Operating Systems CS3523
Programming Assignment 2
ES21BTECH11022

PThreads

High-Level Design of Code

1. Taking Input: The program uses command line arguments to take the input file containing values of k and n. Here n is the size of the sudoku square, and k is the number of threads used. Apart from n and k we also input a matrix from the input file. This is the n * n sudoku matrix that we'll check for correctness. Here we also check for two errors an improper number of command line arguments and the case of fopen returning a NULL pointer. In both cases, the error is printed onto the output file.

2. CreatingThreads: For thread creation, we first initialize an array of k pthread ids. Then the program creates k threads using a for loop. As each thread executes one function with one void* argument, we pass a pointer to a struct type named arg in the code. We create an array of k such arg structs to give to each thread. This struct contains one integer j representing the thread number and the input matrix. Then the for loop is run creating the k threads. Then each thread runs the function func and after completion, we join all threads using pthread_join.

3. Function func: Each thread runs this function. The initialization of the argument of type arg occurs in the for loop which creates the thread itself. This function is a typical sudoku checker but optimized for parallel programming. For a n * n sudoku grid to be valid each of its rows should have all the distinct numbers from 1 to n and so should each column. Also, we check n grids of size m * m (n = m^2) We first proceed to check rows, then columns, and finally grids.

4. Final Output File: For this program, we have only one main output file named "OutMain.tex". Firstly in this file, the logs of each thread are present. These are all intertwined due to parallel execution. Then if all the entries for each row/column/grid are valid then the string “Sudoku is valid” is present. If even one of the entries were invalid we would get a “Sudoku is invalid” comment. And finally, in the end, we have the time taken to execute the program present in seconds.

5. Early Termination: In the case that one of the threads finds an invalid row/column/grid it changes the indicator complete to 1 and preemptively exits the function func. Before returning it also prints to the file which row/column/grid is invalid. If complete is set to 1 we proceed to cancel all the following threads. Else we wait for the threads via the pthread_join.

Low-Level Design of Code

Considering low-level design, we first create k thread ids using an array of tid_t. Then we simply create k threads using,

pthread_create(&threads[i], NULL, perfect, (void*) &args[i]);

Thus one thread is created for each tid with NULL attributes and each of them executes the function func with argument args[i]. The variable args is basically of type struct arg* and to pass it as an argument we perform type casting. Then in the function, we check the sudoku for validity by checking its rows, columns, and grids. This marks the end of tasks for one thread. In the program k such threads are created. Finally, we join all these k threads using,

pthread_join(pthread[i], NULL);

In the context of task splitting I have split the task to be of three types: i) n Row checks ii) n Column checks iii) n Grid checks. Now each thread first checks approximately n/k rows. This is ensured by making the thread j check row numbers that fall into the set

