Parallel Computer Architecture
Spring 2019

Shared Memory Multiprocessors
Memory Coherence

Nikos Bellas

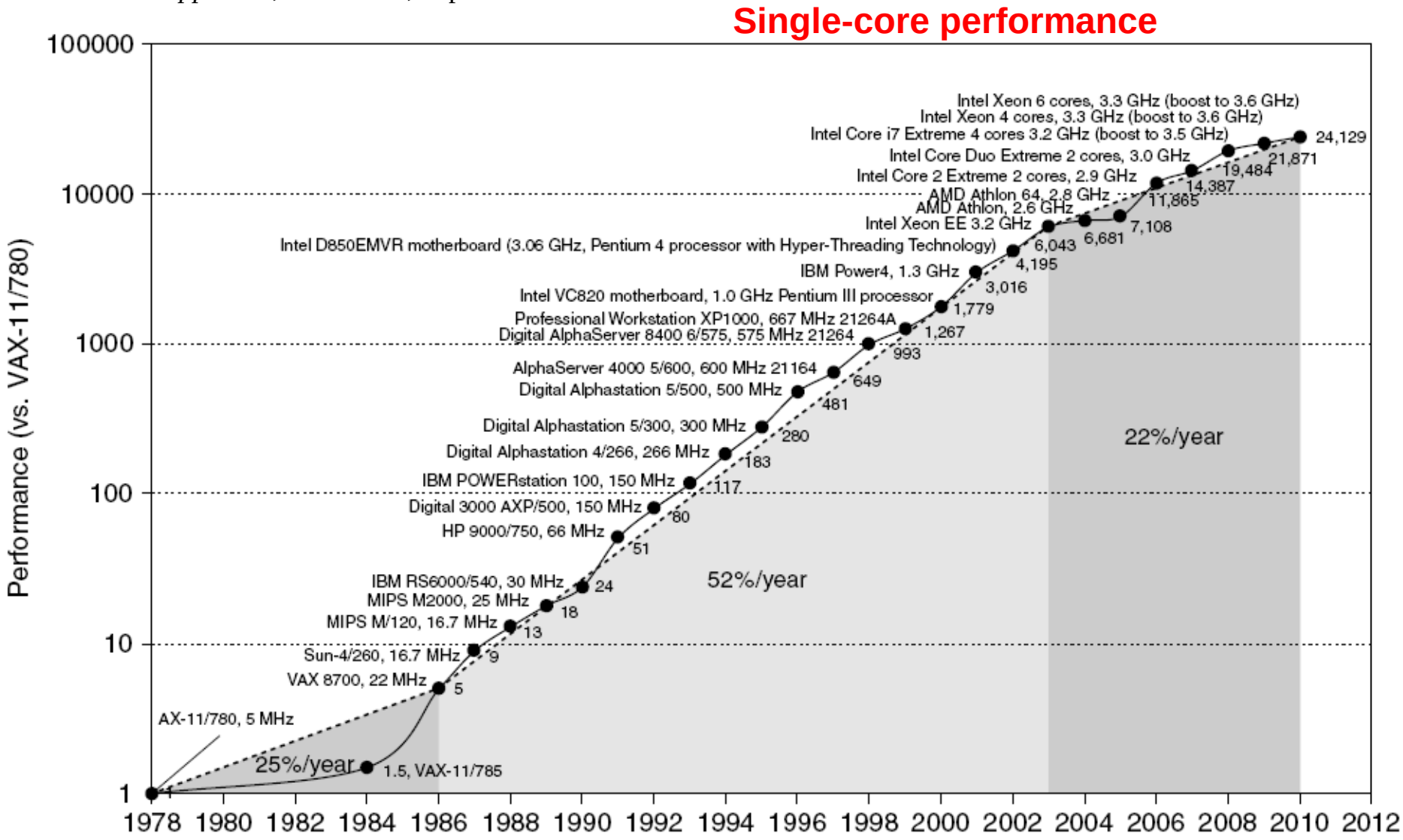
Computer and Communications Engineering Department
University of Thessaly

Introduction to Multiprocessing

Multicore Era

General-purpose unicores have stopped historic performance scaling

From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 5th edition, September 2011



History

"… 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

Chip Name (Year)	Intel Xeon E-2186G (Coffee Lake) (2018)	AMD Ryzen 1950X (2017)	IBM Power9 SMT4 (2017)	ARMv8 Xgene-3 (2018)
Cores/Chip	6	16	24	32
Threads/Core	2	2	4	1
Threads/Chip	12	32	96	32
Max Freq (GHz)	4.7	4.1	4	3

Factors for the Multiprocessor adoption

- Slowdown in uniprocessor performance due to limited ILP
- Growth in multithreaded applications
  - Data bases, file servers, …
- Growing interest in servers, server performance
- Increasing desktop performance less important
  - Except for 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
  - Exploit the huge cost-performance advantages of commodity processors

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 SIMD (single PC: Vector, CM-2, GPUs)
Multiple Instruction, Single Data (MISD) (????)	Multiple Instruction, Multiple Data MIMD (Clusters, SMP servers)

- SIMD ⇒ Data-Level Parallelism
- MIMD ⇒ Thread-Level Parallelism
- MIMD popular because
  - Flexible: N programs or 1 multithreaded program
  - Cost-effective: same MPU in desktop & MIMD machine

What is a "Parallel Computer"

