Shared Symmetric Memory Systems

Computer Architecture

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

Increasing importance of multiprocessors

- There is a decrease in silicon and energy efficiency as more ILP is exploited.
  - Cost of silicon and energy grows faster than performance.
- Increasing interest in high performance servers.
  - *Cloud computing*, *software as a service*, ...
- Data intensive applications growth.
  - Huge amounts of data on the Internet.
  - *Big data analytics*.

TLP: Thread level parallelism

- TLP implies the existence of multiple program counters.
  - Assumes MIMD.
  - Generalized use of TLP outside scientific computing is relatively recent.
  - New applications:
    - Embedded applications.
    - Desktop.
    - High-end servers.

Multiprocessors

- A **multiprocessor** is a computer consisting of highly coupled processors with:
  - **Coordination and use** typically controlled by a **single operating system**.
  - **Memory sharing** through a **single shared memory space**.

- **Software models**:
  - **Parallel processing**: Coupled set of cooperating threads.
  - **Request processing**: Independent process execution originated by users.
  - **Multiprogramming**: Independent execution of multiple applications.

- Most common approach:
  - From 2 to tenths of processors.
  - Shared memory.
    - Implies shared memory.
    - Does not necessarily imply a single physical memory.

- **Alternatives**:
  - **CMP** (*Chip Multi Processors*) or *multi-core*.
  - Multiple chips.
    - Each one may (or may not) be *multi-core*.
  - **Multicomputer**: Weakly coupled processors not sharing memory.
    - Used in large scale scientific computing.

- **Maximizing exploitation** of multiprocessors:
  - With **n** processors, at least **n** processes or threads are needed.

- Threads **identification**:
  - Explicitly identified by programmer.
  - Created by operating system from requests.
  - Loop iterations generated by parallel compiler (e.g. OpenMP).

High-level identification performed by programmer or system software with threads having **enough** number of instructions to execute.

Multiprocessors and shared memory

SMP: Symmetric Multi-Processor

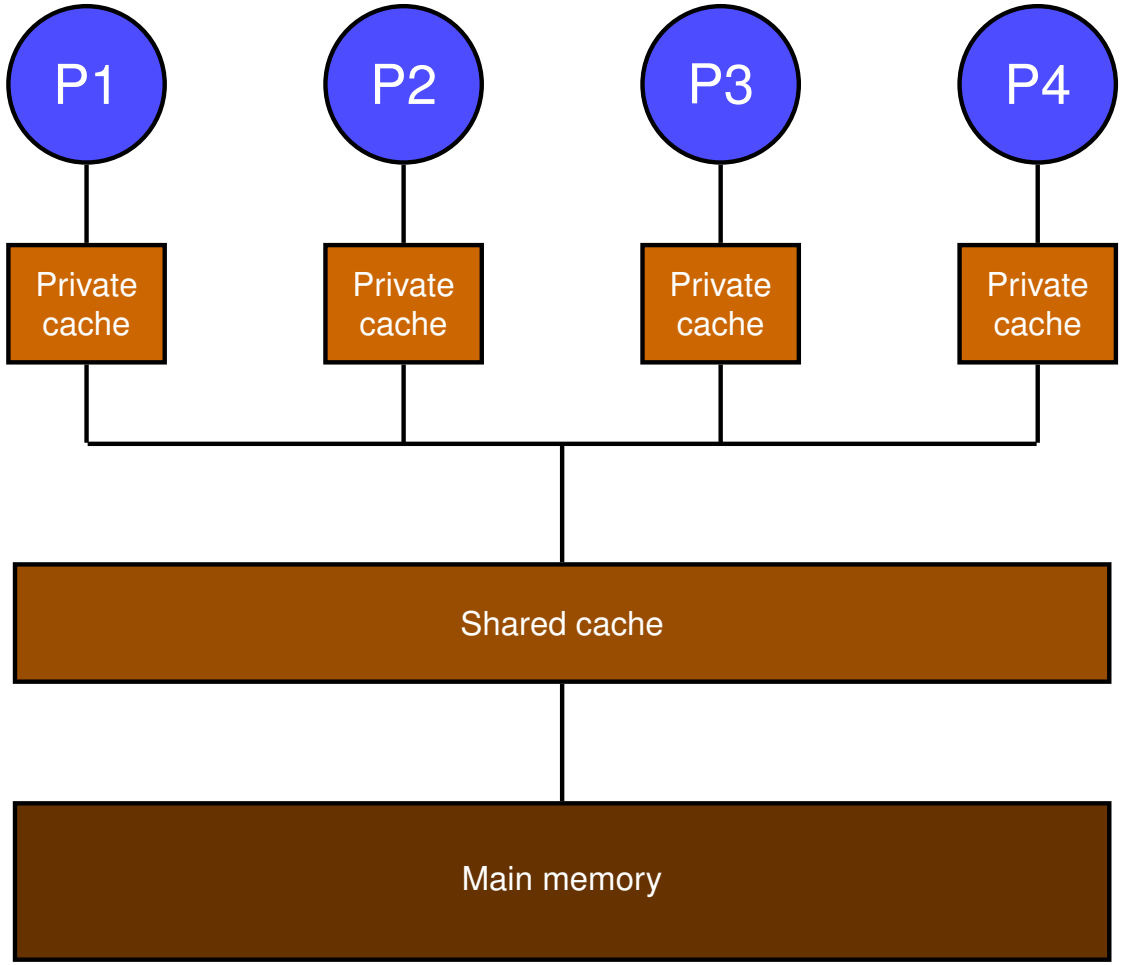
- Centralized shared memory.
- Share a single centralized memory where all have equal access time.
- All multi-cores are SMP.
- **UMA**: Uniform Memory Access
  - Memory latency is uniform.