$$S_j = \{x \mid x = j + ck, c = 0, 1, 2, \ldots, x \leq n\}$$

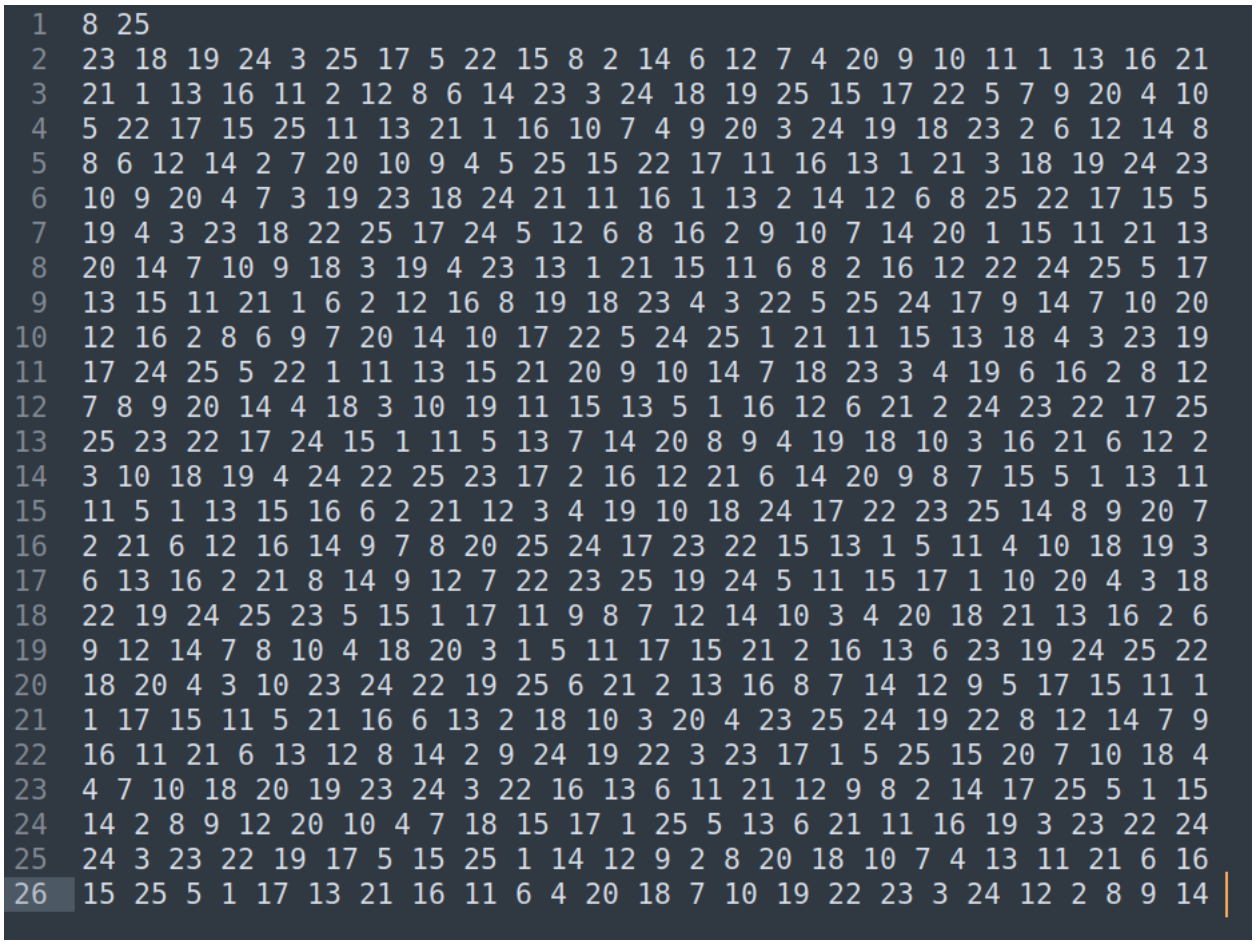
This also holds true for columns and grids. For numbering grids, we first traverse row-wise and move downwards.

For refining the code I have also implemented early termination of threads. As soon as any of the threads finds an invalid row/column/grid it changes the value of the complete variable to 1 and exits. Soon after in the main thread instead of the join procedure we cancel all the threads using the command.

pthread_cancel(threads[i]);

Analysis of Output:

Output file using n = 25 and k = 8
And the following input matrix:



```
8 25
23 18 19 24 3 25 17 5 22 15 8 2 14 6 12 7 4 20 9 10 11 1 13 16 21
21 1 13 16 11 2 12 8 6 14 23 3 24 18 19 25 15 17 22 5 7 9 20 4 10
5 22 17 15 25 11 13 21 1 16 10 7 4 9 20 3 24 19 18 23 2 6 12 14 8
8 6 12 14 2 7 20 10 9 4 5 25 15 22 17 11 16 13 1 21 3 18 19 24 23
10 9 20 4 7 3 19 23 18 24 21 11 16 1 13 2 14 12 6 8 25 22 17 15 5
19 4 3 23 18 22 25 17 24 5 12 6 8 16 2 9 10 7 14 20 1 15 11 21 13
20 14 7 10 9 18 3 19 4 23 13 1 21 15 11 6 8 2 16 12 22 24 25 5 17
13 15 11 21 1 6 2 12 16 8 19 18 23 4 3 22 5 25 24 17 9 14 7 10 20
12 16 2 8 6 9 7 20 14 10 17 22 5 24 25 1 21 11 15 13 18 4 3 23 19
17 24 25 5 22 1 11 13 15 21 20 9 10 14 7 18 23 3 4 19 6 16 2 8 12
7 8 9 20 14 4 18 3 10 19 11 15 13 5 1 16 12 6 21 2 24 23 22 17 25
25 23 22 17 24 15 1 11 5 13 7 14 20 8 9 4 19 18 10 3 16 21 6 12 2
3 10 18 19 4 24 22 25 23 17 2 16 12 21 6 14 20 9 8 7 15 5 1 13 11
11 5 1 13 15 16 6 2 21 12 3 4 19 10 18 24 17 22 23 25 14 8 9 20 7
2 21 6 12 16 14 9 7 8 20 25 24 17 23 22 15 13 1 5 11 4 10 18 19 3
6 13 16 2 21 8 14 9 12 7 22 23 25 19 24 5 11 15 17 1 10 20 4 3 18
22 19 24 25 23 5 15 1 17 11 9 8 7 12 14 10 3 4 20 18 21 13 16 2 6
9 12 14 7 8 10 4 18 20 3 1 5 11 17 15 21 2 16 13 6 23 19 24 25 22
18 20 4 3 10 23 24 22 19 25 6 21 2 13 16 8 7 14 12 9 5 17 15 11 1
1 17 15 11 5 21 16 6 13 2 18 10 3 20 4 23 25 24 19 22 8 12 14 7 9
16 11 21 6 13 12 8 14 2 9 24 19 22 3 23 17 1 5 25 15 20 7 10 18 4
4 7 10 18 20 19 23 24 3 22 16 13 6 11 21 12 9 8 2 14 17 25 5 1 15
14 2 8 9 12 20 10 4 7 18 15 17 1 25 5 13 6 21 11 16 19 3 23 22 24
24 3 23 22 19 17 5 15 25 1 14 12 9 2 8 20 18 10 7 4 13 11 21 6 16
15 25 5 1 17 13 21 16 11 6 4 20 18 7 10 19 22 23 3 24 12 2 8 9 14
```

Output:

```
Thread 1 checks row 1 and is valid
Thread 1 checks row 9 and is valid
Thread 1 checks row 17 and is valid
Thread 1 checks row 25 and is valid
Thread 1 checks column 1 and is valid
Thread 1 checks column 9 and is valid
Thread 1 checks column 17 and is valid
Thread 1 checks column 25 and is valid
Thread 1 checks grid 1 and is valid
Thread 1 checks grid 9 and is valid
Thread 1 checks grid 17 and is valid
Thread 1 checks grid 25 and is valid
Thread 3 checks row 3 and is valid
Thread 3 checks row 11 and is valid
Thread 3 checks row 19 and is valid
Thread 3 checks column 3 and is valid
Thread 3 checks column 11 and is valid
Thread 3 checks column 19 and is valid
Thread 3 checks grid 3 and is valid
Thread 3 checks grid 11 and is valid
Thread 3 checks grid 19 and is valid
Thread 2 checks row 2 and is valid
Thread 2 checks row 10 and is valid
Thread 2 checks row 18 and is valid
Thread 2 checks column 2 and is valid
Thread 2 checks column 10 and is valid
Thread 2 checks column 18 and is valid
Thread 2 checks grid 2 and is valid
Thread 2 checks grid 10 and is valid
Thread 2 checks grid 18 and is valid
Thread 4 checks row 4 and is valid
Thread 4 checks row 12 and is valid
Thread 4 checks row 20 and is valid
Thread 4 checks column 4 and is valid
```

```
35  Thread 4 checks column 12 and is valid
36  Thread 4 checks column 20 and is valid
37  Thread 4 checks grid 4 and is valid
38  Thread 4 checks grid 12 and is valid
39  Thread 4 checks grid 20 and is valid
40  Thread 5 checks row 5 and is valid
41  Thread 5 checks row 13 and is valid
42  Thread 5 checks row 21 and is valid
43  Thread 5 checks column 5 and is valid
44  Thread 5 checks column 13 and is valid
45  Thread 5 checks column 21 and is valid
46  Thread 5 checks grid 5 and is valid
47  Thread 5 checks grid 13 and is valid
48  Thread 5 checks grid 21 and is valid
49  Thread 6 checks row 6 and is valid
```

```
51  Thread 6 checks row 22 and is valid
52  Thread 6 checks column 6 and is valid
53  Thread 6 checks column 14 and is valid
54  Thread 6 checks column 22 and is valid
55  Thread 6 checks grid 6 and is valid
56  Thread 6 checks grid 14 and is valid
57  Thread 6 checks grid 22 and is valid
58  Thread 7 checks row 7 and is valid
59  Thread 7 checks row 15 and is valid
60  Thread 7 checks row 23 and is valid
61  Thread 7 checks column 7 and is valid
62  Thread 7 checks column 15 and is valid
63  Thread 7 checks column 23 and is valid
64  Thread 7 checks grid 7 and is valid
65  Thread 7 checks grid 15 and is valid
66  Thread 7 checks grid 23 and is valid
67  Thread 8 checks row 8 and is valid
68  Thread 8 checks row 16 and is valid
69  Thread 8 checks row 24 and is valid
70  Thread 8 checks column 8 and is valid
71  Thread 8 checks column 16 and is valid
72  Thread 8 checks column 24 and is valid
73  Thread 8 checks grid 8 and is valid
74  Thread 8 checks grid 16 and is valid
75  Thread 8 checks grid 24 and is valid
76  Sudoku is valid
77  The time elapsed is 0.001908 seconds
```

