
EEC 581 Computer Architecture

Lec 9 – Multiprocessors (4.1 & 4.2)

Chansu Yu
Electrical and Computer Engineering
Cleveland State University

Acknowledgement ...

- **Part of class notes are from**
 - David Patterson
 - Electrical Engineering and Computer Sciences
 - University of California, Berkeley
 - <http://www.eecs.berkeley.edu/~pattsrn>
 - <http://www-inst.eecs.berkeley.edu/~cs252>
- **And, from**
 - Mary Jane Irwin
 - Computer Science and Engineering
 - Pennsylvania State University
 - <http://www.cse.psu.edu/~mji>
 - www.cse.psu.edu/~cg431

Outline

- Cray 1
- MP Motivation
- SISD v. SIMD v. MIMD
- Centralized vs. Distributed Memory
- Challenges to Parallel Programming
- Consistency, Coherency, Write Serialization
- Write Invalidate Protocol
- Example
- Conclusion

2007-11-06

3

“The CRAY-1 computer system”

- by R.M. Russell, *Comm. of the ACM*, January 1978
- Number of functional units: 12 (today's: 6~8)
- Clock rate: 80MHz (possible due to its small size)
- Memory size: 8 MB
- Memory latency: 50ns (faster than today's PC)
- Cooling (4x per cubic inch of CDC 7600)
- Instruction set architecture: load/store, 3 address, 24-bit memory address
- Virtual Memory: No

2007-11-06

4

“The CRAY-1 computer system”

- **138 MFLOPS, 250 MFLOPS peak (Pentium 4: 4 GFLOPS)**
- **The CRAY-1 weighs 10,500 lbs (10⁶ more than today)**
- **The CRAY-1 consumes 115kW (10³ of those today), and has a Freon cooling system**
 - Power was limiting factor than as it is today
- **How many Cray-1 were shipped?**
 - 80 Cray-1s of all types were sold, priced from \$5M to \$8M

2007-11-06

5

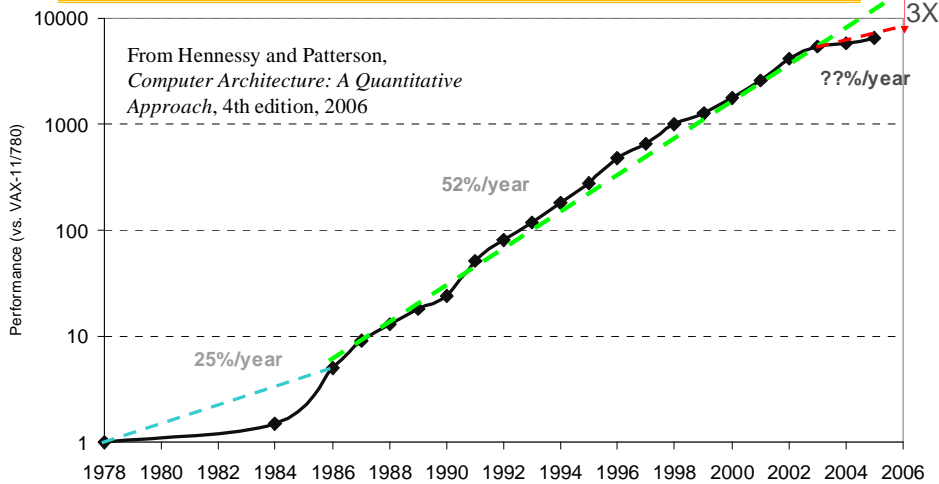
Outline

- Cray 1
- MP Motivation
- SISD v. SIMD v. MIMD
- Centralized vs. Distributed Memory
- Challenges to Parallel Programming
- Consistency, Coherency, Write Serialization
- Write Invalidate Protocol
- Example
- Conclusion

2007-11-06

6

Uniprocessor Performance (SPECint)



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

7

Déjà vu all over again?

“... today’s processors ... are nearing an impasse as technologies approach the speed of light..”

