Name: **Seat Number:** **Sciper:**

CS-307 Midterm Exam

Please do not turn this page until instructed to do so.

Please write your seat number at the top of each page.

You have **150 minutes** in total to answer all questions.

Please write clearly and concisely using a **blue** or **black** pen. Show all work for full credit.

There is an empty sheet at the end that you can use as scratch paper. Please use that first and only ask for extra sheets if you need more.

Total number of pages: **18**

Problem	**Points**
Cache Hierarchies & Cache Coherence [21 points]	
Parallel Computing & SW Optimizations [23 points]	
Cache Coherence Protocol Design [36 points]	
Memory Ordering [20 Points]	
Total [100 points]	

Seat Number:

Caches and Cache Coherence [21 points]

1. Consider an 8-core multiprocessor running a perfectly parallelizable program. Each core runs a single thread and has a private cache which is coherent with others. Assume that a single-threaded (single-core) execution of the program can read data at a maximum rate of 100 GB/s, 60% of which is read from the private cache, and 40% from memory. In case of a multi-threaded (multi-core) execution, each thread can read data at a maximum rate of 100 GB/s, 60% of which is read from private caches, 30% is read from other coherent caches, and the remaining 10% is read from memory. For this question, assume that the access pattern is evenly distributed.

    a. What is the speedup in the overall rate of reading data by the multiprocessor in the multi-threaded execution scenario as compared to single-threaded execution when memory bandwidth is 20 GB/s? [3 points]

    b. Assuming a bus-based coherence protocol, how much total bus bandwidth is required for the multi-threaded scenario? [2 points]

Seat Number:

2. Assume an 8-core multiprocessor system with a per-core 32KB 8-way set-associative L1 data cache. All private caches are connected to a shared L2 cache with 8 banks and use a directory-based MSI write-invalidate coherence protocol. All addresses are 64-bits wide, and the cache block size is 64 bytes. Ignore the L1 instruction cache in this question.

a. In the L1 data cache, how many bits are required for the cache block offset, set index, and cache tag, respectively? [3 points]

b. Consider a sparse directory organization with 8 ways, 4x overprovisioning, and a bit vector to track all the sharers. Calculate how many entries are required in the directory, given the L1 data cache organization above. [2 points]

c. Based on the directory organization in (b), how many sets are in the directory? How much total storage is required for each distributed bank of the directory co-located with an L2 bank? [3 points]

Seat Number:

3. Consider an N-core multiprocessor system with a bus-based MSI write-invalidate cache coherence protocol with write-back caches. Assume that all cache blocks are 64 bytes and are invalid at the start, and the cache capacity is infinite. Calculate the total number of bytes from array A (as a function of N) that are (i) brought into any of the private caches from the bus, and (ii) written back from any of the private caches to the bus, when executing each of the given code snippets. [Hint: consider false sharing] [8 points]

a.
```
uint64 A[N];

//single-threaded version
for(i = 0; i < N; i++){
   A[i] = 1;
}
```

b.
```
uint64 A[N];

#pragma omp parallel for
for(i = 0; i < N; i++){
   A[i] = 1;
}
```

c.
```
uint64 A[N*8];

#pragma omp parallel for
for(i = 0; i < N*8; i+=8){
   A[i] = 1;
}
```

Seat Number:

Parallel Computing & SW Optimizations [23 points]

4. Assume you have a program which is made of four different segments. These segments along with their execution time when run completely serially are shown below:

i. Fully parallelizable part 1 (30 secs)
ii. Serial part (2.5 secs)
iii. Fully parallelizable part 2 (15 secs)
iv. Serial part (2.5 secs)

Note: Assume that according to OpenMP's fork-join model, there is an additional overhead of $0.25 * 10^{-3} * n$ secs when forking or joining n threads. Also assume that only one thread can run on one core.