- A collection of processing elements (PEs) than can communicate and co-operate to solve large problems fast
- A collection of PEs
  - How many? How powerful?
  - Fewer and more powerful? More and weaker?
  - Compare Chip Multiprocessors (i7, Power) and GPUs
- that can communicate
  - How do they communicate
  - Message Passing vs Shared Memory
  - Interconnect architecture (Bus vs Point-to-point interconnects
- and co-operate
  - Synchronization needed to organize computation

What is a “Parallel Computer”

- Granularity of parallelism
  - Grain size:

    Process : > 1e6 instructions

    Tasks/Functions : 1e3 – 1e6 instructions

    Loop iterations: 10 -1e3 instructions

    ILP : few tens instructions

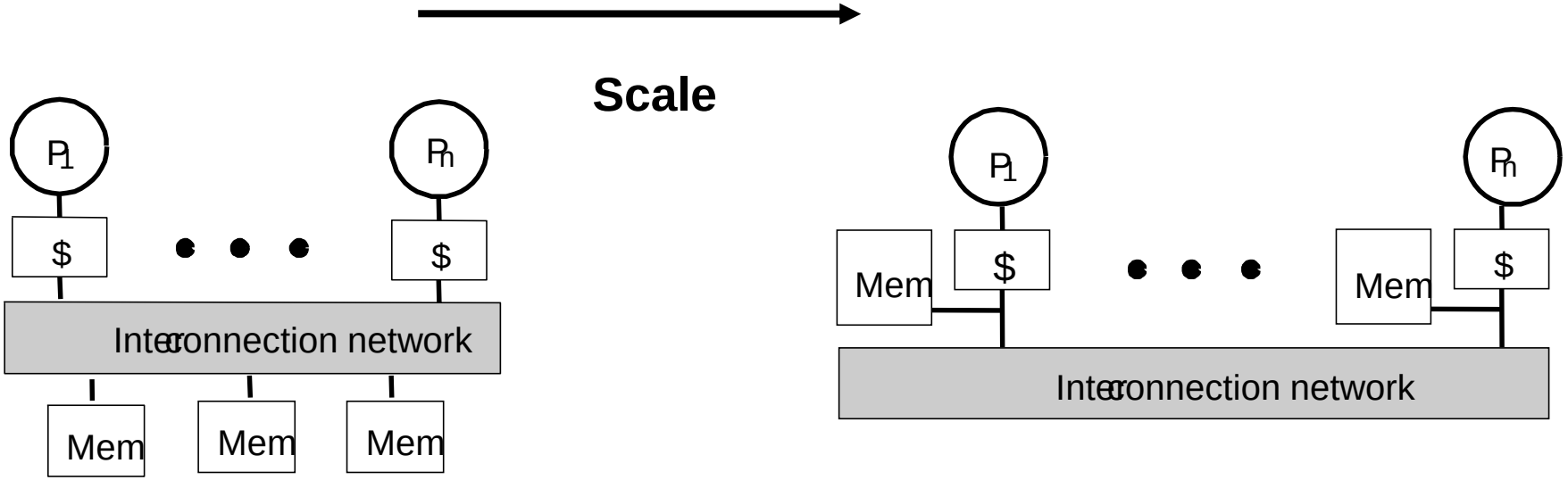
Two Models for Communication and Memory Architecture

1. Communication occurs by explicitly passing messages among the processors: message-passing multiprocessors (aka multicomputers)
   - Modern *cluster* systems contain multiple stand-alone computers communicating via messages
   - Streaming communication can also be classified as a fine-granularity message passing mechanism
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. Symmetric Multiprocessors (SMP)
   - NUMA (Non-Uniform Memory Access time multiprocessor) for shared address, distributed memory MP

Communication and Synchronization

- Parallel processes must co-operate to complete a single task faster
- Requires distributed communication and synchronization
  - Communication is for data values, or “what”
  - Synchronization is for control, or “when”
  - Communication and synchronization are often inter-related
    - » i.e., “what” depends on “when”
- Message-passing bundles data and control
  - Message arrival encodes “what” and “when”
- In shared-memory machines, communication usually via coherent caches & synchronization via atomic memory operations
  - Due to advent of single-chip multiprocessors, it is likely cache-coherent shared memory systems will be the dominant form of multiprocessor in the near future

Centralized vs. Distributed Memory



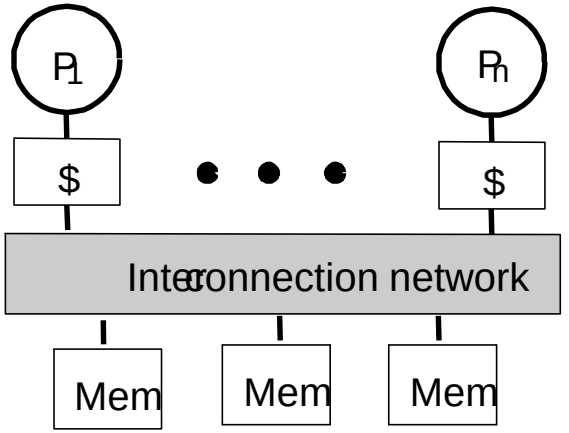
Centralized Memory: UMA

Distributed Memory: NUMA

Symmetric Multiprocessors (SMP)

