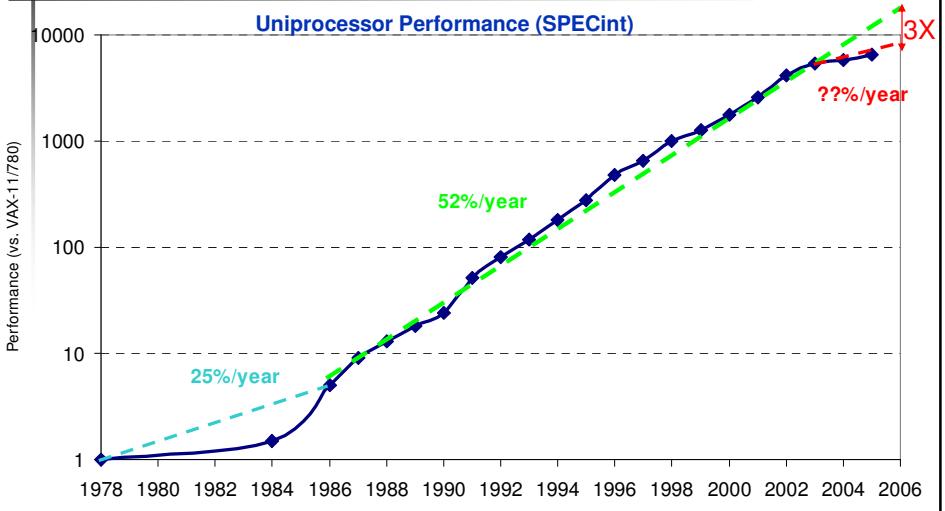
Motivation for Multiprocessors



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Other Factors ⇒ Multiprocessors

- **Growth in data-intensive applications**
  - **Data bases, file servers, …**
- **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**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Flynn's Taxonomy

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

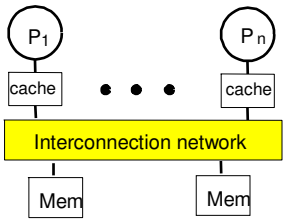
- Flynn classified by data and control streams in 1966

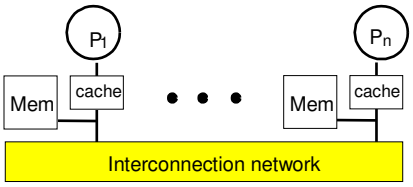
Single Instruction Single Data (SISD) (Classic Uniprocessor)	Single Instruction Multiple Data SIMD (Vector processors, GPUs)
Multiple Instruction Single Data (MISD) (????)	Multiple Instruction Multiple Data MIMD (Dist processors, clusters)

- SIMD ⇒ Data Level Parallelism
- MIMD ⇒ Thread Level Parallelism

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Centralized vs. Distributed Memory





Centralized Memory

- Dozens of processors, maybe hundreds of cores
- Small enough to share a centralized memory
- Memory bandwidth limits scale

Distributed Memory

- Thousands of processors
- Memory distributed among processors

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Centralized Memory Multiprocessor

- Also called symmetric multiprocessors (SMPs)
- Large caches help single memory satisfy memory demands of small number of processors
- Can scale to a few dozen processors by using a switch and by using many memory banks
- Although scaling beyond that is technically possible, it becomes less attractive as the number of processors sharing centralized memory increases

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Distributed Memory Multiprocessor

- Pro: Cost-effective way to scale memory bandwidth
  - If most accesses are to local memory
- Pro: Reduces latency of local memory accesses
- Con: Communicating data between processors more complex
- Con: Must change software to take advantage of increased memory BW

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

- Note that either Centralized or Distributed Memory MPs can use either model, but generally:
  - Shared memory ← → Centralized
  - Message-passing ← → Distributed

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

2 Challenges of Parallel Processing

(1) the speedup we can achieve through parallel processing is highly dependent on the fraction of the program that is "inherently sequential"

- Suppose we want to get 80X speedup from 100 processors. What fraction of the original program can be sequential?

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Amdahl's Law Answers

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{\text{Speedup}_{\text{parallel}}}}$$

$$80 = \frac{1}{\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times \left(\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}\right) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

2 Challenges of Parallel Processing

(2) long latency to remote memory can ruin performance gains

- **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?**
  - **CPI = Base CPI +**
    Remote request rate
    x Remote request cost
  - **CPI = 0.5 + 0.2% x 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**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Challenges of Parallel Processing: How do we resolve them?

