Assignment 2 Report

Tanmay Goyal
AI20BTECH11021

Program Design:

The program begins with reading from the input file ‘input.txt’ and getting the values of the number of threads points and the size of the sudoku board we wish to check. If the file cannot be opened, we return an error. We then acquire the elements on the sudoku board to be checked.

The input.txt file is generated using a python script that takes as arguments the number of threads and the size of the sudoku, generates a random sudoku, and parses it and puts it into the input.txt file in the expected manner.

We then define a class to store the parameters to be passed to the threads, which consist of the following attributes: the size of the sudoku board, the total number of threads, the thread number, and a pointer to the sudoku board given to all threads. We require all this because instead of passing the tasks to be performed to each thread, we let the thread calculate which tasks it is supposed to perform by numbering all the tasks, which is described later.

One of the global arrays used is a result_thread array consisting of maximum size 300, since the maximum size of the board can be 100, and thus, the maximum number of checks we would have to perform is 300. It will simply be a binary array, with 0 representing that specific check was invalid and 1 being the check was valid.

We then check the number of threads. If the size of the sudoku is N, then the total number of tasks would be 3*N; N rows, N columns and N grids. If the number of threads is greater than 3*N, we only

make 3*N threads, since we do not wish to break any row, column, or grid into further subtasks.

We then define the thread_parameter array and initialize the array with the object attributes which are the size of the Sudoku, total number of threads, thread number and the pointer to the sudoku board.

We now start the timer. The time measured is only for thread creation and execution. This is where the execution using OpenMP and Pthreads differs. In the PThreads implementation, we create the number of threads required and call the function check_sudoku, while in the OpenMP implementation, we call the check_sudoku function in parallel.

This is an important step because while the **implementation for OpenMP could be made far easier**, removing the global array and the class and its array, I decided to keep it to **ensure uniformity in comparison between the two.**

The check_sudoku function runs as follows -> It assumes the 3N tasks are numbered from 0 to 3N-1, where tasks having index in [0,N) are the rows, [N,2N) are columns, and [2N , 3N) are grids. We allow each thread to calculate which tasks it is required to do -> for example if the number of tasks are 48, and the total number of threads are 12, then thread 5 will be required to perform the following:

1. Task 5- checking row 5
2. Task 17- checking column 1
3. Task 29- checking column 13
4. Task 41- checking grid 9

For performing the check, we simply take an array with N elements and set it to 0. If we encounter a number during the check, we update the value in the array to 1. If for a particular number, we

encounter the array value to be 1, we find that a repetition has occurred and we can say the check is invalid.

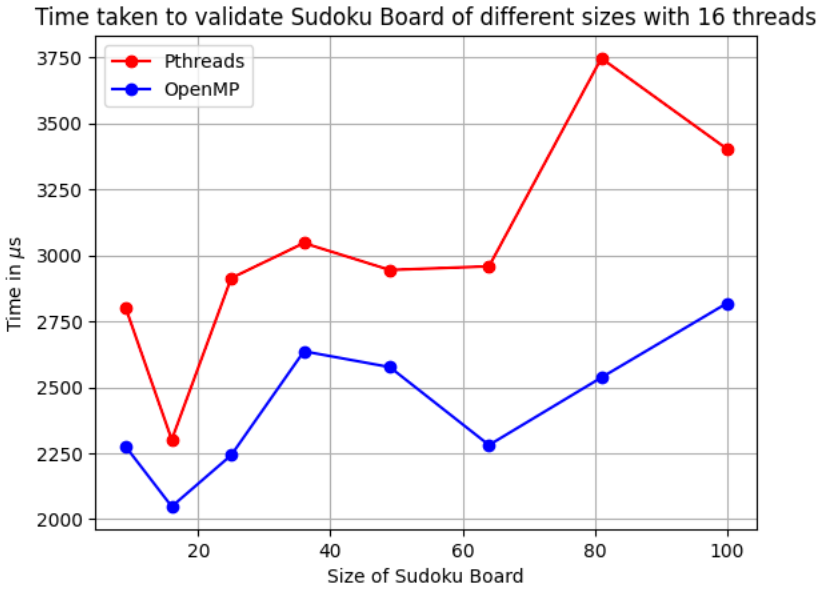
Finally, we print the outputs of each thread, and the time taken and the overall result. We compute the overall result using the fact that the sum of the results_thread array should be exactly 3N if all the checks were valid.

Analysis:

The first graph is as expected; As we increase the board size, the amount of time required for validation also increases.

However, the second graph is not as per expectations; As we increase the number of threads, the time should decrease as the parallelism increases. However, we observe the opposite, time increases as the number of threads increase. I believe this is because increasing the threads does not simply improve performance, it becomes a tradeoff between overhead such as time taken for creation of threads, context switching, time taken to access the common memory, etc. vs the time saved in parallelism. Since all the checking of the rows, columns and grids are constant, majority contribution comes from the overhead, which increases as the number of threads increase.

1. Time Taken for Validation of Sudoku Board using constant number of threads



2. Time taken for Validation of 25 x 25 Sudoku Board using variable number of threads