Operating System–2(CS3523)
Programing Assignment 2

Pettugadi Pranav
CS21BTECH11063

graph-1

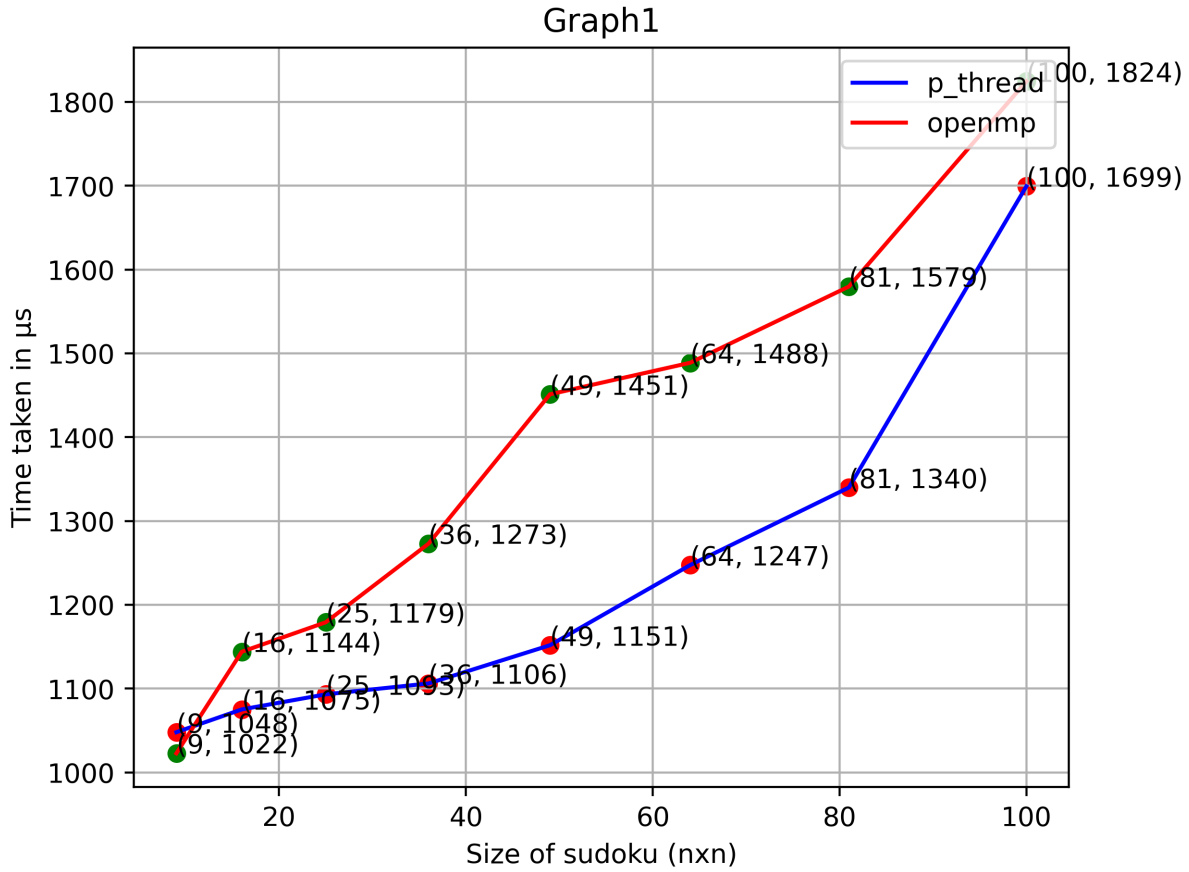

Figure 1: experiment-1

In both pthtread and openmp graph the time increases as the size of Sudoku increases. This is because the bigger the size of sudoko, the more work you have to do. This leads to an increase in the time required complete the program, especially if the program uses a single thread to solve the problem. Using multiple threads can increase the time it takes to complete the program, but the increase is less compare to single thread noticeable because the work can be shared between threads. However, if the program is not optimized for multithreading, or if a lot of overhead is involved in creating and managing threads, then as the problem grows, the overall time required to solve the problem increases. may increase.