1. Application parallelism ⇒ primarily via new algorithms that have better parallel performance
2. Long remote latency impact ⇒ both by architect and by the programmer
   - For example, reduce frequency of remote accesses either by
     - » Caching shared data (HW)
     - » Restructuring the data layout to make more accesses local (SW)

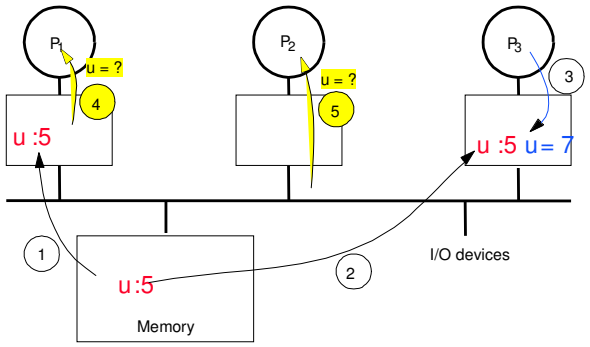
Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Centralized Shared-Memory Architectures

- Caches both
  - Private data -- used by a single processor
  - Shared data -- used by multiple processors
- Caching shared data
  ⇒ reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
  ⇒ cache coherence problem

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

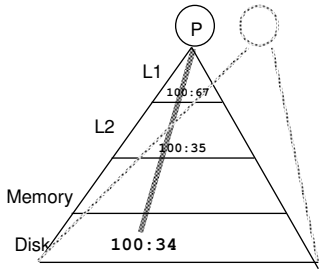
Example Cache Coherence Problem



- 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!

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Defining Coherent Memory System

1. Preserve Program Order: A read by 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. Coherent view of memory: 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 and no other writes to X occur between the two accesses
3. Write serialization: 2 writes to same location by any 2 processors are seen in the same order by all processors
   - For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Basic Schemes for Enforcing Coherence

- Program on multiple processors will normally have copies of the same data in several caches
- Rather than trying to avoid sharing in SW, SMPs use a HW protocol to maintain coherent caches
  - Migration and Replication key to performance of shared data
- Migration - data can be moved to a local cache and used there in a transparent fashion
  - Reduces both latency to access shared data that is allocated remotely and bandwidth demand on the shared memory
- Replication – for shared data being simultaneously read, since caches make a copy of data in local cache
  - Reduces both latency of access and contention for read shared data

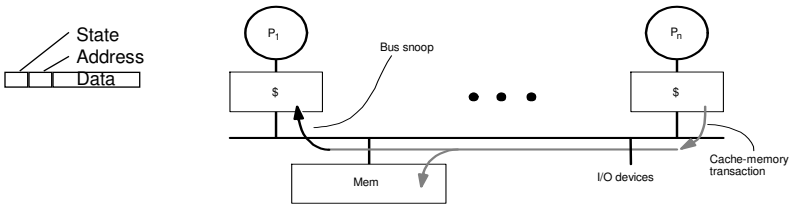
Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

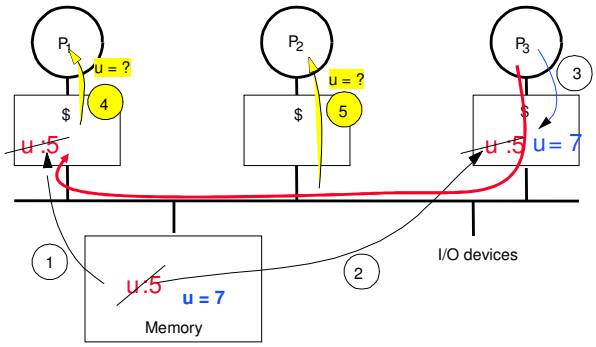
Snoopy Cache-Coherence Protocols



- **Cache Controller "snoops" all transactions on the shared medium (bus or switch)**
  - **relevant transaction if for a block it contains**
  - **take action to ensure coherence**
    - » **invalidate, update, or supply value**
  - **depends on state of the block and the protocol**
- **Either get exclusive access before write via write invalidate or update all copies on write**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Example: Write-thru Invalidate



- **Must invalidate before step 3**
- **Write update uses more broadcast medium BW ⇒ all recent MPUs use write invalidate**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)