David Mitchell, *The Transputer: The Time Is Now* (1989)

- Transputer had bad timing (Uniprocessor performance↑)
⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- “We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing”
Paul Otellini, President, Intel (2005)
- All microprocessor companies switch to MP (2X CPUs / 2 yrs)
⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

Manufacturer/Year	AMD/'05	Intel/'06	IBM/'04	Sun/'05
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	4
Threads/chip	2	4	4	32

Applications Needing “Supercomputing”

- **Energy** (plasma physics (simulating fusion reactions), geophysical (petroleum) exploration)
- **DoE stockpile stewardship** (to ensure the safety and reliability of the nation’s stockpile of nuclear weapons)
- **Earth and climate** (climate and weather prediction, earthquake, tsunami prediction and mitigation of risks)
- **Transportation** (improving vehicles’ airflow dynamics, fuel consumption, crashworthiness, noise reduction)
- **Bioinformatics and computational biology** (genomics, protein folding, designer drugs)
- **Societal health and safety** (pollution reduction, disaster planning, terrorist action detection)

Other Factors ⇒ Multiprocessors

- **Growth in data-intensive applications**
 - Data bases, file servers, ...
- **Growing interest in servers, server perf.**
- **Increasing desktop perf. less important**
 - Outside of graphics
- **Improved understanding in how to use multiprocessors effectively**
 - Especially server where significant natural TLP
- **Advantage of leveraging design investment by replication**
 - Rather than unique design

2007-11-06

10

Flynn's Taxonomy

M.J. Flynn, "Very High-Speed Computers",
Proc. of the IEEE, V 54, 1900-1909, Dec. 1966.

- Flynn classified by data and control streams in 1966

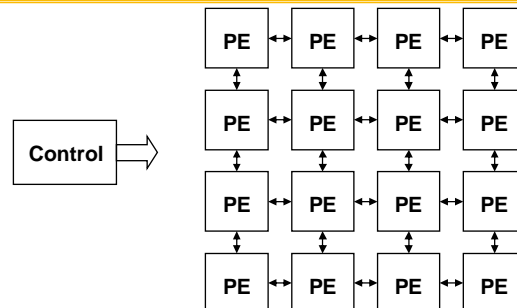
Single Instruction Single Data (SISD) (Uniprocessor)	Single Instruction Multiple Data <u>SIMD</u> (single PC: Vector, CM-2)
Multiple Instruction Single Data (MISD) (????)	Multiple Instruction Multiple Data <u>MIMD</u> (Clusters, SMP servers)

- SIMD ⇒ Data Level Parallelism
- MIMD ⇒ Thread Level Parallelism
- MIMD popular because
 - Flexible: N pgms and 1 multithreaded pgm
 - Cost-effective: same MPU in desktop & MIMD

2007-11-06

11

SIMD Processors



- Single control unit
- Multiple datapaths (processing elements – PEs) running in parallel
 - PEs are interconnected (usually via a mesh or torus) and exchange/share data as directed by the control unit
 - Each PE performs the same operation on its own local data