DSM: Distributed Shared Memory

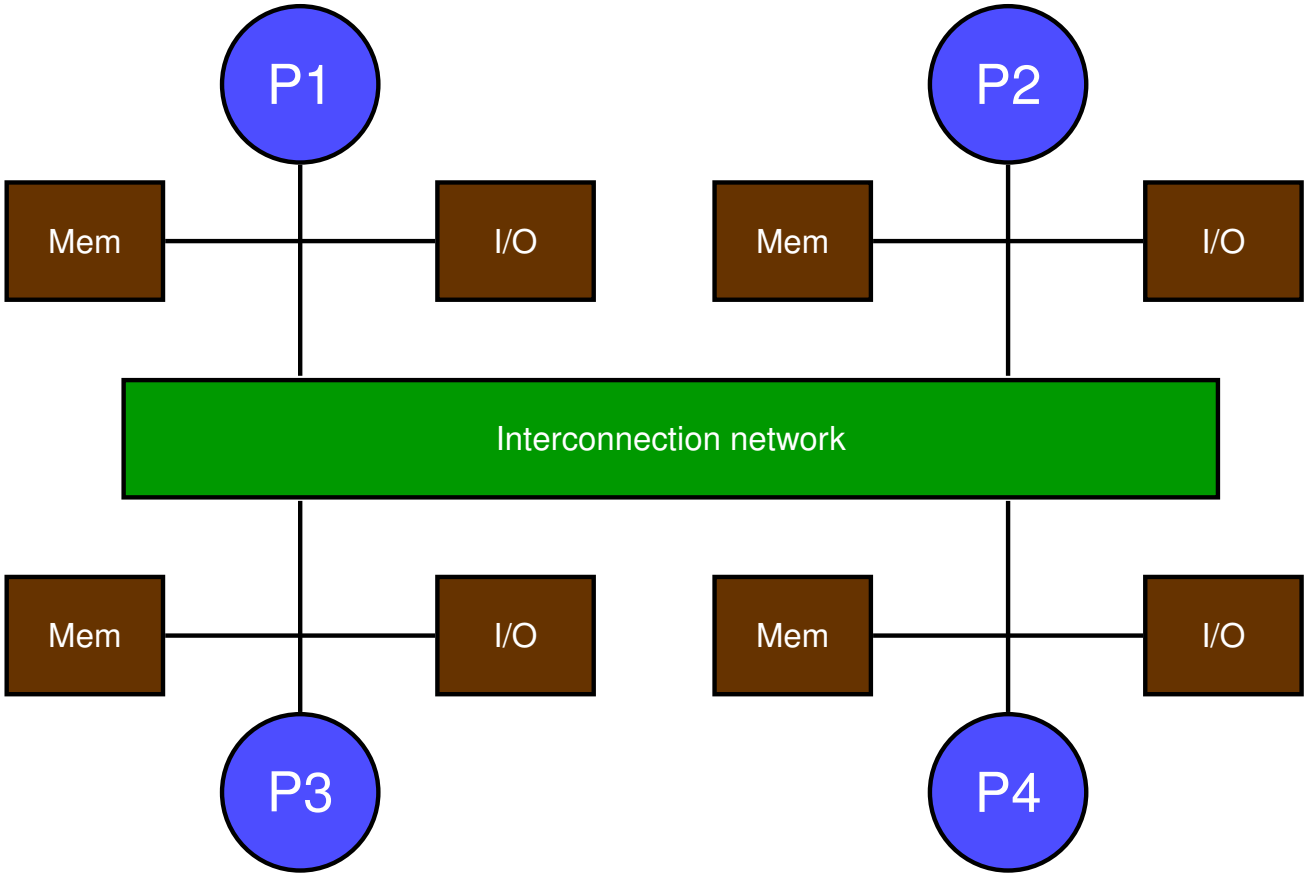
- Memory is distributed across processors.
- Needed when the number of processors is high.
- **NUMA**: Non Uniform Memory Access.
  - Memory latency depends on data location.

