Iomem
use multimemory. SMPs.

classmate
Date 12.2.2020
Page 1

SHARED MEMORY MULTIPROCESSORS

~~AAAAAA~~ Symmetric (shared memory multiprocessor) ~~Ar~~ referred to as SMPs. centralized shared memory multiprocessors.

SMA. (multimemory systems).
most existing multicores are symmetric processors.

these SMPs are refferred to as uniform memory access. (UMA)
multiple multicore chips, we have got separate memory for each multi-core chip. memory is distributed. each multicore chip has its own memory.

we have distributed memory (NUMA) rather than centralized memory. distributed shared memory multiprocessors. these are refferred to as non-uniform memory access. powers referred to as NUCA
challeges of 4 processors.

in the shared system, and any distributed system. (SMP and DSM) the address space is shared.
challenges of parallel processing. (1) there could be limited. parallelism available.

(2) relatively high cost of ~~parallelism~~ communication.

CENTRALIZED SHARED MEMORY ARCHITECTURE

(1) use of multilevel. caches can substantially reduce. the memory bandwidth demands of a processor.

backside or memory bus to distinguish it from the bus used to IO operations. In this case, access to memory is asymmetric.

MULTIPROCESSOR CACHE COHERENCE

arise because of caching of shared data. view of memory by 2 different processors is through their individual caches.

How will a processor come to know which processors are caching the same variables?

memory consistency a memory system is **coherent** with any read of the data item, returns the most recently written value of that data.

2 different aspects of memory system behaviour.

① coherence

② consistency

Coherence ⇒ what value can be returned by a read.

consistency ⇒ determines when a written value will be returned by a read.

a memory system is coherent if

① a read of processor p to location x that follows a write by p to x, with no writes of x by another processor occurring between the write and the read by p always returns the value written by p.

② a read by a processor to location x that follows a write by another processor to x returns the written value if the read and write are sufficiently separated in time, an no other writes to x occur between the two accesses.

③ writes to the same location are serialized, that is 2 writes to the same location by any 2 processors are seen in the same order by all processors.

the issue of exactly when a written value must be seen by a reader is defined by a memory consistency model.

Coherence and consistency are complimentary.

coherence defines the behaviour of reads & writes to the same memory location,

while consistency defines the behaviour of reads and writes to accesses to other memory locations.

ASSUMPTIONS

① a write does not complete until all processors have seen the effect of that write.

② the processor does not change the order of any write wrt. any other memory access.

THE BASIC SCHEMES OF ENFORCING COHERENCY

the coherence problem for multiprocessors and IO.
IO - multiple copies are a rare event.

a program running on multiple processors will normally have copies of the same data in several caches.

On a coherent multiprocessor, the cache provides both migration & replication of shared data items.

multiprocessors adopt a hardware solution by introducing a protocol to maintain coherent caches.

We have 2 classes of protocols.

① directory based (Sharing status of a block is kept in one location called directory)

② snooping.

(1) write invalidate protocol

(2) write update / write broadcast protocol.

Snooping → caches will moniton bus activity.

request	processor	State of the address Cache block.		
request	source. processor		type of Cacheact.	function expl.
① read hit	processor	Shared or modified.	normal hit	read data in local cache.
② read miss	processor	invalid	normal miss	place read miss on bus.
③ read miss	processor	shared.	replacement	address conflict miss. place read miss on bus.
④ read miss	processor	modified.	replacement	address conflict miss. write back blk., then place read miss on bus.
⑤ write hit	processor	modified	normal hit	write data in local cache.

⑥ write hit processor shared coherence place invalidate on bus. these operations are often called, upgrade or ownership misses. because they dont fetch the data, but only change.

⑦ write miss processor invalid normal miss place write miss on bus.

⑧ write miss processor shared replacement address conflict miss. place write miss on bus.

⑨ write miss processor modified replacement. address conflict miss write back block, then place write miss on bus.

① read miss bus shared no action allow shared cache or memory to service the read miss.

② read miss bus modified (problem) coherence attempt to read shared data, so place cache block on bus, write back block, and change state to shared.

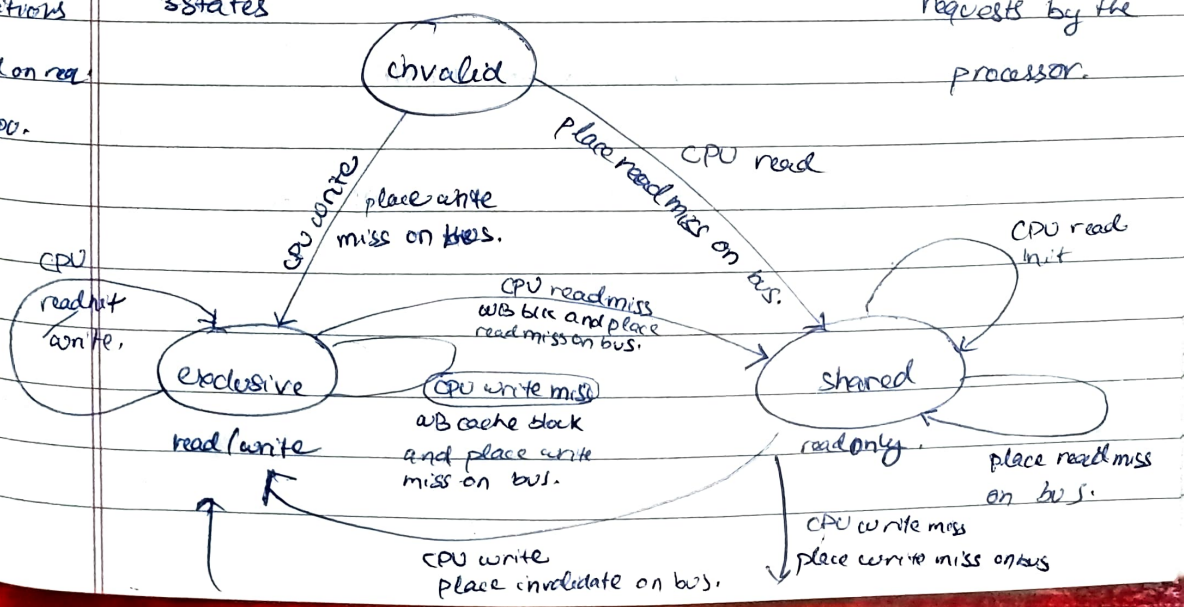
②	invalidate	bus	shared	coherence	attempt to write shared block invalidate the cache block.
④	write miss	bus	shared	coherence	attempt to write shared block, invalidate the cache block.
⑤	write miss	bus	modified	coherence	attempt to write block that is exclusive elsewhere, writeback the cache block and make its state invalid in the local cache.

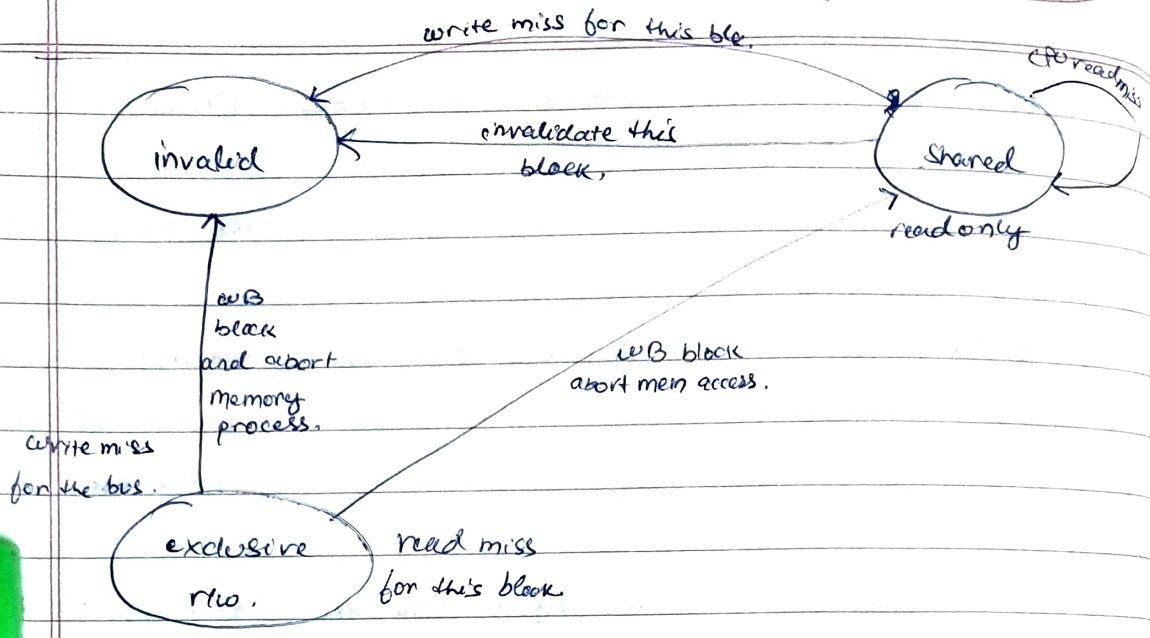
a write invalidate cache coherence protocol for a private writeback cache (not write through), showing the states and the state transition for each block in the cache. (2-4 words).

Cache state transitions based on req. from CPU.

3 states

requests by the processor.





Cache state transitions based on request from the bus.

the protocol assumes, operations are atomic. non-atomic ~~operations~~ actions introduce the possibility that the protocol can deadlock.

MSI - modified shared invalid.

↓ extension

① MESI - optimized certain behaviour and possibly improve performance.
exclusive

② intel i7 procession has got MESIF ← forward.

③ MOESI Owned. AMD opteron.

DIRECTORY BASED CACHE COHERENCE PROTOCOL.

two primary ops. , handling a read miss, and write to a shared clean cache block. (handling a write miss to a blk that is currently shared is a simple combination of these 2.)

a directory must track the state of each blk
- shared. one or more nodes have the block cached. and the value in memory is up to date., as well as in all caches.

- uncached : no node has a copy of the cache block.

- modified : exactly one node has a copy of the cache block and it has written the block., so the memory copy is out of data. this processor is called the owner of the block.

bit vectors -> indicates all processors which are shared.

Simclars FSM

local node is the node where a request originates.
home node -> the mem location & the directory entry of an address reside.

the physical address space is statically distributed. the local node may also be the home node.

a remote node is the node that has a copy of the cache block whether exclusive or shared.

P = requesting node number; A = requested address
D = data contents.

message type	source	dest.	message contents.	function of msg.
read miss	local cache	home dir.	P, A	node P has read miss at A request data and P a read shared.
write miss	local cache	home dir.	P, A	node P has a write miss at address A, request data and make P the exclusive owner.
invalidate	local cache	home dir.	A	request to send invalidate to all remote caches that are caching the block at address A.
invalidate	home dir.	remote cache.	A	invalidate the shared copy of data at address A.
fetch	home dir.	remote cache	A	fetch the blk at address A and send it to its home dirct; change the state of A in the remote cache shared.

7A -

fetch/ invalidate	home dir.	remote cache.	A	fetch the blk at address A and send it to its own directory, invalidate and ~~change~~ the ~~state of~~ A block in the cache.
data value reply	home dir.	local cache	D	return a data value from the home ~~directory~~ memory.
data WB.	remote cache	home dir.	A, D.	WB the data value for address A.