Symmetric Multiprocessors

Implementing a single memory image operated upon by multiple processors is possible at small scales. The SMP is a standard design in which cache coherency allows all processors to see the same image.

Homework

Write two procedures -- **`Put_Task()`** and **`Get_Task()`** -- in any language, e.g. C, and using Fetch&Add for synchronization as follows

- Define `TD[1..n]`, the ToDo array; `n=c*Processors` for some `c`
- Define `FF`, for first free, pointing in `TD` to first free cell
- Define `NA`, for next available, pointing to next task in `TD`
- `Put_Task(a)` takes a task as input, and places it in `TD`; `Get_Task()` returns the next task from `TD` if there is one
- The management of `TD` is completely decentralized

Homework

Strategy: The TD[1..n] is several times larger than the number of processors; system assumes at least 1 producer and 1 consumer

```
Put_Task(a) {
  slot = Fetch&Add(FF,1);
  if slot == n then Fetch&Add(FF, -n);
  if slot > n then slot = slot - n;
  while TD[slot] != 0 do wait(rand());
  TD[slot] = a;
}
```

Put waits if the slot is occupied to avoid overrun; Get waits if slot is empty
No task has is "0"

```
Get_Task { var temp;
  slot = Fetch&Add(NA,1);
  if slot == n then Fetch&Add(NA,-n);
  if slot > n then slot = slot - n;
  while TD[slot] == 0 do wait(rand());
  temp=TD[slot]; TD[slot]=0; return temp;
}
```

Shared Memory

Shared memory was claimed to be a poor model because it is does not scale

- Many vendors have sold small shared memory machines
  - Some like SMPs work well (but modeled poorly by PRAM)
  - Some never worked -- KSR
  - Some worked because of a technology opportunity -- slow processors with a "fast interconnect"
  - Some work on small scale, but not beyond 64 processors and everyone tries to ignore that fact -- Origin-2000
- Many researchers have come up with great ideas, but they still remain unproved

Citation

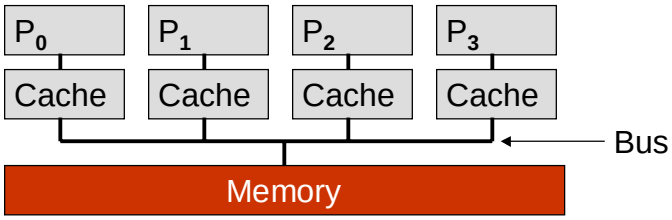
David Culler and J.P. Singh
Parallel Computer Architecture
Morgan Kaufmann, 1999

Share Memory Image

- Previous models of shared memory have literally implemented a single memory unit where all data resides
- Besides being a point of contention, a single memory doesn't permit caching (though "read-only" caching is OK)
- The SMP turns the idea around and exploits caching to implement a shared memory

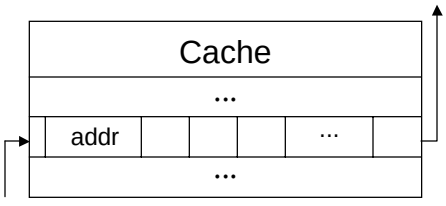
Architecture of an SMP

- A symmetric multiprocessor (SMP) is a set of processor/cache pairs connected to a bus
- The bus is both good news and bad news
  - The (memory) bus is a point at which all processors can "see" memory activity, and can know what is happening
  - A bus is used "serially," and becomes a "bottleneck," limiting scaling



Recall Caches

- Cache blocks (lines) contain several words
- Blocks have state
  - Valid
  - Invalid
  - Dirty = diff from mem
- Cache writing
  - Write through means update memory on all writes
  - Write back means wait and update when block is invalidated
  - "allocate" vs "no-allocate"



Cache Coherence -- The Problem

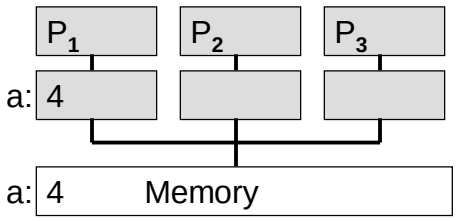
- Processors can modify shared locations without other processors being aware of it unless special hardware is added