2007-11-06

12

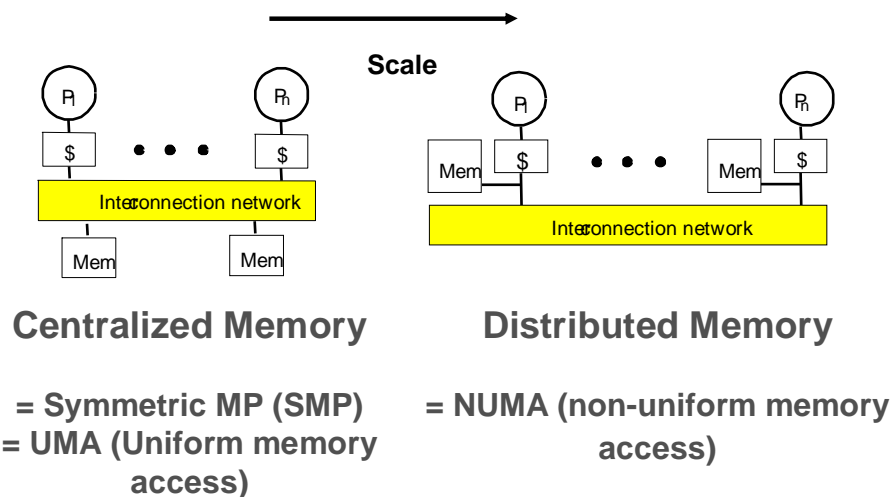
Example SIMD Machines

	Maker	Year	# PEs	# b/ PE	Max memory (MB)	PE clock (MHz)	System BW (MB/s)
Illiac IV	UIUC	1972	64	64	1	13	2,560
DAP	ICL	1980	4,096	1	2	5	2,560
MPP	Goodyear	1982	16,384	1	2	10	20,480
CM-2	Thinking Machines	1987	65,536	1	512	7	16,384
MP-1216	MasPar	1989	16,384	4	1024	25	23,000

2007-11-06

13

Centralized vs. Distributed Memory



2007-11-06

14

2 Models for Communication and Memory Architecture

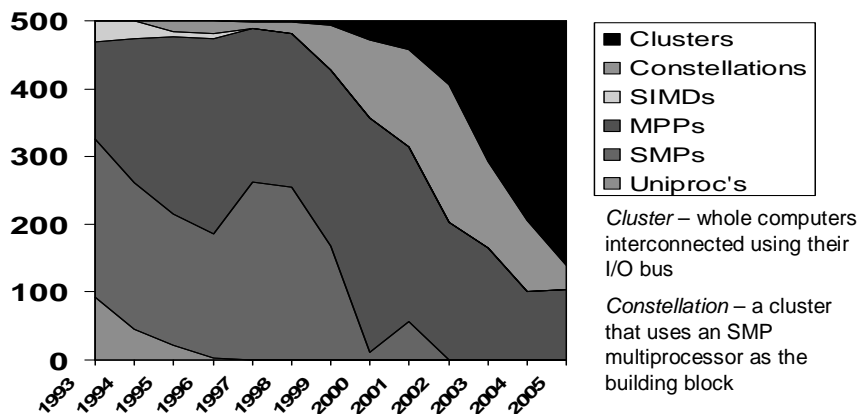
1. **Communication occurs by explicitly passing messages among the processors:**
message-passing multiprocessors
2. **Communication occurs through a shared address space (via loads and stores):**
shared memory multiprocessors either
 - **UMA (Uniform Memory Access time) for shared address, centralized memory MP**
 - **NUMA (Non Uniform Memory Access time multiprocessor) for shared address, distributed memory MP**

2007-11-06

15

Supercomputer Style Migration (Top500)

<http://www.top500.org/lists/2005/11/>



- **In the last 8 years uniprocessor and SIMDs disappeared while Clusters and Constellations grew from 3% to 80%**

16

Challenges of Parallel Processing

- First challenge is % of program inherently sequential
- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?
 - a. 10%
 - b. 5%
 - c. 1%
 - d. <1%

2007-11-06

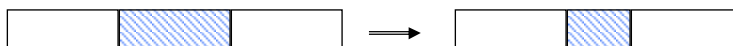
17

Encountering Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F < 1) of the task by a factor S (S > 1) and the remainder of the task is unaffected



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times$$

$$\text{Speedup w/ E} =$$

2007-11-06

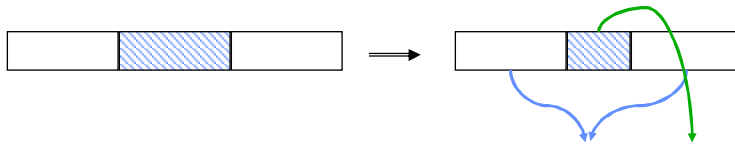
18

Encountering Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F < 1) of the task by a factor S (S > 1) and the remainder of the task is unaffected



