Operating Systems2(CS3523)
Programming Assignment 2: Validating Sudoku Solution

Govinda Rohith Y
CS21BTECH11062

Plot 1:

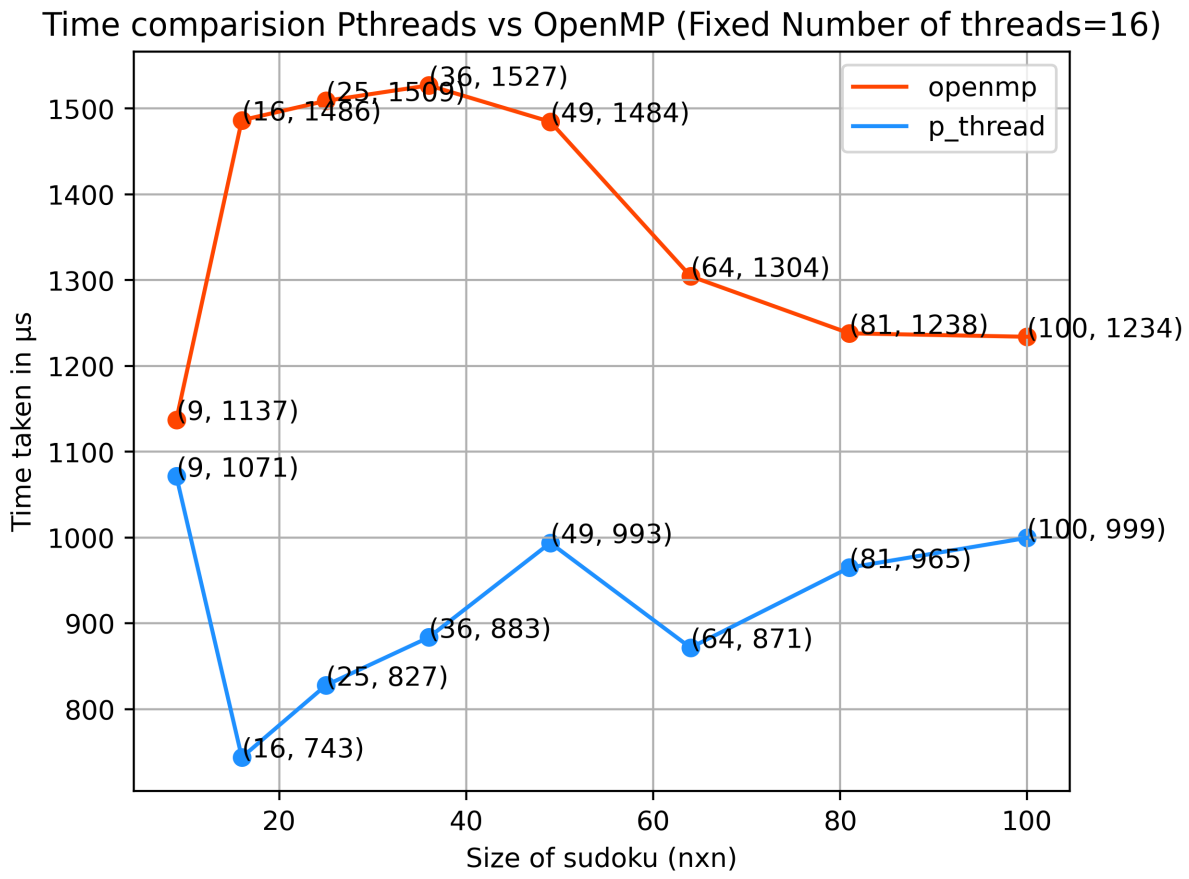

Figure 1: Time comparison P_Thread vs OpenMP Threads (Fixed number of Threads)

Analysis:

PThread plot Analysis:

The trend of the plot by using p_threads is increasing along with some anomalies. This is obvious because as the Sudoku size is increasing the number of checks to be done are increasing so the time taken is increasing as the size of sudoku is increasing.

OpenMP plot Analysis:

The trend of the plot by using OpenMP threads is increasing first and then decreases along with some anomalies.The first increasing trend is obvious because as the Sudoku size is increasing the number of checks to be done are increasing so the time taken is increasing as the size of sudoku is increasing. But then the trend decreases, one reason might be the fast execution of OpenMP threads. OpenMP threads are easily scaled than pthreads that is OpenMP threads are costlier to create(because they are portable) but operate faster compared to PThreads. So may be due to faster execution and perfect thread management the total time consumed decreases. Another reason might be context switching which is decided by OS's CPU Scheduler.

