Validating Sudoku by Multithreading

Aaryan, CO21BTECH11001

Contents of the zip file:

Input files: 9x9.txt, 25x25.txt, 36x36.txt, 49x49.txt, 64x64.txt, 81x81.txt 100x100.txt files contains the sudoku of the size to be given to the program.
Report: Assign2_Report_CO21BTECH11001.pdf
Source code: Assgn2SrcPthreadCO21BTECH11001.c, Assgn2SrcOpenmpCO21BTECH11001.c
Readme file: Assgn2_README_CO21BTECH11001.txt

Working of code:

It can be described as follows:

1. In main function, input is taken from input file using fscanf.
   Let size of sudoku = N
   Let number of threads to be created = k
   A 2D array named sudoku is created and input is taken from file.

2. While using openmp, number of threads to be created is set to k using *omp_set_num_threads(k)*.

3. An array *arguments* is created by using a struct *thread_args,* to pass arguments to the thread.
   i^{th} element of the array *arguments* is passed as an argument to the i^{th} thread.
   The struct *thread_args* has the following contents:
   a. *size*: It is the total number of rows and columns and subgrids that the thread is going to check.
   b. *start_row*: An array to store starting row of row or column or subgrid.

c. *end_row*: An array to store the ending row of row or column or subgrid.
d. *start_col*: An array to store starting column of row or column or subgrid.
e. *end_col* : An array to store the ending column of row or column or subgrid.
f. *check*: An array, which will store whether the row or column or subgrid is valid or invalid.

4. If the size of sudoku is NxN, then there are total 3N elements to be checked (N rows, N columns, N subgrids).

   Let $div = (3N)/k$
   Let $mod = (3N)\%k$

   Distribution is as follows:
   1st thread: (div + 1) elements

   2nd thread: (div+1) elements
   .
   .
   .

   (mod) th thread: (div + 1) elements

   (mod+1) th thread: (div) elements
   .
   .
   .
   (k)th thread: (div) elements

   Size of arrays *start_col, start_row, end_col, end_row, check* is set to the number of things to be checked by the thread.

5. While using pthreads, *runner* function will do the work of a thread. Checking of any element (row or column or subgrid) is done by storing frequency of each element appeared in element.

At last, it is checked if the frequency of every element is 1. If yes, the element is valid, else the element is invalid.

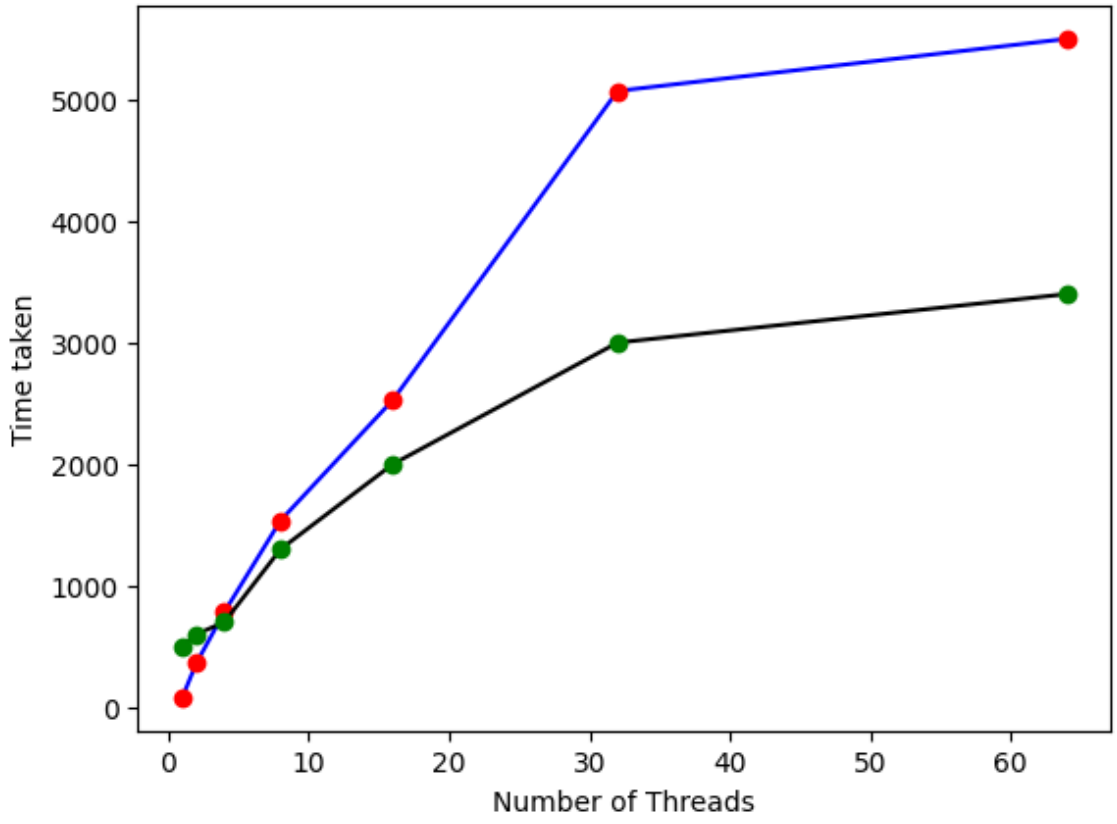
6. While using openmp, same logic is performed by a code inside the parallel part of openmp.

