Assignment 2: Performance Evaluation of OpenMP Programs

Submitted By: Vansh Bhasin

Roll no:102303270

Note: Cache misses of perf stat are not supported by system.

Note: All graphs are generated by python using perf_speedup_plot.sh script and can be found in the repo.

Experiment: Question 1

1. Title
Molecular Dynamics Force Calculation using OpenMP

2. Objective
The purpose of this experiment is to develop a parallel implementation for calculating inter-particle forces in a molecular simulation using OpenMP. The experiment focuses on evaluating how parallel execution improves performance and how synchronization mechanisms affect scalability and efficiency.

3. System & Environment
Processor: Intel Core i7-14650HX
Cores: Multiple Physical and Logical Cores (16 physical / 24 logical)
Operating System: Microsoft Windows
Compiler: g++ with OpenMP support

4. Methodology
The program computes interaction forces between particles using nested loops. OpenMP is used to parallelize the outer loop so multiple threads can perform calculations concurrently. Synchronization techniques such as atomic operations or reduction may be used to ensure correct updates to shared variables. Execution time is recorded for different thread counts to evaluate parallel performance.

5. Experimental Results

Threads	Execution Time (seconds)	Speedup	Efficiency (%)
1	0.036033	1.00	100.00
2	0.020035	1.80	89.92
4	0.020400	1.77	44.16
8	0.021078	1.71	21.37
16	0.020243	1.78	11.12
24	0.021783	1.65	6.89

Note: Speedup $S(p) = T_1 / T_p$. Efficiency $E(p) = S(p) / p$

6. Discussion & Analysis

A. Parallel Speedup and Scalability Behavior

The experimental results show that increasing the number of threads reduces execution time initially, but the improvement is limited beyond a certain point. The serial execution time with 1 thread was 0.036033 seconds, while the fastest execution time achieved was 0.020035 seconds (at 2 threads), resulting in a maximum speedup of 1.80×.

This indicates that parallelization improved performance, but the speedup is far below the ideal linear speedup. Ideally, using 24 threads should result in a 24× speedup, but the measured speedup is significantly lower. This deviation occurs due to inherent limitations in parallel execution, such as synchronization overhead, memory access delays, and thread management costs.

The speedup graph clearly shows that performance improvement slows down after 4 threads and almost plateaus beyond that point. This behavior aligns with Amdahl's Law, which states that the presence of serial portions in the program limits maximum achievable speedup.

B. Efficiency Reduction with Increasing Threads

Efficiency represents how effectively the available threads are utilized. The efficiency dropped significantly as the number of threads increased:

- 2 threads: 89.92%
- 4 threads: 44.16%
- 8 threads: 21.37%
- 16 threads: 11.12%
- 24 threads: 6.89%

This sharp decline in efficiency indicates that additional threads contribute less useful work and spend more time waiting or managing overhead. The efficiency graph confirms this trend, showing a rapid decrease as thread count increases.

The main reasons for this reduction include:

- Thread creation and management overhead
- Synchronization delays between threads
- Increased competition for shared memory and cache
- Limited parallel portion of the code

C. Performance Plateau and Hardware Resource Contention

The execution time decreases significantly when moving from 1 to 4 threads, but after that, the improvement becomes minimal. For example:

- 4 threads: 0.020400 seconds
- 24 threads: 0.021783 seconds

This small difference indicates that the program has reached a performance saturation point.

The Intel Core i7-14650HX processor has multiple physical and logical cores, but all cores share certain hardware resources such as memory bandwidth and cache hierarchy. When too many threads attempt to access memory simultaneously, contention occurs, reducing performance gains.

Additionally, some parts of the program cannot be parallelized and must run sequentially, further limiting speedup.

D. Strong Scaling Characteristics

This experiment demonstrates strong scaling behavior, where the problem size remains fixed while the number of threads increases. Ideally, execution time should decrease proportionally with thread count. However, the observed results show diminishing returns beyond 4 threads.

