1. Source Code For All Programs

- Pipeline.c
- Plot_performance.py
- plot_2.py

2. Workload Explanation

The **Producer-Consumer** workload simulates a **multithreaded pipeline** where multiple producer-consumer pairs interact using **shared buffers**. Each thread pair operates independently to minimize contention.

1. Workload Behavior

- **Producers** generate random data and insert it into a buffer.
- **Consumers** retrieve and process data from the buffer.
- **Synchronization** is managed using **mutex locks** and **condition variables** to ensure thread-safe access.

Each thread performs **100 operations** (producing or consuming items). The buffer size is **100**, preventing buffer overflow or underflow.

2. Characteristics of the Workload

- **Multi-threading & Parallelism**: Each producer-consumer pair runs independently, increasing parallelism.
- **Synchronization Overhead**: Mutex locks and condition variables introduce delays due to waiting threads.
- **Memory Access Patterns**: Frequent buffer read/write operations impact cache efficiency.
- **Scalability Challenge**: Performance may degrade with excessive threads due to increased context switching.

3. Objective of Performance Analysis

- Measure **execution time** across different thread counts.
- Analyze **throughput (items/sec)** to evaluate efficiency.
- Investigate **context switching, cache performance, and CPU cycles** using perf.

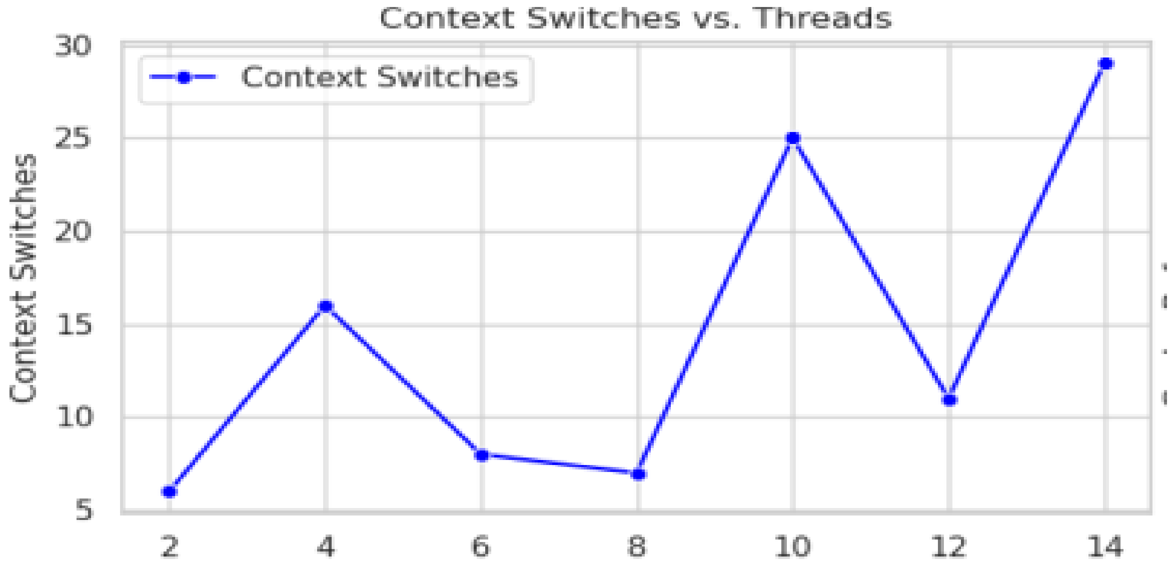
3. Perf Analysis

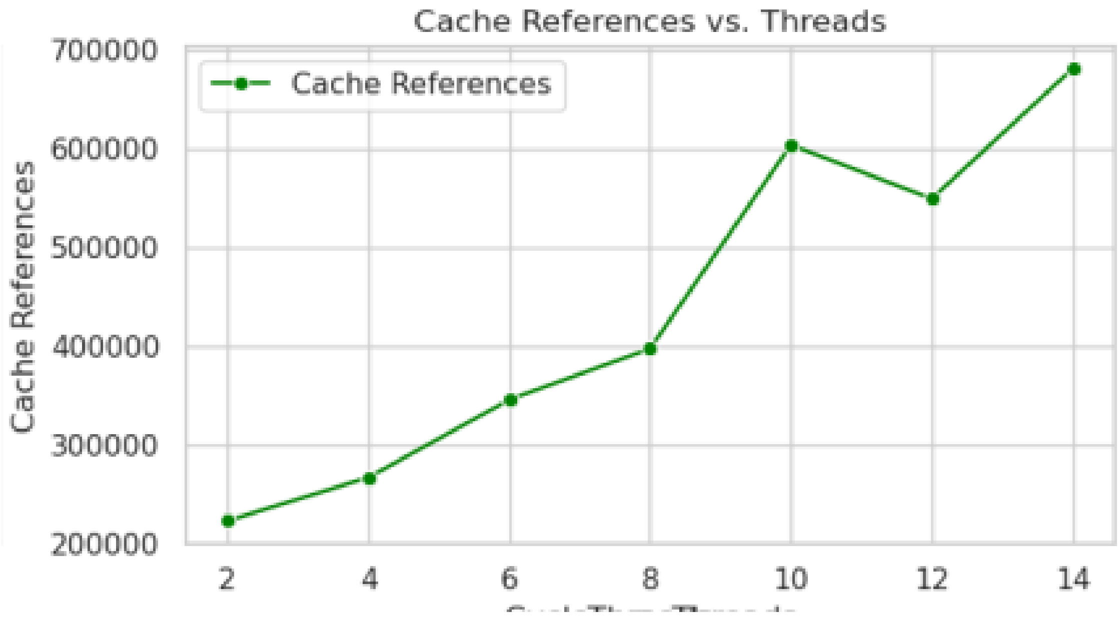
The **Producer-Consumer** workload was analyzed using the perf tool to measure key system performance metrics. The evaluation was conducted across varying thread counts to study how synchronization and parallelism impact execution.

