Multiprocessors

There have been many reasons for the success of shared memory multiprocessors on the market.

In my opinion, the simplicity of programming and the growth of the service sector due to the Internet and the Web were the main ones.

The continuous goal in the design is to achieve scalability and contain consumption.

1

Multiprocessors

Computers consisting of tightly coupled processors whose coordination and usage are controlled by a **single operating system** and that **share memory** through a **shared address space**.

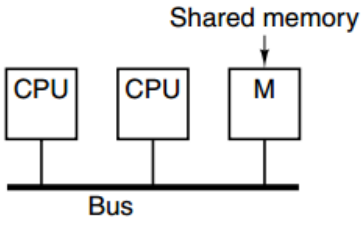
Such systems exploit thread-level parallelism through two differen software models.

- ***The execution of a tightly coupled set of threads collaborating on a single task, which is typically called parallel processing.***
- ***The execution of multiple, relatively independent processes that may originate from one or more users, which is a form of request-level parallelism.***
- ***Request-level parallelism may be exploited by a single application running on multiple processors, such as a database responding to queries, or multiple applications running independently, often called multiprogramming.***

The main rules in design of hw/sw parallel solutions:

1. Limit the overhead of interprocess communication: shared memory and caches.

The programming model



2

The multiprocessors range

- The multiprocessors range in size from a dual processor to dozens and sometimes hundreds of processors and communicate and coordinate through the sharing of memory.
- Although sharing through memory implies a shared address space, it does not necessarily mean there is a single physical memory.
- Such multiprocessors include both single-chip systems with multiple cores, known as multicore, and computers consisting of multiple chips, each of which is typically a multicore.
  - HP, Dell, Cisco, IBM, SGI, Lenovo, Samsung, Oracle, Fujitsu, and many others
- Many multicore processors also include support for multithreading.

3

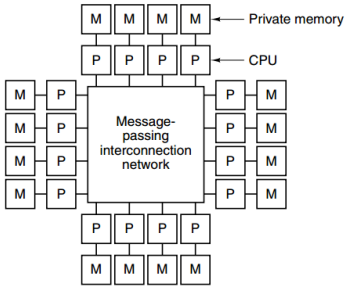
Multicomputers

There are a range of special large-scale multiprocessor systems, sometimes called multicomputers, which are less tightly coupled than the multiprocessors.

The primary use for such multicomputers is in high-end scientific computation, although they are sometimes used for commercial applications filling the niche between multiprocessors and warehousescale computers.

- Examples: the Cray X series and IBM BlueGene

- Scalability: up to hundreds of computer elements.
- Consumption: switching on or off of single computer element
- Communication overhead: linked to the type of network, the size of the system and the process mapping on processors.
- load balancing: main goal of the project.
- Difficulty in designing the software: dynamic load balancing of processors, process mapping on processors to minimize the average delay in communications.



4

Who and how to create processes and threads

To take advantage of an MIMD multiprocessor with n processors, we must usually have at least n threads or processes to execute; with multithreading, which is present in most multicore chips today, that number is 2–4 times higher.

The independent threads within a single process are typically identified:

1. by the programmer in service applications or
2. created by the operating system (from multiple independent requests).
3. Threads may be generated by a parallel compiler exploiting data parallelism with the iterations of a loop.

5

Speedups in any parallel processor

Limitations in available parallelism make it difficult to achieve good speedups in any parallel processor, as our first example shows.

- Suppose you want to achieve a speedup of 80 with 100 processors. What fraction of the original computation can be sequential?
- Amdahl's Law is:

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

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

$$\text{Fraction}_{\text{parallel}} = 0.9975$$

- For simplicity in this example, assume that the program operates in only two modes: parallel with all processors fully used, which is the enhanced mode, or serial with only one processor in use.

- To achieve a speedup of 80 with 100 processors, only 0.25% of the original computation can be sequential!

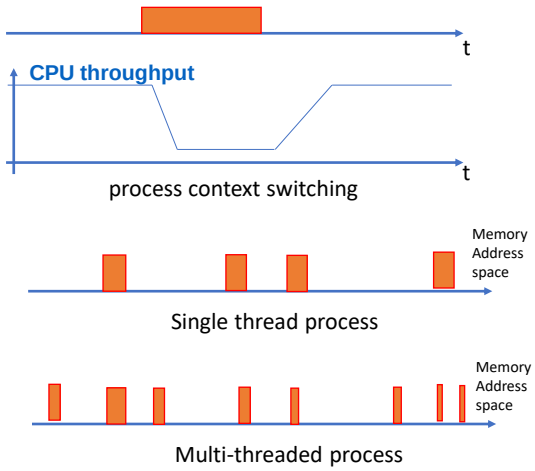
6

Take the overhead into consideration

Take the overhead into consideration. Threads can also be used to exploit data-level parallelism, although the overhead is usually higher than would be seen with a SIMD processor or with a GPU.

- This overhead means that grain size must be sufficiently large to exploit the parallelism efficiently.
- For example, although a vector processor or GPU may be able to efficiently parallelize operations on short vectors, the resulting grain size when the parallelism is split among many threads may be so small that the overhead makes the exploitation of the parallelism prohibitively expensive in a MIMD.
- Cache effects due to multiple threads applications and process context switching.

Cache effects:
unbalanced distribution of misses



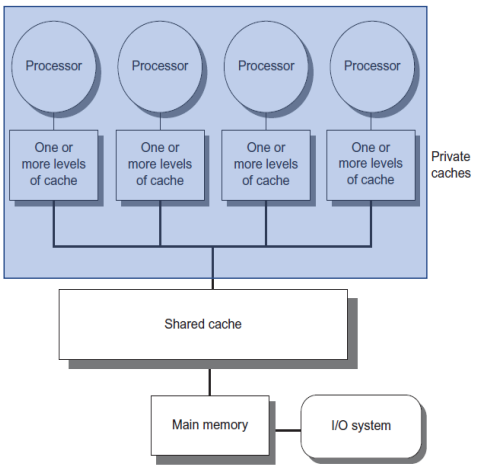
7

SMP **symmetric** (shared-memory) multiprocessors

SMPs, or centralized shared-memory multiprocessors, features small to moderate numbers of cores, typically 32 or fewer.

For multiprocessors with such small processor counts, it is possible for the processors to share a single centralized memory that all processors have equal access to, thus the term **symmetric**.

