Problem 5.2

In the design exploration of a new chip multiprocessor system an important design decision is the choice of cache organization. In Section 5.4.1 the trade-off between shared and private caches are discussed, and in this exercise we will investigate it quantitatively. Assume that a chip multiprocessor has four processor cores. Our options are between using a private or a shared, first-level cache organization, where both of them consume about the same chip resources. The detailed assumptions for each cache organization are given below. The block size for both organizations is 16 bytes.

Private cache organization: Each private cache contains eight block entries, is direct-mapped, and the cache hit time is one cycle. Cache coherence across the private caches is maintained using a simple protocol, according to Section 5.4.2. The time to carry out a snooping action is ten cycles. If a block has to be retrieved from memory it takes 100 cycles.

Shared cache organization: The shared cache organization has many block entries in the private caches. Further, it is portioned into as many banks as the number of processors, and the mapping of memory blocks to banks is round-robin; block address I sis mapped to bank i modulo B, where B is the number of banks. The banks are accessed by the processors using a B x B cross-bar switch (see Chapter 6). To access a cache bank takes two cycles if there is no contention. If a block has to be retrieved from memory it takes 100 cycles.

a) Determine the average memory-access time for each of the organizations in the case when a *single* processor sequentially accesses memory blocks 0 up to 15. Twice in a row. Which organization yields the shortest memory access time, and by how much?
b) Determine the average memory-access time for each of the organizations in the case when *each* processor sequentially accesses memory blocks 0 up to 15 twice in a row. Ignore contention effects. When organization yields the shortest memory access time, and by how much?
c) Number the processors 1, …, 4. Determine the average memory-access time for each of the organizations in the case when processor I sequentially accesses memory blocks 8 x (i-1) up to 8 x (i-1) +7 twice in a row. Ignore contention effects. Which organization yields the shortest memory access time, and by how much?
d) Assume that memory blocks 0 to 7 are initially present in both cache organizations. Now assume that processor 1 modifies all blocks, and processor reads them subsequently. Ignore contention effects. Which organization yields the shortest memory access time, and by how much?
e) Based on your quantitative findings in the previous assignments, under what conditions does a private cache have a performance advantage over a shared cache? Why do many chip multiprocessor designs favor a private first-level organization while using a shared second- or tertiary-level cache?

Answer 5.2:

a) The average memory access time is calculated as follows:

Average access time = Hit time + Miss rate x Miss Penalty

Private cache organization:

The hit time in the private cache is a single cycle and the miss penalty is 100 cycles according to the assumptions. Since the private cache only can host eight blocks and 16 different blocks are accessed twice in a row, all accesses will result in a cache miss (the miss rate is 100%). Therefore, the average memory access time is:

Average = 1 + 100 cycles = 101 cycles

Shared cache organization:

Since there are four processors there are four banks where each bank contains eight blocks. The shared cache can thus host 4x8 = 32 blocks. As a result, the first 16 accesses will miss and the subsequent 16 accesses will hit (the miss rate is 50%). The average memory access time is

Average = 2 + 0.50x100 cycles = 52 cycles

The shared cache is almost twice as fast as the private cache.

b) Private cache organization:

Since each processor accesses 16 different blocks twice in a row, all accesses will miss and the average memory access time will be the same as in a) for the private cache.

Shared cache organization:

One of the processors (say processor 1) will miss on each of the first 16 accesses but will bring the blocks into the shared cache. All other processors (say processor 2, 3, and 4) will hit on each access.

Hence:

Average memory access time for processor 1: (16 x (2 + 100) + 16 x 2)/32 cycles = 52 cycles

Average memory access time for processors 2-4: 2 cycles

Average memory access time = (52 + 3 x 2)/4 = 14.5 cycles

The shared cache is more than 7 times faster.

c) From the assumptions we note that each processor accesses the following blocks:

Processor 1: Blocks 0 through 7 twice

Processor 2: Blocks 8 through 15 twice

Processor 3: Blocks 16 through 23 twice

Processor 4: Blocks 24 through 31 twice

Private cache organization:

Since there are eight blocks in the private cache, the first eight accesses will miss whereas the next eight accesses will hit. Hence:

The average memory access time = (1+100 + 1)/2 cycles = 51 cycles

Shared cache organization:

The exact same miss behavior shows up for the shared cache; the first eight accesses from each processor will miss whereas the next eight accesses will hit. However, the cache hit time is two cycles.

The average memory access time = (2 + 100 + 2)/2 cycles = 52 cycles

d) Private cache organization:

According to the assumptions the eight blocks are already fetched into cache. The eight processor writes result in eight invalidations where each invalidation takes 10 cycles. The subsequent eight processor reads result in coherence misses where each coherence miss takes 10 cycles.

Average memory access time = 1 + 10 cycles = 11 cycles

Shared cache organization:

An advantage of a shared cache is that no cache coherence actions are needed. Each access will take two cycles:

Average memory access time = 2 cycles

e) There was not a single case where the private cache organization outperforms the shared cache. Yet, the disadvantage of the shared cache is its longer cache hit time. Obviously the tradeoff is between the miss rate and hit time. Let's assume that Shared_Hit_Time =KxPrivate_Hit_Time. The private cache organization has an advantage when

Private HT + Private MR x MP < K x Private HT + Shared MR x MP

Private MR x MP/Private HT - Shared MR x MP/Private HT < K - 1

MP (Private MR - Shared MR)/Private HT < K - 1

Assuming that the hit time for the shared cache (Shared HT) is K times that of a private cache and the miss penalty (MP) is 100 times that of the private hit time (Private HT). Then

Private MR - Shared MR < (K-1)/100

K>(Private_MR-Shared_MR)+1

Providing a fast access to a shared cache is not trivial. Also, contention effects which we have not taken into account will make the hit time for the shared cache longer. That's why private first-level caches are the preferred solution whereas having a shared secondary or tertiary cache can provide the benefits shown in this exercise.

Problem 5.3

Consider a shared-memory multiprocessor that consists of eight processors and a private cache associated with each. Cache coherence is maintained by a simple cache protocol according to Section 5.4.2. The bus arbitration mechanism uses a fixed priority based on the identity of the processor/cache unit; when two units with identities i and j simultaneously issue a bus request and i < j, unit i will get access to the bus. However, the bus arbitration mechanism "remembers" the last unit that gained access and that unit cannot get access to the bus in the next bus cycle. Now let each of the eight processors issue a write followed by a read operation to the same location, where each processor writes the same number as its unit identity to that location. What will be returned by the read request issued by each processor?

Table 5.6 Timing and traffic parameters for protocol actions

B is the block size

Request type	Time to carry out protocol action	Traffic
Read hit	1 cycle	N/A
Write hit	1 cycle	N/A
Read request serviced by next level	40 cycles	6 bytes + B
Read request serviced by private cache	20 cycles	6 bytes + B
Read-exclusive request serviced by next level	40 cycles	6 bytes + B
Read-exclusive request serviced by private cache	20 cycles	6 bytes + B
Bus upgrade/update request	10 cycles	10 bytes
Ownership request	10 cycles	6 bytes
Snoop action	5 cycles	N/A

Answer 5.3

Let's number the processors 1,2,...,8. The requests from each processor are issued as follows:

	P1	P2	P3	P4	P5	P6	P7	P8
slot 1	W1	W2	W3	W4	W5	W6	W7	W8
slot 2	R1	R2	R3	R4	R5	R6	R7	R8

The bus arbitration scheme will order the write (Wi) and read (Ri) requests from the processors as follows:

W1, W2, R1, R2, W3, W4, R3, R4, W5, W6, R5, R6, W7, W8, R7, R8.

Since each processor writes a value that corresponds to its identity, the following will be returned: R1 and R2 return 2; R3 and R4 return 4; R5 and R6 return 6; and R7 and R8 return 8.

Problem 5.14

We consider a scalable implementation of a shared-memory multiprocessor using a set of nodes that each contains a processor, a private cache, and a portion of the memory, as shown in Figure 5.19. Cache coherence is maintained using a directory cache protocol, where the directory uses a presence-flag vector associated with each memory block to keep track of which nodes have copies of that block and with the protocol according to Figure 5.20. The time it takes to process a directory request at the home and a remote node is 50 cycles. Further, the latency and traffic of all consistency-induced requests and responses are detailed in Table 5.9, and the block size is 32 bytes.

Table 5.9 Timing and traffic parameters for protocol actions

B is the block size

Request type	**Time to carry out protocol action**	**Traffic**
Read hit	1 cycle	N/A
Write hit	1 cycle	N/A
BusRd	20 cycles	6 bytes
RemRd	20 cycles	6 bytes
RdAck	40 cycles	6 bytes
Flush	100 cycles	6 bytes + B
InvRq	20 cycles	6 bytes
InvAck	20 cycles	6 bytes
UpgrAck	20 cycles	6 bytes

a) Determine the number of cycles needed to handle a cache miss when the home node is the same as the requesting node and the memory copy is clean. Also determine the amount of traffic (in bytes) caused by the coherence transaction.
b) Determine the number of cycles needed to handle a cache miss when the home node is the same as the requesting node and the memory copy is dirty. Also determine the amount of traffic (in bytes) caused by the coherence transaction.
c) Determine the number of cycles needed to handle a cache miss when the home node is different from the requesting node and the memory copy is clean. Also determine the amount of traffic (in bytes) caused by the coherence transaction.
d) Determine the number of cycles needed to handle a cache miss when the home node is different from the requesting node, the memory copy is dirty, and the remote node is the same as the home node. Also determine the amount of traffic (in bytes) caused by the coherence transaction.
e) Determine the number of cycles needed to handle a cache miss when the home node is different from the requesting node, the memory copy is dirty, and the remote node is different from the home node (and of course different from the requesting node). Also determine the amount of traffic (in bytes) caused by the coherence transaction.

Answer 5.14

a) When the home node is the same as the requesting node the miss penalty is the same as the time it takes to process a directory request which is 50 cycles according to the assumptions.

Cache miss time: 1 + 50 = 51 cycles.
Traffic: There is no traffic outside the node.

b) In the case when the memory copy is dirty some other node will have to service the cache miss. The time it takes to do that involves a directory lookup (50 cycles) at the home node, a remote read request (20 cycles), a lookup at the remote node (50 cycles), a flush operation that brings a copy of the block back to the home node, which is the same as the local node (100 cycles), and finally installing the block copy (50 cycles). Hence,

Cache miss time: 1 + 50 + 20 + 50 + 100 + 50 cycles = 271 cycles
Traffic involves sending a remote read request and flushing the block.
Traffic: 6 + 6 + 32 = 44 bytes

c) This scenario involves sending a read request to the home node (BusRd), doing a directory lookup, flushing the block back, and installing it at the requesting node.

Cache miss time: 1 + 20 + 50 + 100+ 50 cycles = 221 cycles
Traffic: 6 + 6 + 32 bytes = 44 bytes

d) This scenario involves sending a read request to the home node, doing a directory lookup, flushing the block back to the requesting node, and installing it there.

Cache miss time: 1 + 20 + 50 + 100 + 50 cycles = 221 cycles
Traffic: 6 + 6 + 32 bytes = 44 bytes

e) This scenario involves sending a read request to the home node (20 cycles), doing a directory lookup (50 cycles), sending a remote read request to the remote node (20 cycles), doing a lookup at the remote node (50 cycles), flushing the block back to the home node (100 cycles), processing it at the directory (50 cycles), flushing the block back to the local node (100 cycles) and installing it there (50 cycles).

Cache miss time: 1 + 20 + 50 + 20 + 50 + 100 + 50 + 100 + 50 cycles = 441 cycles
Traffic: 6 + 6 + 6 + 32 + 6 + 32 bytes = 88 bytes

Problem 7.7

Consider the following reference stream:

r1 w1 r1 w1 r2 w2 r2 w2 r3 w3 r3 w3 r2 w2 w2 r2.

All of the references in the stream are to the same memory location: r, w indicate read or write, respectively, and the digit refers to the processor issuing the reference. We compare four protocols: MSI-invalidate, MSI-update, MESI, and competitive. In MSI and MESI, we introduce a BusUpgrade transaction. This transaction is used when the processor already has a shared (coherent, latest) copy and thus simply needs to invalidate other caches instead of issuing a BusRdX and reloading the block.

Assume that all caches are initially empty, and use the following cost model: read/write cache hit with no bus access – 1 cycle; cache accesses requiring simple transaction on bus (BusUpgrade, BusUpdate) – 20 cycles; misses requiring whole cache block transfer – 150 cycles. The cost of the total bumber of cycles required to execute all accesses in the trace.

Compare the cost of executing the trace on a bus-based machine for each of the four protocols. Explain the differences, given the stream of references and the protocols.

Answer 7.7

Consider the following reference stream:

r1 w1 r1 w1 r2 w2 r2 w2 r3 w3 r3 w3 r2 w2 w2 r2

In the following we designate events/bus transactions as follows:

M: BusRd or BusRdX

H: cache hit

Ud: BusUpd

Ug: BusUpg

1) MSI invalidate protocol

	r1	w1	r1	w1	r2	w2	r2	w2	r3	w3	r3	w3	r2	w2	w2	r2
event	M	H,Ug	H	H	M	H, Ug	H	H	M	H,Ug	H	H	M	H,Ug	H	H
state	S	M	M	M	S	M	M	M	S	M	M	M	S	M	M	M
cycles	150	20	1	1	150	20	1	1	150	20	1	1	150	20	1	1

The total cost of the trace is 689 cycles.

2) MSI update protocol

	r1	w1	r1	w1	r2	w2	r2	w2	r3	w3	r3	w3	r2	w2	w2	r2
event	M	H	H	H	M	H, Ud	H	H,Ud	M	H,Ud	H	H,Ud	H	H,Ud	H, Ud	H
state	M	M	M	M	S	S	S	S	S	S	S	S	S	S	S	S
cycles	150	1	1	1	150	20	1	20	150	20	1	20	1	20	20	1

The total cost of the trace is 557.

3) MESI invalidate protocol

	r1	w1	r1	w1	r2	w2	r2	w2	r3	w3	r3	w3	r2	w2	w2	r2
event	M	H	H	H	M	H, Ug	H	H	M	H,Ug	H	H	M	H,Ug	H	H
state	E	M	M	M	S	M	M	M	S	M	M	M	S	M	M	M
cycles	150	1	1	1	150	20	1	1	150	20	1	1	150	20	1	1

The total cost of the trace is 669.

4) Competitive protocol

	r1	w1	r1	w1	r2	w2	r2	w2	r3	w3	r3	w3	r2	w2	w2	r2
event	M	H,Ud	H	H	M	H,Ud	H	H,Ud	M	H,Ud	H	H,Ud	M	H,Ud	H,Ud	H
state	S0	D	D	D	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0
cycles	150	20	1	1	150	20	1	20	150	20	1	20	150	20	20	1

The total cost of the trace is 745 cycles.
The comparison of the costs is shown in the table below:

Protocol	MSI invalidate	MSI update	MESI invalidate	Competitive
Cost (cycles)	689	557	669	745

In the given reference stream, MSI update has the least cost because it does not need to update other copies when it is the only copy once processor 1 has the block. Also processor 2 does not need to bring the block again after processor 3 modified it because the block in processor 2 is also updated. In case of MESI protocol, it does not need to use BusUpgrade transaction when the block is in E state, while processor 1 send BusUpgrade transaction when it writes in MSI invalidate protocol. In the competitive protocol with the given reference stream, the block in processor 2 is invalidated by w3, and so the block is reloaded by r2. Even though the protocol causes more BusUpdate transactions than other protocols, the updates are not helpful in this case. Hence, it has the highest cost among the four protocols.

Problem 6.1

Consider the mesh interconnection network of Figure 6.2. The network clock frequency is 1 GHz and the phit size is 8 bits. Further, the routing address occupies 2 phits, the payload of a packet is 128 bits, and the envelope is 16 bits.

a) Under the assumption that the sender and receiver overheads are zero and that cut-through switching is used, what is the unloaded end-to-end network latency for sending a packet from the node in the lower-left corner to the node in the upper-right corner?
b) What is the effective bandwidth?
c) Assume that the sender and the receiver overheads are 64 and 128 ns, respectively. What is the effective bandwidth in this case?

Answer 6.1

A packet contains 128 + 16 = 144 bits. With a phit size of 8 bits, it consists of 18 phits of which 2 occupy the routing address.

a) Recall the equation for the unloaded end-to-end latency:

End-to-end packet latency = Sender OH + Time of flight + Transmission time + Routing time + Receiver OH

The packet must traverse 5 links/switches on its way from the source to destination. Under cut-through switching the first two phits must be received at a switch before a routing decision can be taken. The time to buffer them is equivalent to two network cycles which is 2 ns (two 1-GHz clock periods). Therefore, the Time of flight is 2 ns x 5 = 10 ns. An additional cycle is needed for the first phit to reach the destination. Hence, the Time of flight is 11 ns. The rest of the packet (17 phits) will then be delivered with one phit each network clock cycle (1 ns). Hence, the Transmission time is 17 ns as there are 17 additional phits. The routing algorithm is assumed to not cause any overhead so the Routing time is zero. Cut-through routing does not impose any overhead as the packet is pipelined through the switches (with two pipeline stages in each to accommodate the routing address) so the Switching time is zero. Finally, according to the assumptions, the sender and receiver do not impose any overheads. As a result, the unloaded end-to-end latency is 11 + 17 ns = 28 ns.

b) Recall that the Effective bandwidth = Packet size/max(Sender OH, Receiver OH, Transmission time)

The packet size is 128 bits disregarding meta data. The Transmission time is 17 ns. Hence, the Effective Bandwidth is 128/17 Gbits/s = 7.53 GBits/s.

c) With a Sender and Receiver OH of 64 and 128 ns, respectively, the Effective bandwidth will be 128/max(64,128,16) GBits/s = 1 GBit/s.

Problem 6.2

Consider the same network parameters and topology and packet size as in Exercise 6.1, but assume that the switching strategy is store-and-forward.

a) Determine the unloaded end-to-end network latency for sending a packet from the lower-left to the upper-right corner. Assume that the sender and receiver overheads are zero.
b) What is the minimum size buffer required in each switching using store-and-forward switching?
c) What is the effective bandwidth?

Answer 6.2

a) With store-and-forwarding instead of cut-through switching the entire packet has to be buffered at each switch before it is transferred to the next. The number of network cycles to buffer a packet at each switch is equal to the number of phits in a packet, which is 18. With five switches, it will take 5 x 18 ns = 90 ns and then one more cycle until the first phit is received at the destination. The rest of the packet will then arrive at the rate of the network clock frequency with one phit per cycle so the transmission time counted in network cycles equals the number of remaining phits which is 17. As a result, the end-to-end latency is 91 ns + 17ns = 108 ns.

b) Each switch has to accommodate buffer space corresponding to the number of phits which is 18.

c) Assuming that the Sender and Receiver overheads are zero, the Effective bandwidth is 128/17 GBits/s = 7.53 GBits/s.

Problem 6.3

Consider a 16-by-16 torus interconnection network and determine the following interconnection network properties:

a) Network diameter;
b) Bisection bandwidth, assuming that each link has a bandwidth of 100 Mbits/s;
c) The bandwidth per node.

Answer 6.3

a) The network diameter is the longest route (counted in number of links for example) in a particular interconnection network. In a 16-by-16 mesh, the longest route is 15 links in one dimension and then 15 links in the other dimension. In a torus, the end points are connected so this cuts the number of links between the furthest nodes by a factor of two in each dimension. Therefore, the longest route is 16 link traversals in total.
b) If we make a cut across the torus so that it forms two identical network graphs, the cut goes across 32 links; 16 nodes are connected to each other plus the link that connects the end points. With 100 Mbits/s for each link the bisection bandwidth is 3.2 GBits/s.
c) Let's first establish the entire bandwidth across all links. The number of links in an N-by-N mesh is 2N(N-1). Hence, a 16-by-16 mesh has 2 x 16 x 15 = 480 links. For a 16-by-16 torus the number of links that connect the end points is 16+16=32. So in total, there are 480 + 32 = 512 links. The total bandwidth is 512 x 100 Mbits/s = 51.2 Gbits/s. Since the number of nodes is 256, the bandwidth per node is 200 Mbits/s.

Problem 6.4

A design team is contemplating whether to use a non-pipelined (atomic) or a pipelined (split-transaction) bus to connect the private-cache subsystem to the second-level shared cache. They base their analysis on the following data:

- The processors are single-issue and are clocked at 2 GHz;
- The CPI (clock cycles per instruction), disregarding memory stalls, is 1;
- The miss rate per instruction for the private caches is 1%;
- The access time of a second-level cache bank is 20 ns;
- The bus can send an address and return an entire cache block in a single bus cycle;
- The bus cycle time is 2 ns;
- The number of processors is four.

a) What is the bus utilization assuming a non-pipelined bus?
b) What is the bus utilization assuming a pipelined bus?
c) What would be your recommendation based on the available data?

Answer 6.4

a) Assuming that the instruction cache hit rate is 100%, a miss rate per instruction of 1% implies that every hundred instruction is a memory load that misses in the cache. With a single-issue processor clocked at 2 GHz and with an ideal CPI=1 (disregarding memory stalls) it means that every 50 ns each processor will experience a cache miss on average. With 4 processors it means that the time between cache misses is 50ns/4 = 12.5 ns. A bus transaction takes one bus cycle (2ns) to take care of the request, 20 ns to have the second-level cache bank to respond to the miss request, and another bus cycle to return the requested cache block (2ns). So in total it takes 24 ns to carry out a miss request and during this time the bus is busy. This will make the time between consecutive misses longer by the same amount. So the time between misses is 50ns + 24 ns = 74 ns. With four processors the time between misses is 74 ns/4 = 18.5 ns. Since 18.5 ns < 24 ns, the non-pipelined bus is fully utilized (100%).
b) With a pipelined split-transaction bus, the bus can be released once the request is buffered at the second-level cache bank. So the bus will only be busy when the request is transferred and when the block is sent back. These two operations only occupy the bus for 2ns + 2ns = 4ns. Since there is a cache miss request every 18.5 ns (see Problem 5.4 a) the bus utilization is only 4/18.5 = 22%.
c) Based on the analysis in a) and b) it is quite clear that the non-pipelined bus is not an option whereas the pipelined bus can accommodate the bandwidth demands of the four processors quite well. Note, however, that this analysis is simplistic and does not take into account queuing delays.

Problem 6.5

A design team is contemplating what switch degree to use when designing a multi-stage interconnection network that connects 64 nodes of one kind to 64 nodes of another kind. The crossbar switches to be used range from 2-by-2 to 64-by-64 switches. The latency times for the different switches are shown in Table 6.2.

Table 6.2 Latency times for switches as a function of switch degree

Switch degree	Switch cycle time [ns]
2	12
4	15
8	30
64	70

a) Determine the switch degree that will minimize the unloaded end-to-end time of flight.
b) Construct a MIN with the switch degree determined in (a) using a shuffle-exchange interconnection pattern.
c) Highlight two routes that will use the same link or switch resources.

Answer 6.5

a) Let's first derive the end-to-end latency as a function of the switch degree. Given a switch degree of k, the number of stages to connect N nodes of one kind to N nodes of another kind is logk N. In the table below, the number of stages for a given switch degree is derived. The end-to-end-latency is then derived by multiplying the switch-cycle time with number of stages. For the given assumptions, the minimum end-to-end latency is achieved at a switch degree of 4.

Switch degree	Switch-cycle time [ns]	Number of stages	End-to-end latency [ns]
2	12	6	72
4	15	3	45
8	30	2	60
64	70	1	70

b) See Figure 6.6 using 2-by-2 switches. There will be three stages and 64/4 = 16 switches in each stage.
c) By examining Figure 6.6 highlighting two routes that use the same resources in a MIN with 2- by-2 switches it is possible to identify two routes that use the same resources for a MIN using 4-by-4 switches.

Problem 7.3

This problem is about implementing various synchronization primitives using LL (load-linked) and SC (store conditional). The instructions are:

```
LL Rx, A/load mem location A in Rx;
SC Rx, A/conditionally store (Rx) into mem location A
```

The store is conditional because if the processor has received an invalidation or an update for X from the coherence protocol between the LL and the SC, then the store is aborted and the value returned in Rx is 0 (note that the value to store should never be 0 when the lock succeeds). The enforcement of this condition can be implemented with a few registers in the snooper or the network interface, which keep the addresses of LLs and are checked whenever a SC is executed. If an invalidation or an update is received by the interface for the address, SC will fail.

Between LL and SC, instructions can be inserted provided they don't store to memory and they can't trigger unexpected or expected exceptions. For example, if an instruction can have an exception, but the programmer knows that the exception will never happen, then it is ok to use it. A context switch between the LL and SC also causes the SC to fail. Any store must be conditional.

How to implement a test and set with these instructions was discussed in this chapter. In this problem we want to implement other synchronization primitives with LL and SC (even some that do not exist).

a) Show the code for F&ADD X, Rx, a, where a is a small immediate constant added to X.
b) Show the code for F&ADD X, Rx, Ry, where Ry is added to X and Rx returns the value of X before the ADD.
c) Show the code for F&ADD X, Rx, Y, where memory location Y is added to memory location X and the value of X before the ADD is returned in Rx.
d) Show the code for SWAP (Rx, X), whre the values in Rx and X are swapped.
e) Show the code for CAS Rx, Ry, X, where the values in Rx is first compared to the value of X and, if they are equal, the values in Ry and X are swapped.
f) A programmer would like to implement an entire critical section as an atomic operation using LL and SC (instead of using locks). The critical section in one thread is as follows:

```
A += A;
If (A==8) B=B+1;
      Else B=0;
```

A and B are shared writable variables. Assume that the programmer can write the code directly in assembly code. Is there a way to code this critical section with LL and SC? If so, show the code. If not explain why.

Answer 7.3

a) Code for F&A X, Rx, a, where a is a small immediate constant added to X.

```
F&A(X,Rx,a) LL Rx,X
ADDI R1,Rx,a
SC R1,X
BEQZ R1,F&A
Return
```

b) Code for F&A X,Rx,Ry, where Ry is added to X and Rx returns the value of X before the ADD.

```
F&A(X,Rx,Ry)LL Rx,X
ADD R1,Rx,Ry
SC R1,X
BEQZ R1,F&A
Return
```

c) Code for F&A X,Rx,Y, where memory location Y is added to memory location X (X!=Y) and the value of X before the ADD is returned in Rx

```
F&A(X,Rx,Y) LW R1,Y
wait: LL Rx,X
ADD R1,Rx,R1
SC R1,X
BEQZ R1,wait
Return
```

d) Code for SWAP(X,Rx), where the values in Rx and X are swapped

```
SWAP(X,Rx) ADD R2,Rx,R0/save Rx
wait: LL R1,X
SC R2,X /cannot lose Rx
BEQZ R2,wait
ADD Rx,R1,R0
Return
```

e) Code for CAS Rx, Ry, X, where the value in Rx is first compared to the value of X and if they are equal the values in Ry and X are swapped.

```
CAS(Rx,Ry,X) LL R1,X
ADD R2,R1,R0/save X
BNE Rx,R1,exit
ADD R1,Ry,R0
exit: SC R1,X /store old X or Ry
```

```
BEQZ R1,CAS
ADD Ry,R2,R0
Return
```

f) The major problem is that there are two memory stores in this code: one to A and one to B. All stores must be conditional while they are subject to conflicts. In this code the programmer wants the whole code to be a critical section and executed atomically, i.e. B should not be updated if A is updated. Conversely, A should not be updated if B is updated during the if statement. This is very hard (if not impossible) to do with the way LL and SC are defined. Even if an SC monitors the specific address and we execute the two SC's in sequence at the end of the code, an update could still reach the processor for one variable after the SC for the first variable has been already been successful and has updated memory. Too late! It would work if the two SCs could be executed as a critical section.

Problem 8.2

For this problem we use the dependence graph shown in Figure 8.4 with the following modification: we assume that X5 is a cache hit and hence always takes only one cycle to compute.

a) Redo the speculative schedule using interleaved multi-threading and block multi-threading (similar to Table 8.1, and all assumptions remain the same as for generating Table 8.1). Note that when X5 is a cache hit block multi-threading is identical to a single-threaded core, where X executes first and then Y executes.
b) How much faster (in clock cycles) is interleaved multi-threading over block multi-threading in this case?
c) Interleaved multi-threading requires more hardware support for selective flushing and using thread ID for tag matching dependencies. Due to the design complexity, the clock cycle of the inter-leaved multi-threading processor is 20% slower. Does it still makes sense to use interleaved multi-threading for the modified threaded code in Figure 8.4, where X5 is a cache hit?

Answer 8.2

Execution schedule for block (coarse grain) multithreading:

Instruction (latency)	Dispatch (issue Q)	Issue	Register fetch	Exec start	Exec complete	CDB	Retire (T1)	Retire (T2)
X1(2)	1(1)	2	3	4	5	6	7	
X2(2)	1(2)	4	5	6	7	8	9	
X3(4)	2(3)	4	5	6	9	10	11	
X4(2)	2(4)	8	9	10	11	12	13	
X5(1,20)	9(3)	10	11	12	12	13	14	
X6(1)	9(4)	11	12	13	13	14	15	
Y1(2)	15(1)	16	17	18	19	20		21
Y2(3)	15(2)	18	19	20	22	23		24
Y3(2)	16(3)	21	22	23	24	25		26
Y4(2)	16(4)	23	24	25	26	27		28
Y5(3)	24(3)	25	26	27	29	30		31
Y6(1)	24(4)	28	29	30	30	31		32

Execution schedule for interleaved (fine grain) multithreading:

Instruction (latency)	Dispatch	Issue	Register fetch	Exec start	Exec complete	CDB	Retire (T1)	Retire (T2)
X1(2)	1(1)	2	3	4	5	6	7	
X2(2)	1(2)	4	5	6	7	8	9	
Y1(2)	2(3)	3	4	5	6	7		8
Y2(3)	2(4)	5	6	7	9	10		11
X3(4)	8(3)	9	10	11	14	15	16	
X4(2)	8(4)	13	14	15	16	17	18	
Y3(2)	11(3)	12	13	14	15	16		17
Y4(2)	11(4)	14	15	16	17	18		19
X5(1,20)	17(3)	18	19	**20**	20	21	22	
X6(1)	17(4)	19	20	21	21	22	23	
Y5(3)	19(3)	20	21	22	24	25		26
Y6(1)	19(4)	23	24	25	25	26		27

Looking at the issue, execution start and retire stages, interleaved multithreading is five cycles faster than block multithreading. It the clock rate for interleaved MT processor is 20% slower, the time taken by interleaved MT is 27*1.2=32.4 and is still faster than block MT (33).

Problem 8.3

A system architect has three choices for an on-chip interconnection network: a uni-directional ring, a bi-directional ring, and an N x N mesh network. In addition, the architect has two choices for providing coherence between L1 caches: snoop-based or directory-based. There are 64 cores on the chip. All cores have a private L1 cache, but they all share a large L2 cache that is divided into 64 banks, where one bank is attached to each core. The latency to communicate between two cores connected directly by a link is one cycle. The router latency is one cycle per each router that a packet goes through. For directory-based coherence, the access time to the directory is 16 cycles. Assume that there is no contention on any of the wires or to access the directory.

a) Which combination of design choices provides the shortest latency to provide coherence?
b) Now consider a future many-core CMP with 1024 cores on-chip. The latency for directory access increases to 20 cycles. In this case, which combination of design choices provides the shortest latency?

Answer 8.3

Interconnection networks available are uni-directional ring (UR), bi-directional rings (BR) and N*N mesh (NM). Coherence protocols supported are snoop-based (SB) and directory-based (DB). A total of 64 cores are available and router latency is 1 cycle. A directory access takes 16 cycles. To compare let's take both the worst case and the average case latencies. We make the following assumptions: We assume uniform traffic in the network for coherence messages. The latency is only calculated up to the point where the message reaches the target core and proceeds without receiving an acknowledgment. Contention in the network and at the directory is ignored. In SB, each coherence message is sent to all cores, whereas in DB a coherence message is only sent to ONE core and, with the help of directories a requesting core finds out where to send that message. The latencies are given in Table.

		Interconnection network type		
Latency	Coherence type	UR	BR	NM
Worst case	SB	63	32	14
Worst case	DB	16+63	16+32	16+14
Average	SB	63	32	7
Average	DB	16+32	16+32/2	16+7

Problem 8.4

The choice between using locks and transactional memory (TM) semantics is not always obvious, particularly given the hardware complexity of implementing the TM semantics. This exercise illustrates these trade-offs. Consider that two threads T1 and T2 are executing two pieces of code that may need mutual exclusion, since the two threads conditionally modify a shared variable. Each thread executes ten instructions in their respective critical section. If the developer uses locks, the code can run on machine A. If the developer uses transactions, the code can run on machine B. Both A and B are multi-threaded machines that allow T1 and T2 to be executed concurrently. Both machine A and machine B can execute the critical section codes with a CPI of 1. The only difference between the two machines is that machine B's cycle time is 20%longer than that of machine A as it has to support TM semantics. If a transaction fails, there is an additional penalty of 20 cycles to clear the transactional state.

a) If the probability that the two threads concurrently read/modify the shared variable 10% of the time (i.e., 90% of the time there is no sharing), which machine is faster?
b) As the size of the critical section grows, the probability that there is a conflict increases. Hence, if the number of instructions in the critical section increases from 10 to 1000, the probability of contention increases form 10% to 20%. In such a scenario, which machine is faster?

Answer 8.4

T1 and T2 are concurrent threads. machine A has lock support and machine B has TM support. Machine B's clock is 20% longer and the penalty for an aborted transaction is 20 cycles. CPI=1. Assume a lazy TM so that a conflict is detected at the end of a transaction.

a) Each thread has 10 instructions. The probability of a conflict is 10%. In the lock-based system, two transactions take 20 cycles. In the TM system the time to execute two transactions is: 1.2*(10*0.9+0.1*(10+30))=15.6 cycles. Thus TM is better in this case.
b) Now each thread has 1000 instruction. The probability of a conflict is 20%. In the lock-based system two transactions take 2000 cycles. In the TM system the two transactions take: 1.2*(1000*0.8+0.2*(1000+1020))=1444.8 cycles. TM is much better in this case.

Problem 8.5

Consider a future 16-way CMP operating in a power-constrained environment. The CMP can automatically configure itself to run as a 1-, 2-, 4-, 8-, 16-way core CMP but always using a fixed power budget. For instance, it can run as a single-core processor by grabbing the power from the other 15 cores by putting them to sleep and using the additional power to increase its frequency. Assume that sleep and wakeup times are zero, and that power and frequency have a square relationship. For instance, if one core uses the power of all 16 cores, its frequency can increase four-fold. We call this an EPI-throttled CMP.

Consider a partially parallel application. This application starts as a single-threaded application and spends 5% of the time in sequential mode. During the following 40% of the time, the application has 16 threads, and only four threads for the next following 40% of the execution time. During the remaining execution time, the application has only one thread.

a) What is the speedup of this application when it runs on this future CMP compared to running on a single-core machine that uses the same power but operates at a higher frequency using the square relationship?
b) What is the speed of this application when it runs on this future CMP compared to running on a traditional 16-way CMP (again using the same power budget) that does not provide reconfiguration capability?
c) Due to limitations of voltage scaling, assume that in the future power depends linearly on frequency. In this case comment on what benefits (if any) an EPI-throttled CMP provides.

Answer 8.5

Let's use 1 clock cycle of a core in a 16-way CMP as the baseline clock. Then the clock period of a single core that uses the same power as a 16-core CMP is 1/4 and the clock period of a single core that uses the same power as a 4-core CMP is 1/2 (based on the square relationship).

In the case of the EPI-throttled CMP the time is

((5+15)*1/4+40+40*1/2)=65 cycles.

a) For a single core the time is:
   1/4*((5+15)+16*40+4*40)=205 cycles
   The speedup for the EPI-throttled core is 205/65=3.15
b) For a traditional 16-way CMP the time is:
   1*(5+40+40+15)=100
   The speedup for the EPI-throttled core is 100/65=1.53
c) Here power is proportional to frequency. Hence when all cores in a 16-core CMP are active the cycle time is 1. When only one core is active that consumes the same power as the 16-core CMP then its cycle time is 1/16th of the cycle time of 16 active cores in a CMP.
   For the EPI-throttled CMP the time is:
   ((5+15)*1/16+40+40*1/4)=51.25
   For the single core, the time is:
   1/16*(5+16*40+4*40+15)=51.25. Speedup is 1.
   For the traditional 16-way CMP the time is:
   1*(5+40+40+15)=100. Speedup is 1.95

Problem 8.8

a) Consider a simple 5-stage pipeline that is single-threaded. The pipeline treats every cache miss as a hazard and freezes the pipeline. While executing a benchmark assume that a L1 cache miss occurs every 100 cycles, and that each L1 cache misses takes 10 cycles to satisfy if the block is found in L2 or 50 cycles if L2 misses as well. A L2 cache miss occurs after 200 cycles of computation. Assume that the CPI in the absence of cache misses is 1. What is the actual CPI, taking into account cache miss latencies?

b) Consider the same example as in (a), but assume that hardware is now two-way multi-threaded, similar to Figure 8.3. Assume that switching overhead is zero, and that there are two threads with identical cache miss behavior as described in the first case. What is the CPI of each of the two programs on the two-way multi-threaded machine? Did the CPI improve? If yes, explain how. If not, explain why one should bother with the two-way multi-threaded machine?

c) Consider the case in (a), but the switching overhead is five cycles. Again compute the CPI of each thread, and explain why it increases, decreases, or stays the same.

d) Consider the case for which the L2 miss latency jumps from 50 to 500 cycles and the switching overhead jumps from 5 to 50 cycles. Compute the CPI in this machine.

Answer 8.8

For simplicity assume that each thread executes 200 instructions.

a) The CPI is (200+10+50)/200 = 1.3

b) Cache access latencies are hidden because of the total overlap of thread executions with cache misses. The CPI improves to 1.

c) With a switching overhead of 5 cycles the CPI is now:
(100+5+100+5)/200 = 1.05.
This increase reflects the overhead of switching threads.

d) CPI(T1) = (100+50+100+500)/200 = 3.75
CPI(T2) = (100+50+100+500)/200 = 3.75
The CPI of the overall machine is 3.75 as well.