PThread vs OpenMP

From the Figure 1 we can refer that PThreads execute faster than OpenMP threads. This is simply due to the fact that OpenMP thread creation takes more time than that of Pthreads because OpenMP has more features than Pthreads like portability, work-sharing constructs. But for a larger sudoku size greater than 100 the time taken by OpenMP and pthreads becomes almost same which shows the effective utilization of additional features of OpenMP threads.

Anomalies:

PThread plot Anomalies:

There are two observable anomalies in pthread plot. The first one is at when the sudoku size is 9x9, This is because of the my algorithm design. The algorithm design is such that each thread except last thread runs $\lfloor (3 * n)/k \rfloor$(Where $\lfloor x \rfloor$ represents the integer less than or equal to x) of operations and the left over operations is run by last thread. Since n=9 so each thread runs 1 operations and last thread runs 12 operations so it increases overall time. The same reason applies for second anomaly where the sudoku size is 49x49.

OpenMP plot Anomalies:

One anomaly in OpenMP thread is at sudoku size 9x9 as per above explanation the time should be more. But this increase in time is covered up by large OpenMP thread creation time overhead. So time is less compared to 16x16 sudoku.

Plot 2:

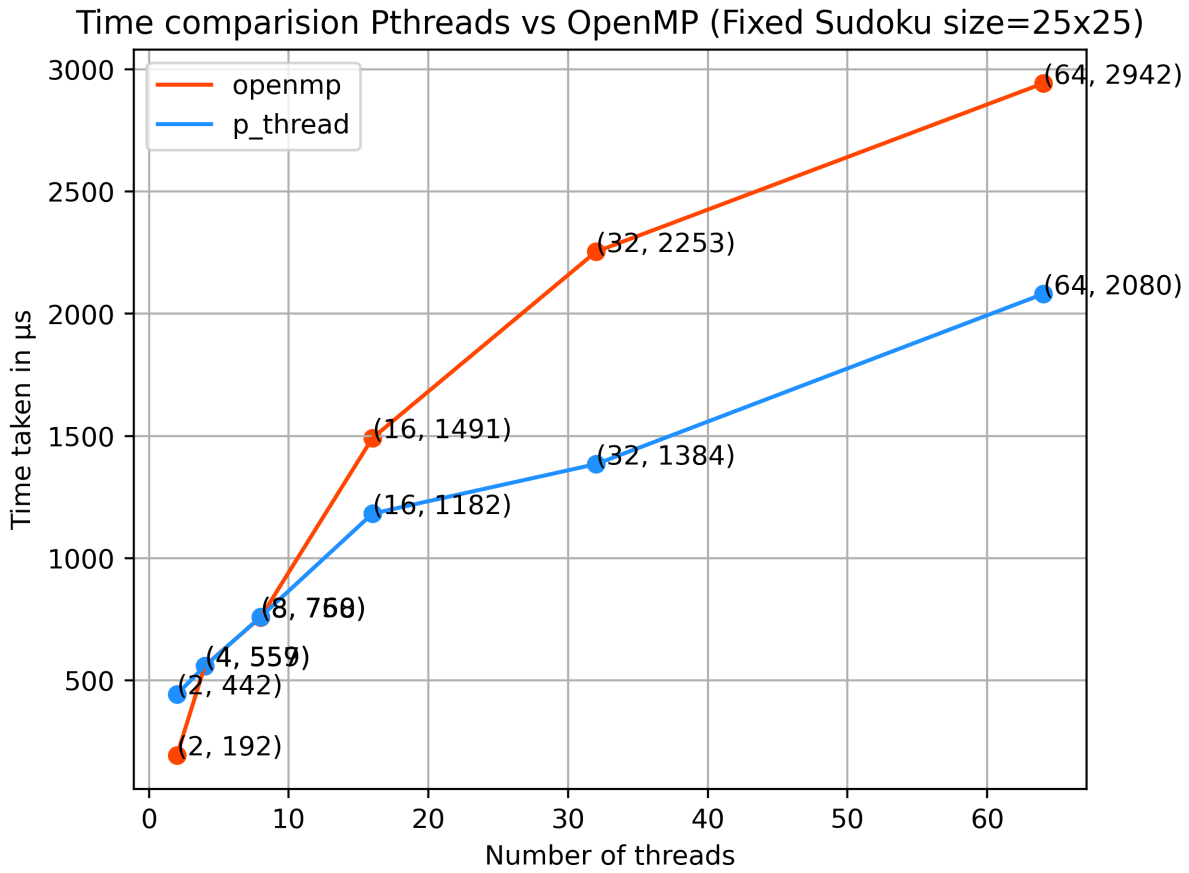

Figure 2: Time comparison P_Thread vs OpenMP Threads (Fixed Sudoku Size)

Analysis:

The trend of both the plots is increasing. This is because of time overhead caused by creating of more number of threads. Since number of threads are increasing which also includes the time burden of creating more number threads which leads to increase in overall total time.

