Quad-Core MESI Cache Coherence Simulator Report : COL216

Krishna Purbia (2023CS51136)
Vishesh Sehrawat (2023CS51152)

April 30, 2025

1 Introduction

This project simulates a quad-core processor where each core has its own L1 cache. Cache coherence is maintained using the MESI (Modified, Exclusive, Shared, Invalid) protocol with a central(atomic) bus. Each cache is write-back, write-allocate, and uses an LRU (Least Recently Used) replacement policy. The simulation tracks core operations cycle-by-cycle, including cache hits, misses, evictions, writebacks, and coherence transactions.

The primary goal is to understand the interaction between caches in a multicore environment and how coherence overhead impacts overall system performance.

1.1 Data Structures

- **CacheLine**: Represents a block in the cache with fields for tag, MESI state, and LRU counter.
- **Cache**: Implements an L1 cache for each core with methods to find lines, handle LRU, and insert/update lines.
- **BusRequest**: Describes requests : BusRd, BusRdX, BusUpgr, and BusWB.
- **Stats**: Collects per-core statistics (reads, writes, misses, evictions, idle cycles, etc.).
- **Dequeue**: Used for processing BUS requests in order they are called.Due to BUSWB we have to use Dequeue otherwise Queue was enough.

1.2 MESI Protocol

The MESI protocol operates as follows:

- **Modified (M)**: Line has been modified; exclusive to this cache; must be written back on eviction.
- **Exclusive (E)**: Line is clean; exclusive copy.

- **Shared (S)**: Line is clean; may exist in multiple caches.
- **Invalid (I)**: Line is invalid.

Transitions happen based on processor read/write operations and bus transactions.

1.3 Simulator Flow

Each core executes one memory operation per cycle if not blocked. Misses trigger bus transactions that other cores snoop to maintain coherence.

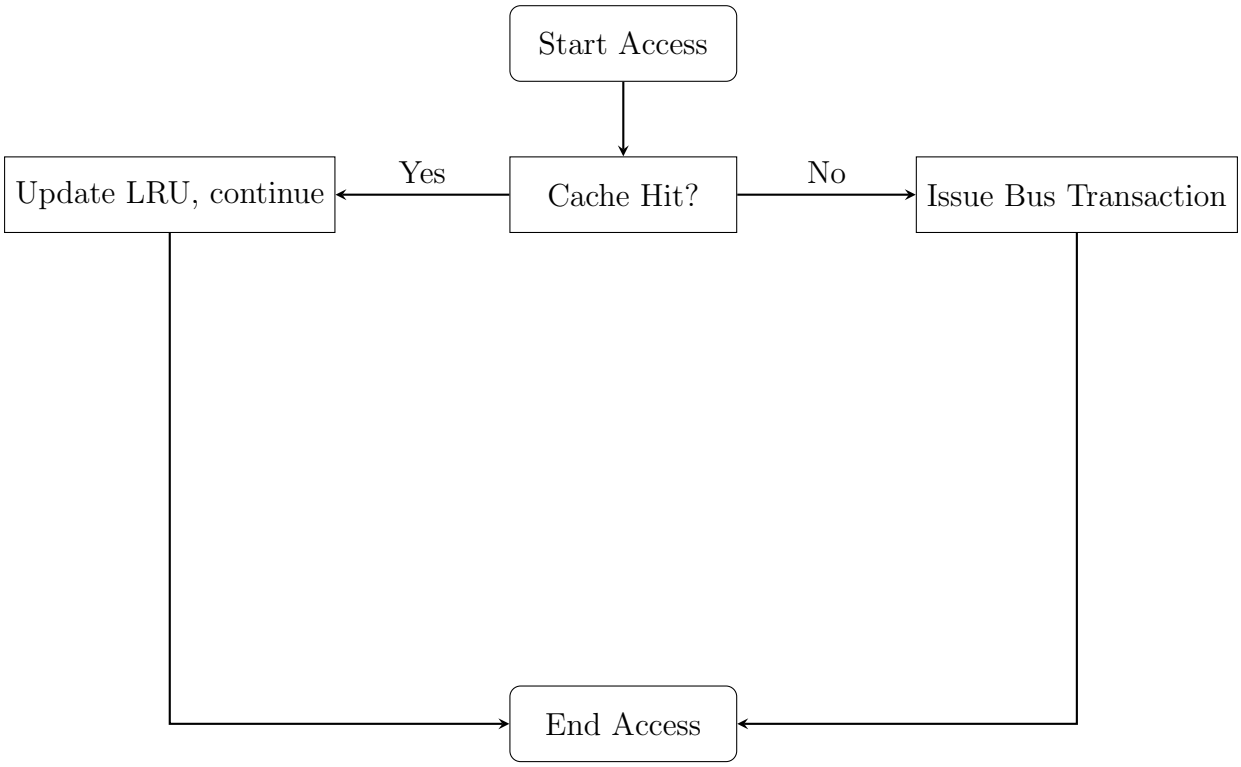

Figure 1: Cache access and coherence handling flow

2 Experimental Results

2.1 Default Setup

- Cache Size: 4KB per core
- Associativity: 2-way set associative
- Block Size: 32 bytes

