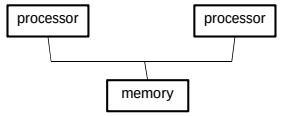
Shared Memory Multiprocessors and Cache Coherence

Kai Shen

Shared Memory Multiprocessors

- Limitation of instruction-level parallelism
  - Dependences
  - Complexity to support high-degree instruction-level parallelism
- Multiple processors sharing memory

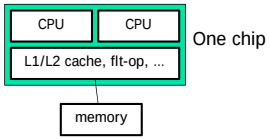


Multiprocessor Architecture: Hardware Multithreading

- Core CPU contains
  - the instruction fetching, decoding, pipelining units etc. (with registers)
- Other peripheral things on processor
  - L1/L2 cache, floating-point units, etc.

⇨ Hardware multithreading

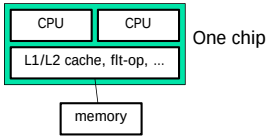
- Multiple CPUs share the same set of peripheral things so they can be manufactured on the same silicon die



Multiprocessor Architecture: Hardware Multithreading

⇨ Hardware multithreading

- Multiple CPUs share the same set of peripheral things so they can be manufactured on the same silicon die

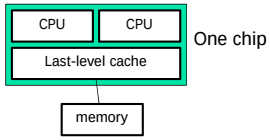


- Benefits:
  - Less manufacturing cost ⇨ more CPUs on the same silicon die area
  - Less power consumption for multiple CPUs
  - Faster CPU-to-CPU coordination and data sharing
- Problems:
  - Resource contention diminish benefits, leads to unpredictable performance

Multiprocessor Architecture: Multicore

- Last-level cache (LLC) is most significant part of the chip

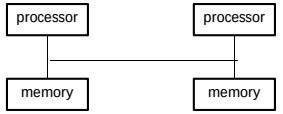
⇨ Multicore: just sharing the LLC allows multiple CPU on the same silicon die



- Between traditional multiprocessor and multithreading
  - Inherit much of the benefits for hardware multithreading
  - Only resource contention is on the shared LLC space

Multiprocessor Architecture: NUMA

- Memory bandwidth is a big problem for large-scale multiprocessor
- Non-Uniform Memory Access
  - Each processor can still access all memory, but accesses are faster to "local memory"

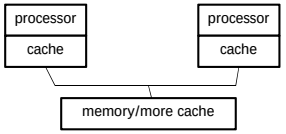


Our Test Machines

- node17, node19-node28
  - two CPU chips, each containing two hardware threads
- node33-node72
  - same layout, faster processors
- node4x2a
  - four CPU chips, each containing two cores, each further containing two hardware threads

Cache Coherence Problem

- In shared memory multiprocessors (except multithreading), each processor has a local cache



- For each data item in memory, additional copies may exist in processor local caches
  - after one processor updates the data, another processor's local copy may be incoherent
  - What is wrong about it?

Cache Coherence

- Coherence means the system semantics is the same as that of a system without processor-local caches
- Multiprocessor cache coherent if there exists a hypothetical sequential order of all operations for each data location:
  - returned value in the read operation is that written by last write in the sequential order
  - the sequential order matches the order of operations from each processor

Cache Coherence Through Bus Snooping

- All caches and memory connected by a shared bus
- Bus snooping
  - Each processor can monitor the bus for activities
- Not always the case (particularly for NUMA)

Bus Snooping For Write-Through Caches

- Cache can be write-through or write-back
- Assume write-through cache
  - Every write goes to the bus
- Bus snooping
  - Each processor monitors the bus for writes
  - If there is a local cached copy of the write target, it is invalidated or updated.
    - Tradeoff between invalidation vs. update?
- What is the hypothetical sequential order?

Coherence on Write-Back Caches

- Write-back caches are much more favored in practice
- Writes do not necessarily go to bus, so we may not directly snoop them

MSI Write-Back Invalidation

- Three states for a cache entry
  - Modified (M) – I modified it, I am the only one who has a copy
  - Shared (S) – I have a clean copy, possibly shared by others
  - Invalid (I)
- Write to a modified cache entry?
- Write to a shared/invalid cache entry?
  - Local write preceded by a read-exclusive (must go to bus, invalidate other caches' copies)
- How to handle a read?
- Operations on bus? Things that are snoop-able.
  - Read-exclusive, read (miss local cache), write back
- What is the hypothetical sequential order?

MESI Write-Back Invalidation

- Modified (M) – I modified it, I am the only one who has a copy
- Shared (S) – I have a clean copy, possibly shared by others

⇨ I have a clean copy. But can't tell if I am the only one who has a copy.

- When does it matter?
  - I read an entry into cache and then write to it
  - With S state, write must be preceded by a read-exclusive
- Add a new state
  - Exclusive clean (E)
- When to assign E to an entry?
  - First read, no other cache has a copy

Directory-based Cache Coherence

- No all multiprocessors use shared bus for memory access
  - It does not scale!
- Large multiprocessors with NUMA
  - Many local memory accesses
  - With the ability of bus snoop, an explicit directory about cache state can be used