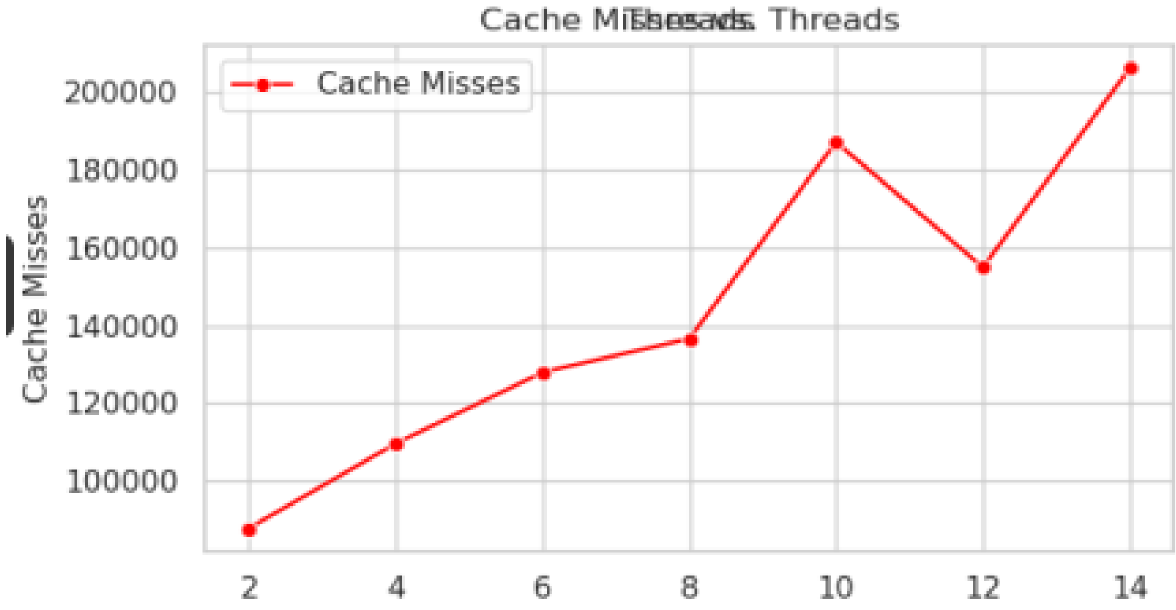
- **Key performance metrics observed–**

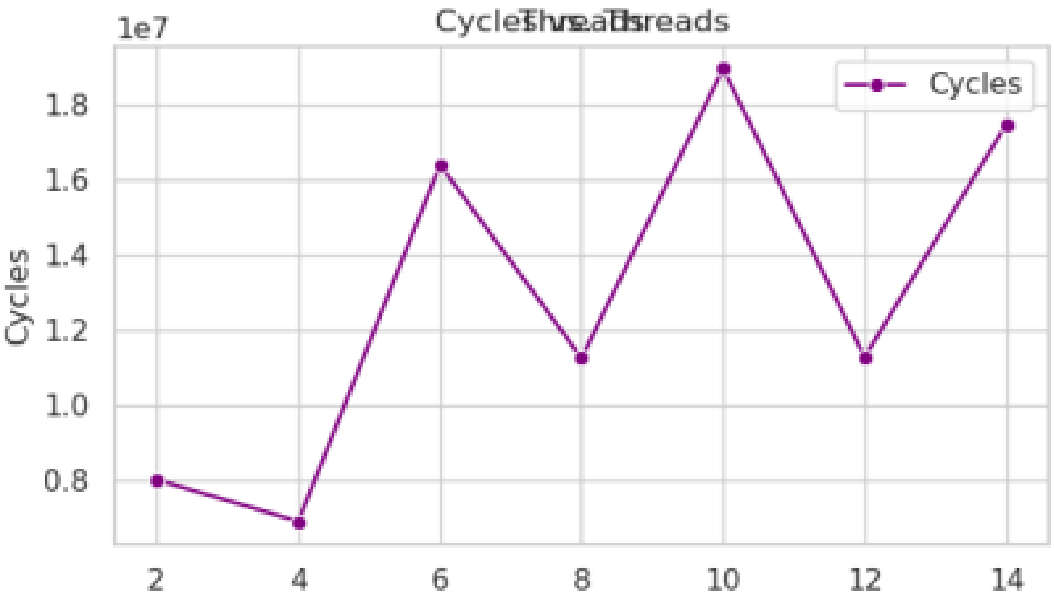
    1. **Execution Time**
        - Measured total runtime for different thread counts.
        - Decreases initially with more threads but stabilizes due to synchronization overhead.
    2. **Throughput (items/sec)**
        - Number of processed items per second.
        - Peaks at an optimal thread count before declining due to contention.
    3. **Context Switches**
        - Indicates scheduling overhead.