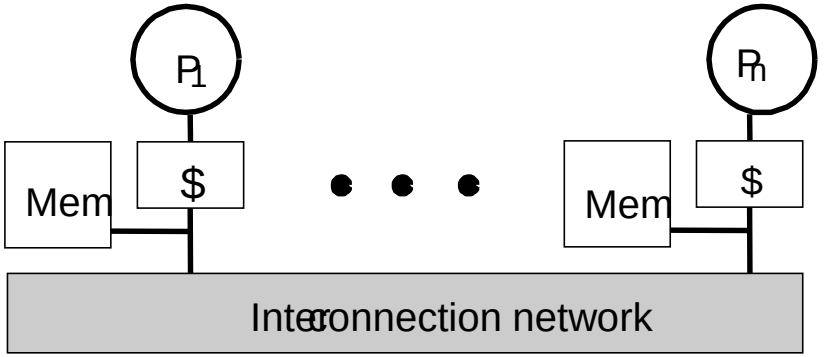
Centralized Memory Multiprocessor

- Single main memory has a symmetric relationship to all processors
- Large caches ⇒ single memory can 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 conceivable, it becomes less attractive as the number of processors sharing centralized memory increases



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:** Software must be aware of data placement to take advantage of increased memory BW



Challenges of Parallel Processing I

- Big challenge is % of program that is inherently sequential
  - What does it mean to be inherently sequential?
- Suppose we want 80X speedup from 100 processors. What fraction of original program can be sequential?

a. 10%

b. 5%

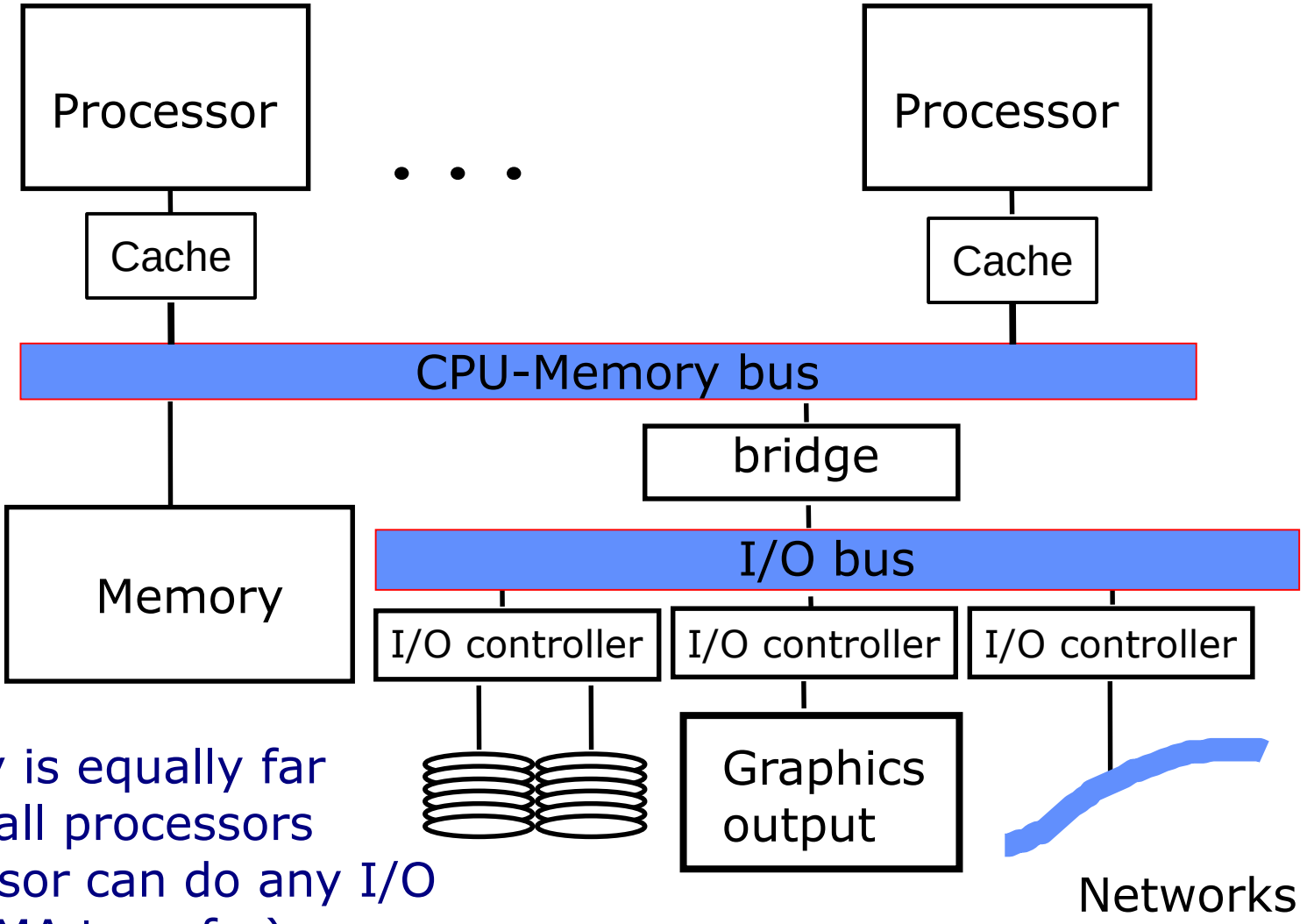
c. 1%

d. <1%

Challenges of Parallel Processing II

- Remote memory accesses can have a detrimental effect on performance
- Suppose in a 2 GHz multiprocessing system each processor requires 200 ns to access a remote memory
- Only 0.2% of the instructions involve a remote access
- Assume base CPI is 0.5
- Each remote access requires 200/0.5 = 400 cycles
- CPI (new) = CPI (base) + Remote Request Rate*Remote Request Cost = 0.5+0.002*400 = 1.3
- Increase of CPI from 0.5 to 1.3!

