Broadcast-Based Cache Writes, Snoop Bus, and Directory Coherence

1. Introduction

In shared-memory multiprocessor systems, maintaining cache coherence is crucial to ensure that processors have a consistent view of shared data. This requires mechanisms for propagating updates or invalidations to other caches when data is written. The three primary techniques to achieve this are **broadcast-based cache writes**, **snoop buses**, and **directory-based coherence**.

2. Broadcast-Based Cache Writes

2.1 Definition

In broadcast-based cache writes, when a processor writes to a cache line, the operation is broadcasted to all other processors in the system. This ensures all caches are updated or invalidated with minimal delay.

2.2 Types of Broadcast-Based Writes

1. **Write-Invalidate Protocol**:
   - The writing processor invalidates the cache line in all other processors.
   - Future reads from other processors result in a cache miss.
2. **Write-Update Protocol**:
   - The new data is broadcasted, and all processors update their caches.

2.3 Mathematical Model

The time required for a broadcast-based cache write can be expressed as:

$$T_{\text{broadcast}} = T_{\text{local}} + T_{\text{propagation}} + T_{\text{update/invalidation}}$$

Where:

- T_{local}: Time to perform the write in the local cache.
- $T_{\text{propagation}}$: Time to propagate the write to other processors.
- $T_{\text{update/invalidation}}$: Time taken by other processors to update or invalidate their caches.

Communication Overhead: The bandwidth $B_{\text{broadcast}}$ required for broadcast:

$$B_{\text{broadcast}} = N \cdot S_{\text{msg}} / T_{\text{cycle}}$$

Where:

- N: Number of processors.
- S_{msg}: Size of the broadcast message.
- T_{cycle}: Time per communication cycle.

3. Snoop Bus

3.1 Definition

A **snoop bus** is a hardware-based mechanism for cache coherence in shared-memory systems. All processors monitor (or "snoop") a shared communication bus for coherence events, such as reads or writes, to maintain a consistent cache state.

3.2 How It Works

1. **Shared Bus**:
   - Processors share a communication bus for memory operations.
2. **Monitoring**:
   - Each cache monitors the bus for read or write operations related to its cached data.
3. **Coherence Actions**:
   - If a processor detects a relevant operation, it updates or invalidates its cache line as needed.

3.3 Coherence Protocols for Snoop Bus

1. **MESI Protocol**:
   - Modified, Exclusive, Shared, Invalid states.
   - Ensures coherence through state transitions based on snoop bus events.
2. **MOESI Protocol**:
   - Adds an "Owned" state to MESI for efficient sharing of modified data.

3.4 Mathematical Model

The total time for a snoop bus operation:

$$T_{\text{snoop}} = T_{\text{bus\_access}} + T_{\text{cache\_action}}$$

Where:

- $T_{\text{bus\_access}}$: Time to detect and process the snoop request.
- $T_{\text{cache\_action}}$: Time to update or invalidate the cache line.

Bandwidth Usage: The snoop bus bandwidth B_{snoop}:

$$B_{\text{snoop}} = \frac{N_{\text{requests}} \cdot S_{\text{msg}}}{T_{\text{cycle}}}$$

Where:

- N_{requests}: Number of memory requests monitored.
- S_{msg}: Size of the coherence message.

4. Directory-Based Coherence

4.1 Definition

Directory-based coherence uses a centralized or distributed directory to track the state of each cache line in the system. The directory maintains information about which caches hold copies of a memory block and whether the block is modified, shared, or invalid.

4.2 How It Works

1. **Directory Entries**:
   - Each memory block has an associated directory entry.
   - Tracks which processors have cached the block and its state (Modified, Shared, Invalid).
2. **Coherence Actions**:
   - Read or write requests are sent to the directory.
   - The directory coordinates updates or invalidations across caches.

4.3 Directory States

1. **Uncached**: No processor has the cache block.
2. **Shared**: The block is cached by multiple processors.
3. **Exclusive**: The block is cached by one processor and matches memory.
4. **Modified**: The block is cached by one processor and differs from memory.

4.4 Mathematical Model

Directory Storage Overhead: The directory storage requirement:

$$S_{\text{directory}} = N_{\text{blocks}} \cdot (P + S_{\text{state}})$$

Where:

- N_{blocks}: Number of memory blocks.
- P: Number of processors (bits to identify processors caching a block).
- S_{state}: Storage required for the state of each block.

Total Access Time: The access time for a directory-based coherence system:

$$T_{\text{directory}} = T_{\text{lookup}} + T_{\text{propagation}} + T_{\text{update/invalidation}}$$

Where:

- T_{lookup}: Time to lookup the directory.
- $T_{\text{propagation}}$: Time to propagate updates or invalidations.
- $T_{\text{update/invalidation}}$: Time taken by caches to update or invalidate data.

Scalability: Directory-based systems scale better than broadcast or snoop systems as the number of processors increases, with communication complexity proportional to $O(\log P)$.

5. Comparison of Techniques

Feature	**Broadcast-Based Writes**	**Snoop Bus**	**Directory Cohere**
Scalability	Limited (high bandwidth usage)	Moderate (bus contention)	High (centralized/d
Communication	High (broadcast to all)	Medium (snoop traffic)	Low (targeted mess
Memory Overhead	Low	Low	High (directory sto
Latency	Low for small systems	Medium	Low (scalable syste

6. Conclusion

Broadcast-based writes, snoop buses, and directory-based coherence are three key techniques for maintaining cache coherence in shared-memory multiprocessor systems. While broadcast and snoop-based systems are simpler and effective for small-scale systems, directory-based coherence scales better for large systems due to reduced communication overhead. Mathematical models help optimize these mechanisms by balancing latency, bandwidth, and storage requirements.