7. Now the main thread will perform the final checking and print the log in the output file.
   The main thread will traverse the array *arguments* and see if all elements are valid. If yes, then sudoku is valid. Else, it is invalid.

8. Now, the main thread will print the time taken and value calculated in *output.txt* file.

Results :

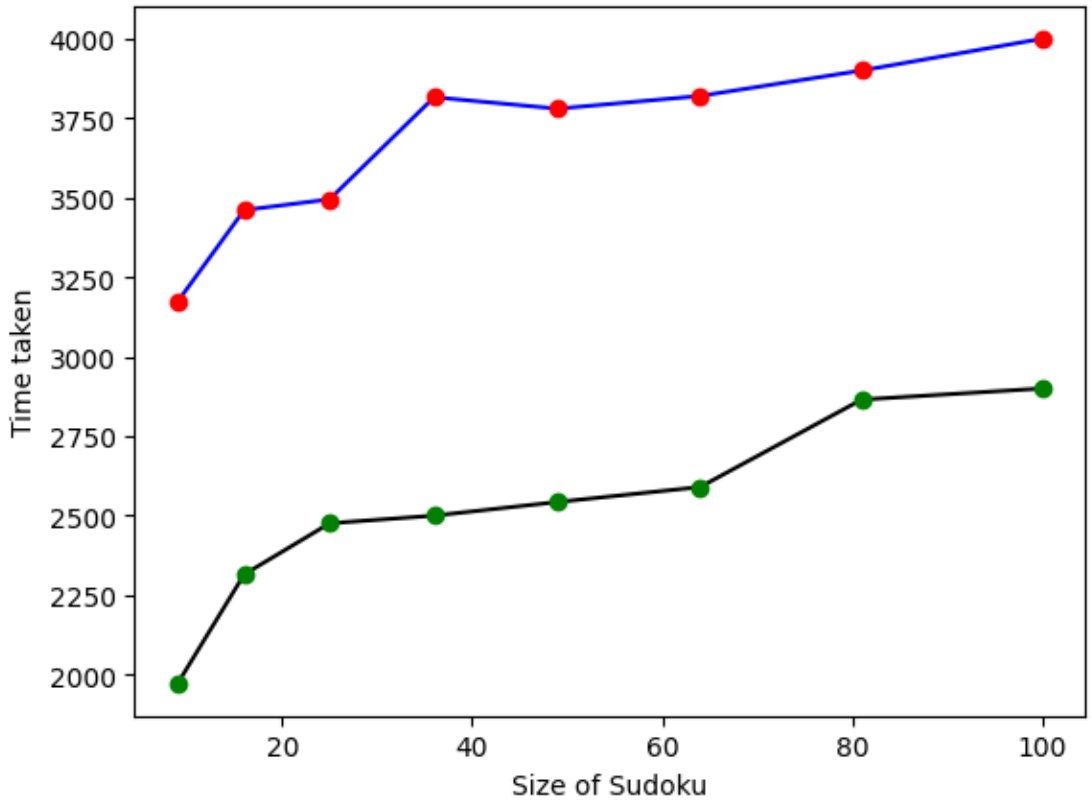
1. When the size of sudoku is kept constant at 25x25 and the number of threads is increased gradually, the time taken to execute mostly increases.



In the above plot, blue curve represent openmp and black curve represent pthread.
The reason for this behavior is that due to small size of sudoku, number of computations to be performed is just 1875. Therefore, the time taken to manage threads adds a significant amount of time to total time and increasing threads increases the total time to execute.

However, when the same plot is plotted for a 2500x2500 sudoku, the time taken decreases with increasing the threds.

2. When the number of threads is kept constant and the size of sudoku is increased, the time taken to execute mostly increases.



In the above plot, blue curve represent openmp and black curve represent pthread.
Again, the reason for this behavior is that due to small sizes of sudoku. Since the number of computations to be performed are quite low, therefore, changing the size of sudoku hardly changes time taken to execute.

Also, the plot may be different for different systems due to the same reason.