# REPORT

## COMPARISON OF PERFORMANCE OF THREADS AND OPENMP

Threads and OpenMP are two commonly used parallel programming models for multi-threaded applications. Both have their own strengths and weaknesses.Threads provide a low-level programming interface for creating and synchronizing threads, but can be difficult to use and manage due to issues such as race conditions and deadlocks. On the other hand, OpenMP provides a higher-level interface for parallel programming that is easier to use, but may be less flexible and have less control over the underlying hardware.
In terms of performance, it can be difficult to compare the two as it depends on the specific application and hardware. In general, OpenMP can lead to better performance as it provides higher-level constructs and optimizations, but may have some performance overhead due to its abstractions. On the other hand, threads can provide lower overhead, but may require more effort to achieve good performance.
(Note the time taken in this analysis is in the order of microseconds)

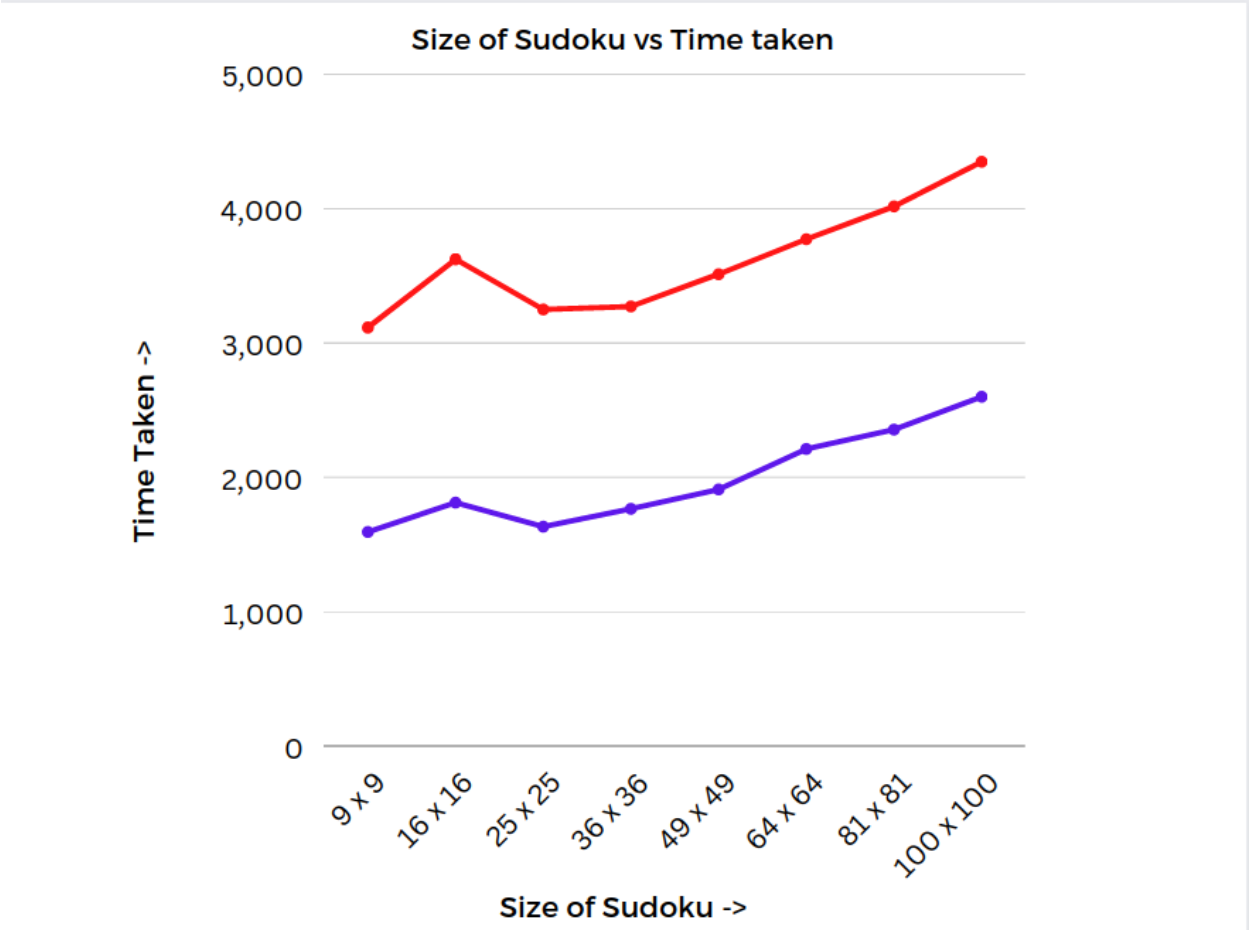
### ANALYSIS 1 : Comparison of time taken as varied sizes of input are given

Given the fact that during the execution of PThread and OpenMP models of parallel programming, the hardware used was the same along with the number of threads. Here, as seen in the graph below, Pthread seems to be performing better (taking lesser time) than OpenMP (contradictory to what is said to happen in general.) The factors for this may include:

1. Overhead: OpenMP introduces additional abstractions and layer of abstraction over PThreads, which can result in additional overhead and slower performance compared to PThreads.
2. Load Balancing : OpenMP uses a static load balancing strategy, which divides the work equally among the threads. On the other hand, PThreads often requires manual load balancing, which can lead to better performance (if optimized correctly).
3. Implementation: The codes (doing the same task) given to both of them were different (in my case: the code using pthreads was modular as compared to the code using OpenMP) and hence this can also influence the performance difference between PThreads and OpenMP.

(Note : Even though OpenMP provides higher-level synchronization constructs, such as barriers and critical sections, that simplify the process of coordinating and synchronizing threads, we cannot argue here as I haven't implemented such constructs in my code.)