        - Increases with higher thread counts due to mutex-based synchronization.
    4. **Cache References & Misses**
        - High cache misses indicate inefficient memory access patterns.
        - Shared buffer accesses lead to frequent cache invalidations.
    5. **CPU Cycles & Instructions**
        - Helps analyze CPU-bound performance.
        - Low instructions per cycle (IPC) suggest waiting on locks rather than execution.

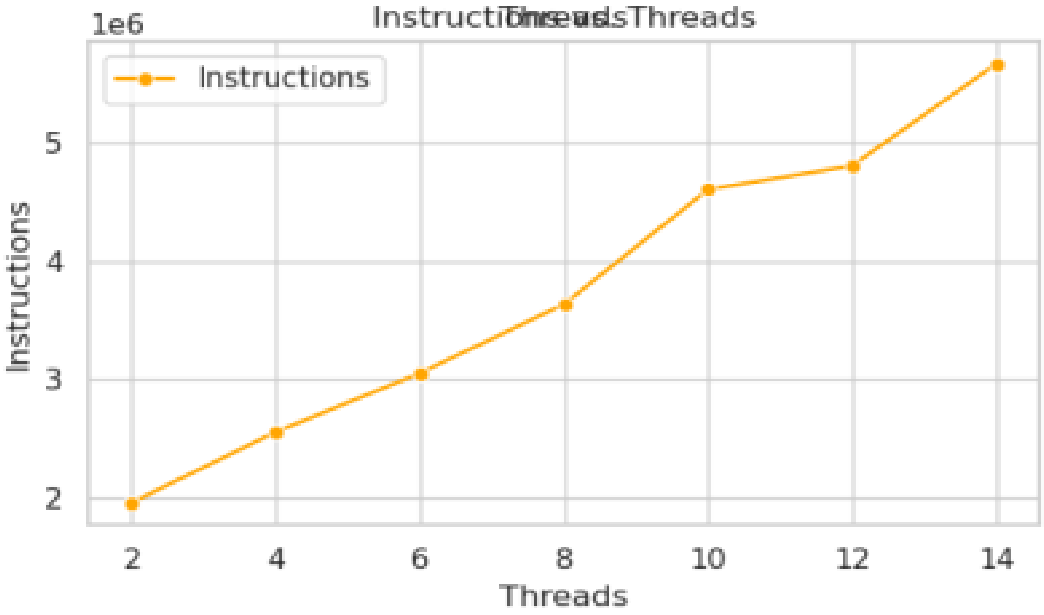
4. Graphs-


Context Switches vs. Threads
Context Switches
Context Switches
30
25
20
15
10
5
2
4
6
8
10
12
14