OpenMP

High-Level Design of Code

1. Taking Input: The program uses command line arguments to take the input file containing values of k and n. Here n is the size of the sudoku square, and k is the number of threads used. Apart from n and k we also input a matrix from the input file. This is the n * n sudoku matrix that we'll check for correctness. Here we also check for two errors an improper number of command line arguments and the case of fopen returning a NULL pointer. In both cases, the error is printed onto the output file.

2. CreatingThreads: Thread creation is really simple using OMP threads. We use the #pragma omp parallel to run the following code block via parallelism without user intervention. We also mention num_threads(k) to specify the number of threads. We also use the #pragma omp for to run the for loop in parallel. This ensures that all the iterations of the for loop are split equally among the threads.

3. Code Block inside pragma omp directive: This code block is a typical sudoku checker. For a n * n sudoku grid to be valid each of its rows should have all the distinct numbers from 1 to n and so should each column. Also, we check n grids of size m * m (n = m^2) We first proceed to check rows, then columns, and finally grids. Now the directive ensures that the task is run in parallel using k threads.

4. Final Output File: For this program, we have only one main output file named "OutMain.tex". Firstly in this file, the logs of each thread are present. These are all intertwined due to parallel execution. Then if all the entries for each row/column/grid are valid then the string “Sudoku is valid” is present. If even one of the entries were invalid we would get a “Sudoku is invalid” comment. And finally, in the end, we have the time taken to execute the program present in seconds.