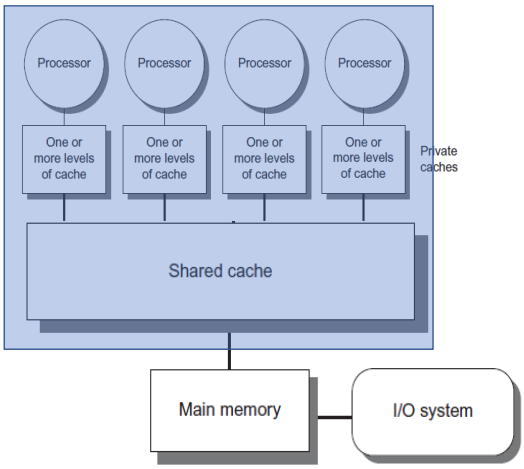
Symmetric means that the scheduler solves the workload balance by means the processes migration.



8

SMP **symmetric** (shared-memory) multiprocessors

SMP architectures are also sometimes called **uniform memory access** (UMA) multiprocessors, arising from the fact that all processors have a uniform latency from memory, even if the memory is organized into multiple banks.

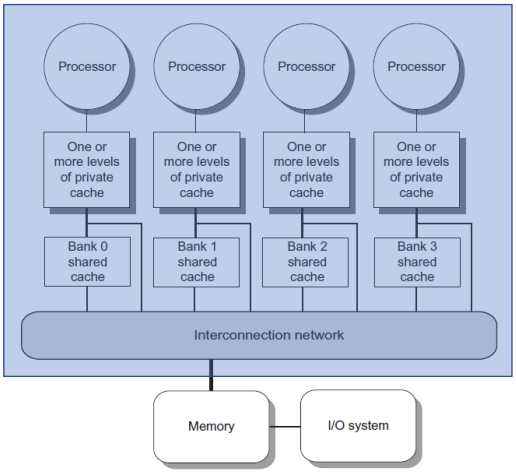


9

SMP **symmetric** (shared-memory) multiprocessors

Some multicores have nonuniform access to the outermost cache, a structure called NUCA for Nonuniform Cache Access, and are thus are not truly SMPs, even if they have a single main memory.

- The IBM Power8 has distributed L3 caches with nonuniform access time to different addresses in L3.



10

DSM, Distributed Shared Memory multiprocessor

In multiprocessors consisting of multiple multicore chips, there are often separate memories for each multicore chip.

- Thus the memory is distributed rather than centralized.

Many multiprocessors with distributed memory have fast access to a local memory and much slower access to remote memory.

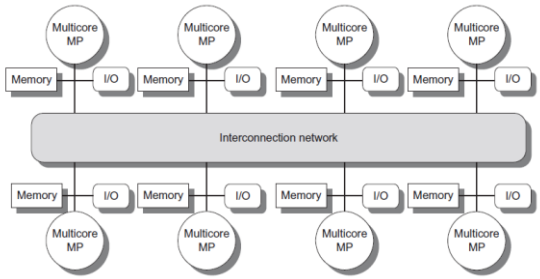
- Often the differences in access time to various remote memories are small in comparison to the difference between the access times to the local memory and to a remote memory.

To support larger processor counts, memory must be distributed among the processors rather than centralized.

- Otherwise, the memory system would not be able to support the bandwidth demands of a larger number of processors without incurring excessively long access latency.

The alternative design approach consists of multiprocessors with physically distributed memory, called distributed shared memory.

A **DSM** multiprocessor is called a **NUMA** (nonuniform memory access) because the access time depends on the location of a data word in memory.



11

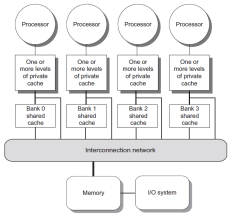
Communication among processes in SMP and DSM system

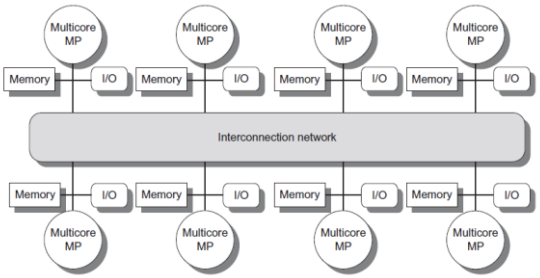
The term shared memory associated with both SMP and DSM refers to the fact that the address space is shared.

- **In both SMP and DSM architectures, communication among threads occurs through a shared address space.**
- A memory reference can be made by any processor to any memory location.
- In some cases, the operations in memory are carried out to hardware through messages.

The clusters and warehouse-scale computers look like individual computers connected by a network, and **the memory of one processor cannot be accessed by another processor** without the assistance of software protocols running on both processors.

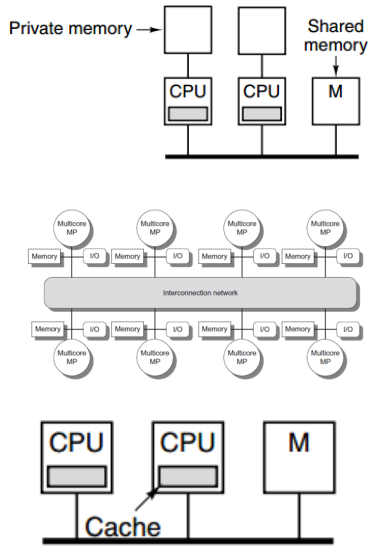
In such designs, message-passing protocols are used to communicate data among processors.





12

Shared memory is the bottleneck for performance and scalability



Private memory and distributed memory

- Distributing the memory among the nodes and private memories increase the bandwidth and reduce the latency to local memory.
  - Increase the complexity
- Cache memory
  - multilevel caches can substantially reduce the memory bandwidth demands of a processor
    - Side effects in context switching
- Multithreaded, out-of-order execution, and speculative execution
  - Cover the latency in access to the shared memory
    - Increase the complexity

13

Reference to a remote memory

Suppose we have an application running on a 32-processor multiprocessor that has a **100 ns** delay to handle a reference to a remote memory.

Processors are stalled on a remote request, and the processor clock rate is 4 GHz.

- If the base CPI (assuming that all references hit in the cache) is 0.5, how much faster is the multiprocessor
  1. If there is no communication versus
  2. if 0.2% of the instructions involve a remote communication reference?

$$\text{CPI} = \text{Base CPI} + \text{Remote request rate} \times \text{Remote request cost}$$
$$= 0.5 + 0.2\% \times \text{Remote request cost}$$

$$\frac{\text{Remote access cost}}{\text{Cycle time}} = \frac{100\,\text{ns}}{0.25\,\text{ns}} = 400\,\text{cycles}$$

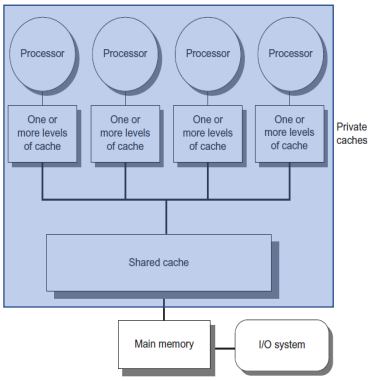
$$\text{CPI} = 0.5 + 0.20\% \times 400$$
$$= 1.3$$

The multiprocessor with all local references is 1.3/0.5 = 2.6 times faster.