Each simulation was repeated 10 times using the same trace files to check consistency.

Simulation Parameters

Parameter	**Value**
Trace Prefix	app1
Set Index Bits	6
Associativity	2
Block Bits	5
Block Size (Bytes)	32
Number of Sets	64
Cache Size (KB per core)	4
MESI Protocol	Enabled
Write Policy	Write-back, Write-allocate
Replacement Policy	LRU
Bus	Central snooping bus

Table 1: Cache Simulation Parameters

Per-Core Statistics

Metric	Core 0	Core 1	Core 2	Core 3
Total Instructions	2,497,349	2,490,468	2,509,057	2,503,127
Total Reads	1,489,888	1,485,857	1,492,629	1,493,736
Total Writes	1,007,461	1,004,611	1,016,428	1,009,391
Total Execution Cycles	2,538,789	2,570,948	2,655,201	2,903,543
Idle Cycles	14,431,309	14,232,852	15,178,932	14,109,780
Cache Misses	39,420	38,847	45,084	39,599
Cache Miss Rate	1.58%	1.56%	1.80%	1.58%
Cache Evictions	34,901	34,298	39,743	35,012
Writebacks	6,689	6,246	9,847	6,481
Bus Invalidations	57	31	102	44
Data Traffic (Bytes)	6,641	6,201	9,763	6,439

Table 2: Core-wise Cache and Execution Statistics

Overall Bus Summary

Metric	Value
Total Bus Transactions	192,077
Total Bus Traffic (Bytes)	29,044

Table 3: Overall Bus Summary

Observation: All Metrics remained constant for all 10 runs . As fix order for execution of cores.

3 Cache Parameter Variation

We varied one parameter at a time:

3.1 Effect of Cache Size

- Sizes tested: 256B, 1KB , 16KB, 4096KB
- Observation: Execution time decreases with cache size as less evictions will happen.