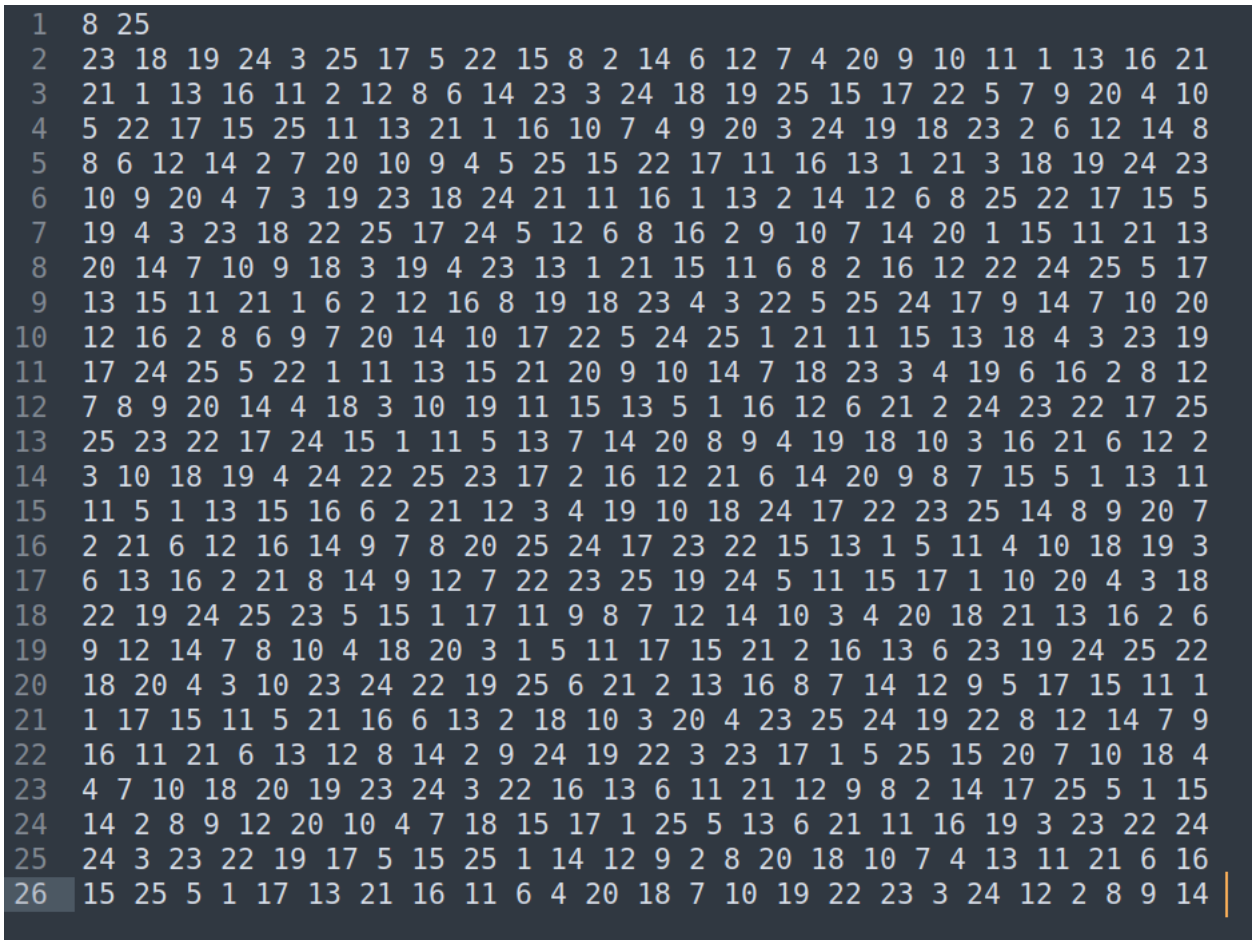
Low-Level Design of Code

Considering the low-level design, we see that the #pragma omp parallel directive is used to run the following code block in parallel. By default, the number of threads is usually set to the number of cores. But by giving the specifier number_threads(k) we can change the number of threads.

Next is the #pragma omp for directive. This ensures that all the iterations of the for loop are split among the k threads. Another specifier that isn’t in the code is the private and shared variables. As all variables used for checking the sudoku matrix are defined within the directive block they are set to private by default.

Analysis of Output:

Output file using n = 25 and k = 8
And the following input matrix:



```
8 25
23 18 19 24 3 25 17 5 22 15 8 2 14 6 12 7 4 20 9 10 11 1 13 16 21
21 1 13 16 11 2 12 8 6 14 23 3 24 18 19 25 15 17 22 5 7 9 20 4 10
5 22 17 15 25 11 13 21 1 16 10 7 4 9 20 3 24 19 18 23 2 6 12 14 8
8 6 12 14 2 7 20 10 9 4 5 25 15 22 17 11 16 13 1 21 3 18 19 24 23
10 9 20 4 7 3 19 23 18 24 21 11 16 1 13 2 14 12 6 8 25 22 17 15 5
19 4 3 23 18 22 25 17 24 5 12 6 8 16 2 9 10 7 14 20 1 15 11 21 13
20 14 7 10 9 18 3 19 4 23 13 1 21 15 11 6 8 2 16 12 22 24 25 5 17
13 15 11 21 1 6 2 12 16 8 19 18 23 4 3 22 5 25 24 17 9 14 7 10 20
12 16 2 8 6 9 7 20 14 10 17 22 5 24 25 1 21 11 15 13 18 4 3 23 19
17 24 25 5 22 1 11 13 15 21 20 9 10 14 7 18 23 3 4 19 6 16 2 8 12
7 8 9 20 14 4 18 3 10 19 11 15 13 5 1 16 12 6 21 2 24 23 22 17 25
25 23 22 17 24 15 1 11 5 13 7 14 20 8 9 4 19 18 10 3 16 21 6 12 2
3 10 18 19 4 24 22 25 23 17 2 16 12 21 6 14 20 9 8 7 15 5 1 13 11
11 5 1 13 15 16 6 2 21 12 3 4 19 10 18 24 17 22 23 25 14 8 9 20 7
2 21 6 12 16 14 9 7 8 20 25 24 17 23 22 15 13 1 5 11 4 10 18 19 3
6 13 16 2 21 8 14 9 12 7 22 23 25 19 24 5 11 15 17 1 10 20 4 3 18
22 19 24 25 23 5 15 1 17 11 9 8 7 12 14 10 3 4 20 18 21 13 16 2 6
9 12 14 7 8 10 4 18 20 3 1 5 11 17 15 21 2 16 13 6 23 19 24 25 22
18 20 4 3 10 23 24 22 19 25 6 21 2 13 16 8 7 14 12 9 5 17 15 11 1
1 17 15 11 5 21 16 6 13 2 18 10 3 20 4 23 25 24 19 22 8 12 14 7 9
16 11 21 6 13 12 8 14 2 9 24 19 22 3 23 17 1 5 25 15 20 7 10 18 4
4 7 10 18 20 19 23 24 3 22 16 13 6 11 21 12 9 8 2 14 17 25 5 1 15
14 2 8 9 12 20 10 4 7 18 15 17 1 25 5 13 6 21 11 16 19 3 23 22 24
24 3 23 22 19 17 5 15 25 1 14 12 9 2 8 20 18 10 7 4 13 11 21 6 16
15 25 5 1 17 13 21 16 11 6 4 20 18 7 10 19 22 23 3 24 12 2 8 9 14
```