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times ((1-F) + F/S)$$

$$\text{Speedup w/ E} = 1 / ((1-F) + F/S)$$

2007-11-06

19

Challenges of Parallel Processing

- First challenge is % of program inherently sequential
- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?

a. 10%

b. 5%

c. 1%

d. <1%

$$\text{Speedup w/ E} = 1 / ((1-F) + F/S)$$

$$80 = 1 / ((1-F) + F/100)$$

$$F = 99.75\%$$

$$\text{or, sequential} = 0.25\%$$

2007-11-06

20

Challenges of Parallel Processing

- **Second challenge is long latency to remote memory**
- **Suppose 32 CPU MP, 2GHz, 200 ns remote memory, all local accesses hit memory hierarchy and base CPI is 0.5. (Remote access = $200/0.5 = 400$ clock cycles.)**
- **What is performance impact if 0.2% instructions involve remote access?**
 - a. **1.5X**
 - b. **2.0X**
 - c. **2.5X**

2007-11-06

21

CPI Equation

- **CPI = Base CPI +
Remote request rate
x Remote request cost**
- **CPI = $0.5 + 0.2\% \times 400 = 0.5 + 0.8 = 1.3$**
- **No communication is $1.3/0.5$ or 2.6 faster than 0.2% instructions involve local access**

2007-11-06

22

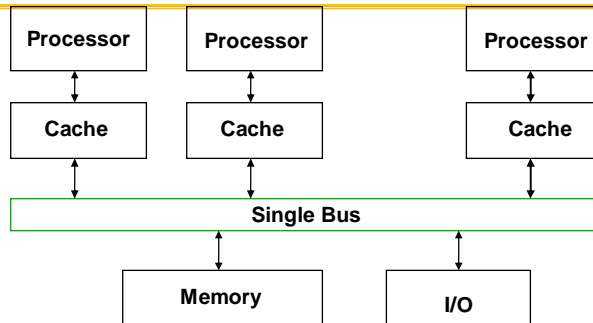
N/UMA Remote Memory Access Times (RMAT)

	Year	Type	Max Proc	Interconnection Network	RMAT (ns)
Sun Starfire	1996	SMP	64	Address buses, data switch	500
Cray 3TE	1996	NUMA	2048	2-way 3D torus	300
HP V	1998	SMP	32	8 x 8 crossbar	1000
SGI Origin 3000	1999	NUMA	512	Fat tree	500
Compaq AlphaServer GS	1999	SMP	32	Switched bus	400
Sun V880	2002	SMP	8	Switched bus	240
HP Superdome 9000	2003	SMP	64	Switched bus	275
NASA Columbia	2004	NUMA	10240	Fat tree	???

2007-11-06

23

Single Bus (Shared Address UMA) MP

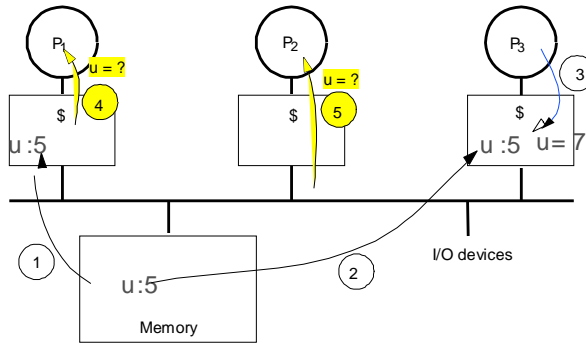


- Caches are used to reduce latency *and* to lower bus traffic
- Must provide hardware to ensure that caches and memory are consistent (cache coherency)
- Must provide a hardware mechanism to support process synchronization (consistency)

2007-11-06

24

Example Cache Coherence Problem



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
 - » Processes accessing main memory may see very stale value
- Unacceptable for programming, and its frequent!