P_1 reads a into its cache

P_3

P_1

P_3



Cache Coherence -- The Problem

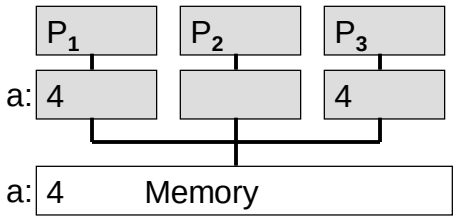
- Processors can modify shared locations without other processors being aware of it unless special hardware is added

P_1 reads a into its cache

P_3 reads a into its cache

P_1

P_3



Cache Coherence -- The Problem

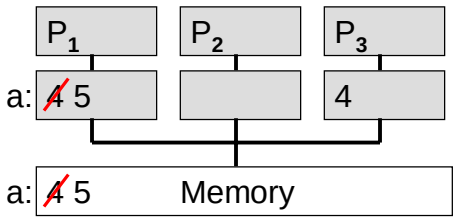
- Processors can modify shared locations without other processors being aware of it unless special hardware is added

P_1 reads a into its cache

P_3 reads a into its cache

P_1 changes a to 5 and writes the result through to main memory leaving P3 with stale data

P_3



Cache Coherence -- The Problem

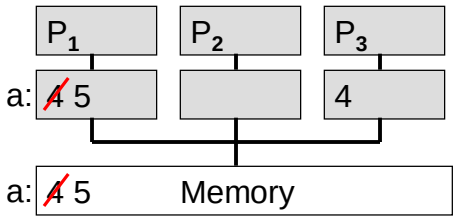
- Processors can modify shared locations without other processors being aware of it unless special hardware is added

P_1 reads a into its cache

P_3 reads a into its cache

P_1 changes a to 5 and writes the result through to main memory leaving P3 with stale data

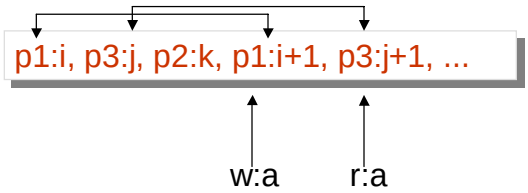
P_3 reads a … incoherent

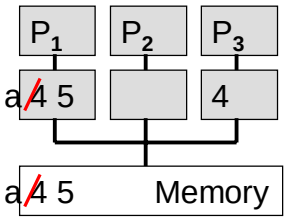


Cache Coherency -- The Goal

A multiprocessor memory system is *coherent* if for every location there exists a serial order for the operations on that location consistent with the results of the execution such that

- The subsequence of operations for any processor are in the order issued
- The value returned by each read is the value written by the last write in serial order





Write Serialization

Implied property of Cache Coherency:

Write Serialization … all writes to a location are seen in the same order by all processors

- For fulfilling “seen by all processors” a bus is a perfect solution

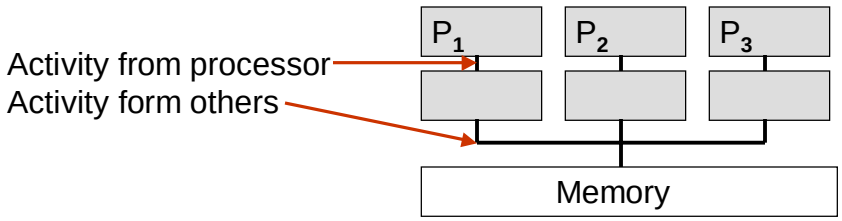
Snooping To Solve Coherency

- The cache controllers can “snoop” on the bus, meaning that they watch the events on the bus even if they do not issue them, noting any action relevant to cache lines they hold
- There are two possible actions when a location held by processor A is changed by processor B
  - Invalidate -- mark the local copy as invalid
  - Update -- make the same change B made

The unit of cache coherency is a cache line or block

Snooping

When the cache controller “snoops” it sees requests by its processor or bus activity by other processors that is not local to them



Snooping At Work I

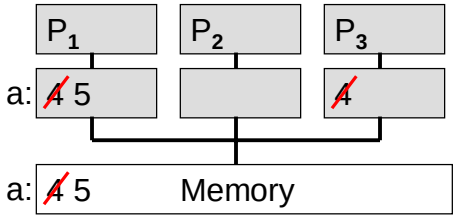
By snooping the cache controller for processor P3 can take action in response to P1's write

