Name: Vishal Vijay Devadiga

Roll Number: CS21BTECH11061

Code Flow

- Opens input file in main and gets values of threads(k), and size of matrix(n) such that $k \not> 3 * n$
- Create k threads that execute the threadsolve(Present in main of openMp file and threadsolve function in pthread file) function. Parent stores the threadID of the threads in an array.
- Each thread checks row/column/grid based on its threadNo.
- There are $3 * n$ operations and is divided among k threads. Threads prints the result directly to the file.
- If a thread finds that given row/column/grid is invalid, it sets a global variable(validsud) to 0. All threads then stop after checking the variable either at the start of an iteration or before printing.
- After all threads finish executing(exits), the parent thread prints whether the sudoku is valid or not and time taken for the calculations above to the output file.

Plots

My system has the processor AMD Ryzen 7 5800u, which has 8 cores and 16 threads. Note that **all measurements of time are in microseconds(μs)**

Some Observations

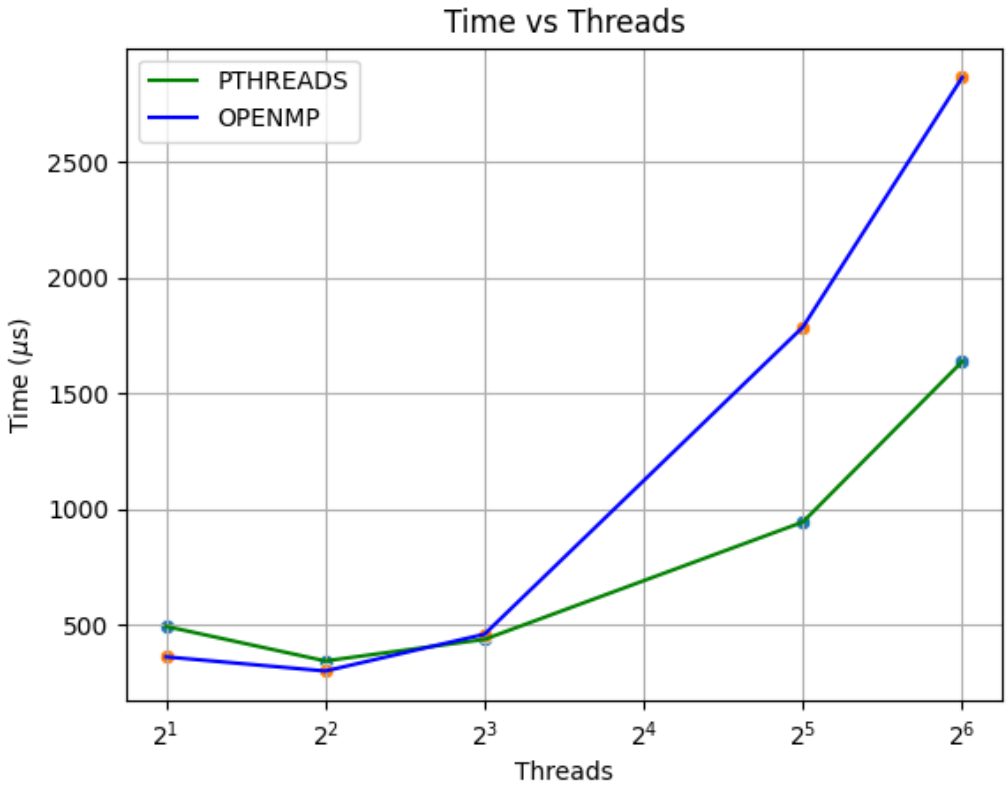
Printing the output file takes the most amount of time in the program. The actual program without printing the output file takes a lot less time.

`fprintf`, `ofstream` are thread-safe depending on the implementation(compiler). It is thread-safe on linux and windows. Thus, there was no need to use a mutex while writing to the output file. Mutexes on a calculative block of code significantly reduces the speedup(**Amadahl's law**).

Increasing the number of threads after a certain amount actually increased the execution time of the program. Example: 512 threads vs 8 threads on my machine. One would expect the 512 threads program to execute in less time, but actually 8 threads program executes in less time.

Pthreads is much lower level in design, that is, we get fine-grained control over thread creation, thread work allocation and so on. Openmp on the other hand, is much faster to code with, as it removes all the pain points with working with pthreads. However, as shown below, openmp is not as fast as pthreads.