- Communication through access to global variables.

SMP: Symmetric Multi Processor



DSM: Distributed Shared Memory



SMP and memory hierarchy

- Why using centralized memory?
  - Multi-level large caches decrease memory bandwidth demand on main memory accesses.

- **Evolution**:
  1. Single-core with memory in **shared bus**.
  2. Memory connection in **separated bus** only for memory.

Cache memory

- **Kinds of data** in cache memory:
  - **Private data**: Data used by a single processor.
  - **Shared data**: Data used by multiple processors.

- **Problem** with shared data:
  - Datum may be replicated in multiple caches.
  - Contention is decreased.
    - Each processors accesses its local copy.
  - If two processors modify their copies . . .
    - Cache coherence?

Cache coherence

Thread 1

```
lw $t0, dirx
addi $t0, $t0, 1
sw $t0, dirx
```

Thread 2

```
lw $t0, dirx
```

- **$t0** initially 1.
- Assuming write through.

Process	Instruction	P1 Cache	P2 Cache	Main memory
T1	Initially	Not present	Not present	1
T1	**lw $t0, dirx**	1	Not present	1
T1	**addi $t0, $t0, 1**	1	Not present	1
T2	**lw $t0, dirx**	1	1	1
T1	**sw $t0, dirx**	2	1	1

Cache incoherence

- Why does incoherence happen?
  - **State duality**:
    - **Global state** → **Main memory**.
    - **Local state** → **Private cache**.

- A memory system is **coherent** if any read from a location returns the **most recent value** that has been written to that location.

- **Two aspects**:
  - **Coherence**: Which value does a read return?
  - **Consistency**: When does a read get the written value?

Conditions for coherence

- **Program order preservation**
  - A read from processor P on location X after a write from processor P on location X, without intermediate writes on X by any other processor Q, always returns the value written by P.
- **Coherent view of memory**:
  - A read from processor P on a memory location X, after a write form other processor Q on location X, returns the written value if both operations are separate enough in time and there are no intermediate writes on X.
- **Writes serialization**:
  - Two writes on the same memory location by two different processors are seen in the same order by all the processors.

Memory consistency

- Defines in which point in time a process reading values will see a written value.

- **Coherence** y **consistency** are complementary:
  - **Coherence**: Behavior of reads and writes on a single memory location.
  - **Consistency**: Behavior of reads and writes with respect to accesses to other memory locations.

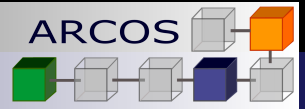
- There are different consistency memory models.
  - **We will have a specific lecture on this problem**

Coherent multiprocessors

- A **coherent multiprocessor** offers:
  - Shared data **migration**.
    - A datum may be moved to a local cache and be used transparently.
    - Decreases remote data access latency and bandwidth demand to shared memory.
  - Shared data **replication** simultaneously read.
    - Performs data copy in local cache.
    - Decreases access latency and read contention.
- **Critical properties for performance**:
  - **Solution**: Hardware protocol for keeping cache coherence.

Kinds of cache coherence protocols

- **Directory** based:
  - Sharing state is kept in a directory.
  - **SMP**: Centralized directory in memory or in LLC (*Last-level cache*).
  - **DSM**: To avoid bottlenecks a distributed directory is used (more complex).
- **Snooping**:
  - Each cache keeps the sharing state of each block that it stores.
  - Caches accessible through a broadcasting device (bus).
  - All caches monitor broadcasting device to determine if they have a copy of the block.



Coherence maintenance

- **Write invalidation**:
  - Guarantees that a processor has **exclusive access** to a block before performing a write.
  - **Invalidates** the rest of copies that other processors might have.
- **Write updates** (*write broadcasting*):
  - Broadcasts all writes to **all caches** to modify block.
  - Makes use of **more bandwidth**.
- Most common strategy $\Rightarrow$ **Invalidation**.

Memory bus use

- **Invalidation**.
  - Processor acquires bus and broadcasts the address to be invalidated.
  - All processors are snooping the bus.
  - Each processor checks if it has in cache the broadcasted address and invalidate it.
- There cannot be two simultaneous writes:
  - Exclusive use of bus serializes writes.
- **Cache misses**:
  - **Write through**:
    - Memory contains the last performed write.
  - **Write back**:
    - If a processor has a modified copy, it sends it to a cache miss from the other processor.