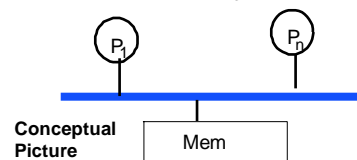
2007-11-06

25

Example Memory Consistency Problem

P ₁	P ₂
/*Assume initial value of A and flag is 0*/	
A = 1;	while (flag == 0); /*spin idly*/
flag = 1;	print A;

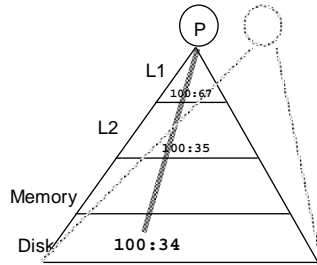
- Intuition not guaranteed by coherence
- expect memory to respect order between accesses to *different* locations issued by a given process
 - to preserve orders among accesses to same location by different processes
- Coherence is not enough!
 - pertains only to single location



2007-11-06

26

Intuitive Memory Model



- **Reading an address should return the last value written to that address**

- Easy in uniprocessors, except for I/O

- **Too vague and simplistic; 2 issues**
 1. **Coherence** defines values returned by a read
 2. **Consistency** determines when a written value will be returned by a read
- **Coherence** defines behavior to same location, **Consistency** defines behavior to other locations

2007-11-06

27

Defining Coherent Memory System

1. **Preserve Program Order:**
P writes A to location X => P reads X: returns A
(with no writes of X by another processor occurring between the write and the read by P)
2. **Coherent view of memory:**
Q writes A to location X => P reads X:
returns A
(if the read and write are sufficiently separated in time and no other writes to X occur between the two)
3. **Write serialization:**
P writes A to X, Q writes B to X => all other processors see the same order of writes

2007-11-06

28

Write Consistency

- For now assume
 1. A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
 2. The processor does not change the order of any write with respect to any other memory access
- ⇒ if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A
- These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order

* See Section 4.6

2007-11-06

29

2 Classes of Cache Coherence Protocols

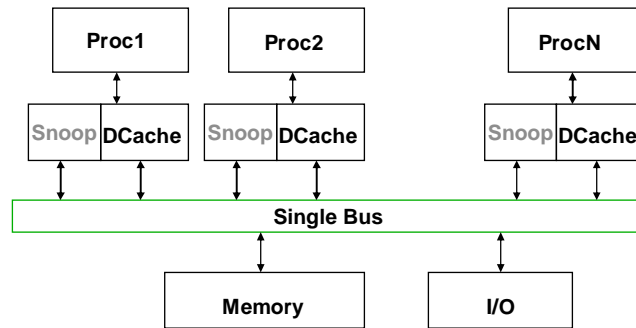
1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
 - All caches are accessible via some broadcast medium (a bus or switch)
 - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

2007-11-06

30

Multiprocessor Cache Coherency

- Cache coherency protocols
 - Bus snooping – cache controllers monitor shared bus traffic with duplicate address tag hardware (so they don't interfere with processor's access to the cache)



2007-11-06

31

Bus Snooping Protocols

- Multiple copies are not a problem when reading
- Processor must have **exclusive** access to write a word
- All other processors sharing that data must be informed of writes

2007-11-06

32

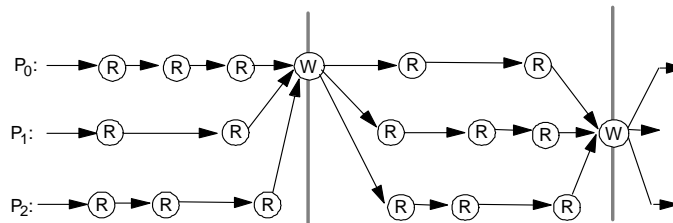
Architectural Building Blocks

- **Cache block state transition diagram**
 - FSM specifying how disposition of block changes
 - » invalid, valid, dirty
- **Broadcast Medium Transactions (e.g., bus)**
 - Fundamental system design abstraction
 - Logically single set of wires connect several devices
 - Protocol: arbitration, command/addr, data
 - ⇒ Every device observes every transaction
- **Broadcast medium enforces serialization of read or write accesses ⇒ Write serialization**
 - 1st processor to get medium invalidates others copies
 - Implies cannot complete write until it obtains bus
 - All coherence schemes require serializing accesses to same cache block
- **Also need to find up-to-date copy of cache block**