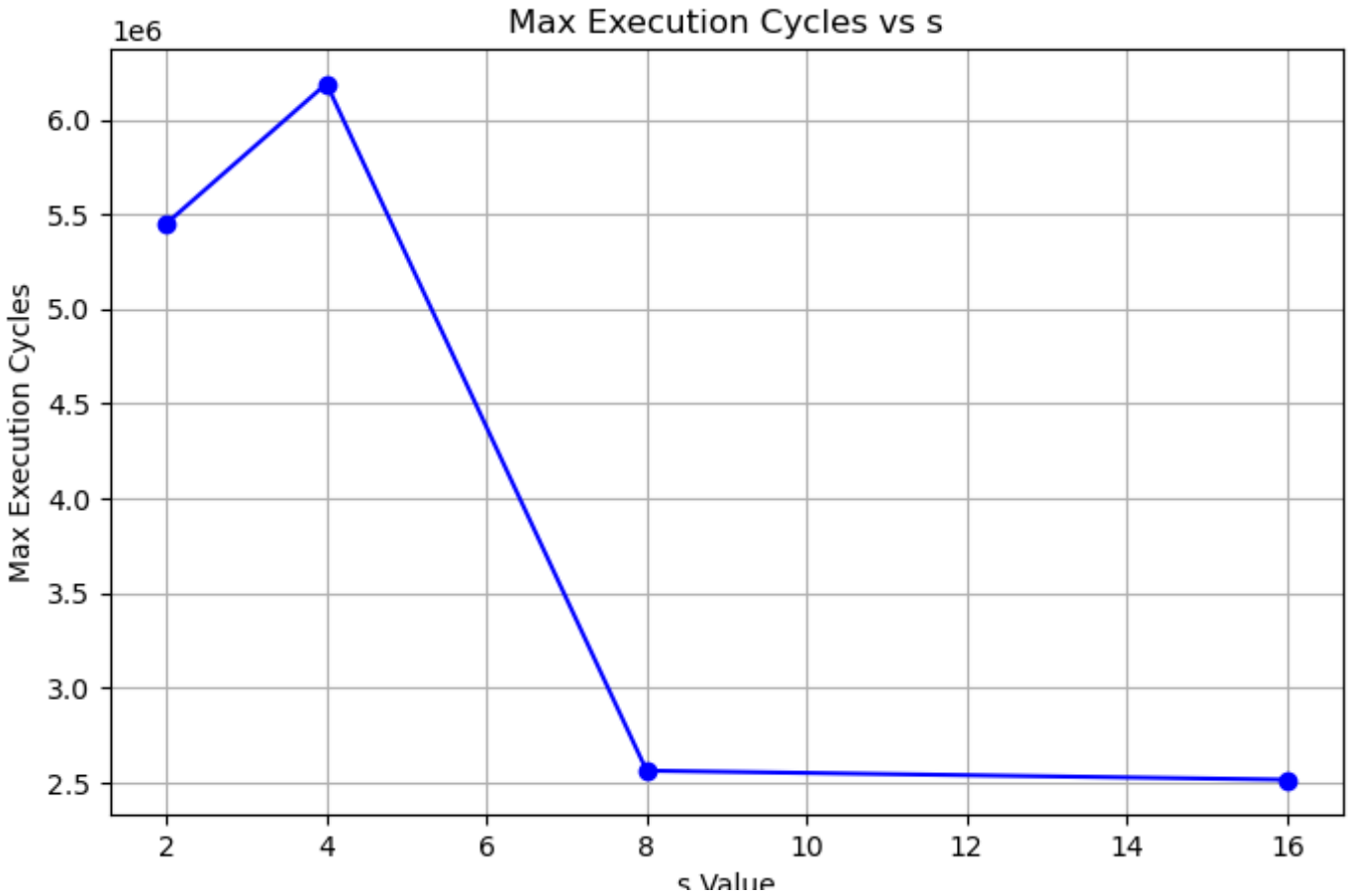

Figure 2: Execution Time vs Cache Size

3.2 Effect of Associativity

- Associativities tested: 1, 2 (default), 4, 8 ways
- Observation: Higher associativity reduces conflict misses .i.e Evictions.