Symmetric Multiprocessors (SMP)



symmetric

- All memory is equally far away from all processors
- Any processor can do any I/O (set up a DMA transfer)

Cache coherence and consistency problems

- Caching of shared data in multiple locations can create *coherence* and *consistency* problems
- A memory system is *coherent* if
  - any read of a data item from any processor returns the most recently written value of that data item, AND
  - writes to the same location are *serialized:* two writes to the same location by any two processors are seen in the same order by all processors.
- Coherence defines the behavior of reads and writes in the same memory location
- The issue of when exactly a written value must be seen by the reader is defined by the *memory consistency model*

Cache coherence problem

Time	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

Cache coherence when two processors access a memory location X.
At the end of step 3, CPU A has updated the value for X in its cache, but CPU B has still the old value.
This is the cache coherence problem

Cache coherence protocol

Example cache coherence protocol based on **cache line invalidation**.

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

First, the two CPUs read the value of X in their internal cache.
Then, CPU A wants to modify the value of X**. Before it does that, it invalidates the copy of X in the cache of CPU B**.
When CPU B reads X again, it misses in the cache, and CPU A provides the most up-to-date version

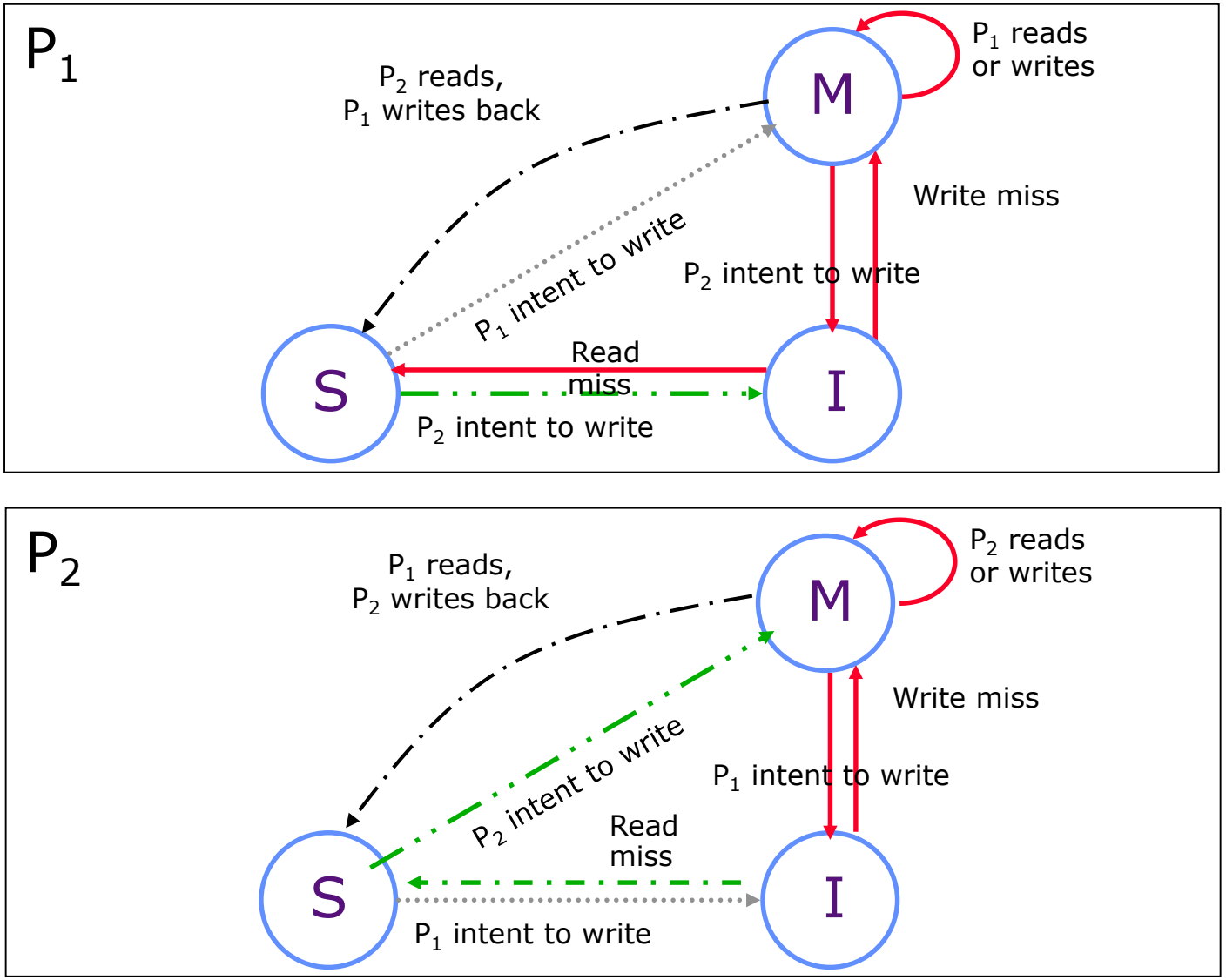
Cache coherence and consistency problems