Implementation

- **Invalidation**:
  - Takes advantage from **validity bit** (V) associated to each block.
- **Writes**:
  - Need to know if there are **other copies** in cache.
    - If there are no other copies **write broadcast** is not needed.
  - **Sharing bit** (S) is added to each associated block.
  - When there is a write:
    - **Bus invalidation** is generated.
    - Transition from **shared state** to **exclusive state**.
    - No need to send new invalidations.
  - When there is a **miss cache** in other processor:
    - Transition from **exclusive state** to **shared state**.

Basic protocol

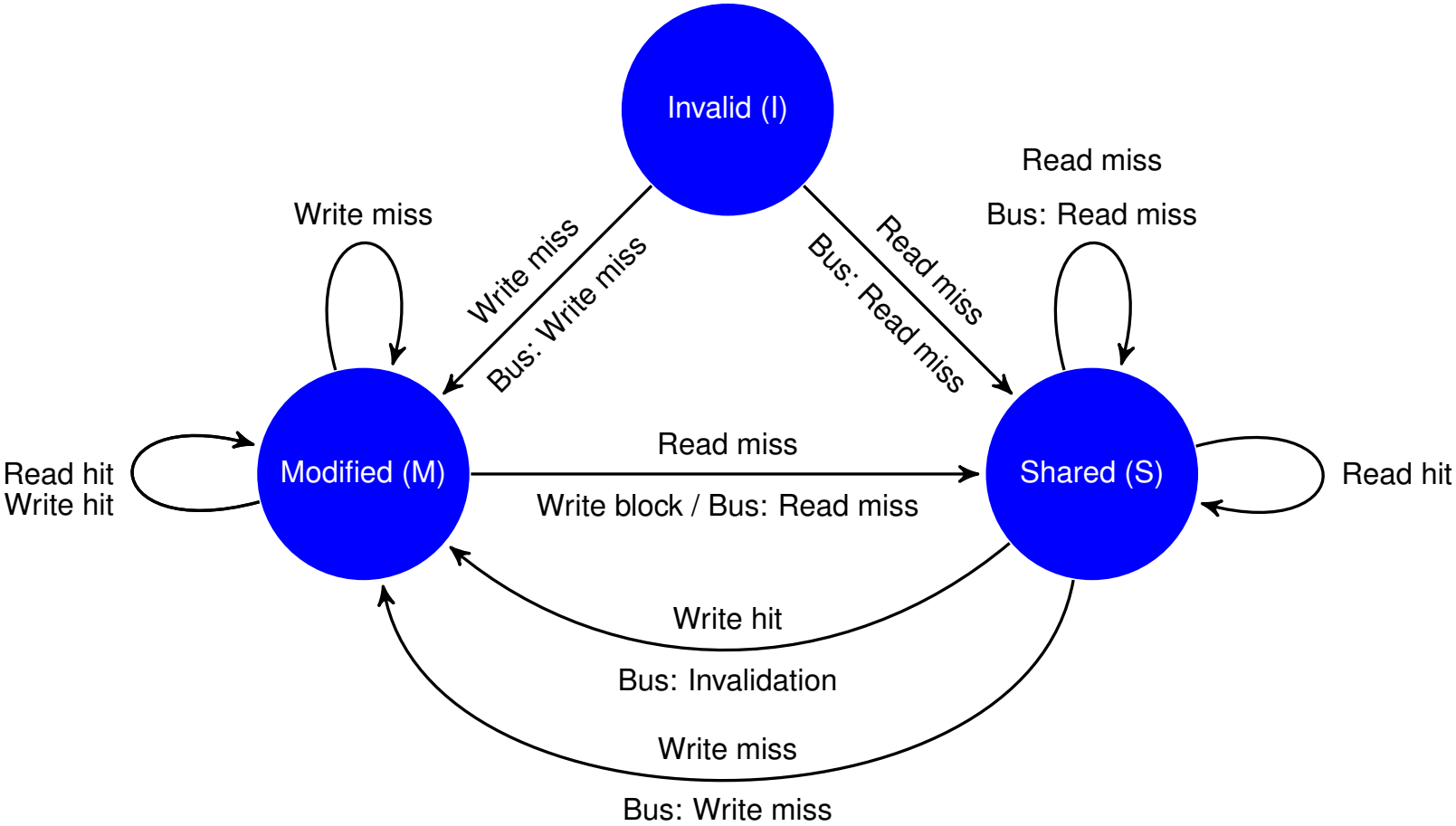
- Based in **state machine** for **each cache block**:
  - **State changes** generated by:
    - Processor requests.
    - Bus requests.
  - **Actions**:
    - State transitions.
    - Actions on the bus.

