Parallel Processors

- Introduction of Multiprocessor Architecture
  - more and more servers using multiprocessors
- Multiprocessor Taxonomy
  - MIMD
  - SIMD
  - SISD

MIMD Alternatives

- MIMD
  - centralized shared memory multiprocessor
  - distributed shared memory multiprocessor

Centralized Shared Memory

- Bus-based multiprocessors
  - multiple processors share a single memory
    - UMA - uniform memory access
  - Caches reduce the traffic requirements
  - Bus interconnects processors, memory, I/O

Cache Coherency Problem

- What happens if shared item is written?
  - old copies reside in cache of other processors
  - stale data values are used
  - need to eliminate old values when writing

```
x, y = 0
x <-- 1
x = x + y
```
Snoopy Coherence Schemes

- Invalidate all copies of a cache block when it is written; only single copy in writer’s cache in the system
- How do you know if there is a copy?
  - snoop in cache on writes/cache misses, invalidate copies
  - snooping limits scalability — misses must check all caches

```
x, y = 0
x <-- 1
invalidate x
x = x + y
```

```
y <-- 1
invalidate y
y = x + y
```

CPU

x, y = 0
x <-- 1
invalidate x
x = x + y

CPU

y <-- 1
invalidate y
y = x + y

snooping path to cache

memory

I/O devices

Snooping Protocol

- All cache controllers monitor or snoop on the bus to determine whether or not they have a copy of the shared block
- Snooping protocols are of two types:
  - write-invalidate: the writing processor invalidates all copies in other caches before changing its local copy; it is then free to update the local data until another processor asks for it
  - write-update: the writing processor broadcasts the new data over the bus; all copies are then updated with the new value. This scheme is also called write-broadcast
- Write-update is like write-through, all writes go over the bus to update copies of the shared data
- Write-invalidate has similar benefits to write-back in terms of reducing demands on bus bandwidth

Single Bus Multiprocessor Using Snooping Protocol

A Simple Cache Coherence Protocol

- A write-invalidate protocol based on write-back policy
- Each cache block is in one of three states:
  - Read only (clean): The cache block is clean (not written) and may be shared
  - Read/Write (dirty): The cache block is dirty (written) and is not shared
  - Invalid: This cache block does not have valid data
A Write-Invalidate CC Protocol

Shared (clean)

Invalid

Modified (dirty)

write-back caching protocol in black

read (hit or miss)

write (hit or miss)

read (hit) or write (hit or miss)

write-back due to read (miss) by another processor to this block

write-back due to read (miss) by another processor to this block

send invalidate

receives invalidate (write by another processor to this block)

send invalidate

write-back caching protocol in black

signals from the processor

coherence additions in blue

signals from the bus

coherence additions in green

A write-invalidate cache-coherence protocol

invalid

clean (read only)

processor write miss

write back dirty block to memory

processor read miss

dirty (read/write)

send invalidate if hit

processor write

Cache state transitions using signals from processor
Commercial Shared-Bus Multiprocessors

- Use write-back cache because write-back reduces bus traffic and thereby allows more processors on a single bus.
- Use write-invalidate as the standard protocol.
- Many processors have instructions such as cache line flush, cache line load, etc to support cache coherence protocol (Pentium, Power PC, MIPS 10000 etc)