14

Design issues for all the multiprocessor models

- Write operations on shared copies in caches
- Exploit the asynchronous features of write
- Low-level atomic mechanism (test&set)
- I/O interrupts
- I/O bus



15

Design issues for all the multiprocessor models

Write operations on shared copies in caches

- Coherence protocol, write-back on private copies and write-through on shared copies

Exploit the asynchronous features of writes

- Memory consistency model

Low-level atomic mechanism (test & set)

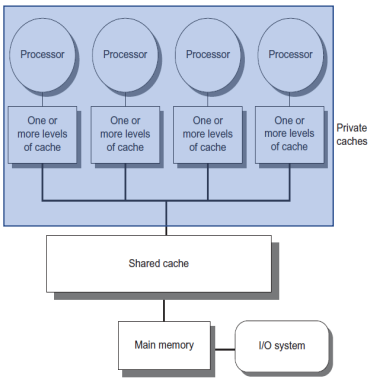
- By using the Coherence protocol

I / O interrupts

- The kernel can decide on which processor to send the interrupt (one fixed or the one on which the lowest priority process is running)

I / O bus

- Specific additional bus to have easy the memory bus



16

Multiprocessor Cache Coherence

When shared data are cached, the shared value may be replicated in multiple private caches.

The coherence problem exists because we have both a global state, defined primarily by the main memory, and a local state, defined by the individual caches, which are private to each processor core.

Because the view of memory held by two different processors is through their individual caches, the processors could end up seeing different values for the same memory location.

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

Write-through cache

If a processor (B) could continuously read an old data value, we would clearly say that memory was incoherent.

17

The design space of the solution

- Definition on a global status for each copy (that is, not only valid and invalid)
- Definition a strategy on local or remote operations

Coherence protocol

Main goals:
- Reduce additional misses
- Reduce the memory operations

- Identify within when the remote coherence actions are to be concluded

Memory consistency model

Main goals:
- Reduce the complexity of additional hardware and
- increase the interval within which operations must do the operations (write asynchronous)

True sharing

False sharing

Unfortunately, the performance depends on
– the type of sharing of the application, the percentage of operation made on the shared areas and
- the behavior of the operating system

18

Coherence and consistency

Coherence and consistency are complementary:

- **Coherence** defines the behavior of reads and writes to the same memory location, while **consistency** defines the behavior of reads and writes with respect to accesses to other memory locations.
- After a write on a shared copy:
  - **Coherence** protocol establishes what operations must be done to make the view of memory consistent by all processors (through their own cache).
  - The **memory consistency model** tells when the operations established by the coherence protocol are to be carried out.

19

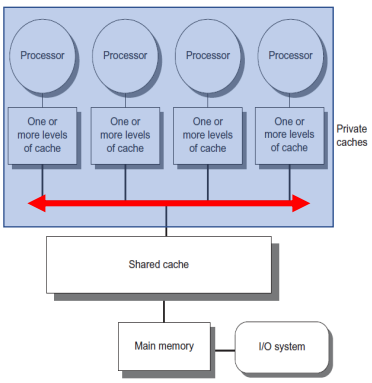
Track the sharing status of a cached copy

Snooping

Every cache that has a copy of the data from a block of physical memory could track the sharing status of the block.

In an SMP, the caches are typically all accessible via some broadcast medium (e.g., a bus connects the per-core caches to the shared cache or memory), and all cache controllers monitor or snoop on the medium to determine whether they have a copy of a block that is requested on a bus or switch access.

Snooping can also be used as the coherence protocol for a multichip multiprocessor, and some designs support a snooping protocol on top of a directory protocol within each multicore.



20

Track the sharing status of a cached copy

Directory based

The sharing status of a particular block of physical memory is kept in one location, called the directory.

In an SMP, we can use one centralized directory, associated with the memory or some other single serialization point, such as the outermost cache in a multicore.

In a DSM, Distributed directories among the processors are used.

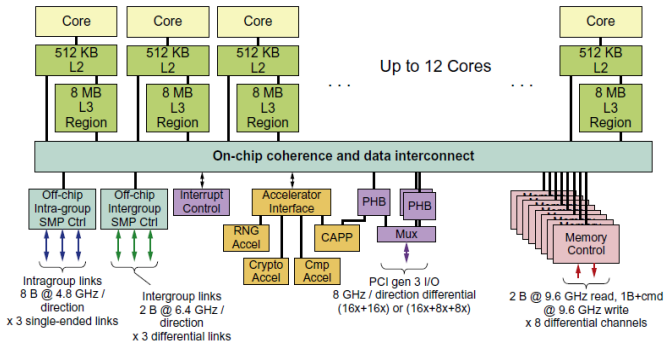
During the execution of the miss, the cache controller asks the status of the copy (not present in the remaining caches, private of a processor or shared among a list of processors) to the directory, updates its directory and all the rest.

21

Two ways to have coherent caches

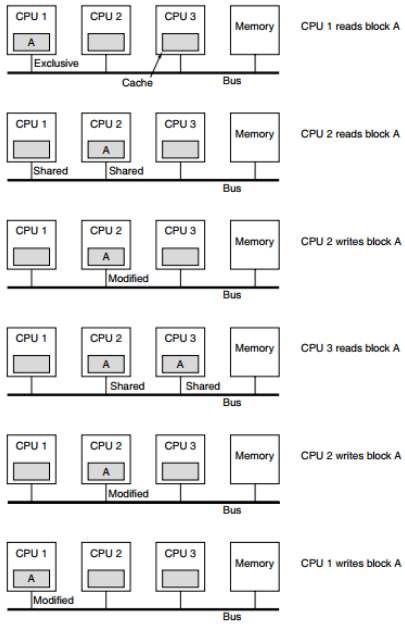
There are two ways to keep cached copies consistent.

- **Write update**. The remaining copies are updated when a processor performs a write operation on a shared copy.
  - The update takes place in broadcast using a bus shared between all the caches.
  - Solution variant: writing in broadcast also updates the copy in shared memory.
  - Cost: the write on the shared bus.