- Simple approach with **three states**:
  - **M**: Block has been modified.
  - **S**: Block is shared.
  - **I**: Block has been invalidated.

Actions generated by processor

Request	State	Action	Description
Read hit	S $\rightarrow$ S	Hit	Read data from local cache
Read hit	M $\rightarrow$ M	Hit	Read data from local cache
Read miss	I $\rightarrow$ S	Miss	Broadcast read miss on bus.
Read miss	S $\rightarrow$ S	Replacement	Address conflict miss. Broadcast read miss on bus.
Read miss	M $\rightarrow$ S	Replacement	Address conflict miss. Write block and broadcast read miss.
Write hit	M $\rightarrow$ M	Hit	Write data in local cache.
Write hit	S $\rightarrow$ M	Coherence	Bus invalidation.
Write miss	I $\rightarrow$ M	Miss	Broadcast write miss on bus.
Write miss	S $\rightarrow$ M	Replacement	Address conflict miss. Broadcast write miss on bus.
Write miss	M $\rightarrow$ M	Replacement	Address conflict miss. Write block and broadcast write miss.

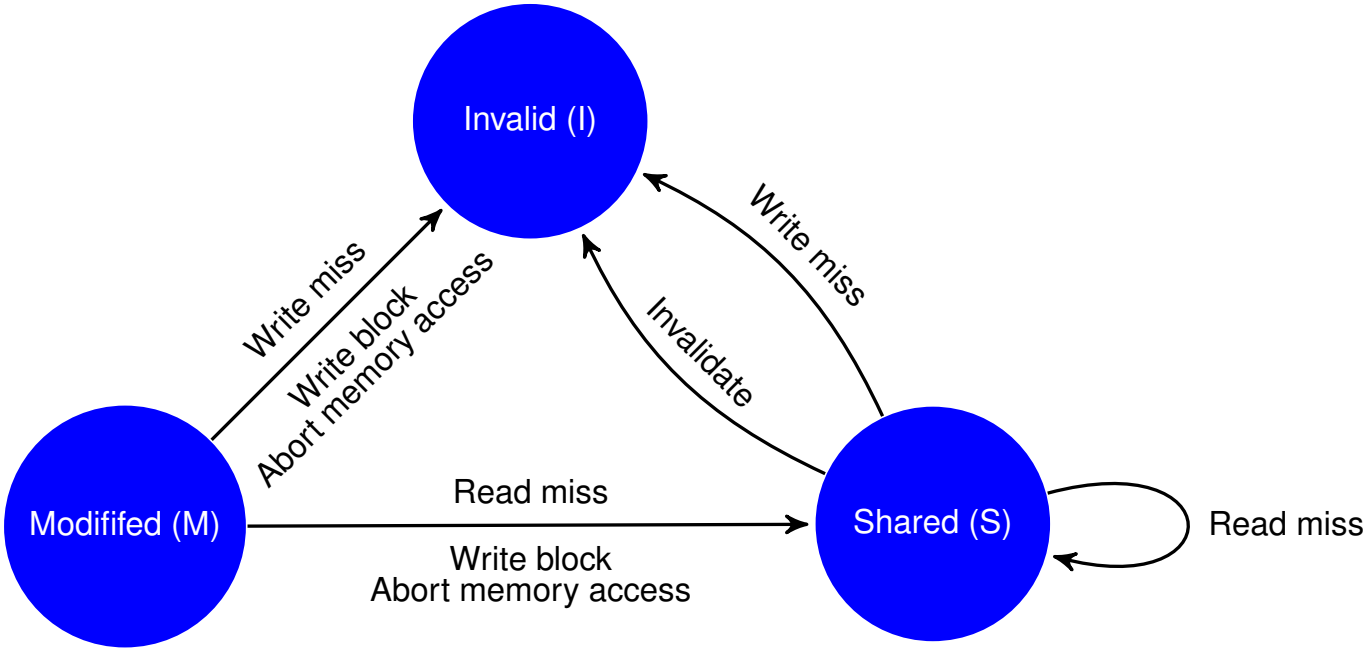
MSI Protocol: Processor actions



Actions generated by bus

Request	State	Action	Description
Read miss	S $\rightarrow$ S	–	Shared memory serves miss.
Read miss	M $\rightarrow$ S	Coherence	Attempt to share data. Place block on bus.
Invalidate	S $\rightarrow$ I	Coherence	Attempt to write a shared block. Invalidate block.
Write miss	S $\rightarrow$ I	Coherence	Attempt to write a shared block. Invalidate block.
Write miss	M $\rightarrow$ I	Coherence	Attempt to write a block that is exclusive elsewhere Write back cache block.

