(a) How many cycles does it take to execute the access sequence under MSI vs. MOESI, assuming the access costs for the protocol transactions to be as in Table 5.6?

(b) Compare the traffic generated by the MSI and MOESI protocols counted in bytes transferred using the data in Table 5.6, and assuming that B is 32 bytes.

(c) Compare the access cost and traffic of MESI in Problem 5.5 with your findings for MOESI in this exercise. What makes the MOESI protocol beneficial and what would remove the performance advantage of MOESI over MESI?

5.7 Assume a shared-memory multiprocessor with a number of processor/private cache units connected by a shared single-transaction bus. Our baseline cache coherence protocol is an MESI protocol, but we want to investigate what performance gain can be achieved with adding *read snarfing* to it according to Section 5.4.4. We want to determine the time and traffic under the execution of a sequence of accesses with a MESI w/o read snarfing with a MESI protocol plus read snarfing by using the parameters in Table 5.6. Consider the following sequence of accesses by the processors:

R1/X, R2/X, R3/X, R4/X, W2/X, R1/X, R3/X, W3/X, R1/1X, R2/X.

(a) How many cycles does it take to execute the access sequence under MESI with and without read snarfing, assuming the access costs for the protocol transactions to be as in Table 5.6?

(b) Compare the traffic generated by the MESI protocol with and without read snarfing using the data in Table 5.6, and assuming that B is 32 bytes.

5.8 Assume a shared-memory multiprocessor with a number of processor/private cache units connected by a shared single-transaction bus. Our baseline cache coherence protocol is a MESI protocol, but we want to investigate what performance gain can be achieved by using an update-based coherence protocol according to Section 5.4.4. We want to determine the time and traffic under the execution of a sequence of accesses with a MESI and with an update-based protocol by using the parameters in Table 5.6. Consider the following sequence of accesses by the processors:

R1/X, R2/X, R3/X, R4/X, W1/X, R2/X, R3/X, W2/X, R1/X, R3/X.

(a) How many cycles does it take to execute the access sequence under the invalidation-based MESI protocol and under the update-based protocol, assuming the access costs for the protocol transactions to be as in Table 5.6?

(b) Compare the traffic generated by the invalidation-based MESI protocol and the update-based protocol using the data in Table 5.6, and assuming that B is 32 bytes.

5.9 Assume a shared-memory multiprocessor with a number of processor/private cache units connected by a shared single-transaction bus. Our baseline cache coherence protocol is a MESI protocol, but we want to investigate what performance gain can be achieved by adding migratory sharing detection/optimization to it according to Section 5.4.4. We want to determine the time and traffic under the execution of a sequence of accesses with a MESI with and without migratory sharing detection/optimization by using the parameters in Table 5.6. Consider the following sequence of accesses by the processors, and assume that the migratory optimization is disabled from the beginning:

R1/X, W1/X, R2/X, W2/X, R3/X, W3/X, R4/X, W4/X.

(a) How many cycles does it take to execute the access sequence under the MESI protocol with and without migratory detection/optimization, assuming the access costs for the protocol transactions to be as in Table 5.6?

(b) Compare the traffic generated by the MESI protocol with and without migratory detection/optimization using the data in Table 5.6, and assuming that B is 32 bytes.

5.10 As we have seen in Section 5.4.5, the stable states (for example, states M, S, and I in an MSI protocol) are not enough to resolve data races. The transient states added to the MSI protocol in Figure 5.15 deal with state transitions from state S to state M. There are, however, other data races that can occur. Add the transient states and the accompanying state transitions to deal correctly with transitions from state I to state M.

5.11 Consider a shared-memory multiprocessor that consists of three processor/cache units and where cache coherence is maintained by an MSI protocol. Table 5.7 shows the access sequence taken by three processors to the same block but to different variables (A, B, C) in that block.

Table 5.7

	Processor 1	Processor 2	Processor 3
1	R_A		
2		R_B	
3			R_C
4	W_A		
5			R_C
6		R_A	
7	W_B		
8			R_A
9			R_B

(a) Classify the misses with respect to cold, true sharing, and false sharing misses.