- Cache coherence protocols used to maintain coherence for multiple processors
- *Snoop based*
  - Every cache with a copy of the shared data, has also a copy of the sharing status of the data
  - No centralized control
  - Cache controllers monitor (snoop) the broadcast medium to determine if they have a copy of the requested block
  - Does not scale very good with larger number of processors.
- *Directory based*
  - The sharing status of a block is kept in a single location – the directory
  - Scales better than snoop-based approaches

Two Processor Example
(Reading and writing the same cache line)

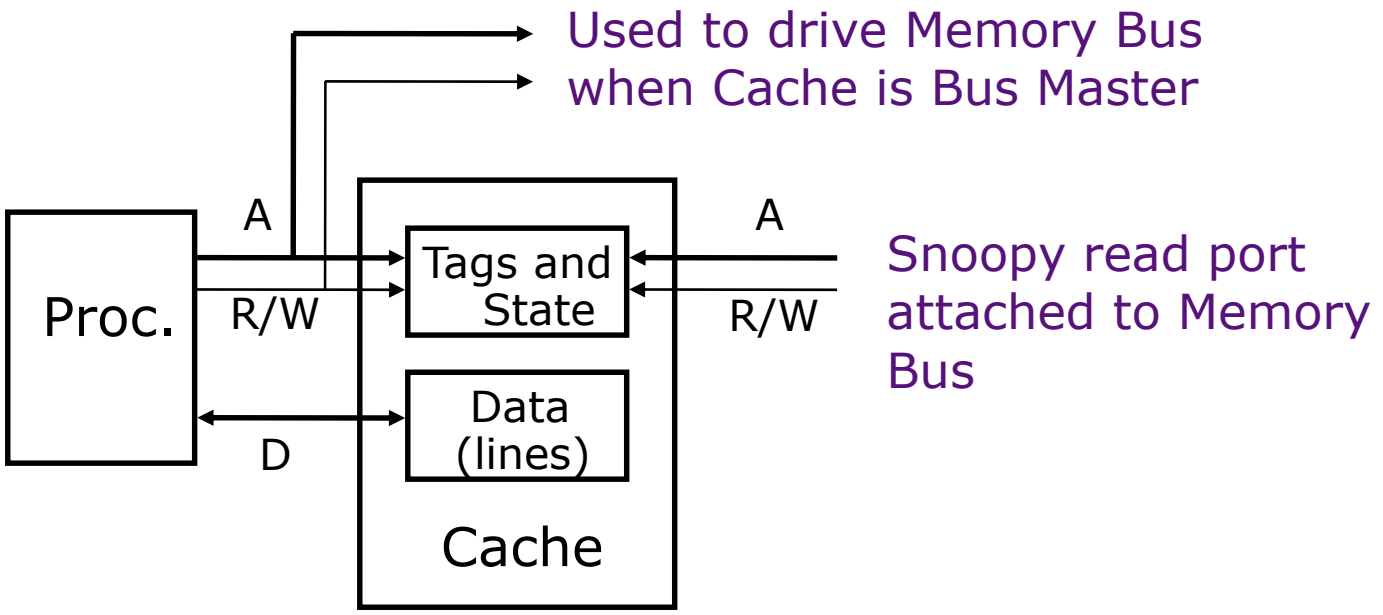
Initially, both P_1 P2 are in (I,I)

P_1 reads (S,I)
P_1 writes (M,I)
P_2 reads (S,S)
P_2 writes (I,M)
P_1 reads (S,S)
P_1 writes (M,I)
P_2 writes (I,M)
P_1 writes (M,I)



Snoopy Cache

- Idea: Have cache watch (or snoop upon) bus transfers, and then "do the right thing"
- Snoopy cache tags are dual-ported