- **Write invalidate**: The remaining copies are invalidated when a processor performs a write operation on a shared copy.
  - The invalidation takes place in broadcast using a bus shared between all the caches.
  - Cost: produces a miss if a processor needs to reuse the copy.
  - It is the most used protocol in multicores.

22



MESI PROTOCOL

This protocol enforces write serialization.

For a write, we require that the writing processor has exclusive access, preventing any other processor from being able to write simultaneously.

If two processors do attempt to write the same data simultaneously, one of them wins the race (we'll see how we decide who wins shortly), causing the other processor's copy to be invalidated.

For the other processor to complete its write, it must obtain a new copy of the data, which must now contain the updated value.

Modified: the entry is valid; memory is invalid; no copies exist.

Exclusive: no other cache holds the block; memory is up to date.

Shared: multiple caches may hold the block; memory is up to date.

Invalid: the block is not cached

23

MESI PROTOCOL

Modified: the entry is valid; memory is invalid; no copies exist.

Exclusive: no other cache holds the block; memory is up to date.

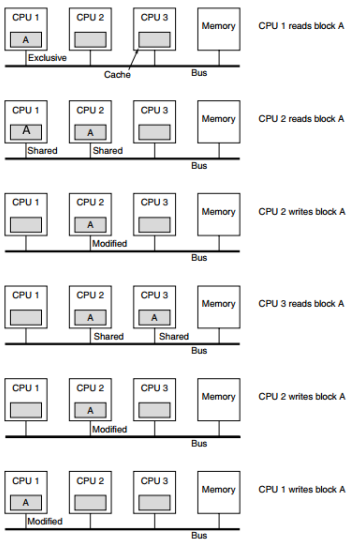
Shared: multiple caches may hold the block; memory is up to date.

Invalid: the block is not cached

MESI Protocol State Transition Table

State	*This Processor* Load	Store	*Other Processor* Load Miss	Store Miss
Invalid (I)	Miss → S or E	Miss → M	---	---
Shared (S)	Hit	Miss → M	---	→ I
Exclusive (E)	Hit	**Hit → M**	Send Data → S	Send Data → I
Modified (M)	Hit	Hit	Send Data → S	Send Data → I

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓



24

MOESI PROTOCOL

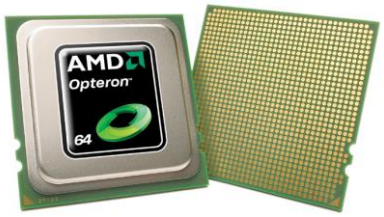
Modified: the entry is valid; memory is invalid; no copies exist.

Owner: multiple caches may hold the block; memory is NOT up to date.

Exclusive: no other cache holds the block; memory is up to date.

Shared: multiple caches may hold the line.

Invalid: the block is not cached



AMD Opteron: 2003

	M	O	E	S	I
M	✗	✗	✗	✗	✓
O	✗	✗	✗	✓	✓
E	✗	✗	✗	✗	✓
S	✗	✓	✗	✓	✓
I	✓	✓	✓	✓	✓

25

MSI PROTOCOL

Modified: the entry is valid; memory is invalid; no copies exist.

Shared: one or multiple caches may hold the block; memory is up to date.

Invalid: the cache entry does not contain valid data

MSI Protocol State Transition Table

State	*This Processor* Load	Store	*Other Processor* Load Miss	Store Miss
Invalid (I)	Miss → S	Miss → M	---	---
Shared (S)	Hit	Miss → M	---	→ I
Modified (M)	Hit	Hit	Send Data → S	Send Data → I

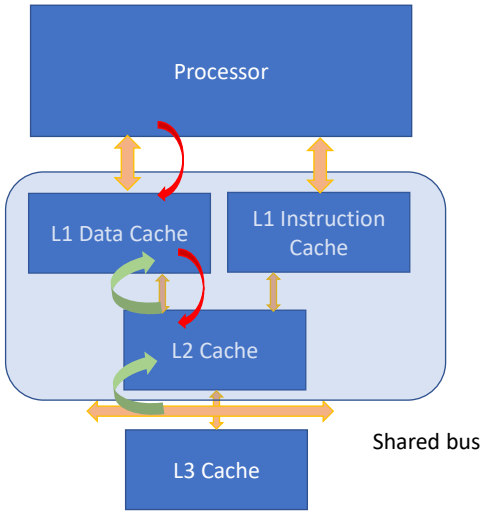
	M	S	I
M	✗	✗	✓
S	✗	✓	✓
I	✓	✓	✓

26

Duplicate the tags

Every bus transaction must check the cache-address tags, which could potentially interfere with processor cache accesses.

One way to reduce this interference is to duplicate the tags and have snoop accesses directed to the duplicate tags.



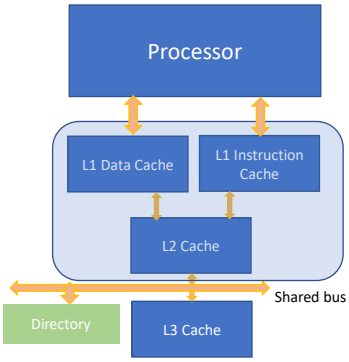
27

Directory

Every bus transaction must check the cache-address tags, which could potentially interfere with processor cache accesses.

Another approach is to use a directory at the shared L3 cache; the directory indicates whether a given block is shared and possibly which cores have copies.

With the directory information, invalidates can be directed only to those caches with copies of the cache block.



Shared One or more nodes have the block cached, and the value in memory is up to date (as well as in all the 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 date. The processor is called the owner of the block.

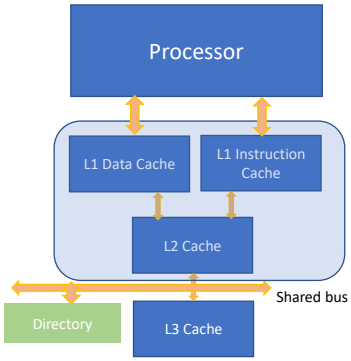
28

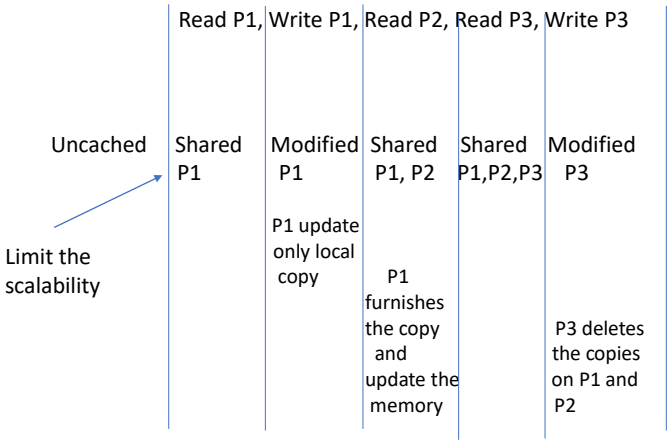
Directory

Shared One or more nodes have the block cached, and the value in memory is up to date (as well as in all the 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 date. The processor is called the owner of the block.