This confirms that parallel efficiency is constrained by:

- Serial code portions
- Synchronization overhead
- Memory access limitations
- Thread scheduling overhead

7. Conclusion

The experiment demonstrates that parallelization using OpenMP significantly improves execution performance compared to serial execution. execution time was reduced from 0.036033 seconds to 0.020035 seconds with 2 threads, achieving a maximum speedup of 1.80×.

However, the results also show that speedup is not linear and efficiency decreases as thread count increases. The optimal performance improvement was observed between 2 and 4 threads, where efficiency remained relatively higher.

Beyond this point, performance gains became minimal due to synchronization overhead, limited parallel workload, and hardware resource contention such as shared memory bandwidth and cache usage.

These results confirm the theoretical principles of parallel computing, particularly Amdahl's Law, which explains why speedup is limited by the serial portion of a program. The experiment highlights the importance of selecting an appropriate number of threads to achieve optimal performance rather than simply using the maximum available threads.

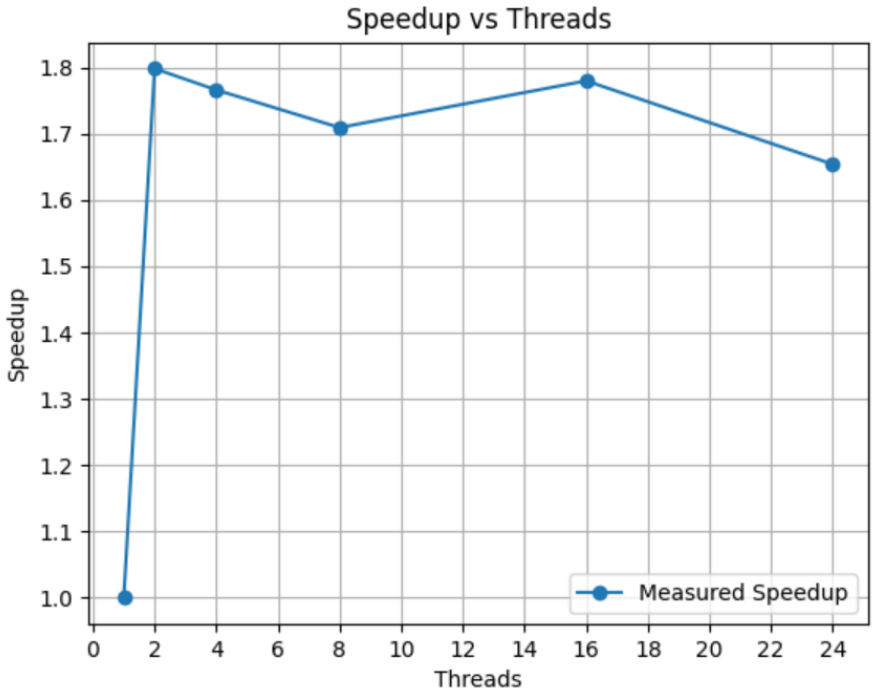

Fig 1: Speedup vs Threads for Molecular Dynamics Simulation

Fig 2: Execution Time vs Threads for Molecular Dynamics Simulation

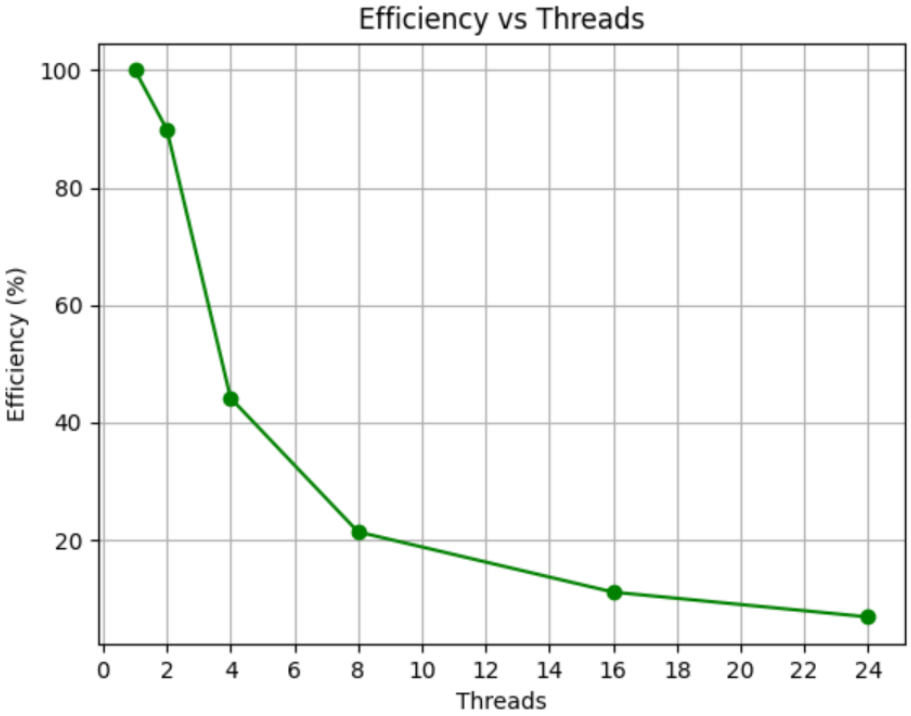

Fig 3: Efficiency vs Threads for Molecular Dynamics Simulation

Experiment: Question 2

1. Title

DNA Sequence Alignment using OpenMP

2. Objective

The objective is to parallelize the sequence alignment algorithm using OpenMP to improve execution efficiency. This experiment investigates how dependencies and synchronization affect the performance of parallel dynamic programming algorithms.

3. System & Environment

Processor: Intel Core i7-14650HX
Cores: Multiple Physical and Logical Cores (16 physical / 24 logical)
Operating System: Microsoft Windows
Compiler: g++ with OpenMP support

4. Methodology

The alignment matrix is computed using parallel execution across independent cells where possible. OpenMP scheduling techniques are applied to distribute the workload among threads. Execution time and performance metrics are recorded for analysis.

5. Experimental Results

Threads	Execution Time (seconds)	Speedup	Efficiency (%)
1	0.429647	1.00	100.00
2	0.262174	1.64	81.94
4	0.236217	1.82	45.47
8	0.227738	1.89	23.58
16	0.272366	1.58	9.86
24	1.017733	0.42	1.76

