEDULE:

Week	Goals
April 2 - April 8	● Research and test different sequential algorithms ● Decide on a sequential algorithm to build off of ● Create the tests for the algorithm ○ Create 8 different tests ■ 9x9 (random, sparse) ■ 16x16 (random, sparse) ■ 25x25 (random, sparse) ■ 36x36 (random, sparse) ○ Write the code for timing the performance ● Write the code to check for correctness of the algorithm
April 9 - April 15	● Research possible methods for parallelization ● Complete the code for the first parallel algorithm
April 16 - April 22	● Milestone Report (due Wednesday, April 19th at 9:00AM) ● Do an analysis on the first parallel algorithm ● Start the second parallel algorithm
April 23 - April 29	● Complete the second parallel algorithm ● Do an analysis on the second parallel algorithm ● Do a comparison analysis on the two parallel algorithms
April 30 - May 5	● Final Report (due Thursday, May 4) ● Poster Session (Friday, May 5) ● Create the poster ● Create the demo