Output:

```
Thread 1 checks row 1 and is valid
Thread 1 checks row 2 and is valid
Thread 1 checks row 3 and is valid
Thread 1 checks row 4 and is valid
Thread 2 checks row 5 and is valid
Thread 5 checks row 14 and is valid
Thread 2 checks row 6 and is valid
Thread 6 checks row 17 and is valid
Thread 5 checks row 15 and is valid
Thread 3 checks row 8 and is valid
Thread 6 checks row 18 and is valid
Thread 4 checks row 11 and is valid
Thread 6 checks row 19 and is valid
Thread 3 checks row 9 and is valid
Thread 3 checks row 10 and is valid
Thread 4 checks row 12 and is valid
Thread 4 checks row 13 and is valid
Thread 2 checks row 7 and is valid
Thread 7 checks row 20 and is valid
Thread 8 checks row 23 and is valid
Thread 7 checks row 21 and is valid
Thread 8 checks row 24 and is valid
Thread 7 checks row 22 and is valid
Thread 8 checks row 25 and is valid
Thread 5 checks row 16 and is valid
Thread 1 checks column 1 and is valid
Thread 8 checks column 23 and is valid
Thread 5 checks column 14 and is valid
Thread 7 checks column 20 and is valid
Thread 1 checks column 2 and is valid
Thread 8 checks column 24 and is valid
Thread 1 checks column 3 and is valid
Thread 5 checks column 15 and is valid
Thread 1 checks column 4 and is valid
```

```
35  Thread 5 checks column 16 and is valid
36  Thread 4 checks column 11 and is valid
37  Thread 6 checks column 17 and is valid
38  Thread 4 checks column 12 and is valid
39  Thread 3 checks column 8 and is valid
40  Thread 4 checks column 13 and is valid
41  Thread 6 checks column 18 and is valid
42  Thread 6 checks column 19 and is valid
43  Thread 7 checks column 21 and is valid
44  Thread 3 checks column 9 and is valid
45  Thread 7 checks column 22 and is valid
46  Thread 3 checks column 10 and is valid
47  Thread 2 checks column 5 and is valid
48  Thread 8 checks column 25 and is valid
49  Thread 2 checks column 6 and is valid
50  Thread 2 checks column 7 and is valid
51  Thread 1 checks grid 1 and is valid
52  Thread 8 checks grid 23 and is valid
53  Thread 5 checks grid 14 and is valid
54  Thread 4 checks grid 11 and is valid
55  Thread 2 checks grid 5 and is valid
56  Thread 1 checks grid 2 and is valid
57  Thread 2 checks grid 6 and is valid
58  Thread 1 checks grid 3 and is valid
59  Thread 8 checks grid 24 and is valid
60  Thread 1 checks grid 4 and is valid
61  Thread 2 checks grid 7 and is valid
62  Thread 7 checks grid 20 and is valid
63  Thread 7 checks grid 21 and is valid
64  Thread 8 checks grid 25 and is valid
65  Thread 6 checks grid 17 and is valid
66  Thread 3 checks grid 8 and is valid
67  Thread 4 checks grid 12 and is valid
```

```
67  Thread 4 checks grid 12 and is valid
68  Thread 3 checks grid 9 and is valid
69  Thread 4 checks grid 13 and is valid
70  Thread 3 checks grid 10 and is valid
71  Thread 5 checks grid 15 and is valid
72  Thread 6 checks grid 18 and is valid
73  Thread 5 checks grid 16 and is valid
74  Thread 6 checks grid 19 and is valid
75  Thread 7 checks grid 22 and is valid
76  Sudoku is valid
77  The time elapsed is 0.003571 seconds
```