graph-2

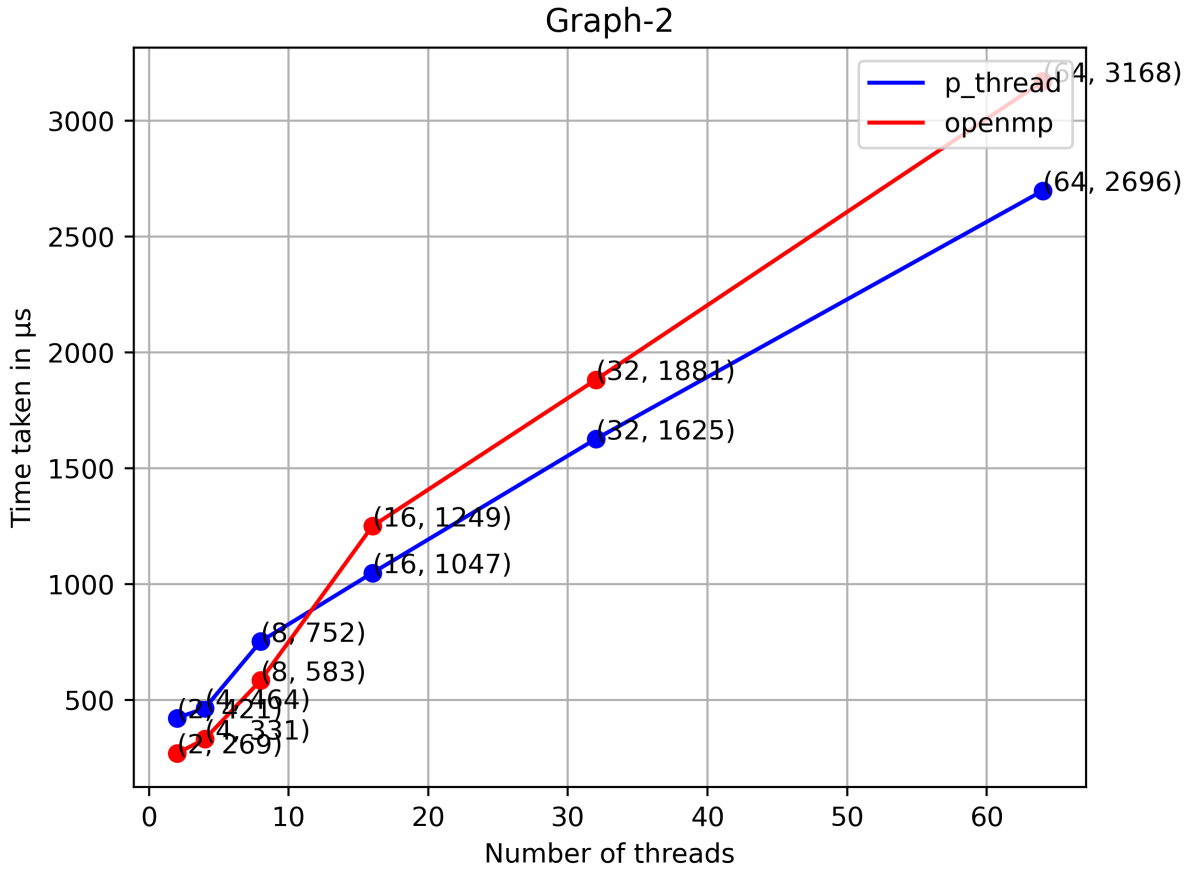

Figure 2: experiment-2

In both pthtread and openmp graph the time is increasing as the number of threads increases for several reasons.

Overhead associated with creating and managing threads: As the number of threads increases, the overhead associated with creating and managing threads becomes more noticeable and can increase execution time.

pthread vs openmp

In the both graph-1 and graph-2 the time for openmp is more than pthread in most cases because

Pthreads allow more fine-grained control over thread execution and synchronization, resulting in better performance in certain cases. If your application requires specific synchronization or scheduling behavior that OpenMP does not provide, pthreads will perform better.

program-design

In main function I have taken input from file "input.txt" k(no.of threads),n(size of sudoko). Then I have created 2 dimensional array sudoko with size "n" taken input from "input.txt".whole program is

same for both pthreads and openmp but creating threads is different. Then I have created k threads using "pthread_create" in pthreads and using "pragma omp parallel num_threads(k)" in openmp.

There are 3*n tasks to perform by k threads. I have distributed these 3*n tasks among k threads as for ith thread if i<(remainder of (3*n)/k)—- ((k*0)+i+1...(k*([(3*n)/k]+1))+i+1) tasks and if i>=(remainder of (3*n)/k) —-((k*0)+i+1...(k*[(3*n)/k])+i+1) tasks in function func1. For "1 to n" tasks we have check row validity ,"n+1 to 2*n" tasks we have to check coloumn validity and "2*n+1 to 3*n" tasks we have check grid validity.

These threads perform their respected tasks by invoking fuctions "row_checker" or "col_checker" or "grid_checker" and assign 0 (if invalid) or 1(valid) in result_array(globaly declared) to tasks respective index.

After all threads complete theirs tasks, Then the main thread open a file output.txt and write each threads task whether rows or columns or grids are invalid or valid.the timer is started when before threads created and stoped when threads complete their tasks.

analysis of algorithm

The time complexity for the programs is O((n$\hat{2}$log(n))/k)

which is less than single thread time complexity (O(3*n)because work is shared among different threads in multithread appication.