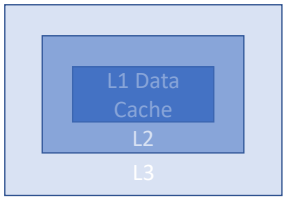


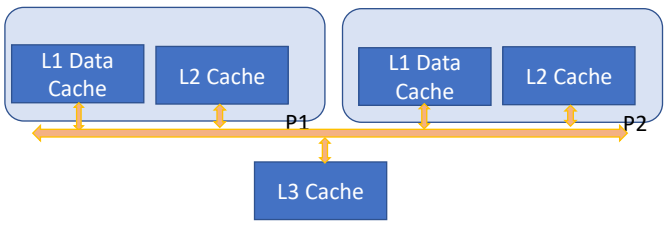


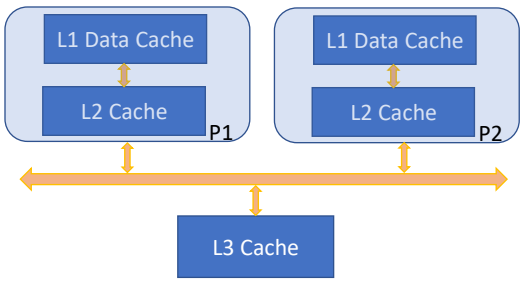
29

The inclusion property

The protocol requires that L3 must always have a copy of any data item in L1 or L2, a property called inclusion.

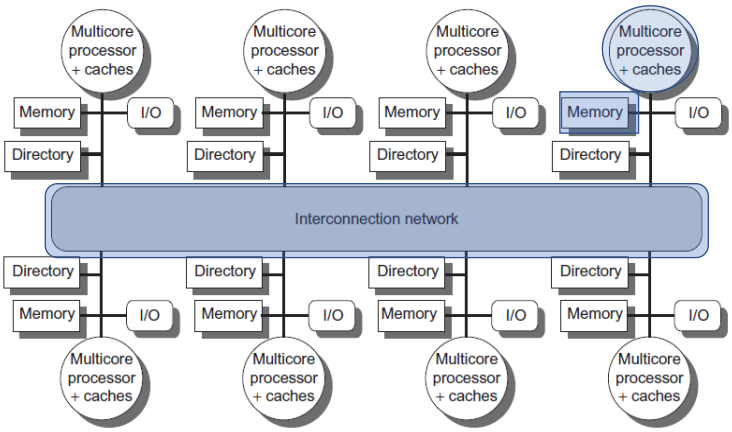






30

Directory-Based Cache Coherence Protocols



1. higher bandwidth interconnection schemes.
2. multiple, independent shared memories to allow larger numbers of cores.
3. Nodes are multiprocessor shared scheme.

Independent virtual/physical translation mechanism

31

Limitations in Symmetric Shared-Memory Multiprocessors

As the number of processors in a multiprocessor grows, or as the memory demands of each processor grow, any centralized resource in the system can become a bottleneck.

For multicores, a single shared bus became a bottleneck with only a few cores.

As a result, multicore designs have gone to

1. higher bandwidth interconnection schemes, as well as,
2. multiple, independent memories to allow larger numbers of cores.

For example:

- The IBM Power8, which has up to 12 processors in a single multicore, uses 8 parallel buses that connect the distributed L3 caches and up to 8 separate memory channels.
  - Power8 has nonuniform access time for both L3 and memory.
- The Xeon E7 uses three rings to connect up to 32 processors, a distributed L3 cache, and two or four memory channels (depending on the configuration).
  - The Xeon E7 can operate as if access times were uniform; in practice, software systems usually organize memory so that the memory channels are associated with a subset of the cores.
- 3. The Fujitsu SPARC64 X+ uses a crossbar to connect a shared L2 to up to 16 cores and multiple memory channels.
  - The SPARC64 X+ is a symmetric organization with uniform access time.

The Snooping bandwidth at the caches can also become a problem because every cache must examine every miss, and having additional interconnection bandwidth only pushes the problem to the cache. To understand this problem, consider the following example.

32

AMD Opteron

The AMD Opteron represents another intermediate point in the spectrum between a snooping and a directory protocol.

Memory is directly connected to each multicore chip, and up to four multicore chips can be connected.

- The system is a NUMA because local memory is somewhat faster.

The Opteron implements its coherence protocol using the point-to-point links to broadcast up to three other chips.

- Because the interprocessor links are not shared, the only way a processor can know when an invalid operation has completed is by an explicit acknowledgment.
- Thus the coherence protocol uses a broadcast to find potentially shared copies, like a snooping protocol, but uses the acknowledgments to order operations, like a directory protocol.

Because local memory is only somewhat faster than remote memory in the Opteron implementation, some software treats the Opteron multiprocessor as having uniform memory access.

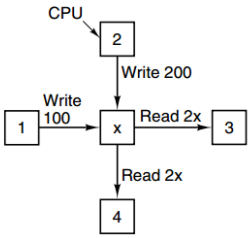
33

Memory Consistency Model

34

Memory Consistency

- **Strict consistency**: any read to a location *x* always returns the value of the most recent write to *x*.
- **Sequential consistency**: in the presence of multiple read and write requests, some interleaving of all the requests is chosen by the hardware (nondeterministically), **but all CPUs see the same order.**



W100	W100	W200
W200	R3 = 100	R4 = 200
R3 = 200	W200	W100
R3 = 200	R4 = 200	R3 = 100
R4 = 200	R3 = 200	R4 = 100
R4 = 200	R4 = 200	R3 = 100