Note: Speedup $S(p) = T_1 / T_p$. Efficiency $E(p) = S(p) / p$

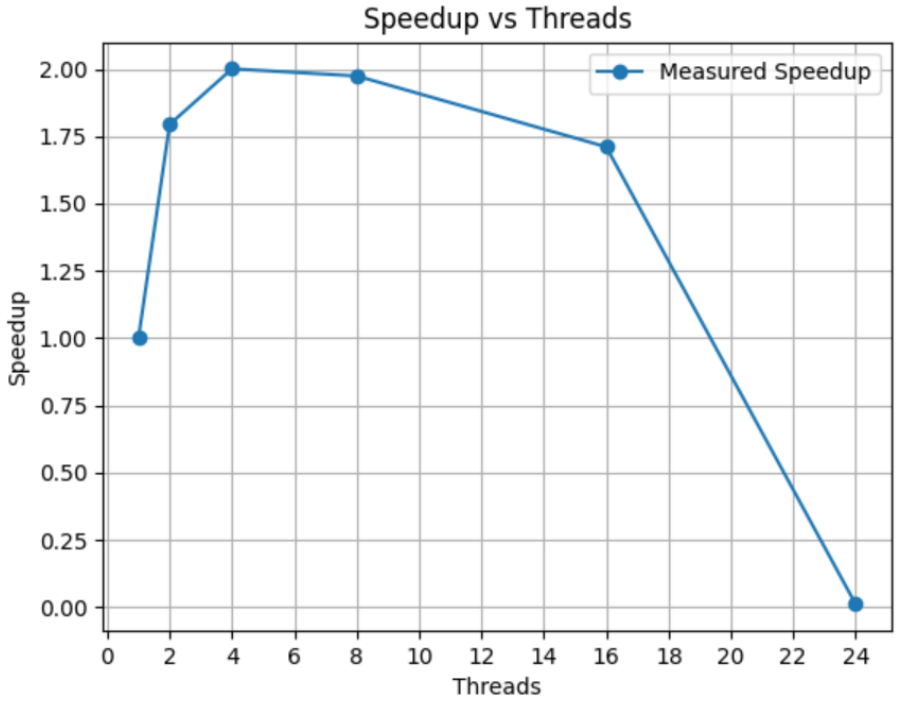

Fig 4: Speedup vs Threads for DNA Sequence Alignment

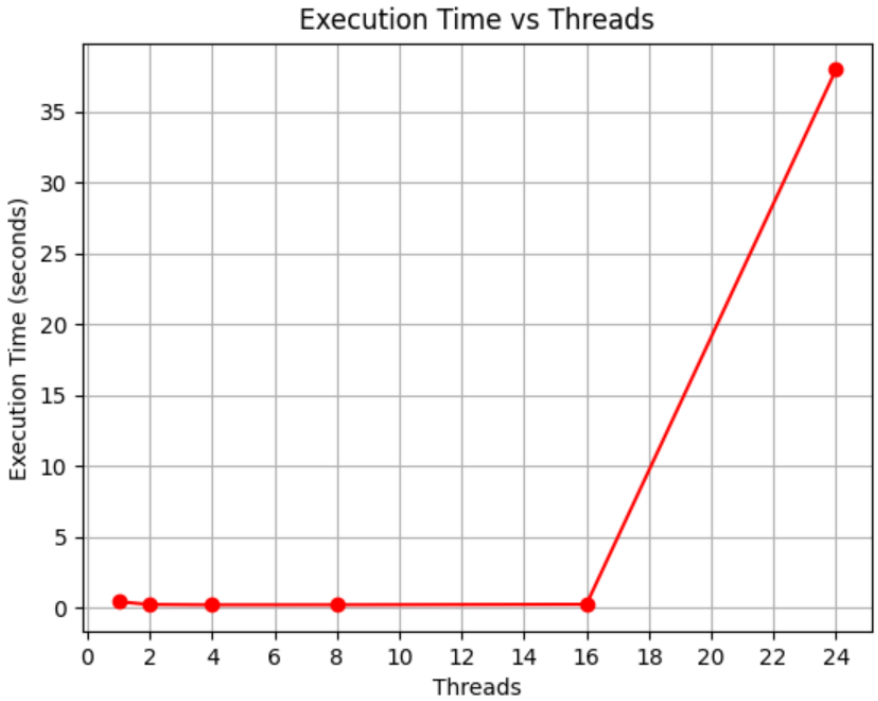

Fig 5: Execution Time vs Threads for DNA Sequence Alignment

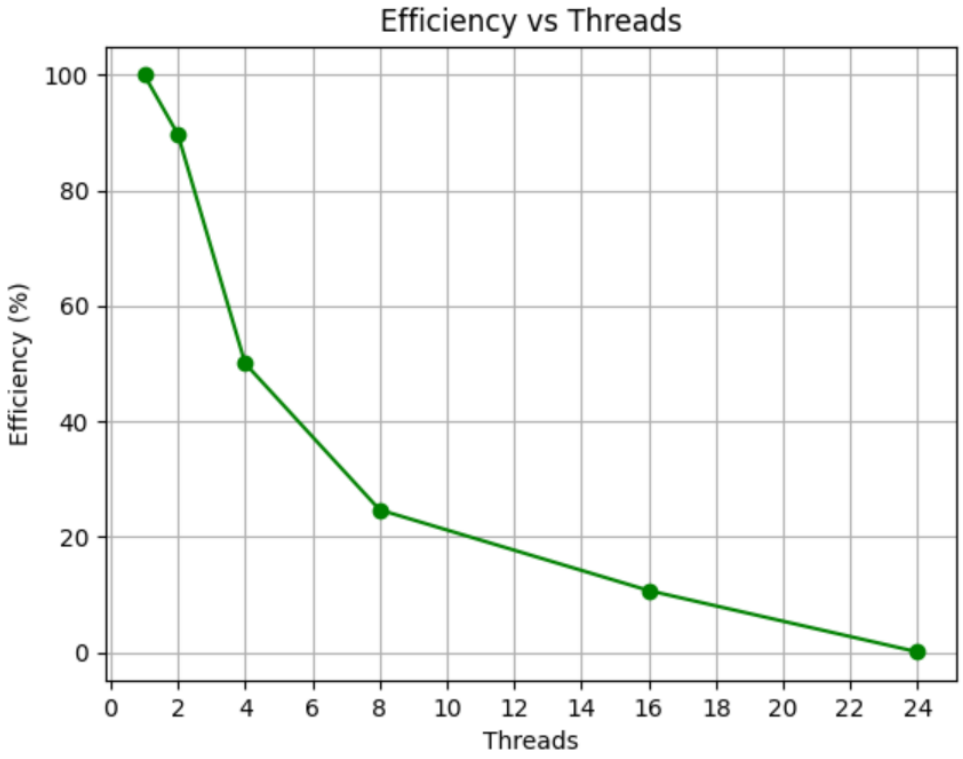

Fig 6: Efficiency vs Threads for DNA Sequence Alignment

6. Discussion & Analysis

A. Parallel Speedup and Optimal Thread Utilization

The experimental results demonstrate that parallel execution significantly improves performance compared to serial execution, but only up to a certain number of threads. The serial execution time with 1 thread was 0.429647 seconds, while the minimum execution time of 0.227738 seconds was achieved using 8 threads, resulting in a maximum speedup of 1.89×.Therefore, optimal thread count = 8 threads.

This indicates that parallelism effectively reduces computation time initially. However, beyond 8 threads, performance begins to degrade instead of improving. The speedup graph clearly shows that speedup increases from 1 to 8 threads but decreases when using 16 and 24 threads. This behavior occurs due to increasing synchronization overhead and resource contention among threads.

The observed speedup is lower than the ideal linear speedup predicted by parallel computing theory, confirming the presence of serial portions and overhead, as described by Amdahl’s Law.

B. Efficiency Degradation with Increasing Threads

Parallel efficiency measures how effectively threads contribute to useful work. The efficiency values show a steady decline as the number of threads increases:

- 2 threads: 81.94% efficiency
- 4 threads: 45.47% efficiency
- 8 threads: 23.58% efficiency
- 16 threads: 9.86% efficiency
- 24 threads: 1.76% efficiency