MSI Protocol: Bus actions

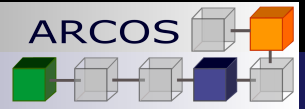


MSI protocol complexities

- Protocol assumes that operations are **atomic**.
  - **Example**: It is assumed that a miss can be detected, bus acquired, and response received in a single action without interruption.
- If operations are not **atomic**:
  - Possibility for a deadlock or data race.
- **Solution**:
  - Processor sending invalidation keeps **bus ownership** until invalidation arrives to the rest of processors.

Extensions to MSI

- **MESI**:
  - Add **exclusive state** (E) signaling that a block lives in a single cache but is not modified.
  - Writing of an **E** block **does not generate invalidations**.
- **MESIF**:
  - Adds **forward state** (F): Alternative to **S** signaling which node must answer each request.
  - Used by Intel Core i7.
- **MOESI**:
  - Adds **owned state** (O) signaling that block in memory is not updated.
  - Avoids memory writes.
  - Used by AMD Opteron.

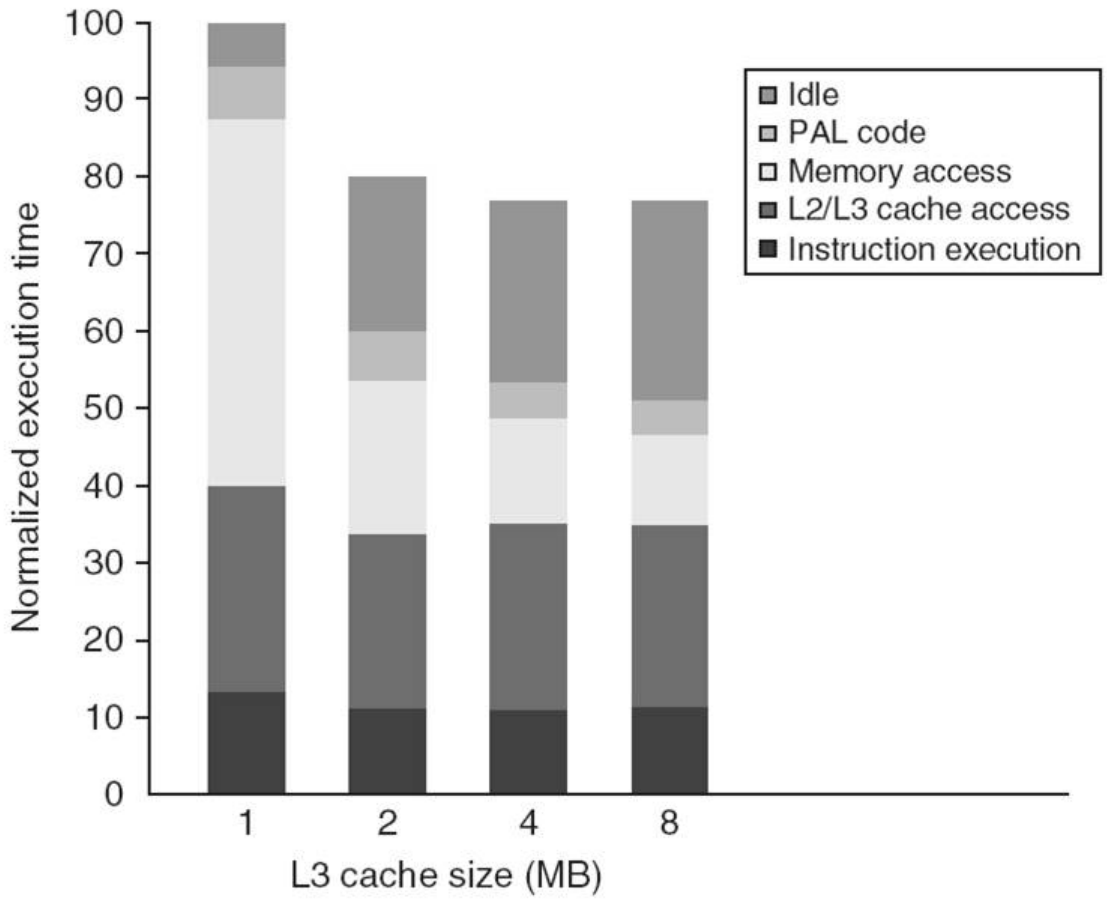


Performance

- Use of cache coherence policies has impact on miss rate.