(b) Which of the misses could be ignored and still guarantee that the execution is correct?

(c) Determine the fraction of essential traffic resulting from the access sequence using the parameters in Table 5.6, and assuming that the block size is 32 bytes.

5.12 Consider a shared-memory multiprocessor that consists of three processor/cache units and where cache coherence is maintained by an MSI protocol. The private caches are direct-mapped. Table 5.8 shows the access sequence taken by three processors to four variables (A, B, C, and D), where A, B, and C belong to the same block and D belongs to a different block. The two blocks map to the same entry in the caches, and the cache is full initially.

Table 5.8

	Processor 1	Processor 2	Processor 3
1	R_A		
2		R_B	
3			R_C
4	W_A		
5			R_D
6		R_B	
7	W_B		
8			R_C
9		R_B	

(a) Classify the misses with respect to cold, replacement, true sharing, and false sharing misses.

(b) Which of the misses could be ignored and still guarantee that the execution is correct?

5.13 Section 5.4.7 discusses the important notion of translation lookaside buffer (TLB) consistency. We want to determine the time it takes to make all TLBs in a shared-memory multiprocessor system consistent when the virtual-to-physical page mapping has changed. Assume that there are four TLBs that have a mapping to the page. Further, assume that it takes 1000 cycles to invoke the page-fault handler, 100 cycles to send an interprocessor interrupt, 200 cycles to invoke a software handler on each processor to shoot down a TLB entry, 20 cycles to invalidate possible block entries for the physical page in each cache, and 100 cycles for each processor to send back an acknowledgment to the processor that executes the page-fault handler to notify it that the TLB entries and all its traces in the caches are removed. How long does it take to carry out the TLB shootdown operation?

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.

5.15 Now consider instead the directory-based protocol of Figure 5.21, but use the latency and traffic parameters from Table 5.9 and assume that the block size is 32 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.

(f) In what cases do the latency and traffic results differ from those in Exercise 5.14?.

5.16 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 protocol according to Figure 5.20, where the directory uses a presence-flag vector associated with each memory block to keep track of which nodes have copies of that block. 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.

(a) Determine the number of cycles needed to handle the coherence transaction resulting from a write to a block in state Shared when the home node is the same as the requesting node and the memory copy is the only copy. Also determine the amount of traffic (in bytes) caused by the coherence transaction.

(b) Determine the number of cycles needed to handle the coherence transaction resulting from a write to a block in state Shared when the home node is the same as the requesting node and the number of sharers is four. Also determine the amount of traffic (in bytes) caused by the coherence transaction.

(c) Determine the number of cycles needed to handle the coherence transaction resulting from a write to a block in state Shared when the home node is different from the requesting node and the memory copy is the only copy. Also determine the amount of traffic (in bytes) caused by the coherence transaction.

(d) Determine the number of cycles needed to handle the coherence transaction resulting from a write to a block in state Invalid 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 the coherence transaction resulting from a write to a block in state Invalid 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. Also determine the amount of traffic (in bytes) caused by the coherence transaction.

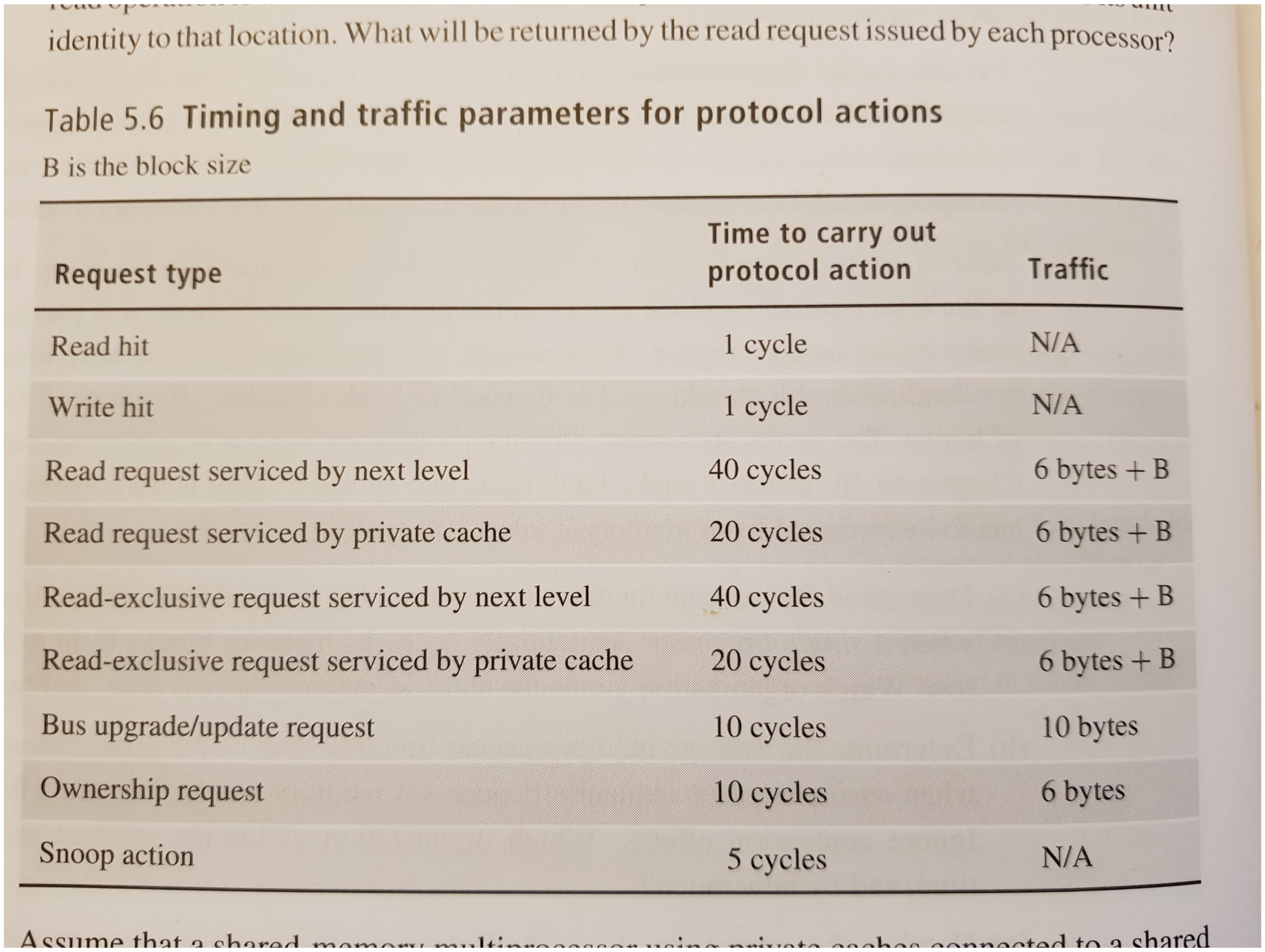
5.17 In the design exploration of a new chip-multiprocessor system that is designed to use 128 processors, the architecture team has decided to use a hierarchical organization according to Figure 5.23. Each cluster consists of eight processors with their private caches. Each cluster also has a shared second-level cache. Different alternatives to maintain coherence inside and across the clusters are contemplated. The block size is 32 bytes. Determine the overhead in maintaining directory information, regarding the second-level cache as well as memory for the following alternatives:

(a) a presence-flag-based directory cache protocol inside each cluster and across clusters;

(b) a limited-pointer directory cache protocol with two pointers inside each cluster and a coarse-vector directory protocol that partitions the clusters into groups of four clusters in each across clusters;

(c) a cache-centric directory cache protocol according to Figure 5.22 inside each cluster and a limited-pointer scheme with four pointers across clusters.

5.18 Consider a cache-centric directory cache protocol according to Figure 5.22. We want to determine the time it takes to carry out an invalidation if N copies are maintained. Assume that a request and a response take K cycles. What would be the number of cycles under the same assumptions for a presence-flag-based directory cache protocol?



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

Assume that a shared memory multiprocessor using private caches connected to a shared

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?

8.3 A system architect has three choices for an on-chip interconnection network: a uni-directional ring, a bi-directional ring, and an $N \times N$ mesh network. In addition, the