a. Write down the expression for theoretical speedup when the program is parallelized using n threads. [2 points]

b. How many threads are needed to achieve maximum speedup? What is the value of the maximum speedup? Please answer both. (Hint: $\frac{dx^a}{dx} = a * x^{a-1}$) [4 points]

c. Let's say you want to run this program on a multiprocessor with:
12 baseline normal cores
2 high performance cores that are 2x faster each than a baseline core
2 ultra performance cores that are 4x faster each than a baseline core
Find the maximum speedup possible assuming that there can be load imbalance between threads when executing parallel sections of code. [4 points]

Seat Number:

5. Consider the following code snippet running on a multiprocessor system with 16 cores:

```
int A[NTHREADS], B[N]; // N is a very large number.
int i=0, j=0;

while( i++ < N ) {
  #pragma omp parallel for
  for (j = 0; j < N; j++){
    int tid = omp_get_thread_num();

    if (j < (N/2) && j % 2 == 0)
      A[tid] += do_complex_calculation(B[j], i);

    A[tid] += do_simple_calculation(B[j], i);
  }
}
```

When structured in the above fashion, this code has *three* major performance bottlenecks. List each one of them and explain why it will result in suboptimal performance. [6 points]

Seat Number:

6. Please answer the following questions while considering a multiprocessor scenario.

a. Define false sharing and true sharing, and briefly describe the differences. [3 points]

b. Consider the given code snippet that naïvely implements a histogram algorithm.

```
int histogram[N_BUCKETS];

#pragma omp parallel for
for(int i = 0; i < INPUT_SIZE; i++){
      unsigned bucket = calc_hist(input.next());
      #pragma omp critical
      histogram[bucket]++;
}
```

Assume that the *calc_hist* function distributes the input values evenly in all the buckets of the histogram.

i. Explain how the cache miss rate changes because of true sharing and false sharing when the number of buckets in the histogram increases. [2 points]

ii. Explain the changes that you would make to optimize the code. [2 points]

Seat Number:

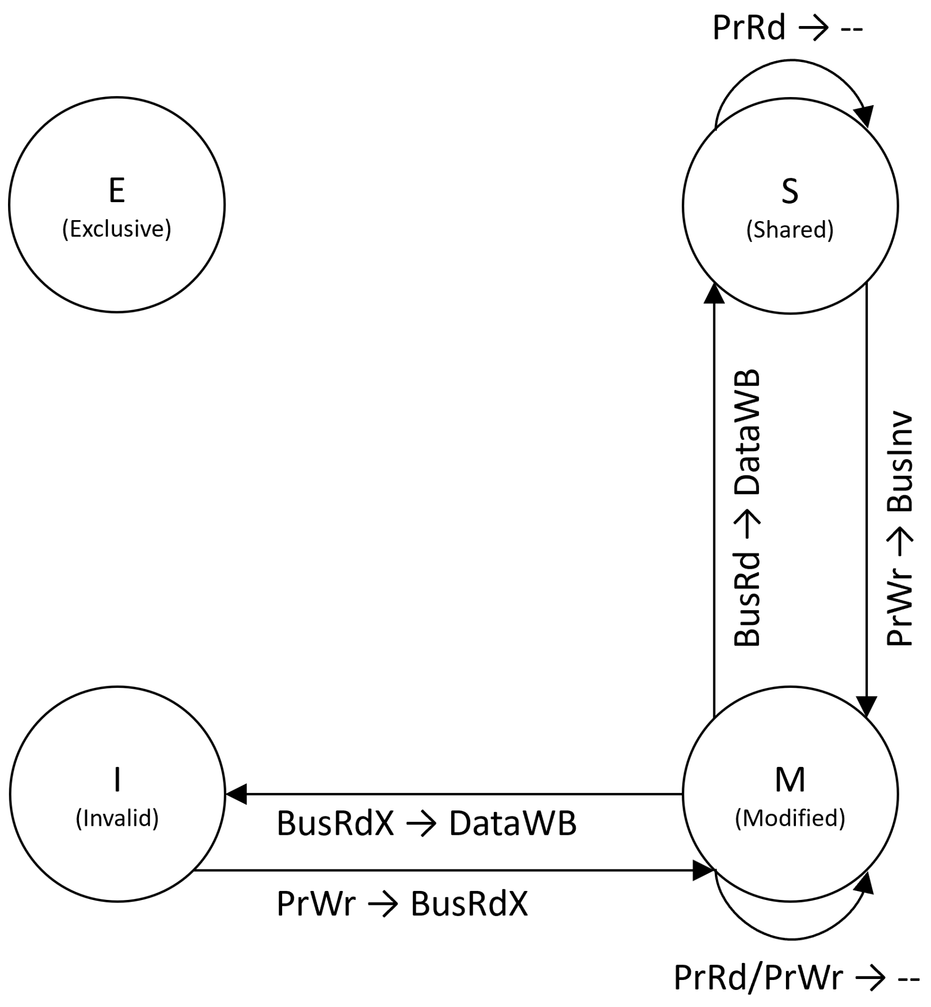
Cache Coherence Protocol Design [36 points]