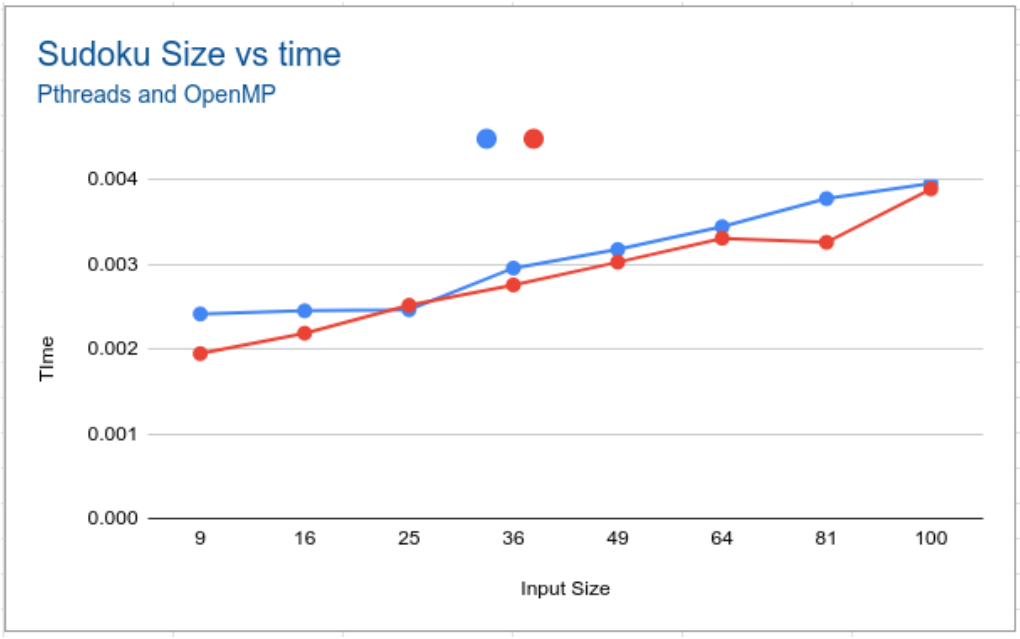
Graphs:

Now I have plotted graphs of Sudoku size vs time taken and Number of threads vs time taken. Each of these was done for both OpenMP and Pthread and compared. Here are the following graphs:

Sudoku Size	TIme Readings					Average Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
9	0.002567	0.002313	0.002498	0.002515	0.002181	0.0024148
16	0.002699	0.002842	0.002295	0.002219	0.002205	0.002452
25	0.002417	0.002887	0.002346	0.002346	0.002325	0.0024642
36	0.003083	0.00304	0.00288	0.002994	0.002781	0.0029556
49	0.003541	0.003359	0.002975	0.002714	0.00329	0.0031758
64	0.003791	0.003538	0.003344	0.003034	0.003517	0.0034448
81	0.003866	0.003638	0.003928	0.003747	0.003703	0.0037764
100	0.004808	0.004061	0.004407	0.004082	0.002432	0.003958

Sudoku Size	TIme Readings					Average Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
9	0.002527	0.001482	0.001335	0.001774	0.002613	0.0019462
16	0.002903	0.001981	0.001548	0.003154	0.001352	0.0021876
25	0.002308	0.002318	0.002085	0.002892	0.002981	0.0025168
36	0.002291	0.003522	0.002025	0.002924	0.003022	0.0027568
49	0.00256	0.002592	0.00383	0.002545	0.003605	0.0030264
64	0.003876	0.002088	0.003037	0.003965	0.003576	0.0033084
81	0.004143	0.00237	0.003705	0.002594	0.003489	0.0032602
100	0.004007	0.004213	0.0025	0.004094	0.004642	0.0038912