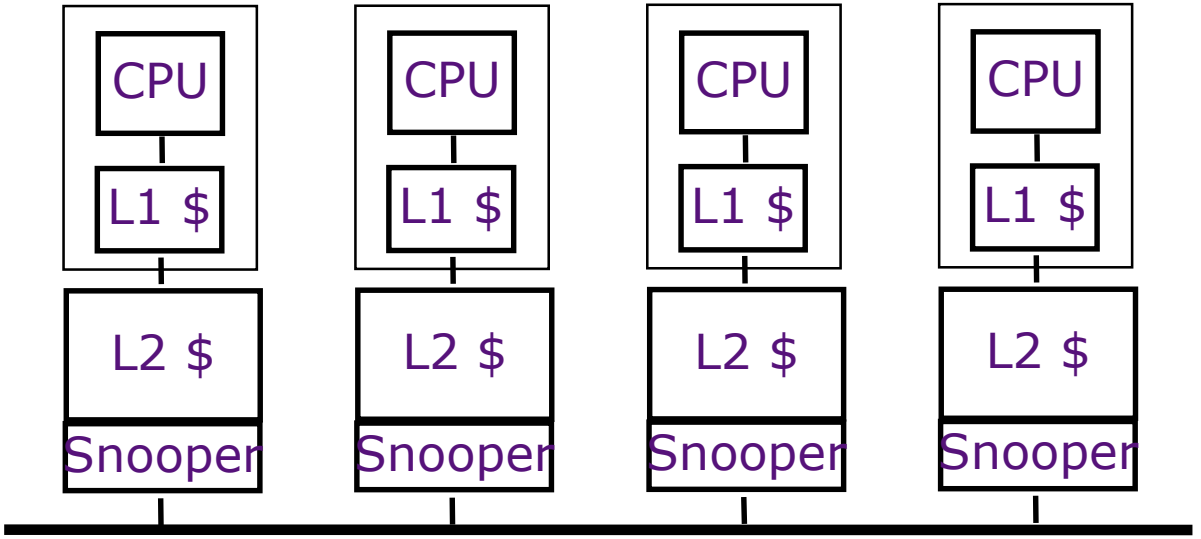
Implementation details

- The MSI protocol assumed that all steps are *atomic*
- Things are not so straightforward (e.g in split transaction buses):
  - In a write miss to location X from the *Modified* state:
    - The CPU misses in location X, and the cache controller requests the bus (1-2 cycles)
    - Waiting for bus arbitration.... (~5 cycles)
    - The bus becomes available, and the cache controller writes back to memory the old data of location X. The bus is released (1 cycle)
    - The cache controller requests the bus again to read in the new data (1-2 cycles)
    - Waiting for bus arbitration.... (~5 cycles)
    - The bus becomes available, and the cache controller reads in the new data from the memory or the owner (3-4 cycles)

Implementation details

- Atomic operations require a *read-modify-write* instruction
- In general, a read-modify-write instruction requires two memory (bus) operations without intervening memory operations by other processors
- In a multiprocessor setting, bus needs to be locked for the entire duration of the atomic read and write operation
  - expensive for simple buses
  - *very expensive* for split-transaction buses
- The solution is a more complex protocol with intermediate states for complex transitions.

Optimized Snoop with L2 Caches



- Processors often have at least two-level cache
  small L1, large L2 (both on chip)
- *Inclusion property:* all entries in L1 must be in L2
  invalidation in L2 ==> invalidation in L1
- Snooping on L2 does not affect CPU-L1 bandwidth
  However, invalidation of L1 may stall the CPU
- Core i7 has inclusive L2 and L3 caches

Performance of Symmetric Shared-Memory Multiprocessors

- Cache performance in SMPs is combination of:
  - Uniprocessor cache miss traffic
  - Coherence misses
    - 4th C: *coherence miss*
    - Joins Compulsory, Capacity, Conflict

Coherence Misses

True sharing misses arise from the communication of data through the cache coherence mechanism

- Processors P1 and P2 share data X1
- Processor P1 wants to write shared data X1 and invalidates data in P2
- Later, Processor P2 wants to read X1 → true sharing cache miss

False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into

- Data X1 and X2 are in the same cache block
- Processor P1 wants to write shared data X1 and invalidates the whole block, including X2
- Later, Processor P2 wants to read X2 → false sharing cache miss
- Miss would not occur if block sizes were 1 word

False Sharing

A cache block contains more than one word

Cache-coherence is usually done at the **block-level** and not word-level (for simplicity)

Suppose P_1 writes $word_i$ and later P_2 reads $word_k$ and
both words have the same block address.

if (i==k) true sharing
if (i <> k) false sharing

Example: True v. False Sharing v. Hit?

- **Assume x1 and x2 in same cache block.**
- **Initially, both cache lines in *shared* state *(S,S)***
- **Any miss that would occur if the block size were one word is designated a sharing miss**

Time	P1	P2	Status
1	Write x1		(M,I) Write Hit
2		Read x2	(S,S) False shar. miss; x1 irrelevant to P2
3	Write x1		(M,I) Write Hit
4		Write x2	(I, M) False shar. miss; x1 irrelevant to P2
5	Read x2		(S,S)True shar. miss; invalidate x2 in P1