The efficiency graph clearly illustrates this downward trend. Initially, threads are efficiently utilized, but as thread count increases, the overhead of managing threads, synchronization delays, and competition for shared hardware resources reduces efficiency.

At higher thread counts, threads spend more time waiting rather than performing useful computation, resulting in poor efficiency.

C. Performance Degradation at Higher Thread Counts

A significant performance degradation is observed when increasing threads beyond 8. At 16 threads, execution time increases to 0.272366 seconds, and at 24 threads, execution time rises dramatically to 1.017733 seconds, which is slower than the serial execution time.

This degradation occurs due to several factors:

- Increased thread synchronization overhead
- Context switching overhead
- Memory bandwidth limitations
- Cache contention between threads
- Thread management overhead

The Intel Core i7-14650HX processor contains multiple physical and logical cores. While physical cores improve performance, excessive use of logical threads can increase contention for shared CPU resources such as cache and memory bandwidth.

As a result, using too many threads introduces more overhead than performance benefit.

D. Strong Scaling Behavior and Amdahl's Law

This experiment demonstrates strong scaling, where the problem size remains constant while the number of threads increases. Ideally, execution time should decrease proportionally with thread count. However, the results show diminishing returns beyond 8 threads.

This confirms the theoretical limitation described by Amdahl's Law, which states that the serial portion of a program limits overall speedup. As the number of threads increases, the impact of the serial portion and parallel overhead becomes more significant, reducing efficiency and performance gains.