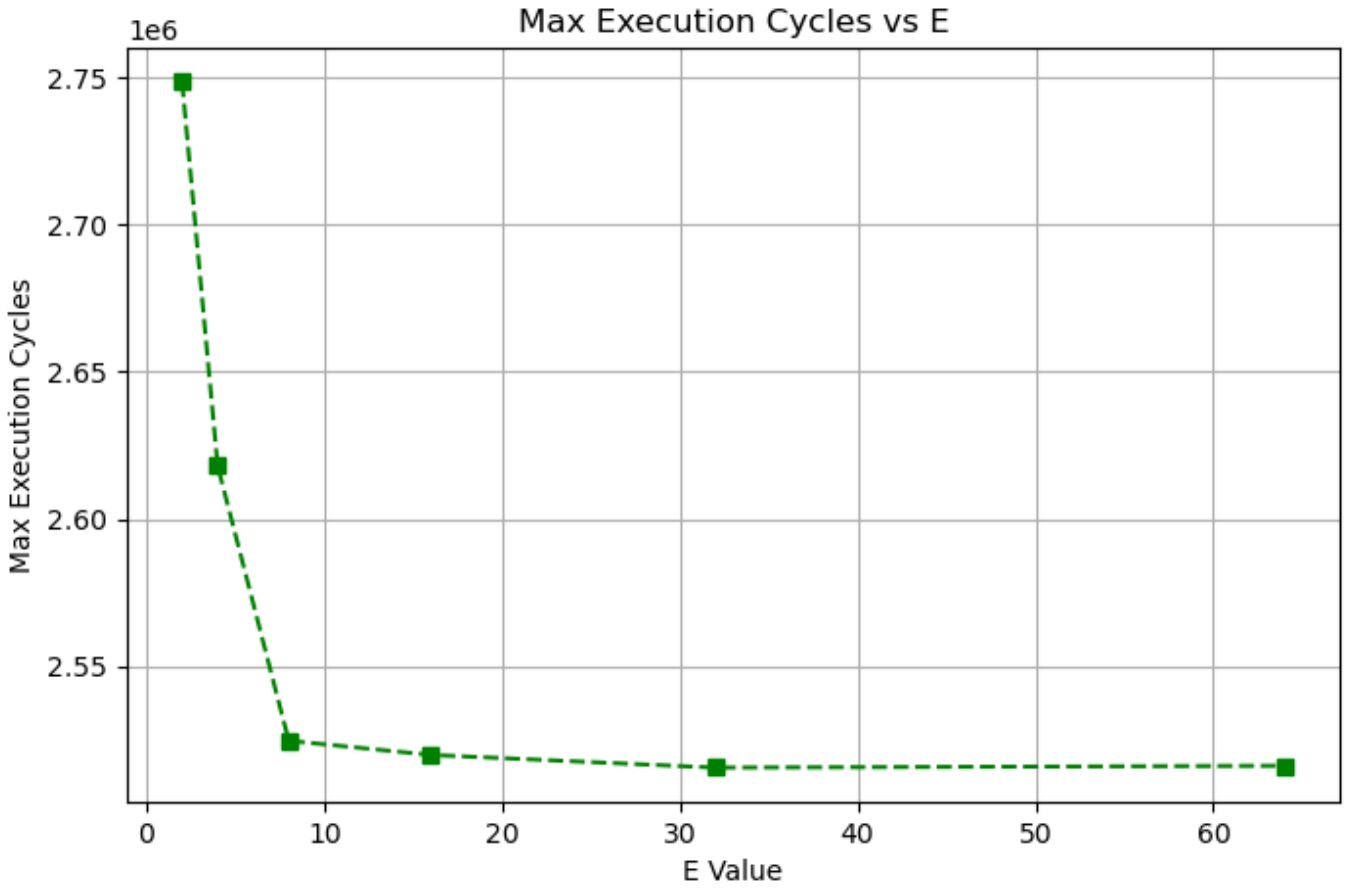

Figure 3: Execution Time vs Associativity

3.3 Effect of Block Size

- Block sizes tested: 4B, 16B , 256B, 65536B
- Observation: Due to increase (2*N) overhead the execution time increases.

Figure 4: Execution Time vs Block Size

4 Assumptions

- Used BusWD , BusRD and BusRdx for bus transactions only.
- When CoreX goes (M to S) due to some read miss in other core , the data is shared to other core through Cache to Cache transfer after writback.
- Cache uses BusRdx instead of BusUpgr when the memory refrence is Shared State.
- Cache to Cache transfers does'nt take place during Write instructions.
- Cores halt on cache misses until the request is resolved. Snooping for bus transactions continues during stalls.
- No, both cores do not stall during a cache-to-cache transfer. The requesting core stalls, waiting for the data. The responding core only stalls for the cycles it takes to put the block on the bus (typically 2N), and this counts as execution for the responding core.

  Only the requester is considered blocked for the full duration of the miss

- Both cores do not stall during a cache-to-cache transfer.The requesting core stalls, waiting for the data. The responding core only stalls for the cycles it takes to put the block on the bus (typically 2N), and this counts as execution for the responding core.
- At any point, the bus processes only one request at a time, whether it is a read miss, a write miss, a snooping invalidate, or a data transfer. New bus requests (including snooping actions) are only initiated when the bus becomes completely free. Thus, snooping and bus transactions are serialized, and happen strictly one after the other.
- Didn't model invalidates as time-taking events.
- INVAlID state happens at last of invalidations BusRequests for that memory refernce .
- When CoreX have memory refernce of address Y in M state and goes to S state due BusRdX of CoreZ , first writeback happens and then Cache to Cache Transfer.
- CoreX is condiered in execution either when there is Cache Hit or CoreX processing actively its own Busrequest after Data transfer.

5 Explanations

- Dequeue is used for management of BusRequets in order they were called. BusWB signals are pushed in front as they are part of either BusRd or BusRdx. Also, any MODIFY state will force BusRd or BusRdx to first writeback MODIFY and then again call BusRd or BusRdx.
- Majorly implemenataion is breaked in three parts : Core operations , BusRequest intialization , BusRequest execution.
- Invalidations are executed immediately.
- BusUpgr is skipped .Since , Cache to Cache transfers were not allowed.

6 Conclusion

We successfully implemented a MESI-based coherent L1 cache simulator for a quad-core system. The simulator tracks detailed performance statistics, and experiments show the influence of cache parameters on overall execution time. Larger caches and higher associativity generally improve performance by reducing miss and coherence penalties.

Future improvements could include adding non-blocking caches, multi-level caches (L2), and better bus arbitration models.

7 Refernces

- Wikipedia, “MESI protocol”
- IIT Kanpur, CS 610 Snoopy Protocol,CSE IIT Kanpur
- Bus Snooping, Wikipedia: “Snoopy cache monitors the bus transactions ”
- Stanford CS149, Memory Consistency Lecture: “Remote write: M→I, write back data”