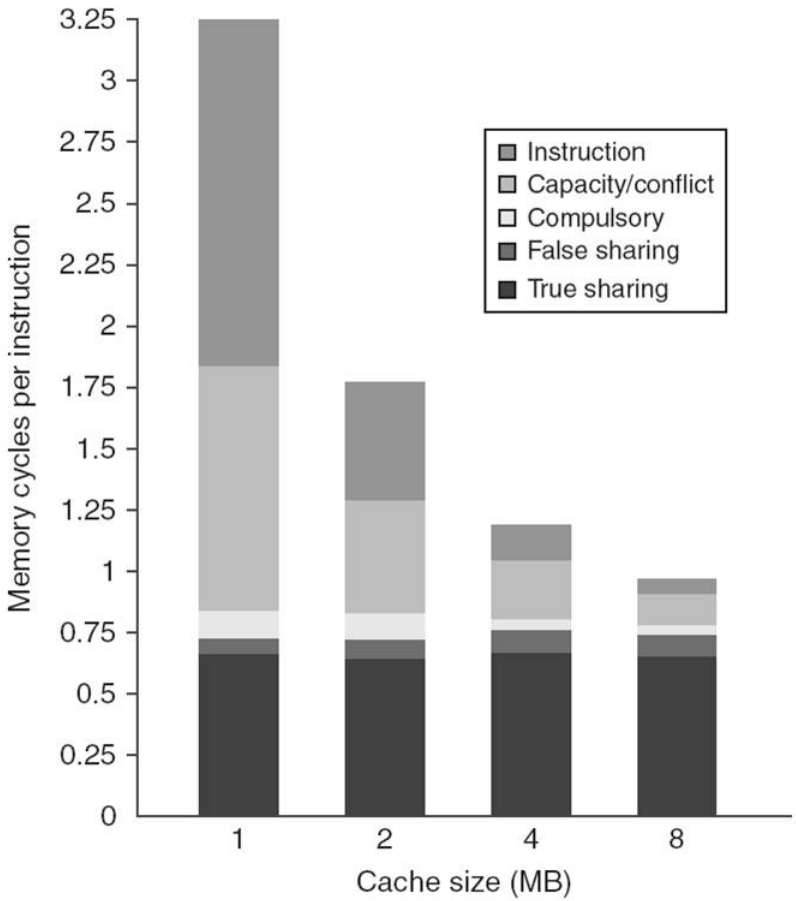
- **Coherence misses emerge**:
  - **True sharing** misses:
    - A processor writes to a shared block and invalidates.
    - A different processor reads a shared block.
  - **False sharing** misses:
    - A processor writes a shared block and invalidates it.
    - A different processor reads a different word from the same block.

Performance when increasing L3



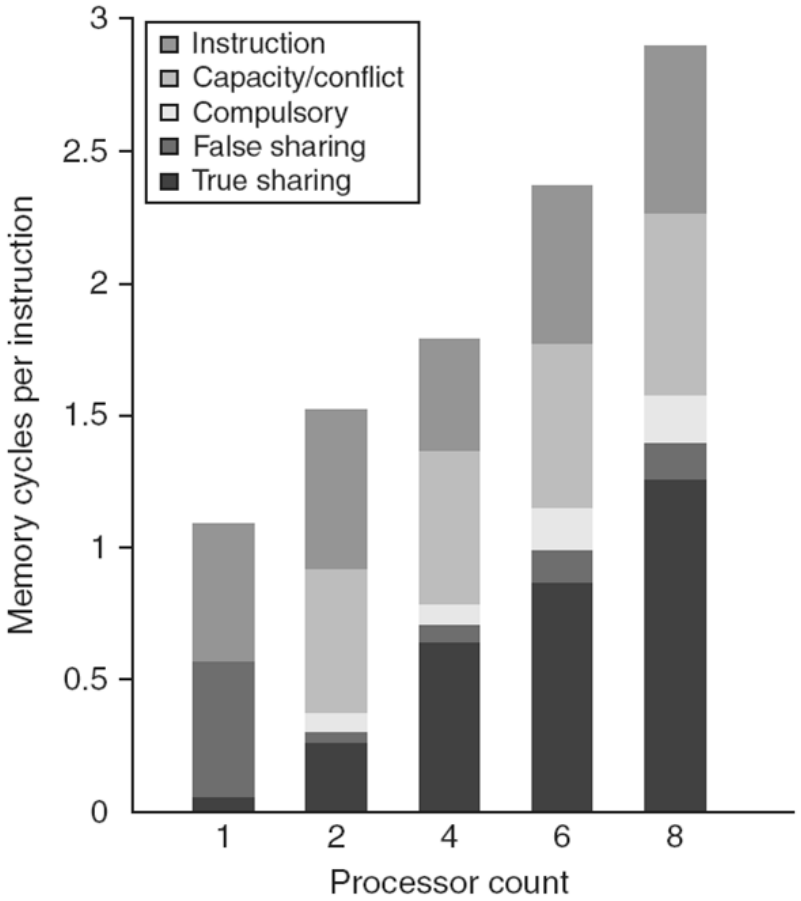
Source: Computer Architecture: A Quantitative Approach. 6 Ed
Hennessy and Patterson. Morgan Kaufmann. 2017.