7. Conclusion

The experiment successfully demonstrates the effect of parallel execution on improving program performance using OpenMP. The execution time decreased from 0.429647 seconds in serial execution to 0.227738 seconds with 8 threads, achieving a maximum speedup of 1.89×.

The results indicate that parallelization significantly improves performance up to an optimal thread count. In this experiment, the optimal performance was achieved at 8 threads, where execution time was minimized and speedup was highest.

However, increasing the number of threads beyond this point resulted in performance degradation due to synchronization overhead, memory contention, and thread management costs. At 24 threads, execution time increased substantially, demonstrating that excessive parallelism can negatively impact performance.

These findings confirm the importance of selecting an appropriate number of threads to achieve optimal performance. The experiment also validates Amdahl's Law, which explains why speedup is limited by serial execution components and parallel overhead.

Experiment: Question 3

1. Title

Heat Diffusion Simulation using OpenMP

2. Objective

This experiment aims to simulate heat transfer across a 2D surface using parallel computation. The objective is to study performance improvements and identify limitations caused by memory bandwidth.

3. System & Environment

Processor: Intel Core i7-14650HX
Cores: Multiple Physical and Logical Cores (16 physical / 24 logical)
Operating System: Microsoft Windows
Compiler: g++ with OpenMP support

4. Methodology

The heat diffusion equation is applied iteratively across a grid. OpenMP is used to parallelize the spatial computation while maintaining correctness. Performance metrics are measured for evaluation.

5. Experimental Results

Threads	Execution Time (seconds)	Speedup	Efficiency (%)
1	0.512705	1.00	100.00
2	0.314733	1.63	81.45
4	0.248889	2.06	51.50
8	0.243940	2.10	26.27
16	0.227889	2.25	14.06
24	0.321451	1.59	6.65

Note: Speedup $S(p) = T_1 / T_p$. Efficiency $E(p) = S(p) / p$

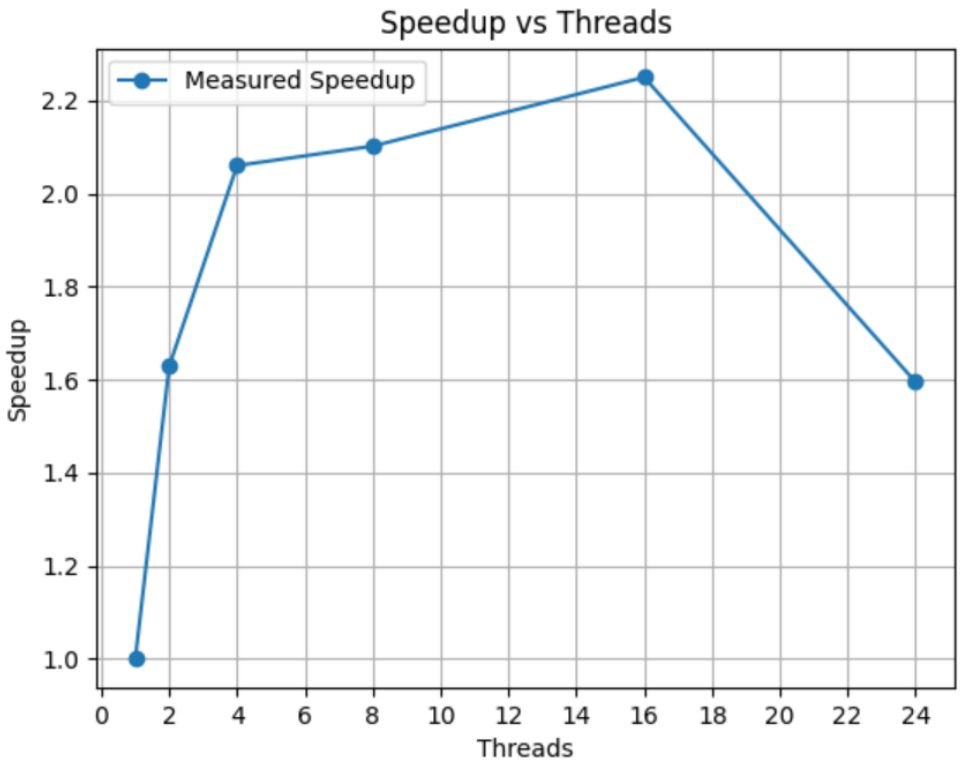

Fig 7: Speedup vs Threads for Heat Diffusion Simulation

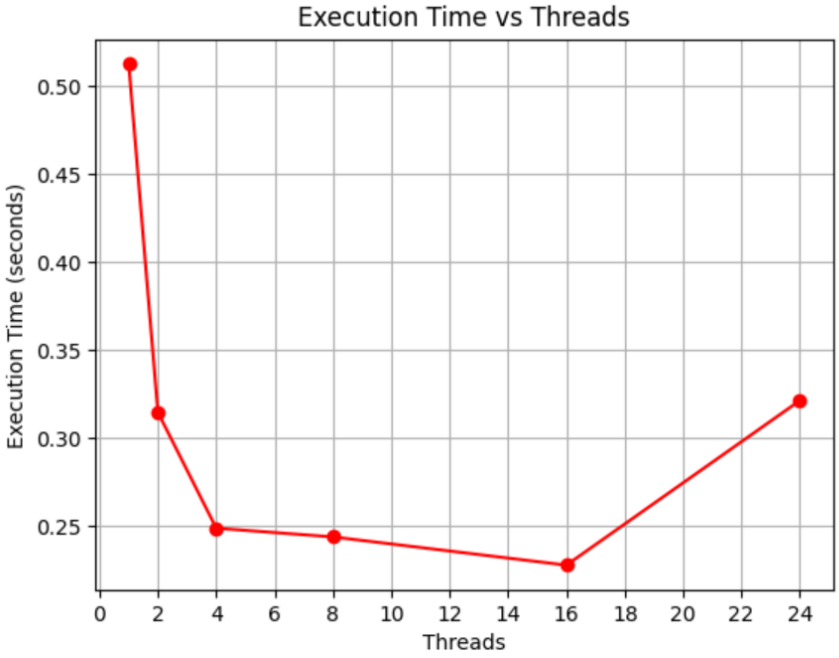

Fig 8: Execution Time vs Threads for Heat Diffusion Simulation

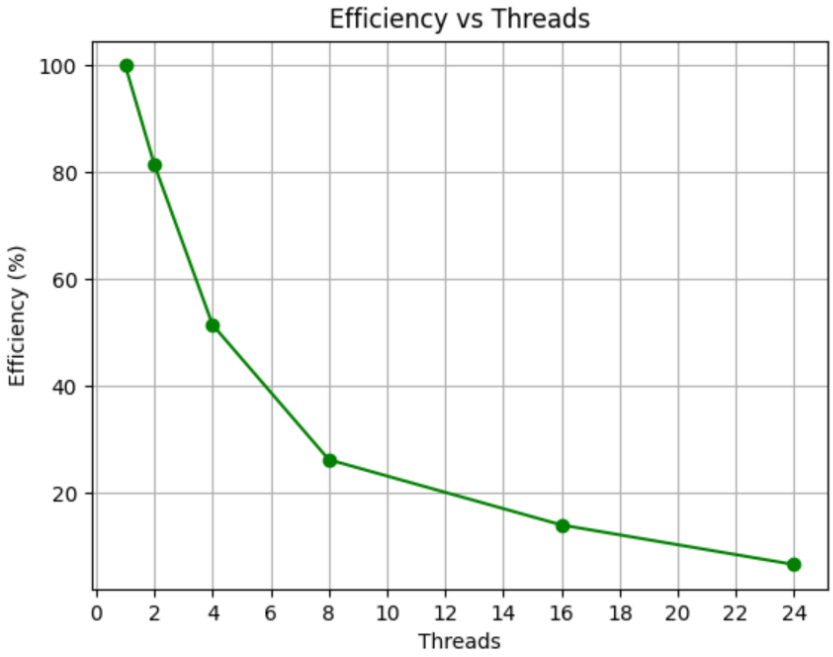

Fig 9: Efficiency vs Threads for Heat Diffusion Simulation

6. Discussion & Analysis

A. Parallel Performance Improvement and Speedup Behavior

The experimental results demonstrate that parallel execution significantly improves performance compared to serial execution. The execution time decreased from 0.512705 seconds with 1 thread to 0.227889 seconds with 16 threads, achieving a maximum speedup of 2.25×.