35

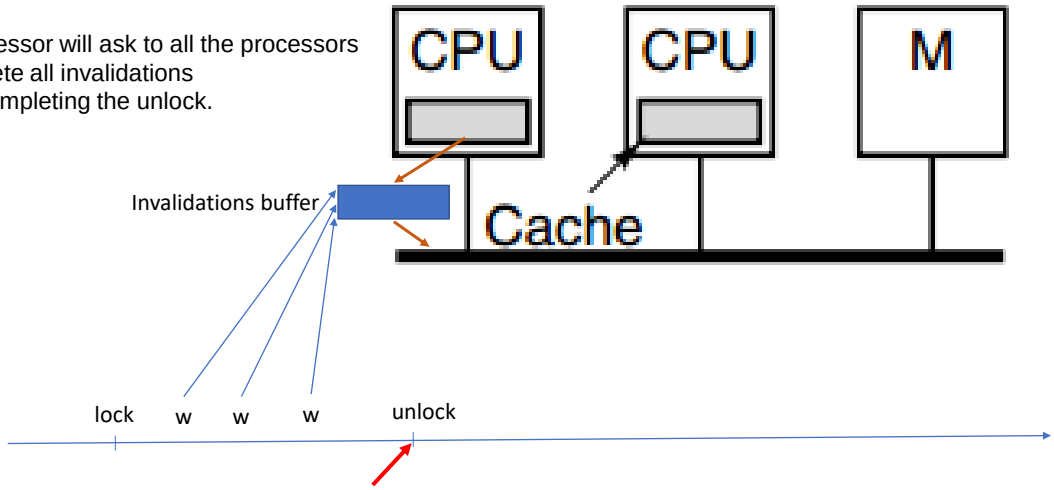
Memory Consistency model

- **Processor Consistency**
  1. Writes by any CPU are seen by all CPUs in the order they were issued.
  2. For every memory word, all CPUs see all writes to it in the same order.
- **Weak Consistency**: no order in write operation but synchronization point.
- **Release Consistency:** write before a release are performed before a new acquisition.

36

Memory Consistency Model (I)

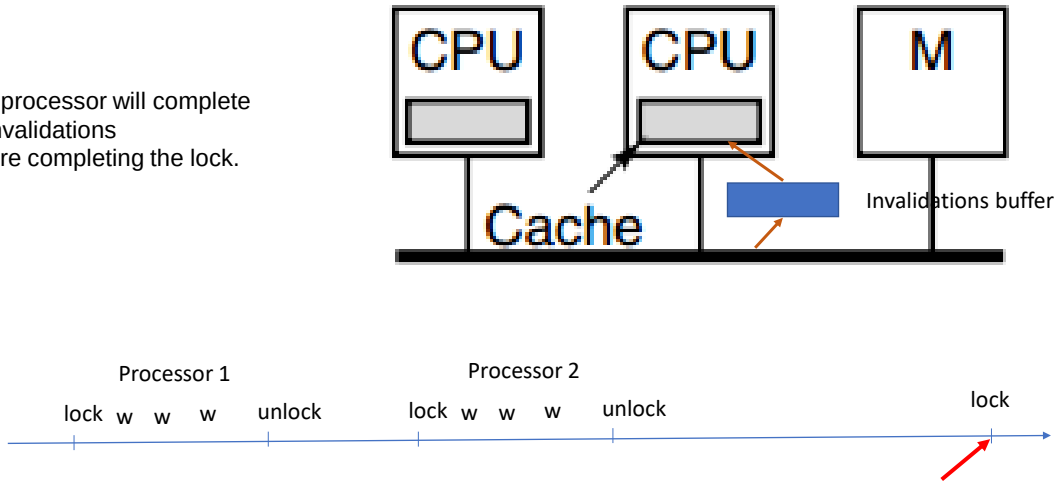
The processor will ask to all the processors to complete all invalidations before completing the unlock.



37

Memory Consistency Model (II)

The processor will complete all invalidations before completing the lock.



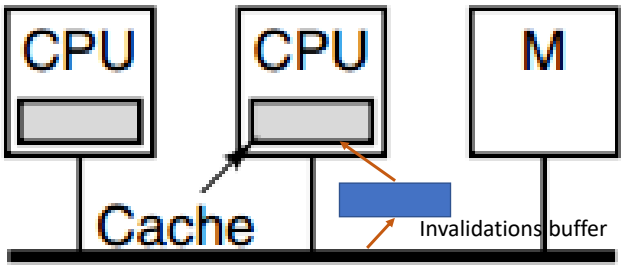
38

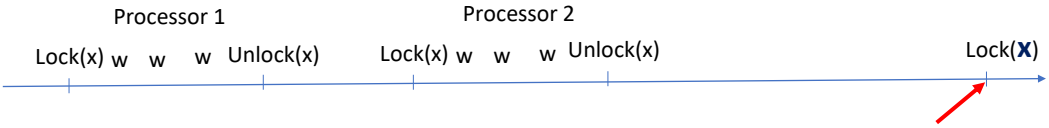
Memory Consistency Model (III)

The programmar must specify all the shared variables.

Invalidations of shared variables are labeled with the current value of the critical section x within which they were created.

The processor will complete all invalidations labeled with x before completing the lock(x).





39

Synchronization

40

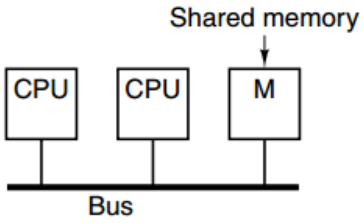
Lock and unlock operations in multiprocessor

Lock and unlock can be used to create mutual exclusion in the kernel, as well as to implement more complex synchronization mechanisms at application level.

Synchronization mechanisms are built with software routines that rely on hardware-supplied synchronization instructions.

For smaller multiprocessors or low-contention situations, the key hardware capability is an uninterruptible instruction or instruction sequence capable of atomically retrieving and changing a value.

In high-contention situations, synchronization can become a performance bottleneck because contention introduces additional delays and because latency is potentially greater in such a multiprocessor.



41

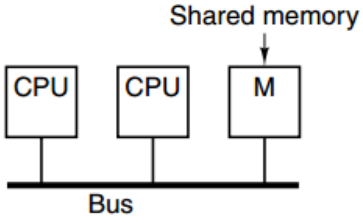
Atomically read and modify a memory location

The key ability we require to implement synchronization in a multiprocessor is a set of hardware primitives with the ability to atomically read and modify a memory location.

Atomic exchange (swap, test&set, fetch&increment)

For example, consider two processors that each try to do the exchange simultaneously:

- This race is broken because exactly one of the processors will perform the exchange first, returning 0, and the second processor will return 1 when it does the exchange.
- The key to using the exchange (or swap) primitive to implement synchronization is that the operation is **atomic**: the exchange is indivisible, and two simultaneous exchanges will be ordered by the write serialization mechanisms.



42

Verify if the pair of instructions was effectively atomic

An alternative is to have a pair of instructions where the second instruction returns a value from which it can be deduced whether the pair of instructions was executed as though the instructions were atomic.

The pair of instructions is effectively atomic if it appears as though all other operations executed by any processor occurred before or after the pair.

In **RISC V**, the pair of instructions includes a special load called a **load reserved** (also called load linked or load locked) and a special store called a **store conditional**.