Anomaly: We can see a peak in the time taken by both OpenMP and PThread, as opposed to the general trend of increasing time taken with increasing size of input (specifically in the sudoku of size 16 x 16). It can occur due to factors like load imbalance. Load imbalance can occur when some threads have more work to do than others, leading to unequal distribution of processing time and potential idle time for some threads. The time taken later on shows a declination which could be due to increased contention for shared resources, memory bandwidth limitations and some underlying hardware limitations.

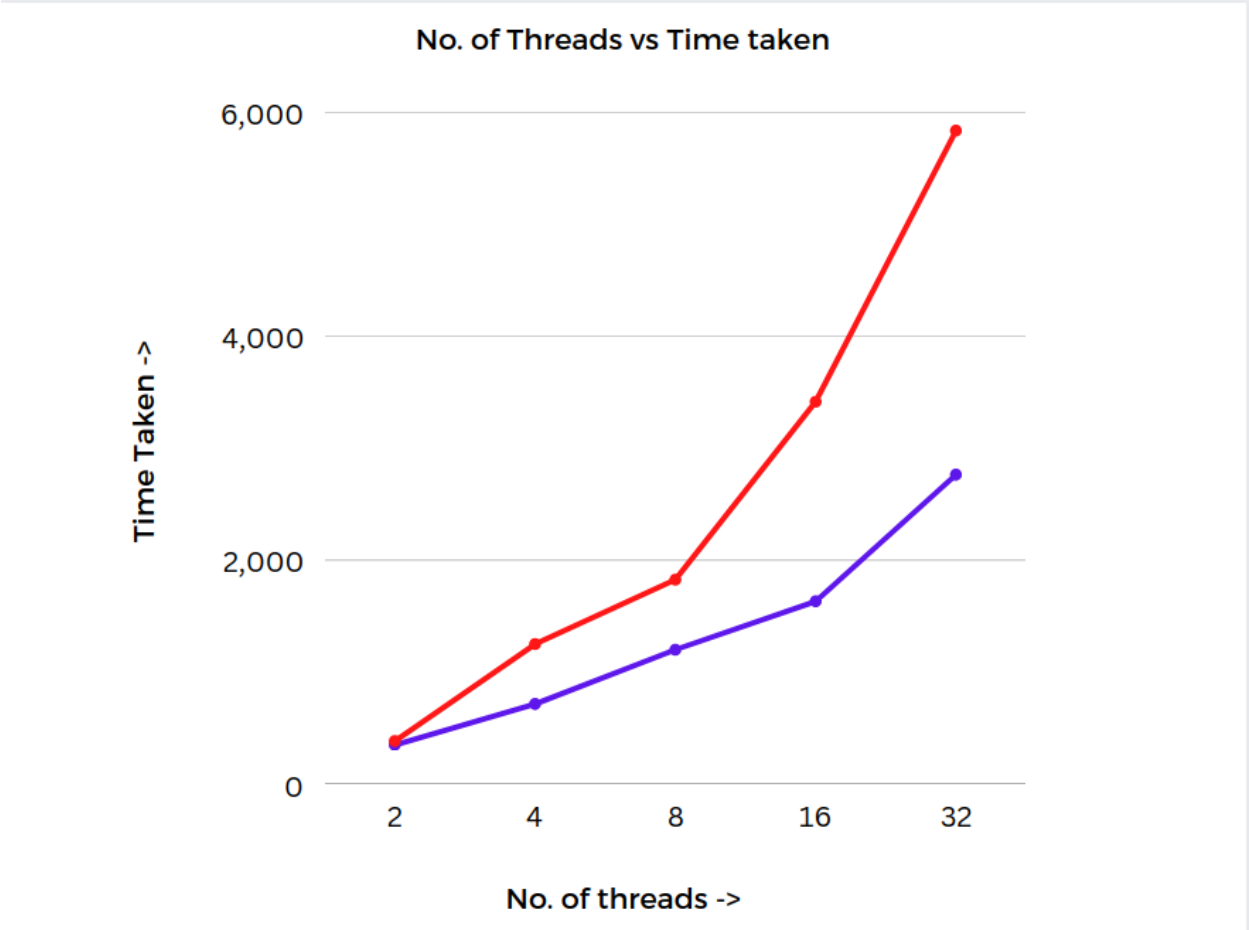



(Blue line refers to Pthreads while the Red line refers to OpenMP)

## ANALYSIS 2 : Comparison of time taken as varied number of threads are used

As it can be seen from the graph below, the time taken to complete the task as the number of threads are increased can be due to the following factors:

1. Overhead: Increasing the number of threads can result in additional overhead from coordination and communication between threads, which can slow down the overall performance which can be seen pronounced in OpenMP as it provides higher-level synchronization constructs.
2. Load Balancing: As the number of threads increases, the workload may become increasingly unevenly distributed among the threads, resulting in decreased performance.
3. Synchronization: Increased number of threads can lead to more frequent synchronization between threads, which can slow down the performance.
4. With more number of threads, multiple threads attempt to access the same data, resulting in increased cache misses and decreased performance.




(Blue line refers to Pthreads while the Red line refers to OpenMP)

Note: In general as is usually the case where OpenMP performs better than PThreads, here in my analysis, it appears to perform worser than Pthreads is mainly because of the implementation. While the code for Pthreads is modular and more organised, the code for OpenMP isn't.

Conclusion : Well the performance comparison between Pthreads and OpenMP is not a straightforward matter and one cannot make direct conclusions as to which one is faster and which one is more efficient. Instead, a thorough evaluation of the specific requirements of the application and the performance characteristics of the hardware is necessary to make an informed decision.