Benchmark Data Summary:

Threads	Commands	Completion Time (ms)	Average Latency (ms)	CPU Time (ms)	Memory Usage (KB)
1	100	9.00	4	40.16	2316
1	1000	13.00	20	222.83	2316
1	10000	57.00	60	830.84	2316
2	100	11.00	3	884.52	2732
2	1000	24.00	20	1102.61	2780
2	10000	64.00	76	2182.69	2836
4	100	9.00	10	2340.10	3712
4	1000	44.00	25	2803.16	3744
4	10000	99.00	116	5407.41	3748
8	100	17.00	23	5779.96	4876
8	1000	25.00	34	6687.95	5720
8	10000	225.00	210	14500.06	5720
16	100	25.00	37	15346.95	8520
16	1000	48.00	53	17354.62	9904
16	10000	285.00	316	32181.63	9904

Observations:

1. **Initial Improvement with Increased Threads**:
    - **From 1 to 2 Threads (100 Commands)**:
        - Average Latency decreases from 32 ms to 4 ms.
        - Completion time for 100 commands drops from 200 ms to 9 ms.
        - This indicates that adding a second thread allows tasks to be processed in parallel, effectively reducing latency.
2. **Deterioration Beyond Optimal Thread Count**:
    - **From 2 to 4 Threads (1000 Commands)**:
        - Average Latency increases from 20 ms (2 threads) to 25 ms (4 threads).
        - Completion time for 1000 commands increases from 13 ms (1 thread) to 44 ms (4 threads).
    - **From 4 to 8 Threads (10000 Commands)**:
        - Average Latency increases from 76 ms (2 threads) to 116 ms (4 threads), and up to 210 ms (8 threads).
        - Completion time for 10000 commands increases from 64 ms (2 threads) to 99 ms (4 threads) and further to 225 ms (8 threads).
        - These trends show that beyond a certain point, adding more threads does not yield the same level of improvement and may actually increase the latency and completion time.
3. **Significant Increase in CPU Time**:
    - **CPU Time** increases exponentially with more threads. From 40.16 ms (1 thread, 100 commands) to 32181.63 ms (16 threads, 10000 commands).

    - The significant increase in CPU time suggests high overhead due to thread management and context switching as more threads are added.

4. **Memory Usage Increases with More Threads**:
    - Memory usage grows from 2,316 KB (1 thread) to 9,904 KB (16 threads).
    - Each additional thread consumes more memory resources, leading to a rise in memory usage as the number of threads increases.

Conclusions

- **Adding Threads Improves Latency Up to a Point:**
    - Introducing additional threads can reduce latency by allowing parallel processing.
    - This is effective until the number of threads matches the number of CPU cores.
- **Excessive Threads Increase Latency:**
    - Beyond the optimal number of threads, latency increases due to overhead.
    - The system spends more time managing threads than executing actual work.
- **Optimal Thread Count Depends on System Resources:**
    - The ideal number of threads often corresponds to the number of physical or logical CPU cores.
    - For example, on a dual-core system, using 2 threads may yield the best performance.
- **Need for Efficient Thread Management:**
    - Implementing thread pools or limiting the number of active threads can help maintain low latency.
    - Avoid unnecessary thread creation and destruction.

Final Thoughts

These observations highlight the importance of balancing concurrency with system capabilities. While multi-threading can enhance performance, it must be managed carefully to avoid introducing latency through overhead and resource contention. By fine tuning the number of threads and optimizing resource usage, you can achieve better performance and lower latency in your application.