Load reserved loads the contents of memory given by rs1 into rd and creates a reservation on that memory address.

Store conditional stores the value in rs2 into the memory address given by rs1.

- If the reservation of the load is broken by a write to the same memory location, the store conditional fails and writes a nonzero rd; if it succeeds, the store conditional writes 0.
- If the processor does a context switch between the two instructions, then the store conditional always fails.

The same function can be achieved via a bus.

The serialization of the bus guarantees the atomicity of the request and the reading of the status.

43

Implementing Locks Using Coherence

```
lockit:  lr    x2,0(x1)   ;load reserved
         bnez  x2,lockit  ;not available-spin
         addi  x2,R0,#1   ;locked value
         sc    x2,0(x1)   ;store
         bnez  x2,lockit  ;branch if store fails
```

Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied.
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied.
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

44

Performance of Symmetric Shared-Memory Multiprocessors

In a multicore using a snooping coherence protocol, several different phenomena combine to determine performance.

The overall cache performance is a combination of the behavior of **uniprocessor cache miss** traffic and the traffic caused by **communication**, which results in **invalidations and subsequent cache misses**.

Changing the processor count, cache size, and block size can affect these two components of the miss rate in different ways, leading to overall system behavior that is a combination of the two effects.

45

Additional misses

The first source is the true sharing misses that arise from the communication of data through the cache coherence mechanism.

- In an invalidation-based protocol, the first write by a processor to a shared cache block causes an invalidation to establish ownership of that block.
- Additionally, when another processor attempts to read a modified word in that cache block, a miss occurs and the resultant block is transferred.
- Both these misses are classified as true sharing misses because they directly arise from the sharing of data among processors.

The second effect, called false sharing occurs when a block is invalidated (and a subsequent reference causes a miss) because some word in the block, other than the one being read, is written into.

Time	P1	P2
1	Write z1	
2		Read z2
3	Write z1	
4		Write z2
5	Read z2	

Assume that words **z1 and z2 are in the same cache block**, which is in the shared state in the caches of both P1 and P2.

46

Additional invalidations due to false sharing

A process (or kernel) operates on a private variable.

The process (or kernel) migration can produce two copies on two caches. Copies must be kept consistent even if one of them will never be used.

time	Processor 1	Processor 2	Cache (MSI protocol)
1	Process B, Read x		Processor 1 has a copy Shared
2			Process B migrates from Processor 1 to Processor 2
3		Process B, Read x	Processor 1 and 2 have the copy Shared
4		Process B, Write x	An invalidation must be sent to the Processor 1. Processor 2 has the copy Modified

47

Additional updates due to false sharing

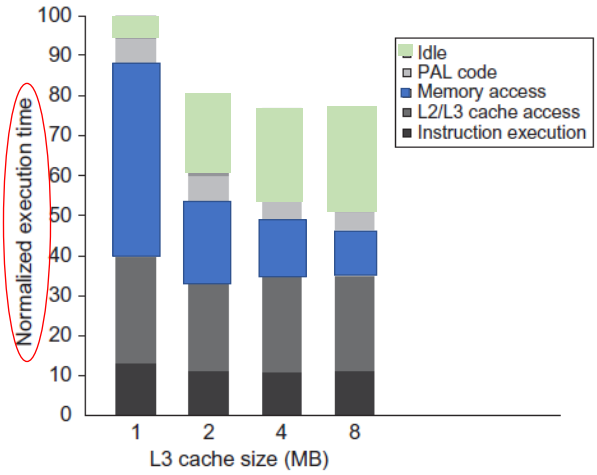
A process (or kernel) operates on a private variable.

The process (or kernel) migration can produce two copies on two caches. Copies must be kept consistent even if one of them will never be used.

time	Processor 1	Processor 2	Cache (Update protocol)
1	Process B, Read x		Processor 1 has a copy
2			Process B migrates from Processor 1 to Processor 2
3		Process B, Read x	Processor 1 and 2 have the copy Shared
4		Process B, Write x	An update must be sent to the Processor 1.

48

Online transaction-processing (OLTP) workload



Each processor in the Alpha-Server 4100 is an Alpha 21164, which issues up to four instructions per clock.

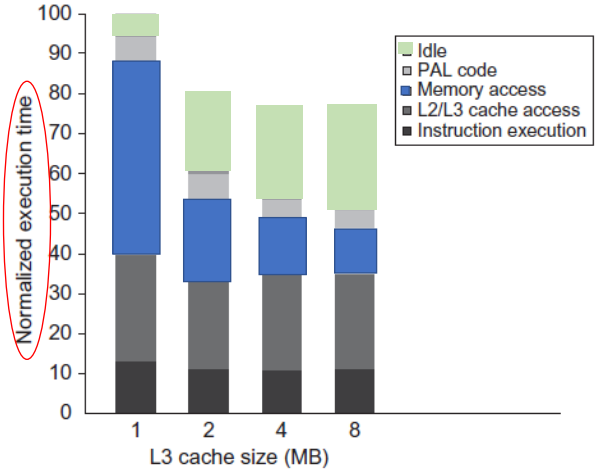
The basic structure of the system, consisting of a four-issue processor and a three-level cache hierarchy, is very similar to the multicore Intel i7.

We focus our attention on the online transaction-processing (OLTP) workload using Oracle 7.3.2 as the underlying database. The workload consists of a set of client processes that generate requests and a set of servers that handle them.

In case of 1Mbyte L3 cache size, 71% of the execution time is spent in user mode, 18% in the operating system (PAL code), and 11% idle, primarily waiting for I/O.

49

Online transaction-processing (OLTP) workload

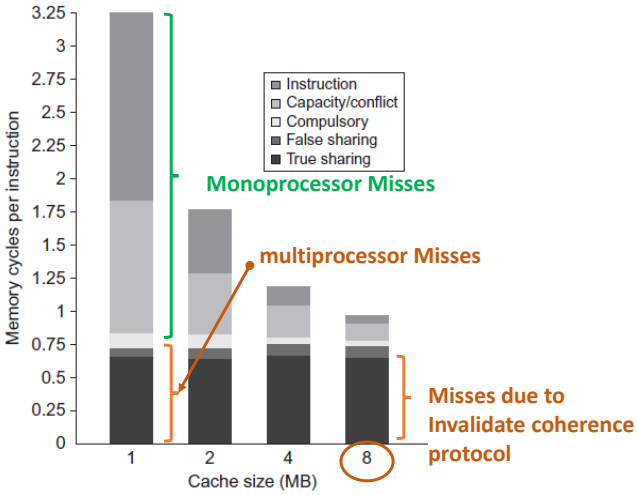


1 to 8 Mbytes per processor
The execution time is improved as the L3 cache grows because of the reduction in L3 misses.
Almost all of the gain occurs in going from 1 to 2 Mbytes (or 4 to 8 Mbytes of total cache for the four processors).
There is little additional gain beyond that, despite the fact that cache misses are still a cause of significant performance loss with 2 Mbytes and 4 Mbytes caches.