SMP Performance
Normalized Exec. Time vs L3 Cache Size

- Alpha Server 4100
  4 Processor SMP, 4-issue per cycle per core, 300 MHz cores

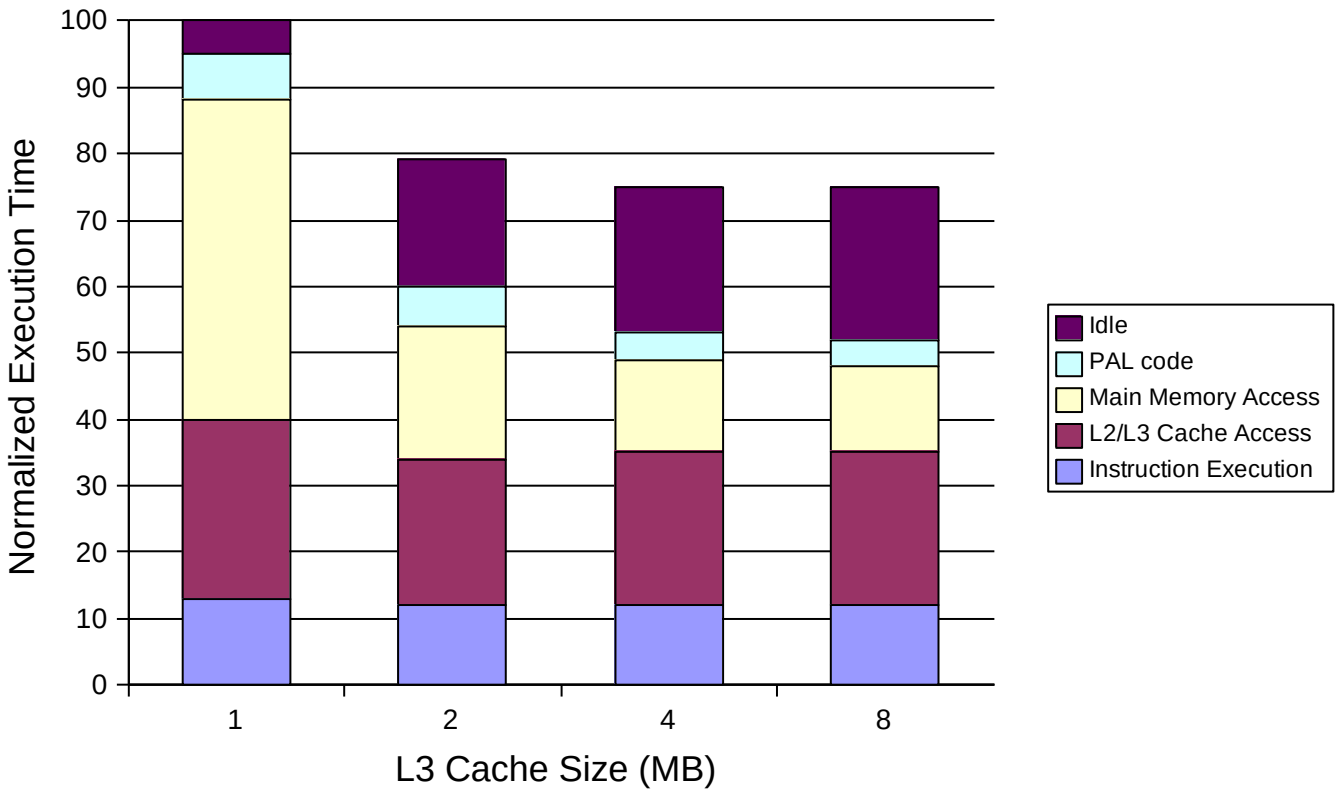
- Three-level cache hierarchy for each core – L3 is off-chip
- L2 Cache is unified, 96 KB

- Commercial Workload: OLTP (online transaction processing workload) modeled after the TPC-B benchmark suite

- Cache misses decrease with larger L3 Caches, but not beyond 2 MBs.

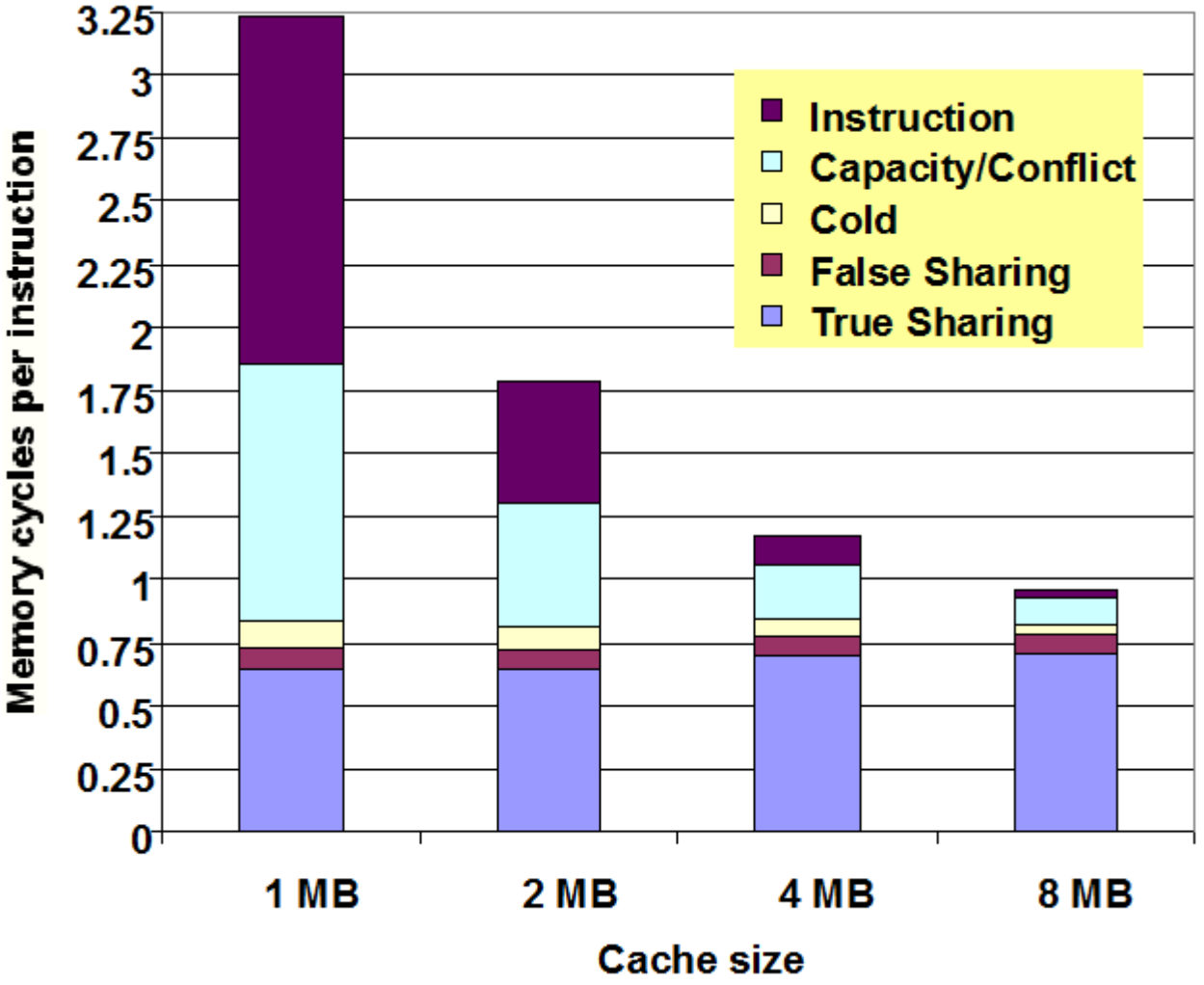
- Idle time increases with larger L3 cache because the I/O effect becomes stronger – Fewer memory stalls require more server processes to keep the processors busy

