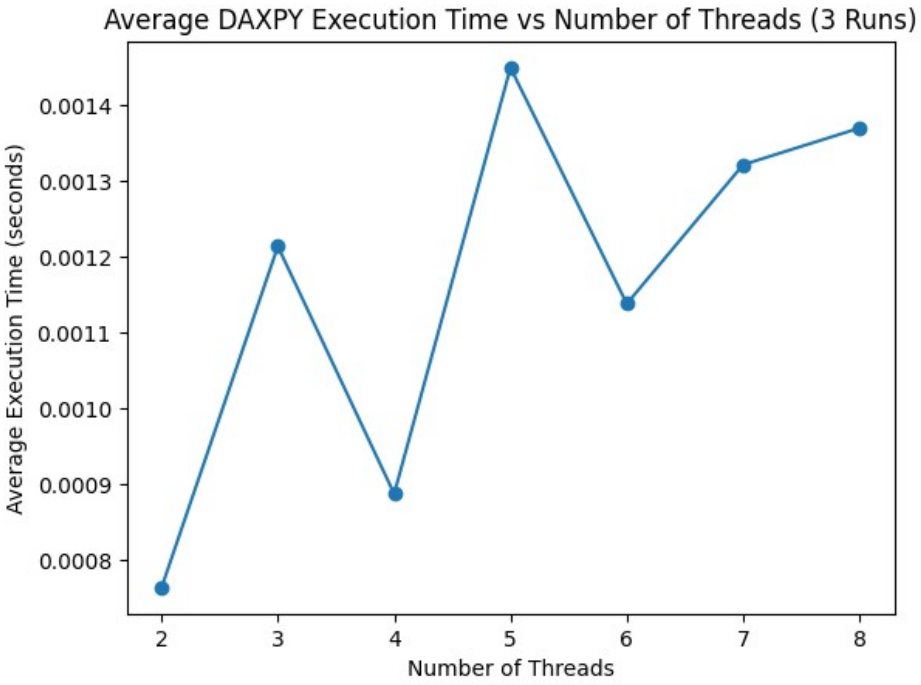
Q1. DAXPY Loop(./q1)

Threads	Time (s)
2	0.000764
3	0.001214
4	0.000888
5	0.001450
6	0.001138
7	0.001322
8	0.001370



Inferences

1. The maximum performance is achieved at 2 threads. All higher thread counts result in slower execution due to overhead to memory banwidth saturation and OpenMP overhead. Since DAXPY performs very little computation per memory access. i.e. highly memory bound.

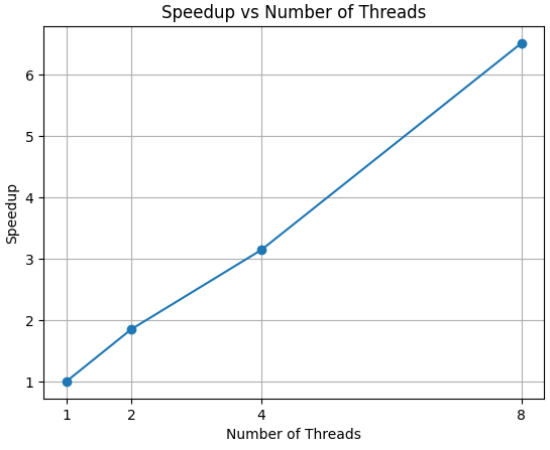
Q2. Matrix Computation(./q2)

N	Sequential		Parallel		Optimised		Effieciency
	Time(sec)	Speedup	Time(sec)	Speedup	Time(sec)	Speedup	
512	0.45	1x	0.08	5.996x	0.09	5.299x	44%
1024	6.82	1x	0.69	9.888x	0.67	10.124x	84%
2048	97.03	1x	16.91	5.737x	11.83	8.202x	68%

Total Threads = 12

Threads	Time(sec)	Speedup
1	5.16	1x
2	2.79	1.852x
4	1.64	3.146x
8	0.79	6.518x

N = 1024



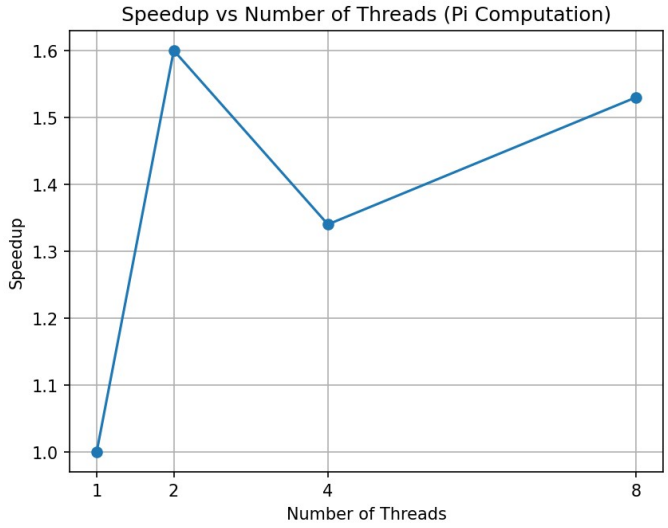
Inferences:

1. Speedup Efficiency
   The optimised version achieves **6.11x speedup** on 12 threads, giving **~51% efficiency.** It shows successful parallelization.
2. Limited CPU Utilization
   Despite using 12 threads, only 3.26 CPUs were utilised effectively. It happens due to multiple threads compete for the same RAM bandwidth, causing cores to stall.
   This is practical demonstration of Amdahl's Law combined with Memory wall(adding more threads does not increase performance once memory bandwidth becomes bottleneck).
3. Impact of using transpose matrix
   The tranpose matrix implementation improves speedup from 5.13x to 6.11x
   Transposing matrix Y converts column-wise access into row-wise access, improving spatial locality and reducing cache misses. As a result, the CPU spends more time computing then waiting fore memory access.
4. High IPC
   IPC of **~2.75** shows **CPU executes effectively** when data is available.
5. Low Branch Miss rate
   Branch miss rate is less than 0.1%. It depicts the nested loops are highly predictable, control flow is not a bottlenect

Q3. Calculation of π (./q3)

num_of_steps = 10,000

Threads	Time(sec)	Speedup
1	0.001969	1x
2	0.001228	1.60x
4	0.001466	1.34x
8	0.001285	1.53x



Inferences

1. Accuracy of π remains constant for different thread counts
   pi = 3.141593

2. Limited Speedup with Increase in threads
   The calculation of π workload is small. Thread creation, synchronization overhead dominates execution time. As a result, adding more threads does not improve performance. Low effective CPU Utilization.

3. Low effective CPU Utilization
   Most threads spend their idle/waiting. The computation per iteration is small, hence cores finish work quickly and stall on synchronization, The program is overhead-bound rather compute-bound.
4. IPC degradation with Higher Threads.
   Instructions per cycle decrease when moving from 1-2 threads (~2.3) to 4-8 threads (~1.8).

   With more threads, contention and synchronization increase, reducing instruction throughput. Indicates diminishing returns from parallelism for this workload.

5. Speedup Curve (Amdahl's Law)
   The speedup curve bends downward because the sequential portion dominates. Each iteration performs very little computation