Contributions to L3 misses



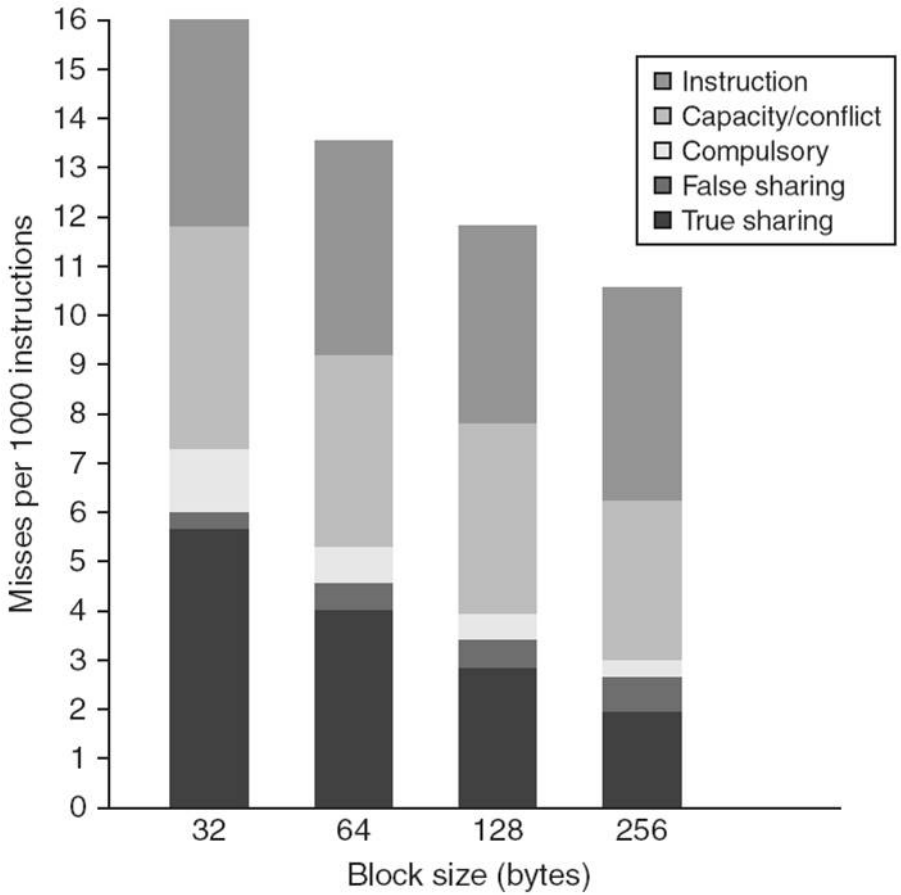
Source: Computer Architecture: A Quantitative Approach. 6 Ed
Hennessy and Patterson. Morgan Kaufmann. 2017.

Increasing number of processors



Source: Computer Architecture: A Quantitative Approach. 6 Ed Hennessy and Patterson. Morgan Kaufmann. 2017.

Increasing block size



Source: Computer Architecture: A Quantitative Approach. 6 Ed Hennessy and Patterson. Morgan Kaufmann. 2017.

Summary

- Multiprocessor as computer with multiple highly coupled processors with coordination, use, and memory sharing.
- Multiprocessors classified into SMP (Symmetric multiprocessors) and DSM (Distributed Shared Memory).
- Two aspects to consider in memory hierarchy: coherence and consistency.
- Two alternatives in cache coherence: directory and snooping.
- Snooping protocols do not require a centralized element.
  - But they generate more bus traffic.

References

- **Computer Architecture. A Quantitative Approach**
  5th Ed.
  Hennessy and Patterson.
  Sections: 5.1, 5.2, 5.3.

- **Recommended exercises**:
  - 5.1, 5.2, 5.3, 5.4, 5.5, 5.6.

Shared Symmetric Memory Systems

Computer Architecture

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid