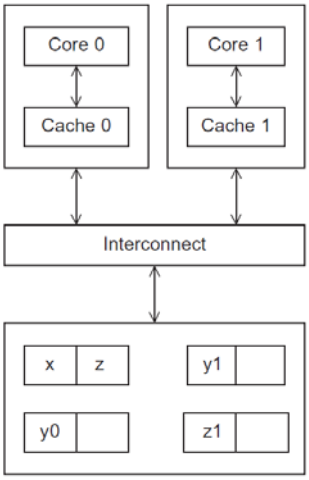
Cache coherence

- Programmers have no control over caches and when they get updated.



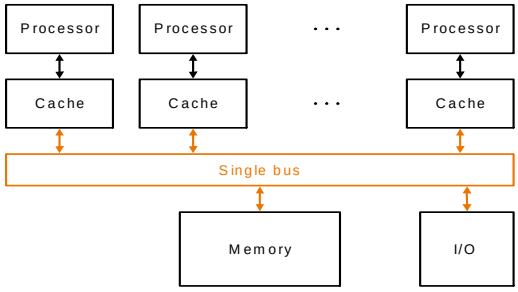
x = 2; /* initially */

Time	Core 0	Core 1
0	`y0 = x;`	`y1 = 3*x;`
1	`x = 7;`	Statement(s) not involving x
2	Statement(s) not involving x	`z1 = 4*x;`

y0 eventually ends up = 2
y1 eventually ends up = 6
z1 = ???

The problem of cache coherence in SMPs

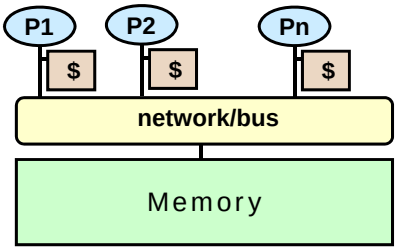
- Different caches may contain different value for the same memory location.



Time	Event	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
0				1
1	CPU A Reads X	X = 1		1
2	CPU B reads X	X = 1	X = 1	1
3	CPU A stores 0 into X	X = 0	X = 1	0 if write through 1 if write back

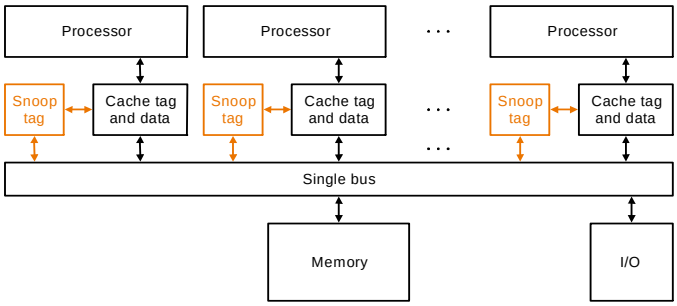
Approaches to cache coherence

- Do not cache shared data
- Do not cache writeable shared data
- Use snoopy caches (if connected by a bus)
- If no shared bus, then
  - Use broadcast to emulate shared bus
  - Use directory-based protocols (to communicate only with concerned parties, not with everybody – directory keeps track of concerned parties)



3

Snooping cache coherence protocols



- Each processor monitors the activity on the bus
- **On a read**, all caches check to see if they have a copy of the requested block. If yes, they may have to supply the data.
- **On a write**, all caches check to see if they have a copy of the data. If yes, they either invalidate the local copy, or update it with the new value.
- Can have either *write back* or *write through* policy (the former is usually more efficient, but harder to maintain coherence).

4

An Example Snoopy Protocol

- Invalidation protocol for write-back caches
- Each block of memory can be:
  - Clean in one or more caches and up-to-date in memory, or
  - Dirty in exactly one cache, or
  - Uncached: not in any caches
- Correspondingly, we record the sate of each block in a cache as:
  - **Invalid** : block contains no valid data,
  - **Shared** : a clean block (can be shared by other caches), or
  - **Exclusive/Modified**: a dirty block (cannot be in any other cache)

MSI protocol = Modified/shared/invalid
Makes sure that if a block is dirty in one cache, it is not valid in any other cache and that a read request gets the most updated data

5

An Example Snoopy Protocol (cont.)

- A read miss to a block in a cache, C1, generates a bus transaction -- if another cache,C2, has the block "exclusively", it has to write back the block before memory supplies it. C1 gets the data from the bus and the block becomes "shared" in both caches.
- A write hit to a shared block in C1 forces a write back -- all other caches that have the block should invalidate that block – the block becomes "exclusive" in C1.
- A write hit to an exclusive block does not generate a write back or change of state.
- A write miss (to an invalid block) in C1 generates a bus transaction
  - If a cache, C2, has the block as "shared", it invalidates it
  - If a cache, C2, has the block in "exclusive", it writes back the block and changes it state in C2 to "invalid".
  - If no cache supplies the block, the memory will supply it.
  - When C1 gets the block, it sets its state to "exclusive"

6

Example

- Assumes that blocks B1 and B2 map to same cache location L.
- Initially neither B1 or B2 is cached
- Block size = one word

Event	In P1's cache	In P2's cache
	L = invalid	L = invalid
P1 writes 10 to B1 (write miss)	L ← B1 = 10 (exclusive)	L = invalid
P1 reads B1 (read hit)	L ← B1 = 10 (exclusive)	L = invalid
P2 reads B1 (read miss)	L ← B1 = 10 (shared)	L ← B1 = 10 (shared)
P2 writes 20 to B1 (write hit)	L = invalid	L ← B1 = 20 (exclusive)
P2 writes 40 to B2 (write miss)	L = invalid	B1 is written back L ← B2 = 40 (exclusive)
P1 reads B1 (read miss)	L ← B1 = 20 (shared)	L ← B2 = 40 (exclusive)

7

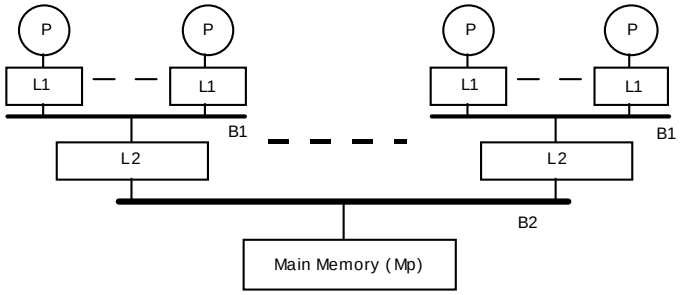
Example (cont.)

Event	In P1's cache	In P2's cache
	L ← B1 = 20 (shared)	L ← B2 = 40 (exclusive)
P1 writes 30 to B1 (write hit)	L ← B1 = 30 (exclusive)	L ← B2 = 40 (exclusive)
P2 writes 50 to B1 (write miss)	L ← invalid	B2 is written back L ← B1 = 50 (exclusive)
P1 reads B1 (read miss)	L ← B1 = 50 (shared)	B1 is written back L ← B1 = 50 (shared)
P2 reads B2 (read miss)	L ← B1 = 50 (shared)	L ← B2 = 40 (shared)
P1 writes 60 to B2 (write miss)	L ← B2 = 50 (exclusive)	L = invalid

Describe the messages that show up on the bus after each event
Message types: read request, write request, write back

8

Hierarchical Snooping

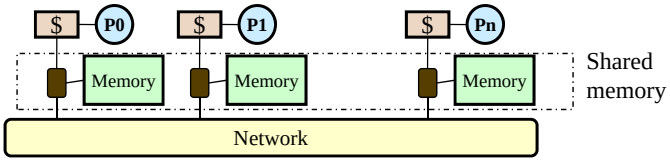


- B1 busses follow standard snooping protocol to keep L1 copies consistent with the content of its shared L2
- The B2 bus keeps copies in L2 consistent with memory
- A change of state of a cache line in L2 can be triggered by a transaction on one of the two busses (B1 or B2) and may require propagation to the other bus.
- Is the inclusion property needed (any block in L1 should also be in L2)??

9

Directory-based Coherence

- Assumes a shared memory space which is physically distributed

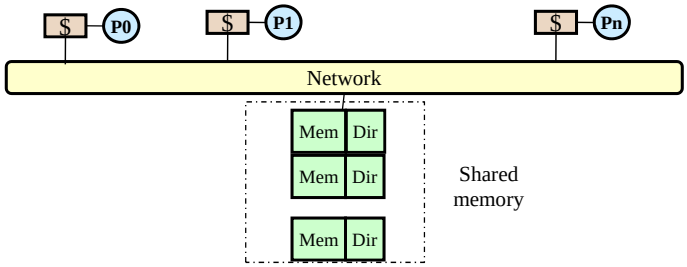


- Idea: Implement a "directory" that keeps track of where each copy of a block is cached and its state in each cache (note that with snooping, the state of a block was kept only in the cache).
- Processors must consult directory before loading blocks from memory to cache
- When a block in memory is updated (written), the directory is consulted to either update or invalidate other cached copies.
- Eliminates the overhead of broadcasting/snooping (bus bandwidth) – Hence, scales up to numbers of processors that would saturate a single bus.
- But is slower in terms of latency

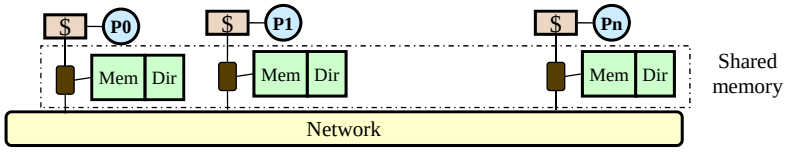
10

Directory-based Coherence

- The memory and the directory can be centralized



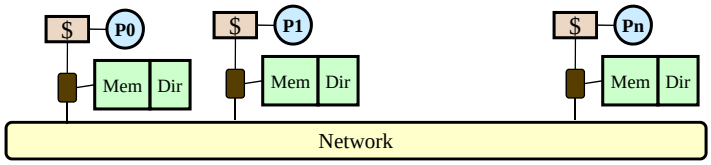
- Or distributed



- Alternatively, the memory may be distributed but the directory can be centralized.
- Or the memory may be centralized but the directory can be distributed (as we will discuss in the case of CMP with private L2 caches)

11

Distributed directory-based coherence



- The location (home) of each memory block is determined by its address.
- A controller decides if access is Local or Remote

- As in snooping caches, the state of every block in every cache is tracked in that cache (exclusive/dirty, shared/clean, invalid) – to avoid the need for write through and unnecessary write back.

- In addition, with each block in memory, a directory entry keeps track of where the block is cached. Accordingly, a block can be in one of the following states:
  - *Uncached:* no processor has it (not valid in any cache)
  - *Shared/clean:* cached in one or more processors and memory is up-to-date
  - *Exclusive/dirty*: one processor (owner) has data; memory out-of-date

12

Enforcing coherence

- Coherence is enforced by exchanging messages between nodes
- Three types of nodes may be involved
  - Local requestor node (L): the node that reads or write the cache block
  - Home node (H): the node that stores the block in its memory -- may be the same as L
  - Remote nodes (R): other nodes that have a cached copy of the requested block.
- When L encounters a **Read Hit,** it just reads the data
- When L encounters a **Read Miss,** it sends a message to the home node, H, of the requested block – three cases may arise:
  - The directory indicates that the block is "not cached"
  - The directory indicates that the block is "shared/clean" and may supply the list of sharers
  - The directory indicates that the block is "exclusive"

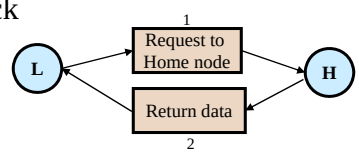
13

What happens on a read miss?

(when block is invalid in local cache)

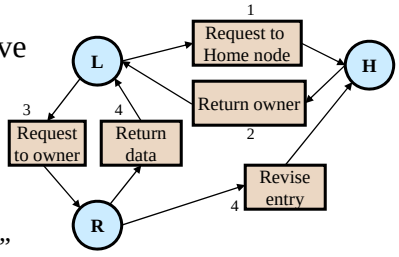
(a)Read miss to a clean or uncached block

-- L sends request to H
-- H sends the block to L
-- state of block is "shared" in directory
-- state of block is "shared" in L



(b) Read miss to a block that is exclusive (in another cache)

-- L sends request to H
-- H informs L about the block owner, R
-- L requests the block from R
-- R send the block to L
-- L and R set the state of block to "shared"
-- R informs H that it should change the state of the block to "shared"



14

What happens on a write miss?

(when block is invalid in local cache)

(a) Write miss to an uncached block

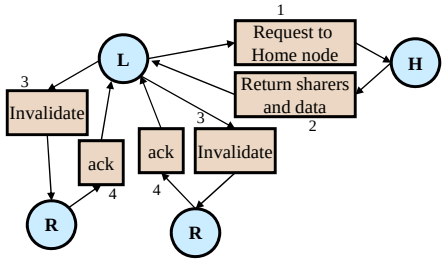
-- similar to a read miss to an uncached block except that the state of the block is set to "exclusive"

(b) Write miss to an block that is exclusive in another cache

-- similar to a read miss to an exclusive block except that the state of the block is set to "exclusive"

(c) Write miss to a shared block

-- L sends request to H
-- H sets the state to "exclusive"
-- H sends the block to L
-- H sends to L the list of other sharers
-- L sets the block's state to "exclusive"
-- L sends invalidating messages to each sharers (R)
-- R sets block's state to "invalid"



15

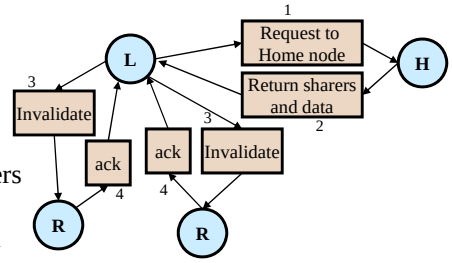
What happens on a write hit?

(when block is shared or exclusive in local cache)

(a) If the block is "exclusive" in L, just write the data

(b) If the block is "shared" in L

-- L sends a request to H to have the block as "exclusive"
-- H sets the state to "exclusive"
-- H informs L of the block's other sharers
-- L sets the block's state to "exclusive"
-- L sends invalidating messages to each sharers (R)
-- R sets block's state to "invalid"

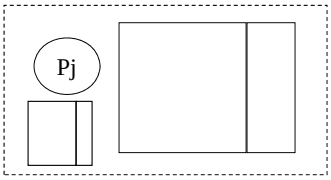


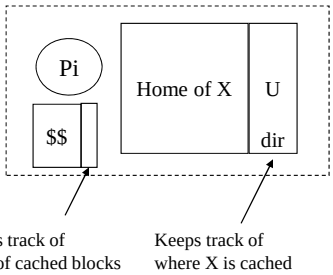
A degree of complexity that we will ignore:
We need a "busy" state to handle simultaneous requests to the same block. For example, if there are two writes to the same block – it has to be serialized.

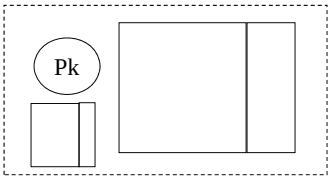
16

Directory-based coherence - example

Case 1:
X is in the *uncached* (U) state in home directory





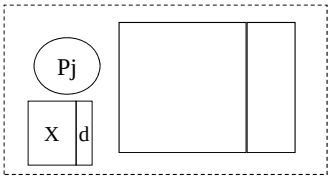


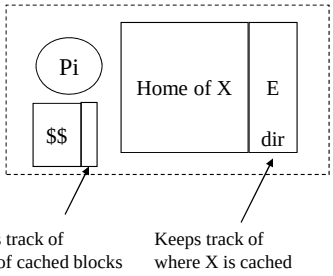
Possible scenario:
- Pj reads X
- Then Pj writes to X

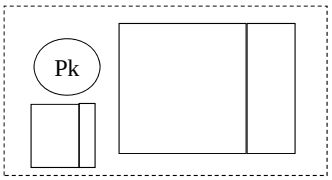
17

Directory-based coherence - example

Case 2:
X is *exclusive* (E) in home directory
and owned by Pj (dirty, d, in Pj)





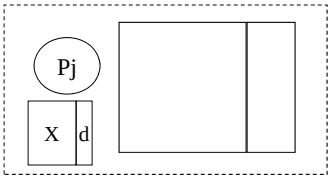


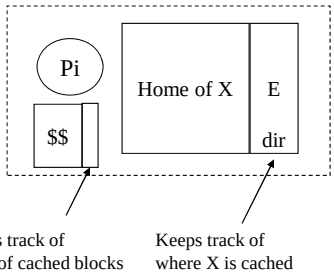
Trace the state of X if:
- Then Pk reads X

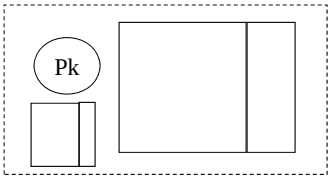
18

Directory-based coherence - example

Case 3:
X is *exclusive* (E) in home directory and owned by Pj (dirty, d, in Pj)





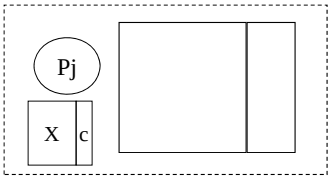


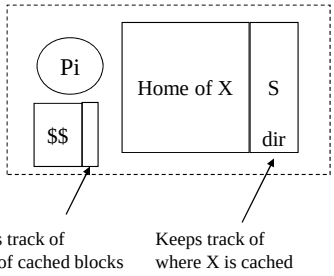
Trace the state of X if:

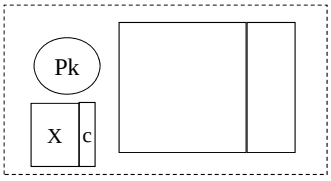
- Pk writes to X

Directory-based coherence - example

Case 4:
X is *shared* (S) in home directory and clean (c) in Pj and PK







Trace the state of X if:

- Pj reads X
- Then Pk writes into X

The MESI protocol

- As described earlier, a block can have three states (specified in the directory)
  - *Invalid or Uncached :* no processor has it (not valid in any cache)
  - *Shared/clean:* cached in one or more processors and memory is up-to-date
  - *Modified/dirty*: one processor (owner) has data; memory out-of-date
- The MESI protocol divides the Exclusive state to two states
  - *Invalid or uncached:* no processor has it (not valid in any cache)
  - *Shared:* cached in more than one processors and memory is up-to-date
  - *Exclusive*: one processor (owner) has data and it is clean
  - Modified: one processor (owner) has data, but it is dirty
- If MESI is implemented using a directory, then the information kept for each block in the directory is the same as the three state protocol:
  - Shared in MESI = shared/clean but more than one sharer
  - Exclusive in MESI = shared/clean but only one sharer
  - Modified in MESI = Exclusive/Modified/dirty
- However, at each cached copy, a distinction is made between shared and exclusive.

21

The MESI protocol

- On a read miss (local block is invalid), load the block and change its state to
  - "exclusive" if it was uncached in memory
  - "shared" if it was already shared, modified or exclusive
    - if it was modified, get the clean copy from the owner
    - if was modified or exclusive, change the copy at the previous owner to "shared"
- On a write miss, load the block and mark it "modified"
  - Invalidate other cached copies (if any)
- On a write hit to a "modified" block, do nothing
- On a write hit to an "exclusive" block change the block to "modified"
- On a write hit to a "shared" block change the block to "modified" and invalidate the other cached copies.
- When a modified block is evicted, write it back .

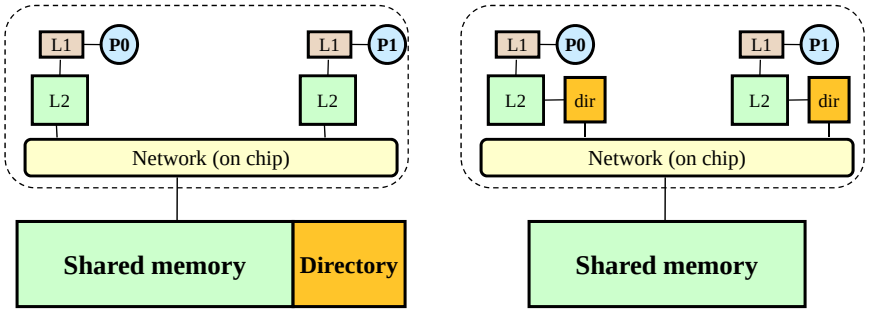
22

The MESI protocol

- If MESI is implemented as a snooping protocol, then the main advantage over the three state protocol is when a read to an uncached block is followed by a write to that block.
  - After the uncached block is read, it is marked “exclusive”
  - Note that, when writing to a shared block, the transaction has to be posted on the bus so that other sharers invalidate their copies.
  - But when writing to an exclusive block, there is no need to post the transaction on the bus
  - Hence, by distinguishing between shared and exclusive states, we can avoid bus transactions when writing on an exclusive block.
  - However, now a cache that has an “exclusive” block has to monitor the bus for any read to that block. Such a read will change the state to “shared”.
- This advantage disappears in a directory protocol since after a write onto an exclusive block , the directory has to be notified to change the state to modified.