P_1 reads a into its cache

P_3 reads a into its cache

P_1 changes a to 5 and writes through to main memory; P_3 sees the action and invalidates the location

P_3



Snooping At Work II

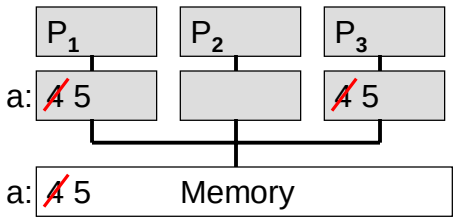
By snooping the cache controller for processor P3 can take action in response to P1's write

P_1 reads a into its cache

P_3 reads a into its cache

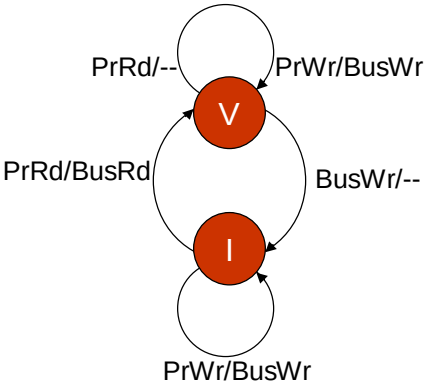
P_1 changes a to 5 and writes through to main memory; P_3 sees the action and invalidates the location or updates it

P_3



Write-through Coherency

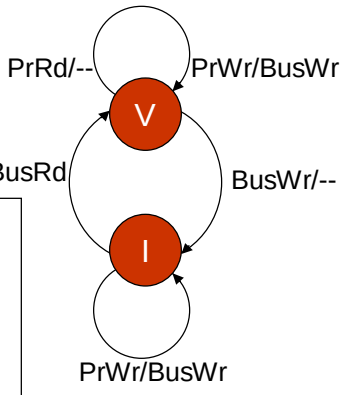
- State diagrams show the protocol



States of a cache line
  V is valid
  I is invalid
Transactions
  Reads (Rd) or Writes (Wr) by processor or bus
Labeling A/B
  If A is observed
    Then transaction B is generated

Applying the WT Protocol

- Consider the transactions



P_1 reads a into its cache

P_3 reads a into its cache

P_1 changes a to 5 and writes through to main memory

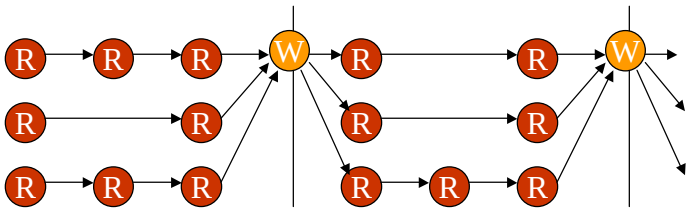
P_3 sees the action and invalidates the location

P_2 reads a into its cache

P_1 : I --> V
PrRd/BusRd
P_3 : I --> V
PrRd/BusRd
P_1 : V --> V
PrWr/BusWr
P_3 : V --> I
BusWr/--
P_2 : I --> V
PrRd/BusRd

Partial Order On Memory Operations

Write bus transactions define a global sequence of events; between writes processors can read … any total order produced by interleaving



Memory Consistency

- What should it mean for processors to see a consistent view of memory?
- Coherency is too weak because it only requires ordering with respect to individual locations, but there are other ways of binding values together

Coherency requires only that the 0 --> 1 transition of a be seen eventually by P1

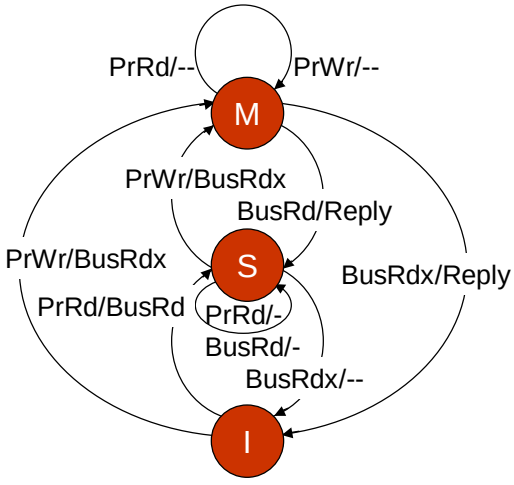
```
P0 : [a, flag initially 0]
a      := 1;
flag   := 1;
P1 :
while(flag != 1)do; -- spin
print (a);
```