From the Figure 2 we can also observe that the total time taken by OpenMP threads is more than time taken by PThreads. To create small number of threads PThreads consume more time than OpenMP threads,But to create more threads OpenMP consumes more time than PThreads.Which shows that creation of OpenMP threads consumes more time than creation of PThreads.So to execute task by threads more than number of cores of systems it is better to use PThreads in the other case it is better to use OpenMP threads.

Anomalies:

There are no anomalies in OpenMP Plot.

For PThread plot the increase rate between thread number 16 and 32 is less compared to increase rate of remaining graph. This is because my system has 8 cores(16 logical CPU's) and since the number of threads are between 16 and 32 the threads are perfectly managed so the time rate decreases.

Low Level Design:

For Both PThreads and OpenMP threads functions are same. For a given input N and K,The sudoku is stored in 2d array `int **sudoku`.Now K threads are created. Pthreads are created in function`bool p_ approach()` and OpenMP threads are created in function `void omp_ approach()`.

For a given sudoku there are 3*N number of checks to be performed in total. So each thread except last thread do $\lfloor (3*N)/K \rfloor$(Where $\lfloor x \rfloor$ represents the integer less than or equal to x) number of operations and the last thread does remaining number of operations. The array `int *eval` of size 3*n+1 contains 3*n+1 integers (intiliased to 0) . This array will be used by main thread to evaluate whole sudoku.

The main algorithm design is such that for given sudoku rows are uniquely identified by number between 1 and N and columns are uniquely identified by number between N+1 and 2*N and the sub grids are uniquely identified by number between 2*N+1 and 3*N(Example column 3 is identified by N+3 and grid 3 by 2*N+3). A for loop is run between the numbers 1 and 3*N and based on iterator which uniquely identifies row, column or grid as stated before and functions `bool row_ eval(int row)`, `bool col_ eval(int col)`,`bool grid_ eval(int grid)` are called based on for loop iterator (Like if iterator is >N and <=2*N the function bool col_eval() is called). Each function checks sudoku accordingly and if the row or column or grid, is valid the function assigns `eval[i]=i` and if it is not valid `eval[i]=-i` where i is unique number identified by that row or column or grid as stated before. For example consider column 3 is valid , column 3 is uniquely identified by i=N+3 so `eval[N+3]=N+3`.

Note:To check sudoku the three eval functions `bool row_ eval(int row)`, `bool col_ eval(int col)`,`bool grid_ eval(int grid)` a boolean array `bool *arr` of size N is intiliased with 0(false). A loop is run through row or column or grid of sudoku and if a number is present in sudoku let it be m(>0 and <=N) then arr[N] is made true.If there are repetetions they are identified as if a number is repeated it can be identified from boolean array.Then if a performed check is valid then `eval[i]=i` else `eval[i]=-i`(refer previous paragraph).

After all threads completes their checks. The main thread runs a for loop in array `int *eval`. If eval[i]=i then it writes into file as it is valid else not valid. The timer is started before creation of threads and it is ended when all threads completes their checks. Then main thread writes into the "output.txt" file accordingly.

Analysis of Algorithm

For a given input N and K and sudoku of size NxN.To evaluate by single process(Without multi threads) the time complexity is O(3*N) and space complexity is O(3*N). But by creating K threads and each thread sharing the work the space complexity becomes O(4*N). But the time complexity is O((3*N)/K) which is quite faster compared to single process computation when K is comparable to number of cores. This shows due to multi-threading the time is saved but on compensation of extra space consumed to store some extra information(to maintain thread). The use of multi-threading is beneficial only when K is comparable to number of cores of the system, if K much greater than number of cores, the threads have to be waiting for other threads to complete as decided by OS's CPU scheduler, Which increases the time taken, which destroys the benefits of multi-threading.Here the master-slave architecture is used usage of peer to peer architecture can saves us some more time.