23

Example of distributed directories in CMPs

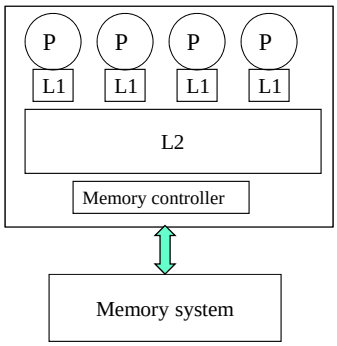


- If **each L2 module is private to the corresponding core**, then on chip directories may be used as replacement for a centralized directory
- Only cache blocks that are on chip need to have directory entries
- Each cache blocks that is on chip is associated with a directory entry.
  - location of directory entry (called its home) is determined by the address.

24

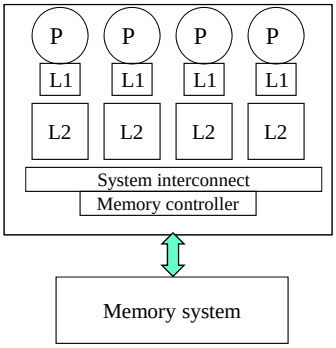
Cache organization in multicore systems

Shared L2 systems



- Examples: Intel Core Duo Pentium
- Uses MESI (Modified, Exclusive, Shared, Invalid) cache coherence protocol to keep the L1 data coherent

Private L2 systems



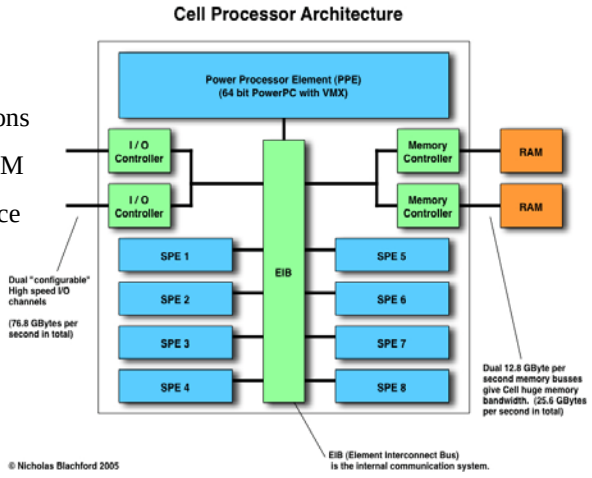
- Examples: AMD Dual Core Opteron
- Uses MOESI (M + Owned + ESI) cache coherence protocol to keep the L2 data coherent (L1 is inclusive to L2)

25

The cell (heterogeneous chip design)

http://www.blachford.info/computer/Cell/Cell2_v2.html

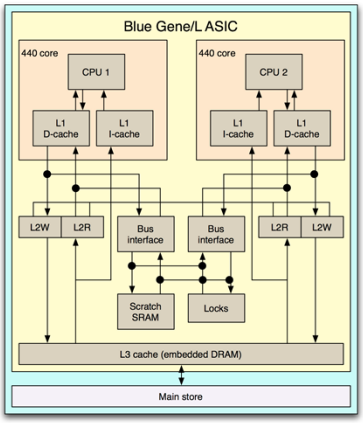
- Target video games
- SPE supports vector operations
- Each SPE has 256KB of RAM
- SPE has its own address space
- DMA transfers between memories

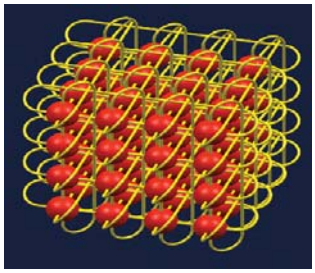


26

The BlueGene (a supercomputer)

http://en.wikipedia.org/wiki/Blue_Gene





- Dual core nodes (power PC)
- 65,536 nodes
- 3D torus interconnection (six ports per node)
- Distributed address space
- Multi-hop message passing
- Tree network for collectives and synchronization

27

PSC's SGI UV ("Blacklight")

- ***Distributed shared memory system***
- ***32 TB shared memory facilitates applications characterized by large memory requirements and complex access patterns. World's largest shared memory machine.***
- ***256 Blades, each with 2 Intel Xeon X7560 eight core processors***
- ***4096 cores***
- ***2.27 GHz -- 37 Tflops***

See architecture details at: http://www.sgi.com/pdfs/4192.pdf

See video at
http://www.youtube.com/watch?v=NhtQTsiFMYI&lr=1&user=sgicorp

28

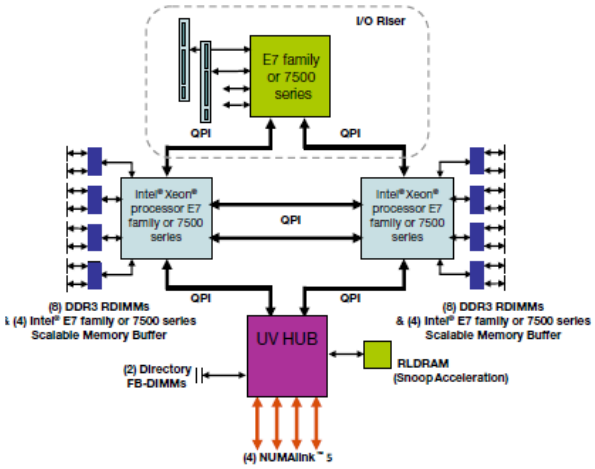


Figure 1. Block-level diagram of SGI UV compute blade.

29

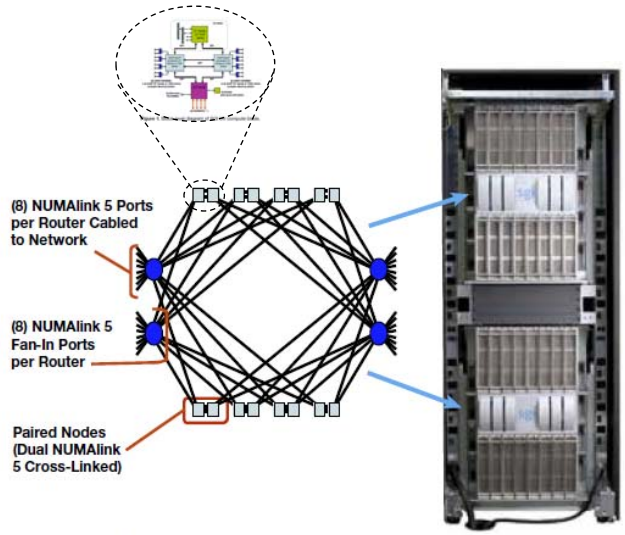


Figure 3a: SGI UV platform blade enclosure, 32 socket interconnect topology, two enclosures per rack.

30

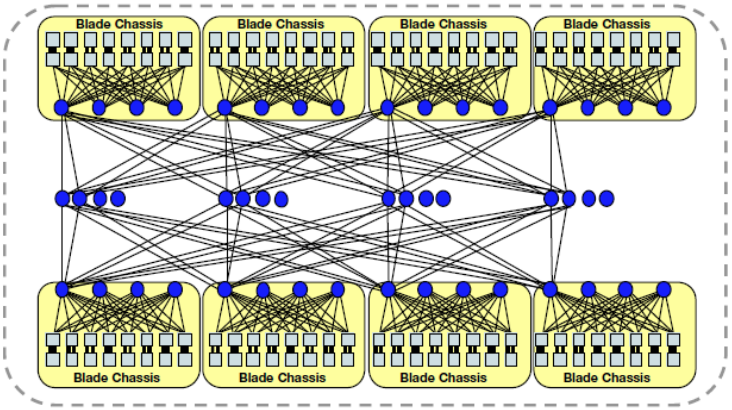


Figure 3b. 256 socket (up to 2,048 core) fat-tree configuration of the SGI UV platform using 16 external 16-port NUMAlink 5 routers (only ¼ of cables are shown). 1,024 socket systems are created with 4x as many blade chassis and 2x as many external routers in the same three-level router topology.

31