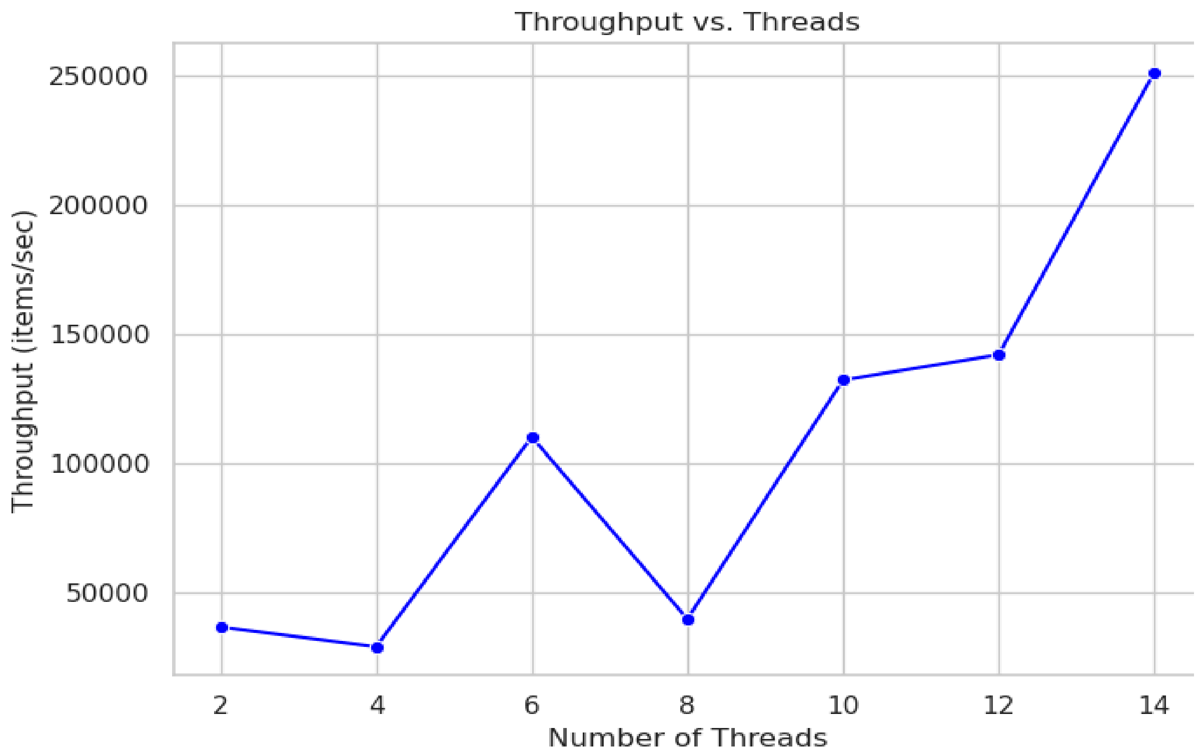

Cache References vs. Threads
Cache References
Cache References
700000
600000
500000
400000
300000
200000
2
4
6
8
10
12
14


Cache MiThesads. Threads
Cache Misses
200000
180000
160000
140000
120000
100000
Cache Misses
2
4
6
8
10
12
14


CycleThreadsThreads
1e7
Cycles
1.8
1.6
1.4
1.2
1.0
0.8
Cycles
2
4
6
8
10
12
14


Instructions vs. Threads
1e6
Instructions
5
4
3
2
Instructions
2
4
6
8
10
12
14
Threads


Execution Time vs. Threads
0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
Execution Time (seconds)
2
4
6
8
10
12
14
Number of Threads



5. Screenshots of perf tool outputs for each program.

perf_2.txt

```
Execution Time: 0.005414 seconds
Throughput: 36941.26 items/sec

 Performance counter stats for './pipeline 2':

                 6      context-switches
           222,782      cache-references
            87,906      cache-misses                     #   39.46% of all cache refs
         8,013,472      cycles
         1,954,127      instructions                     #    0.24  insn per cycle

       0.011720948 seconds time elapsed

       0.001115000 seconds user
       0.003347000 seconds sys
```