The idle time also grows as cache size is increased, reducing some of the performance gains.
This growth occurs because, with fewer memory system stalls, more server processes are needed to cover the I/O latency.
The workload could be retuned to increase the computation/ communication balance, holding the idle time in check.
The PAL code is a set of sequences of specialized OS-level instructions executed in privileged mode; an example is the TLB miss handler.

50

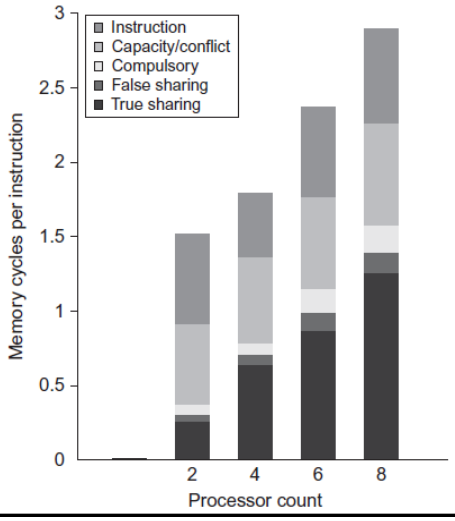
Monoprocessor vs multiprocessor Misses



- The two largest sources of L3 memory access cycles with a 1 Mbytes L3 are instruction and capacity/conflict misses.
  - With a larger L3, these two sources shrink to be minor contributors.
- The compulsory, false sharing, and true sharing misses are unaffected by a larger L3.
  - Thus, at 4 and 8 Mbytes, the true sharing misses generate the dominant fraction of the misses.

51

Memory access cycles increases as processors increases

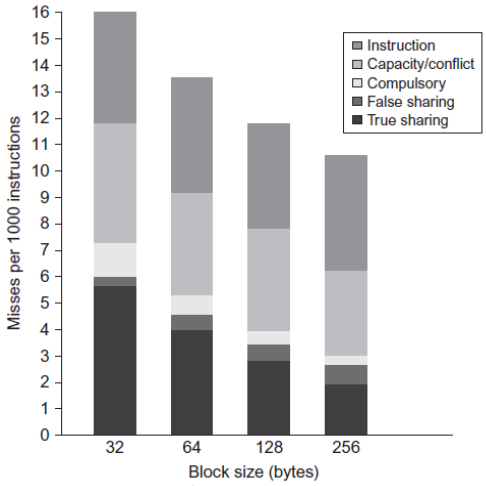


The contribution to memory access cycles increases as processor count increases primarily because of increased true sharing.

The compulsory misses slightly increase because each processor must now handle more compulsory misses.

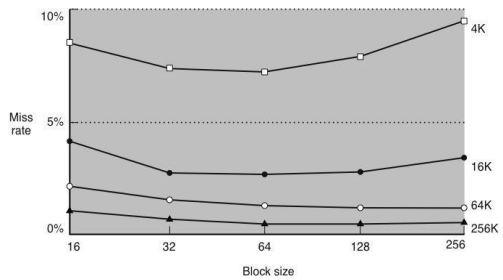
52

Misses and block size



Multiprocessor

- The number of misses per 1000 instructions drops steadily as the block size of the L3 cache is increased, making a good case for an L3 block size of at least 128 bytes. The L3 cache is 2 MiB, two-way set associative.

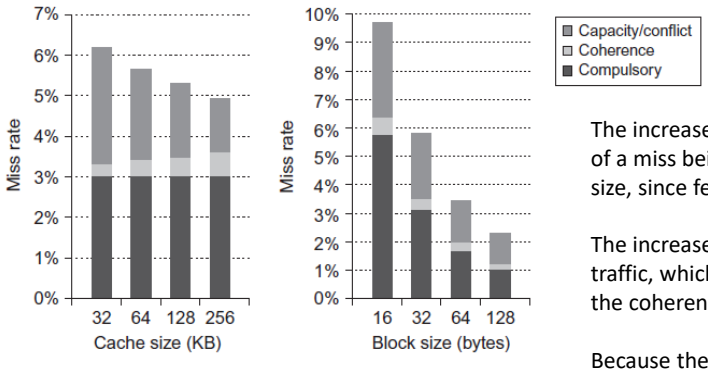


Monoprocessor

53

The components of the kernel data miss rate of the L1 data cache:

when the multiprogramming workload is run on eight processors



The increase in coherence misses occurs because the probability of a miss being caused by an invalidation increases with cache size, since fewer entries are bumped due to capacity.

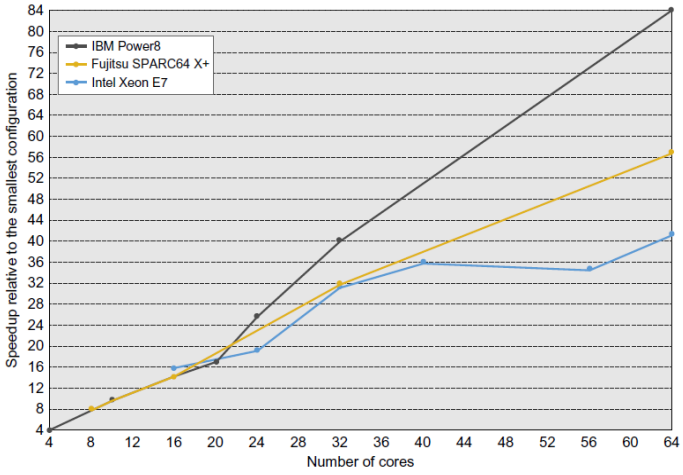
The increased block size has a small reduction in coherence traffic, which appears to stabilize at 64 bytes, with no change in the coherence miss rate in going to 128-byte lines.

Because there are no significant reductions in the coherence miss rate as the block size increases, the fraction of the miss rate caused by coherence grows from about 7% to about 15%.

54

Scalability of distributed shared memory (DSM)

Each core is based on 4 processors



- The performance scaling on the SPECintRate benchmarks for four multicore processors as the number of cores is increased to 64.
- Performance for each processor is plotted relative to the smallest configuration and assuming that configuration had perfect speedup.
- Although this chart shows how a given multiprocessors scales with additional cores, it does not supply any data about performance among processors.
- There are differences in the clock rates, even within a given processor family.

55