2007-11-06

35

Ordering



- **Writes establish a partial order**
- **Doesn't constrain ordering of reads, though shared-medium (bus) will order read misses too**
 - any order among reads between writes is fine, as long as in program order

2007-11-06

36

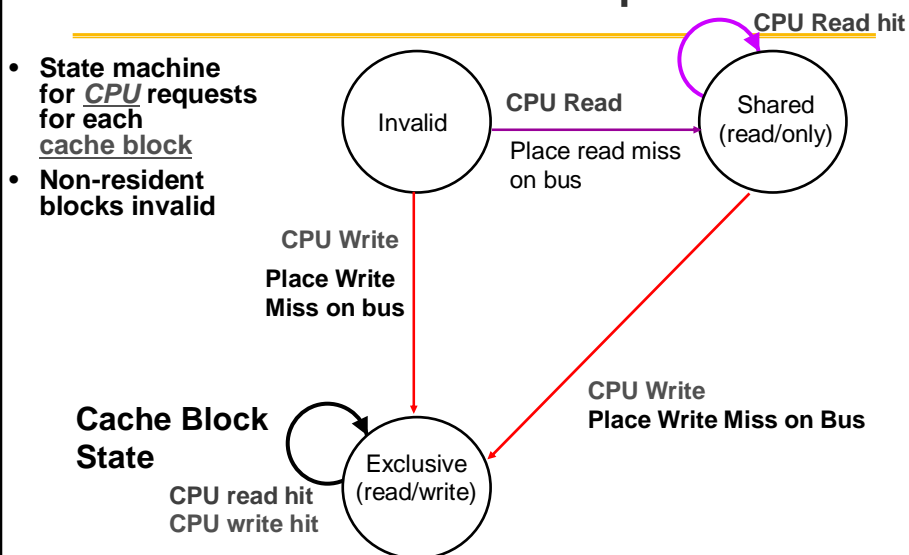
Example Write-invalidate Snoopy Protocol

- **Invalidation protocol, write-back cache**
 - Snoops every address on bus
 - If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
- **Each memory block is in one state:**
 - Clean in all caches and up-to-date in memory (Shared)
 - OR Dirty in exactly one cache (Exclusive)
 - OR Not in any caches
- **Each cache block is in one state (track these):**
 - Shared : block can be read
 - OR Exclusive : cache has only copy, its writeable, and dirty
 - OR Invalid : block contains no data (in uniprocessor cache too)
- **Read misses: cause all caches to snoop bus**
- **Writes to clean blocks are treated as misses**