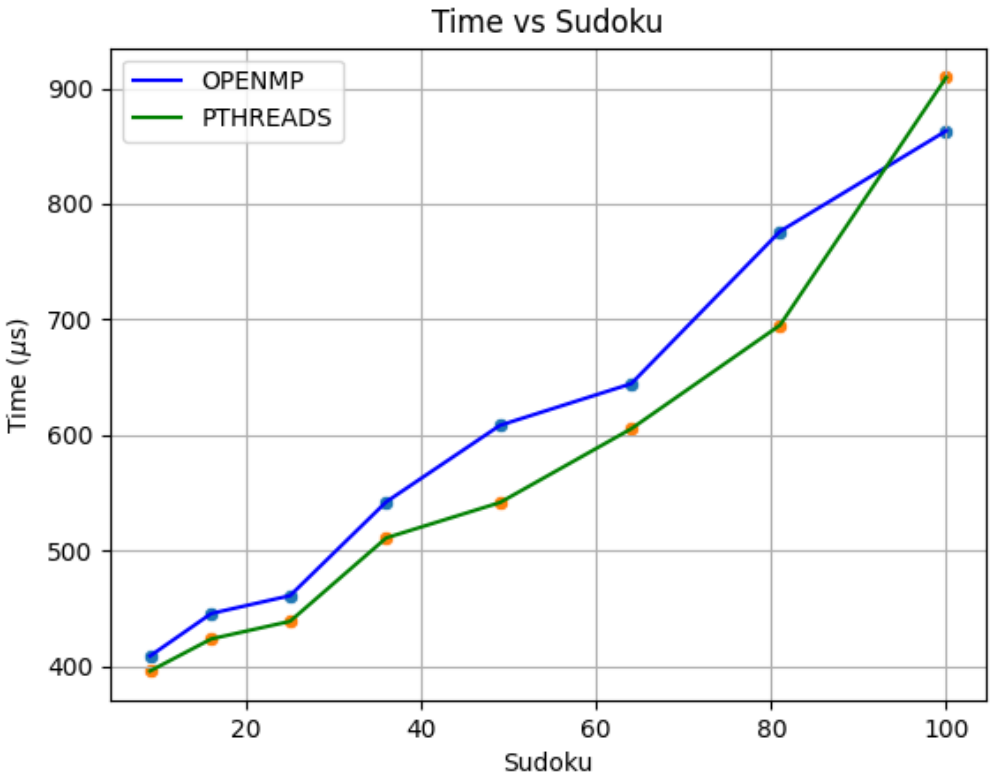
Time vs Threads



Threads	Sudoku	API	T_1	T_2	T_3	T_4	T_5	Average Time
2	25 x 25	OPENMP	364	322	395	354	382	363.4
4	25 x 25	OPENMP	303	288	307	295	318	302.2
8	25 x 25	OPENMP	458	426	510	444	467	459
32	25 x 25	OPENMP	1778	1849	1601	1827	1878	2233.2
64	25 x 25	OPENMP	2643	3105	2763	3080	2725	2863.2
2	25 x 25	PTHREADS	490	522	445	532	485	494.8
4	25 x 25	PTHREADS	294	363	382	381	309	345.8
8	25 x 25	PTHREADS	405	481	470	431	407	438.8
32	25 x 25	PTHREADS	925	1102	812	1041	846	945.2
64	25 x 25	PTHREADS	1492	1501	1845	1775	1574	1637.4

By the graph, it is indicative that increasing the number of threads does increase the performance of the program, but not in a linear way. After a certain limit, increasing the number of threads does not yield increased performance and might even degrade performance. Amadahl's law(Sequential code in the parent thread limits the performance increase,), the number of cores in my system(8 cores) and the increased context switching is the reason why the performance does not increase after using more than 16 threads for program, rather degardes even more.

Time vs Points



Sudoku	Threads	API	T_1	T_2	T_3	T_4	T_5	Average Time
9 x 9	8	OPENMP	416	408	436	366	418	408.8
16 x 16	8	OPENMP	469	485	402	448	424	445.6
25 x 25	8	OPENMP	458	426	510	444	467	459
36 x 36	8	OPENMP	540	504	576	532	559	542.2
49 x 49	8	OPENMP	653	591	594	578	626	608.4
64 x 64	8	OPENMP	651	664	617	613	677	644.4
81 x 81	8	OPENMP	841	733	783	760	764	776.2
100x100	8	OPENMP	810	896	865	901	844	863.2
9 x 9	8	PTHREADS	385	377	410	416	391	394.8
16 x 16	8	PTHREADS	410	432	400	459	417	423.6
25 x 25	8	PTHREADS	405	481	470	431	407	438.8
36 x 36	8	PTHREADS	503	494	541	492	525	511
49 x 49	8	PTHREADS	550	567	562	474	555	541.6
64 x 64	8	PTHREADS	564	642	607	622	592	605.4
81 x 81	8	PTHREADS	719	701	622	698	735	695
100x100	8	PTHREADS	1041	942	869	822	873	909.4

Size of sudoku vs Time is a linear graph, since the tasks are equally distributed between the threads and threads is constant between the tests.