- Performance unchanged for L3 Cache larger than 2 MB



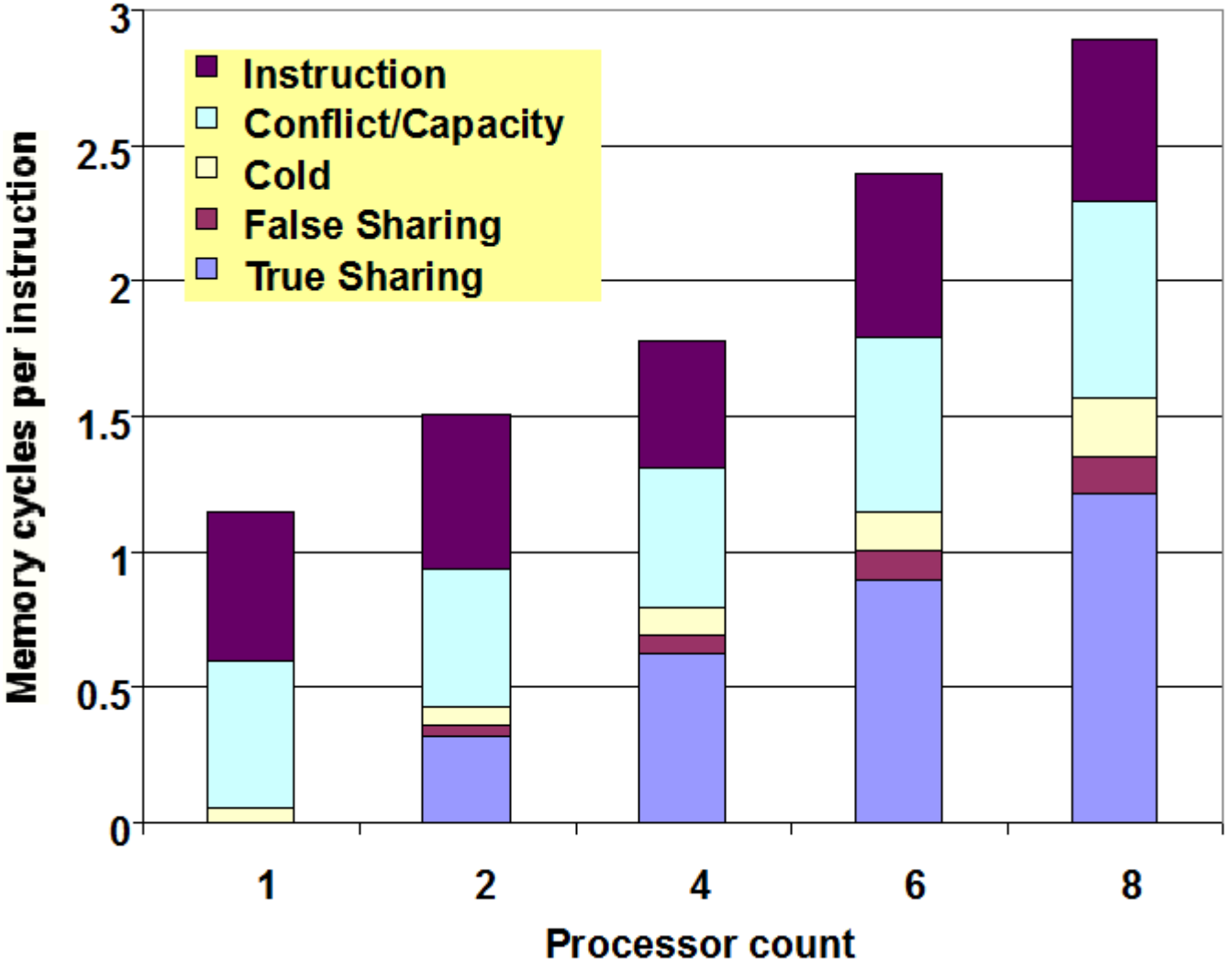
SMP Performance
Memory Cycles vs L3 Cache Size

- What is the reason for high execution overhead due to L3 Caches?
- Instruction and Capacity/Conflict misses the main miss contributors in smaller L3 Caches
- True sharing misses dominate for larger L3 Caches.
- True sharing and Cold misses untouched by larger L3 Caches



SMP Performance
Memory Cycles vs # of processors

- 2 MB, 2-way set associative L3 cache
- True sharing, false sharing increase going from 1 to 8 CPUs



SMP problems

- Snooping requires message broadcasting to all processors
- This increases the bandwidth and does not scale well with increasing number of processors
- The broadcast medium (e.g. bus) and the memory become bottlenecks
- Snooping used for up to about 10 processors