2007-11-06

37

State Machine – CPU Requests



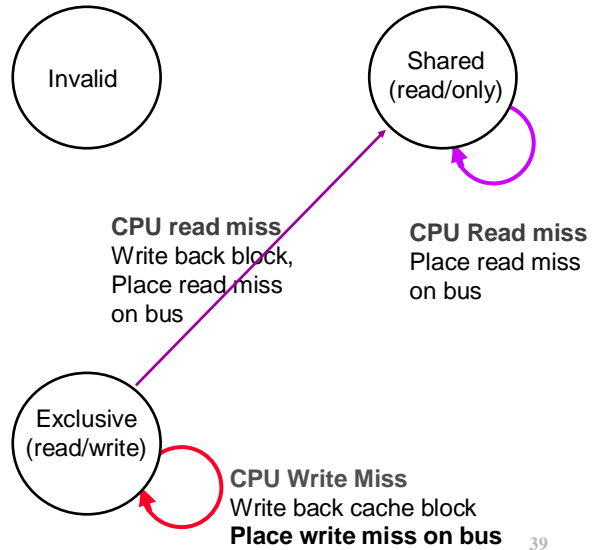
2007-11-06

38

State Machine - Block-replacement

- State machine for CPU requests for each cache block

Cache Block State



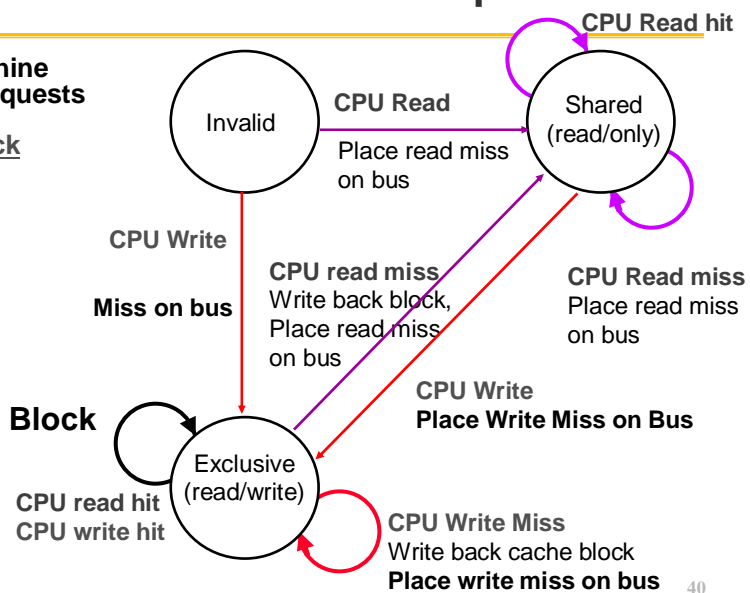
2007-11-06

39

State Machine – All CPU Requests

- State machine for CPU requests for each cache block

Cache Block State

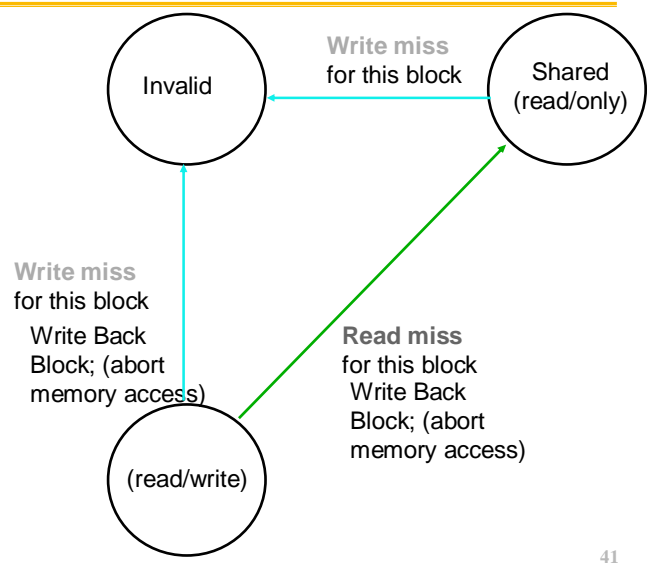


2007-11-06

40

State Machine - Bus Request

- State machine for cache block

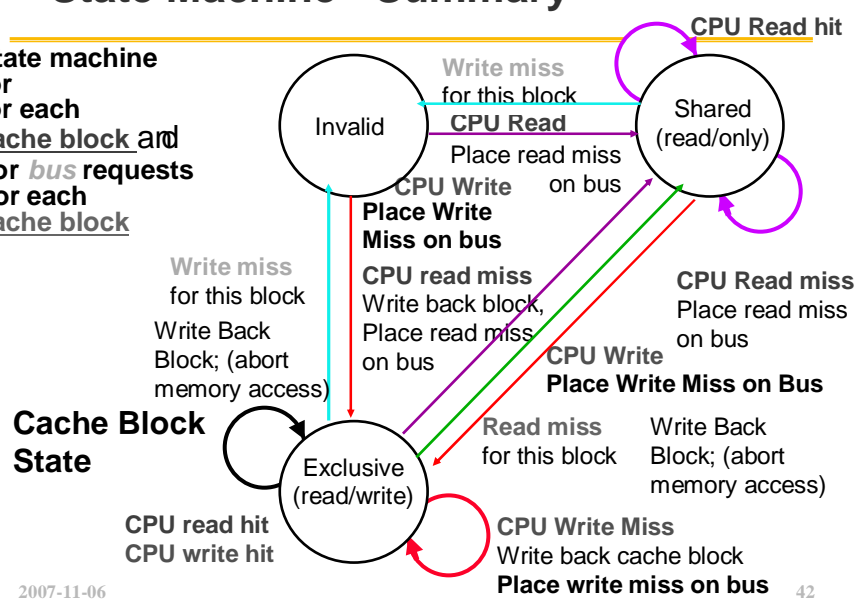


2007-11-06

41

State Machine - Summary

- State machine for each cache block and for *bus* requests for each cache block



2007-11-06

42

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block,
initial cache state is invalid

2007-11-06

43

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

2007-11-06

44

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

2007-11-06

45

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

2007-11-06

46

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

2007-11-06

47

Example

step	P1			P2			Bus			Memory		
	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20	A1	20

