

Cache Coherence Protocols

MSI Protocol

The MSI protocol is one of the most basic cache coherence protocols, employing three states: Modified (M), Shared (S), and Invalid (I). In this protocol, the states manage data consistency between caches and memory but come with significant overhead for data sharing and modification.

In MSI, the inefficiencies become clear when data is frequently shared and written to, requiring invalidations during writes and write-backs to memory when data is shared between caches. The simplicity of MSI results in high coherence traffic, especially in workloads where data is shared and modified frequently.

States

- **Modified (M):** The cache line is locally modified and is inconsistent with main memory. No other cache holds a valid copy, and the memory is stale.
- **Shared (S):** The cache line is valid, can be shared by multiple caches, and matches the data in memory. The memory holds the correct data.
- **Invalid (I):** The cache line is not valid in the cache, and the processor must fetch the data from another cache or from memory.

BUS Signals

When a processor reads from a cache line that is in the Invalid (I) state, a **BusRd (Bus Read)** signal is issued by the CPU. This is a bus request sent to retrieve the data, either from memory or another cache.

If other caches hold the data in the Shared state, they will supply the data, and the cache line in the requesting cache transitions to Shared (S). If no other caches hold the data, it is fetched from memory and transitions to Shared if it's only read, or to Modified (M) if a write will follow.

BUS Signals

When a processor writes to a cache line that is in the Invalid (I) or Shared (S) state, it issues a **BusRdX (Bus Read Exclusive)** signal, meaning the cache must retrieve the data in exclusive mode and invalidate any other copies in other caches.

This request ensures that no other cache holds the line in Shared anymore, and the requesting cache transitions to Modified (M). The BusRdX ensures that the processor now has exclusive access to the data for writing.

The BusRdX is called "exclusive" because it not only reads the data but also ensures that the requesting processor is the only one allowed to modify it, invalidating copies in other caches.

BUS Signals

BusUpgr (Bus Upgrade) is another bus signal used when a cache line is already in the Shared state and the processor wants to write to it. Since the cache already has the data, the cache doesn't need to fetch it again, but the BusUpgr signal is issued to invalidate the line in other caches, upgrading the line from Shared (S) to Modified (M) in the current cache.

MESI Protocol: Optimizing Private Data Handling

The MESI protocol improves upon MSI by adding a fourth state: Exclusive (E). This state optimizes situations where a cache holds data exclusively, allowing for more efficient data modification without unnecessary invalidations. The key idea is that data can be exclusively owned by a single cache without any need for other caches to be invalidated during a write, provided no other caches hold the data.

MESI optimizes for cases where the data is privately held by a single cache. This reduces invalidation traffic and improves performance when a cache can modify its data without needing to communicate with others. However, MESI still has inefficiencies when it comes to handling modified data that needs to be shared across caches, as it still requires a write-back to memory before the cache line can be shared.

Additional State

Exclusive (E): The cache line is valid and matches the data in memory, but no other caches hold it. The cache can transition the data directly from Exclusive to Modified (M) if the processor writes to it, without issuing any bus requests. This eliminates unnecessary bus traffic for private data modifications.

BUS Signals

When a processor reads a cache line and no other cache has the data, the cache line enters the Exclusive (E) state. If the processor writes to the line later, it transitions directly from Exclusive to Modified, avoiding any bus transactions because the data is already exclusive to this cache. This reduces unnecessary BusRdX or BusUpgr requests that were prevalent in MSI for similar cases.

When another processor issues a BusRd to a cache line in the Exclusive state, the cache responds with the data, and the line transitions to Shared (S) in both the reading cache and the cache that held it exclusively. This allows for efficient read-sharing, while still maintaining control over when the data transitions to a shared state.

MOSI Protocol: Addressing Write-Back Inefficiency

The MOSI protocol introduces the Owner (O) state to reduce the number of memory write-backs needed when modified data is shared between caches. In MESI, when a cache line is in the Modified state and another cache requests the data, the modified data must be written back to memory before it can be shared. The MOSI protocol solves this by allowing the modified data to be supplied directly by the cache holding the most recent version, without requiring a write-back to memory.

Additional State

Owner (O): The cache line is shared with other caches, but this cache holds the most recent (dirty) version of the data. The memory is out-of-date, but the cache holding the Owner state is responsible for supplying the data to other caches. This reduces the need to write modified data back to memory.

Advantage by reducing memory writebacks

In MOSI, if a processor reads a cache line that is modified in another cache, the cache holding the Modified line transitions to the Owner (O) state, and the requesting cache enters the Shared (S) state. The Owner cache supplies the data directly to the requesting cache without writing it back to memory first. This improves performance in shared modified data scenarios, where frequent memory write-backs would otherwise slow down the system.

The Owner state reduces the overhead of write-backs, especially in cases where multiple caches need access to modified data. Instead of writing modified data back to memory when sharing it, the cache that modified the data simply transitions to Owner and keeps supplying it as long as other caches request it.

MOESI Protocol: Combining the Best of MESI and MOSI

In MOESI, if a cache line is held privately, it can reside in the Exclusive (E) state, allowing the processor to modify it without any coherence traffic, as in MESI. If the data becomes shared and modified, the cache transitions to the Owner (O) state, optimizing the case where the most recent version of the data needs to be shared across caches. Other caches holding the line move to the Shared (S) state, and the cache holding the Owner (O) state supplies the data.

This dual mechanism allows MOESI to handle both private and shared data efficiently. For private data, the Exclusive state prevents unnecessary invalidations. For shared modified data, the Owner state eliminates unnecessary memory write-backs, reducing overhead in read-sharing scenarios.

MESIF Protocol: Optimizing Read-Heavy Workloads

The MESIF protocol is designed to optimize systems where data is read frequently across multiple caches. It adds the Forward (F) state, which designates one cache to respond to read requests for shared data, reducing bus contention when multiple caches hold the same cache line in the Shared (S) state.

Additional State

Forward (F): The cache line is shared, but only this cache is responsible for supplying data in response to read requests. Other caches holding the line in Shared remain passive and do not respond.

When a cache line is read by multiple processors and resides in the Shared state across multiple caches, only the cache in the Forward (F) state responds to future BusRd requests. This reduces the number of responses to read requests, minimizing bus traffic and improving efficiency in read-heavy workloads.

MESIF is particularly beneficial in situations where multiple caches frequently read the same data, as it prevents multiple caches from responding to every read request. By designating one cache to handle the responses, it minimizes unnecessary coherence traffic.

MOSIF

The MOSIF protocol integrates both the Owner and Forward states to handle both read-heavy and write-heavy scenarios effectively.

When data is frequently modified and shared, the Owner state ensures that modified data can be shared without being written back to memory. When data is frequently read across multiple caches, the Forward state ensures that only one cache responds to read requests, reducing bus contention.

MOESIF

The MOESIF protocol represents the most sophisticated version of the cache coherence protocols, combining the Modified (M), Owner (O), Exclusive (E), Shared (S), Invalid (I), and Forward (F) states. In MOESIF, private data is handled efficiently with the Exclusive state, while shared modified data benefits from the Owner state, and read-heavy workloads are optimized by the Forward state. This allows MOESIF to handle the widest range of workloads with minimal coherence traffic and bus contention.

In MOESIF, a processor read from shared data leads to only the cache in the Forward state responding, while shared modified data can be supplied by the Owner cache without requiring a write-back to memory. This combination allows for maximum efficiency, but also comes with the highest level of complexity, requiring advanced cache management to handle the six states effectively.