Basic Write-back Snoopy Cache Design

- Write-back protocols are more complex than write-through because modified data remains in the cache
- Introduce more cache states to handle that
  - Modified, or dirty, the value differs from memory
  - Exclusive, no other cache has this location
- Consider an MSI protocol with three states:
  - Modified -- data is correct locally, different from memory
  - Shared (Valid) -- data at this location is correct
  - Invalid -- data at this location not correct

MSI Protocol

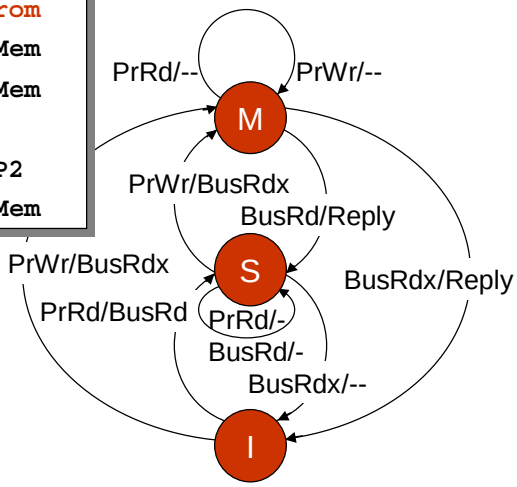
- Rdx means that the cache holds a modified value of the location and asks for exclusive permission to read
- Reply means put the value on the bus for another processor to read



Conceptually: Manage dirty value within caches

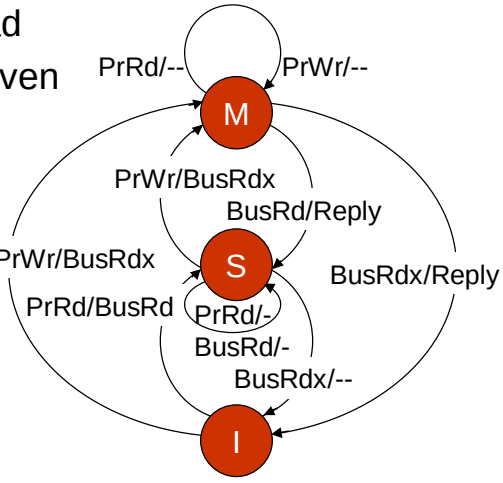
MSI Protocol In Action

Proc					Data
Action	P0	P1	P2	Bus	From
P0:r a	S	-	-	BRd	Mem
P2:r a	S	-	S	BRd	Mem
P2:w a	I	-	M	BRdx	
P0:r a	S	-	S	BRd	P2
P1:r a	S	S	S	BRd	Mem



Critique of MSI

Bad: 2 bus ops to load and update a value even without any sharing



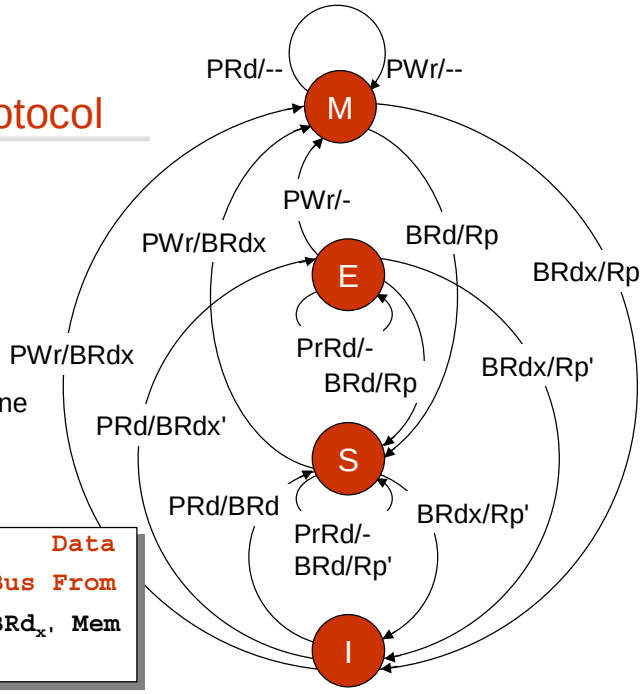
Proc				Data
Action	P0	Pi	Bus	From
P0:r a	S	-	BRd	Mem
P0:w a	M	-	BRdx	