The speedup graph shows a steady increase in speedup from 1 thread to 16 threads, indicating effective utilization of parallel computing resources. This confirms that OpenMP successfully distributed the computational workload across multiple processor cores, reducing overall execution time.

However, the speedup is still lower than the ideal linear speedup due to unavoidable overhead and hardware limitations. According to parallel computing theory and Amdahl's Law, the presence of serial operations and synchronization delays restricts maximum achievable performance.

B. Efficiency Reduction with Increasing Threads

Parallel efficiency decreases as the number of threads increases. The efficiency values observed are:

- 2 threads: 81.45% efficiency
- 4 threads: 51.50% efficiency
- 8 threads: 26.27% efficiency
- 16 threads: 14.06% efficiency
- 24 threads: 6.65% efficiency

The efficiency graph clearly shows a downward trend. This reduction occurs because additional threads introduce overhead such as thread scheduling, synchronization delays, and competition for shared hardware resources.

As thread count increases, each thread performs less useful work relative to the overhead required to manage parallel execution, resulting in lower efficiency.

C. Optimal Thread Count and Performance Saturation

The best performance was achieved at 16 threads, where execution time reached its minimum value of 0.227889 seconds. This indicates that the Intel Core i7-14650HX processor effectively utilized its physical cores up to this level.

However, increasing the thread count to 24 threads resulted in performance degradation, with execution time increasing to 0.321451 seconds and speedup decreasing to 1.59×.

This performance degradation occurs due to several hardware limitations:

- Increased contention for memory bandwidth
- Cache contention between threads
- Thread scheduling and management overhead
- Increased synchronization and communication delays

Logical threads share execution resources with physical cores, and excessive threading increases overhead rather than improving performance.

D. Memory Bandwidth Limitations and Memory-Bound Behavior

The heat diffusion simulation is a memory-intensive application. Each computation step requires frequent reading and writing of grid data stored in main memory. As multiple threads attempt to access memory simultaneously, the memory subsystem becomes a performance bottleneck.

This limitation is evident from the execution time graph, where execution time decreases initially but increases again at higher thread counts. This behavior indicates memory bandwidth saturation, where additional threads cannot improve performance due to limited data transfer capacity.

As a result, the program becomes memory-bound rather than compute-bound, limiting scalability.

E. Strong Scaling Characteristics

This experiment demonstrates strong scaling, where the problem size remains fixed while increasing the number of threads. Ideally, execution time should decrease proportionally with thread count. However, the results show diminishing returns beyond 16 threads.

This confirms the theoretical prediction that parallel performance is limited by:

- Serial execution components
- Synchronization overhead
- Memory access limitations
- Thread management overhead

7. Conclusion

The experiment successfully demonstrates the effectiveness of parallel computing using OpenMP for heat diffusion simulation. The execution time was reduced from 0.512705 seconds in serial execution to 0.227889 seconds with 16 threads, achieving a maximum speedup of 2.25×.

The results indicate that parallel execution significantly improves performance up to an optimal thread count. In this experiment, the best performance was achieved at 16 threads, which effectively utilized the processor’s physical cores.

However, increasing the number of threads beyond this point resulted in performance degradation due to memory bandwidth limitations, synchronization overhead, and resource contention. At 24 threads, execution time increased, demonstrating that excessive parallelism can reduce performance.

These results validate the theoretical principles of parallel computing and confirm that optimal performance depends on selecting an appropriate number of threads. The experiment also

highlights the impact of hardware limitations, particularly memory bandwidth, on parallel scalability.