perf_4.txt

```
Execution Time: 0.013643 seconds
Throughput: 29319.06 items/sec

 Performance counter stats for './pipeline 4':

                16      context-switches
           266,670      cache-references
           109,683      cache-misses                     #   41.13% of all cache refs
         6,909,402      cycles
         2,552,311      instructions                     #    0.37  insn per cycle

       0.014234687 seconds time elapsed

       0.000000000 seconds user
       0.003300000 seconds sys
```

perf_6.txt

```
Execution Time: 0.005429 seconds
Throughput: 110517.59 items/sec

 Performance counter stats for './pipeline 6':

                 8      context-switches
           345,365      cache-references
           127,959      cache-misses                     #   37.05% of all cache refs
        16,401,226      cycles
         3,049,603      instructions                     #    0.19  insn per cycle

       0.011863928 seconds time elapsed

       0.000000000 seconds user
       0.005838000 seconds sys
```

perf_8.txt

```
Execution Time: 0.019971 seconds
Throughput: 40058.08 items/sec

 Performance counter stats for './pipeline 8':

                 7      context-switches
           397,086      cache-references
           136,616      cache-misses                     #   34.40% of all cache refs
        11,258,390      cycles
         3,632,918      instructions                     #    0.32  insn per cycle

       0.031770227 seconds time elapsed

       0.000729000 seconds user
       0.004379000 seconds sys
```

perf_10.txt

```
Execution Time: 0.007547 seconds
Throughput: 132502.98 items/sec

 Performance counter stats for './pipeline 10':

                25      context-switches
           603,307      cache-references
           187,160      cache-misses                     #   31.02% of all cache refs
        18,952,759      cycles
         4,604,097      instructions                     #    0.24  insn per cycle

       0.009369498 seconds time elapsed

       0.001422000 seconds user
       0.005689000 seconds sys
```

perf_12.txt

```
Execution Time: 0.008437 seconds
Throughput: 142230.65 items/sec

 Performance counter stats for './pipeline 12':

                11      context-switches
           548,790      cache-references
           154,947      cache-misses                     #   28.23% of all cache refs
        11,280,430      cycles
         4,804,881      instructions                     #    0.43  insn per cycle

       0.007223084 seconds time elapsed

       0.001005000 seconds user
       0.003017000 seconds sys
```

perf_14.txt

```
Execution Time: 0.005566 seconds
Throughput: 251527.13 items/sec

 Performance counter stats for './pipeline 14':

                29      context-switches
           680,232      cache-references
           206,057      cache-misses                     #   30.29% of all cache refs
        17,472,720      cycles
         5,667,387      instructions                     #    0.32  insn per cycle

       0.015733204 seconds time elapsed

       0.000000000 seconds user
       0.007264000 seconds sys
```

6. Discussion of results, including bottlenecks and optimization opportunities.

The performance analysis of the Producer-Consumer workload revealed several key insights regarding multithreading efficiency, synchronization overhead, and resource utilization.

- ## **Bottlenecks–**

1. **Synchronization Overhead**

    - The use of mutexes and condition variables leads to contention when multiple threads attempt to access the shared buffer.
    - As thread count increases, the time spent waiting for locks rises, reducing the benefits of parallelism.

2. **Context Switching Overhead**

    - Higher thread counts increase the number of context switches, adding scheduling overhead.
    - Excessive context switching leads to wasted CPU cycles, negatively impacting performance.

3. **Cache Performance Issues**

    - Shared buffer access leads to cache invalidations and higher cache misses, especially at larger thread counts.
    - Increased memory contention results in more frequent data movement between caches and RAM, slowing execution.

4. **Throughput Saturation**

    - Throughput increases initially but plateaus beyond an optimal number of threads.
    - At very high thread counts (e.g., 50+ threads), throughput declines due to lock contention and scheduling overhead.

Optimization Opportunities

1. **Reducing Lock Contention**

- Use **lock-free data structures** (e.g., atomic operations, concurrent queues) to minimize waiting time.
- Implement **fine-grained locking** instead of a single lock per buffer to allow more parallel access.

2. **Batch Processing**

    - Instead of **locking per item**, producers could insert **multiple items at once**, reducing synchronization frequency.
    - Consumers can process batches of items before signaling producers.

3. **Improving Cache Efficiency**

    - Use **thread-local buffers** before transferring data to the shared buffer to reduce cache invalidations.
    - Optimize memory layout to improve **spatial locality**, reducing cache misses.

4. **Reducing Context Switches**

    - Use **thread pinning** to keep threads on specific CPU cores, reducing cache migration overhead.
    - Increase **buffer size** to reduce contention and allow more concurrent processing.