Assumes A1 and A2 map to same cache block,
but A1 != A2

2007-11-06

48

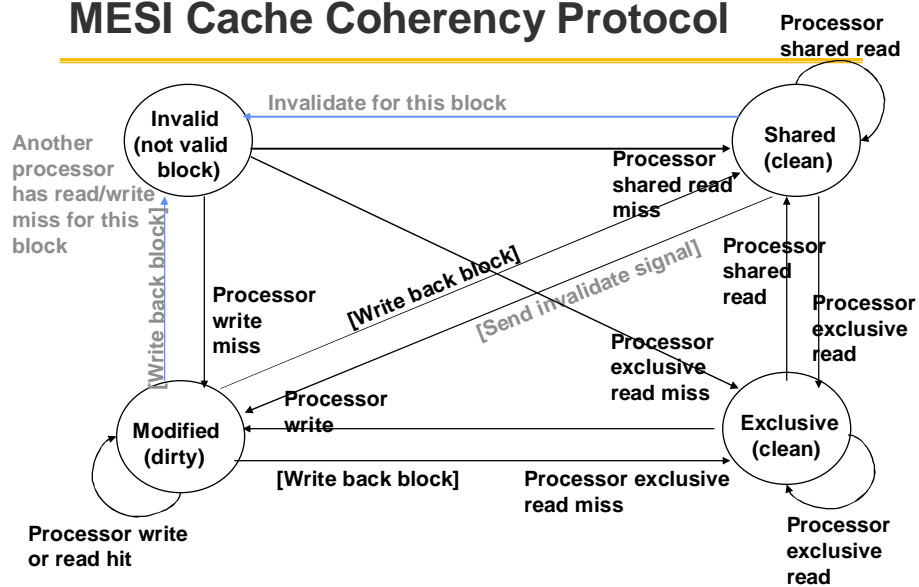
Other Coherence Protocols

- There are many variations on cache coherence protocols
- Another write-invalidate protocol used in the Pentium 4 (and many other micro's) is MESI with four states:
 - Modified – same
 - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
 - » Since there is only one copy of the block, write hits don't need to send invalidate signal
 - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
 - Invalid – same

2007-11-06

49

MESI Cache Coherency Protocol



2007-11-06

50

Conclusions

- **“End” of uniprocessors speedup => Multiprocessors**
- **Parallelism challenges: % parallelizable, long latency to remote memory**
- **Centralized vs. distributed memory**
 - Small MP vs. lower latency, larger BW for Larger MP
- **Message Passing vs. Shared Address**
 - Uniform access time vs. Non-uniform access time
- **Snooping cache over shared medium for smaller MP by invalidating other cached copies on write**
- **Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)**
- **Shared medium serializes writes**
 - ⇒ Write consistency

2007-11-06

51