7. Suppose we are designing a multiprocessor with three constituent processors: P0, P1, and P2 using a bus-based cache coherence protocol. Our multiprocessor currently uses a **3-state MSI protocol** with write-back/write-allocate caches. We want to reduce the amount of bus traffic in our multiprocessor, and as such, we plan to enhance our multiprocessor to use the **4-state MESI protocol** by adding the "Exclusive" (E) state. Recall that by adding the E state, the protocol can transition to Modified (M) without using a BusInv transaction if a block is the only (locally) cached copy in the system. As discussed in the lecture, the protocol uses a "shared line" on the bus to track if there are existing copies. On any BusRd transaction, caches that have copies of the block raise the shared line.

a. Given the following **partial** state transition diagram (shown in the next page) for a 4-state MESI protocol, complete the diagram with all of the missing transitions. Use the {Event → Action} notation and "-" to indicate the transitions where no action is taken. [15 points]

PrRd: Processor read	BusRdX: Bus read exclusive (a.k.a read-and-invalidate)
PrWr: Processor write	BusRd: Bus read
BusInv: Bus invalidate	DataWB: Writeback dirty data on the bus

Seat Number:


PrRd → --
E
(Exclusive)
S
(Shared)
BusRd → DataWB
PrWr → BusInv
I
(Invalid)
M
(Modified)
BusRdX → DataWB
PrWr → BusRdX
PrRd/PrWr → --

Seat Number:

b. Each cache is direct-mapped and holds four cache blocks, each block holds four words, and a word is one byte. To simplify the illustration, the cache address tag contains the full address in **decimal**, and each word is shown as two **hexadecimal** characters. Assume the initial cache and the memory state is as illustrated below. In the table, "St" stands for the coherence state, and "T" for the cache tag. **Note that the words in a cache block are ordered from right to left (e.g., in the example below the value at address 100 is 0x00, and at 103 is 0x42).**

0	**P0**						**P1**						**P2**					
Block	St	T	Data				St	T	Data				St	T	Data			
B0	I	100					E	100	0x42	0xFF	0x10	0x00	I	100				
B1	M	104	0x04	0x42	0x12	0x10	I	104					I	104				
B2	S	108	0x44	0x68	0x44	0x20	I	108					S	108	0x44	0x68	0x44	0x20
B3	I	112					I	112					M	112	0x54	0x66	0x01	0x25

Memory				
Addr	Data			
100	0x42	0xFF	0x10	0x00
104	0x04	0x18	0x12	0x10
108	0x44	0x68	0x44	0x20
112	0x54	0x66	0x01	0x22

What is the resulting state (i.e., coherence state, tags, and data) of the caches and memory after each memory operation? Treat each action below as a sequence of loads and stores. **Leave the cells empty when the state does not change.** [14 points]

Seat Number:

P1: Read 115

1	P0						P1						P2					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

P1: Store 113 <= 0x04

2	P0						P1						P2					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

Seat Number:

P2: Read 102

3	P0						P1						P2					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

P1: Store 108 <= 0x26

4	P0						P1						P2					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

Seat Number:

P2: Store 102 <= 0xAE

7	**P0**						**P1**						**P2**					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

P2: Store 104 <= 0x17

7	**P0**						**P1**						**P2**					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

Seat Number:

P1: Read 104

7	P0						P1						P2					
Block	St	T	Data				St	T	Data				St	T	Data			
B0		100						100						100				
B1		104						104						104				
B2		108						108						108				
B3		112						112						112				

Memory				
Addr	Data			
100				
104				
108				
112				

Seat Number:

c. For each memory operation, specify all the bus transactions, if any, requested by the bus-side cache controllers (e.g., BusRd, BusInv) upon each memory operation. If there are no transactions for a memory operation, indicate it by using '-'. [7 points]

Memory Operation	Bus Transactions
P1: Read 115	
P1: Store 113 <= 0x04	
P2: Read 102	
P1: Store 108 <= 0x26	
P2: Store 102 <= 0xAE	
P2: Store 104 <= 0x17	
P1: Read 104	

Seat Number:

Memory Ordering [20 points]

8. Assume a processor that implements two different memory consistency models: (1) Modified sequential consistency model that does not implement cache peeking but does use a Store Buffer (SB). (2) Processor consistency model that does implement cache peeking and uses the Store Buffer (SB) as well. Assume a 2-entry store buffer (SB) and 2-entry load/store queue (LSQ). Moreover, assume a 1-cycle execution latency for a cache hit and a 100-cycle execution latency for a cache miss. Also, assume memory fence operations take 1-cycle to resolve. Note that only one instruction can enter, and only one instruction can exit each LSQ and the SB in a given cycle. Additionally, a single instruction cannot enter and leave the same structure in the same cycle. Finally, assume instructions are removed from the LSQ and SB based on the memory ordering specifications, and the ROB can accommodate all instructions shown in the program.

For each memory consistency model, show the changes in the LSQ and SB for the given program. **Indicate the cycle number at which each change occurs,** knowing that the first instruction enters the hardware structures at cycle 0. Please denote peeking operations when they happen by using '(P)' in your diagrams. E.g., ST D (P).

Please note that "ST X" and "LD X" denote stores/loads involving address X, and "A", "B", "C", etc. each refer to distinct memory regions. The actual value stored is irrelevant. In each case, the list of operations below specifies whether a cache hit or miss occurs. You may have more space than the number of changes in the hardware structures.

Executed program:

1. ST A (miss)
2. ST B (miss)
3. LD A (miss)
4. ST C (hit)
5. LD D (miss)
6. ST E (miss)
7. LD F (hit)
8. FENCE (show in both the models)
9. LD G (miss)
10. ST H (hit)

Seat Number:

a. Modified Sequential Consistency [10 points]

Cycle 0	**Cycle**	**Cycle**	**Cycle**	**Cycle**	**Cycle**
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	**Cycle**	**Cycle**	**Cycle**	**Cycle**	**Cycle**
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	**Cycle**	**Cycle**	**Cycle**	**Cycle**	**Cycle**
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	**Cycle**	**Cycle**	**Cycle**	**Cycle**	**Cycle**
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Seat Number:

b. **Processor Consistency** [10 points]

Cycle 0	Cycle	Cycle	Cycle	Cycle	Cycle
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	Cycle	Cycle	Cycle	Cycle	Cycle
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	Cycle	Cycle	Cycle	Cycle	Cycle
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Cycle	Cycle	Cycle	Cycle	Cycle	Cycle
LSQ	**LSQ**	**LSQ**	**LSQ**	**LSQ**	**LSQ**
SB	**SB**	**SB**	**SB**	**SB**	**SB**

Seat Number:

(This page is intentionally left blank. You can use it as you wish.)