- Add an Exclusive State, opposite of Shared

Break

Illinois Protocol

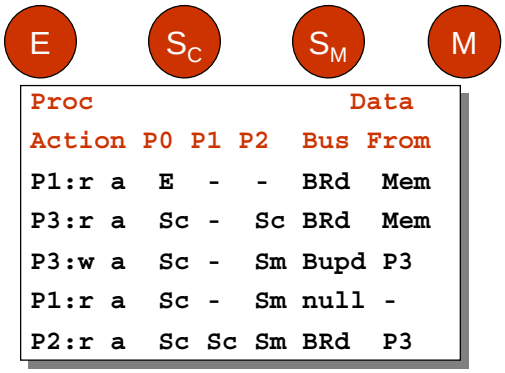
P=processor
B=Bus
Rd=Read
Rdx=Read Ex
Rd_S=Read Sh
$Rd_{S'}$=Read Ex
Rp=Reply
Rp=Reply Someone



Proc				Data
Action	P0	Pi	Bus	From
P0:r a	E	-	BRd_x,	Mem
P0:w a	M	-		

Alternative … Updating

- One caching issue is “invalidation” vs “update”: Dragon



Proc					Data
Action	P0	P1	P2	Bus	From
P1:r a	E	-	-	BRd	Mem
P3:r a	Sc	-	Sc	BRd	Mem
P3:w a	Sc	-	Sm	Bupd	P3
P1:r a	Sc	-	Sm	null	-
P2:r a	Sc	Sc	Sm	BRd	P3

Invalidation vs Update

1 Repeat k times: P1 writes V, P2-Pp read V

… perhaps representing work allocation

2 Repeat k times: P1 writes V M times, P2 reads

… perhaps representing sharing pair

invl = 6B, update = 14B, miss = 70B

P = 16, M = 10, k = 10

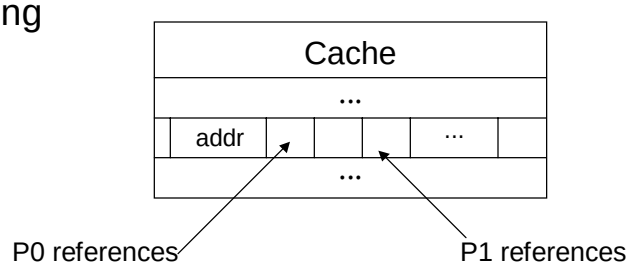
U1: 1,260B	I1:10,624B
U2: 1,400B	I2: 824B

Implications of Blocksize

- Larger blocks exploit spatial locality better
- Bus transactions take more time with larger blocks
- Fewer large blocks for a given amount of memory or more small blocks
- There are implications on sharing

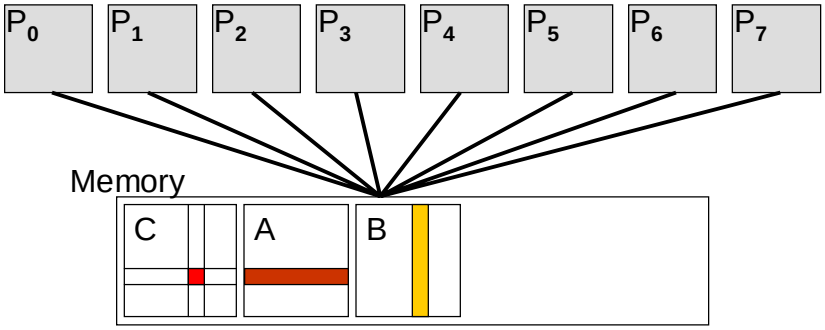
True/False Sharing

- If two processors reference the same cache line and the same word, they are “truly” sharing
- If two processors reference the same cache line but a different word, they are “falsely” sharing



Discussion

- What is the best model of an SMP … PRAM?



Summary

- SMPs solve shared memory by snooping
- Key to SMP's success is the bus, a site for serializing memory references
- Buses work, but only for a small number (64 is upper limit, but fewer is better) of processors
- Relative to the two requirements of shared memory -- acceptable costs, coherency -- the SMP meets both