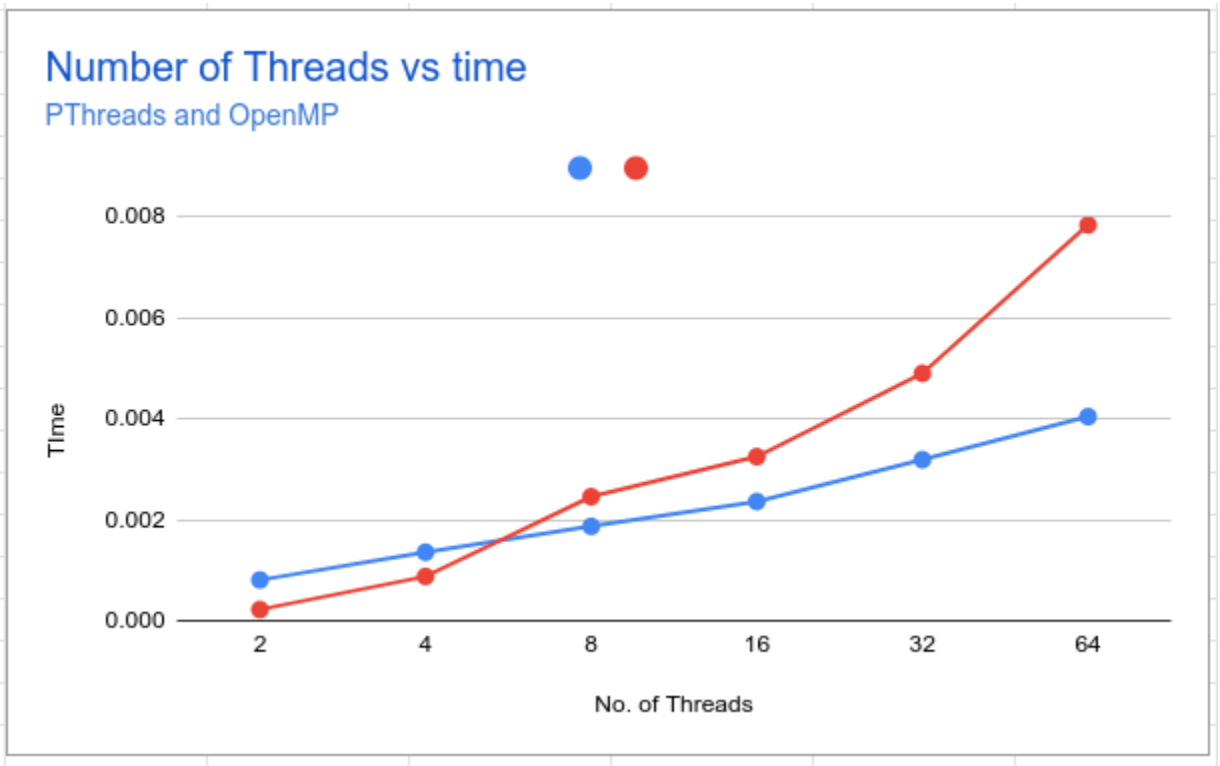
The first table is for Pthreads while the second graph is for OpenMP.



Thread numbers	Time Readings					Average TIme
	time1	time2	time3	time4	time5	
2	0.000922	0.000737	0.000747	0.000671	0.000955	0.0008064
4	0.001525	0.001201	0.001402	0.001573	0.001076	0.0013554
8	0.001969	0.001768	0.001926	0.001959	0.001719	0.0018682
16	0.001751	0.002499	0.00244	0.002541	0.00257	0.0023602
32	0.003098	0.003318	0.003316	0.002911	0.003304	0.0031894
64	0.003322	0.004214	0.004091	0.004953	0.003632	0.0040424

Thread numbers	Time Readings					Average TIme
	time1	time2	time3	time4	time5	
2	0.000281	0.000279	0.000161	0.000201	0.000181	0.0002206
4	0.000797	0.000946	0.001009	0.000823	0.000803	0.0008756
8	0.002784	0.002624	0.001738	0.002639	0.002498	0.0024566
16	0.002682	0.003093	0.003378	0.00326	0.003829	0.0032484
32	0.005289	0.005018	0.004375	0.004754	0.005059	0.004899
64	0.007982	0.008021	0.007817	0.007272	0.0081	0.0078384

The first table is for Pthreads while the second graph is for OpenMP.



In both the graphs blue line graph corresponds to Pthreads while red line graph corresponds to OpenMP threads.