EECS 570 Midterm Exam - Solutions
Winter 2025

Name: _____ Uniqname: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

#	Question	Points
1	Short answers	/ 10
2	Vector Processing	/ 5
3	Synchronization	/ ~~25~~ 23
4	Coherence Protocol Optimization	/ 20
5	Memory Consistency Model	/ 15
6	Transactional Memory	/ 15
Total		/ ~~90~~ 88

NOTES:

- 5 pages of notes (front and back) is allowed.
- Calculators are allowed, but no PDAs, portables, cell phones, etc.
- Don't spend too much time on any one problem.
- You have 80 minutes for the exam.
- There are 12 pages in the exam (including this one). Please ensure you have all pages.
- Be sure to show work and explain what you've done when asked to do so.

1. Short Answer [10 points]

a) State two reasons why a parallel program might not achieve linear speedup in a multicore system. (Linear = speedup of *p* with *p* processors) [2 points]

Sequential portion of the program (Amdahl's law)
Synchronization overhead
Load imbalance across cores

b) Explain a type of parallel programming sharing pattern that would benefit from a write-update protocol compared to an invalidation-based protocol. [2 points]

Producer-Consumer
Shared work-queue (work stealing)

In general, patterns where there are one or more threads that read data written (produced) by another thread

c) State one advantage of message passing over shared memory. [1 point]

Enables expert programmers, who understands program's behavior, to explicitly control data-movement and optimize for locality. This can result in scalable performance.

No need for complex hardware in the form of cache coherence.

No synchronization related bugs like data-races

Fault tolerance. If one process fails, other processes may be isolated.

d) In a parallel program, 4 threads reach a barrier after 5 ms each. The barrier takes 2 ms to complete. What is the total elapsed time from when the first thread arrives at the barrier until all threads have passed the barrier? [1 point]

7 ms

We decided the phrasing of this question was not clear enough. There were several different interpretations that were reasonable. As such all students were awarded a point here.

e) How does the choice of inclusive, exclusive, or non-inclusive caches affect cache coherence protocol complexity in a multi-core system? [2 points]

Inclusive: Easier to implement cache coherence protocol

Exclusive: Higher complexity for cache coherence protocol implementation

Non-inclusive: Highest complexity as both caches need to be checked for all coherence actions.

f) Consider a program that has 10% sequential code, while the remaining 90% is embarrassingly parallel. What is the maximum possible speedup that can be achieved for this program? And, what is the speedup that can be achieved in a system with 9 processors?
[2 points]

Use Amdahl's law

Maximum speedup: 10

Speedup with 9 processors: 5

2. Vector Processing [5 points]

Consider the following vectorized code:

```
void vector_add(const float* a, const float* b, float* result, int size) {
    for (int i = 0; i < size; i += 8) {
        __m256 va = _mm256_loadu_ps(&a[i]);   // Load 8 floats from array a
        __m256 vb = _mm256_loadu_ps(&b[i]);   // Load 8 floats from array b
        __m256 vres = _mm256_add_ps(va, vb);  // Vectorized addition
        _mm256_storeu_ps(&result[i], vres);   // Store result
    }
}
```

a. State two reasons why vector addition increases performance compared to scalar addition. [3 pts]

Vector processor with 8 ALUs enables concurrent execution of 8 operations

Reduces the number of instructions (about 1/8th), including the for-loop's bookkeeping instructions.

b. Why would loads to `a[i]` and `b[i]` be slower if those arrays are not memory aligned?

(A memory address is said to be aligned to a specific boundary if the address is a multiple of that boundary size. For vectorized instructions, the boundary size is the byte width of the vector register being used.) [2 points]

Unaligned loads can span across multiple cache blocks. As a result, a processor may need to issue more than one cache read request.

This incurs additional hardware complexity, which can also increase processor's clock cycle and area.

3. Synchronization [~~25 points~~] [23 points]

You have a shared variable *max_val* that keeps track of the maximum value encountered by multiple threads. Write and analyze a thread-safe concurrent function called *update_max* that updates *max_val* to a new value, but only if the new value is greater than the current value of *max_val*.

Use the following CAS (Compare-and-Swap) atomic operation:

```
CAS(sh_variable_address, expected, new_value)
```

CAS checks if the value at *address* is equal to *expected*.

- If they are equal, it updates the value at *address* to *new_value* and returns true.
- If they are not equal, it does nothing and returns false.

a) Complete the following pseudo-code: [5 points]

```
void update_max(volatile int *max_val, int new_val) {
    int old_val;
    do {
           old_val = *max_val;

           if (new_val <= old_val) {
                 return;
           }
     } while ( !CAS(max_val, old_val, new_val) );
}
```

Example implementation of update_max. Other correct answers are possible.

b) Explain how you used CAS operation to ensure correct updates to max_val. [3 pts]

By using CAS, we are able to atomically check if the value has been updating between reading its value and (assuming it was greater at time of read) attempting to write the new max. If any update did occur, possibly with value still lower than our local new_val, we will retry load, check condition, and perform CAS again.

c) Describe potential performance issues that could arise if many threads try to update *max_val* simultaneously. [2 points]

As with any atomic RMW instruction, under high contention performance degrades due to the thundering herd phenomenon. (answers can vary)

d) Among the following lock algorithms, circle the locks: ~~[2 points X 5]~~ [2 points X 4]

(circle all that apply)

i) that provides fairness.

test&set test&test&set **ticket lock** **array-based lock** **MCS lock**

ii) that require the number of threads that might acquire the lock to be known in advance.

test&set test&test&set ticket lock **array-based lock** MCS lock

iii) don't ensure forward progress if the operating system were to deschedule a thread waiting to acquire the lock.

test&set test&test&set **ticket lock** **array-based lock** **MCS lock**

iv) require the instruction set architecture to provide an atomic memory operation of some kind in order to implement the lock.

test&set **test&test&set** **ticket lock** **array-based lock** **MCS lock**

e) Identify all pairs of instructions that constitute a data-race in the following code snippets. All variables are shared, except ones with *tmp* as their prefix. If there is no data-race, then say "**None**". [5 points]

i)
```
atomic int flag = 0;  int value = 0

Thread-1          Thread-2

I1. value = 1;        I3. while (flag == 0);
I2. flag  = 1;        I4. tmp1 = value;
```

Data race pairs: **None** _____

ii)
```
value = 0; key = 0;

Thread-1          Thread-2

I1. value = 1;        I5. lock(m)
```

```
I2. lock(m)          I6.        key++;
I3.        key++     I7. unlock(m)
I4. unlock(m)        I8.        value = 3;
```

Data race pairs: **I1 - I8**

4. Coherence protocol optimizations [20 points]

a. Identify a property of a cache block that obviates the need to maintain coherence. [3 pts]

One of the following answers:

Read-only

Thread-local

Singular LLC. Does Not need to maintain coherence at this level.

b. Which part of the system would you extend to identify the property in question (a), and how? [3 pts] [5 pts]

Read-only solution (thread-local is similar):

Operating system:

Assume every cache block is read-only at the beginning of execution.

Page-table and TLB track read-only pages using 1-bit meta-data.

Load miss checks TLB to determine if the miss is to read-only data.

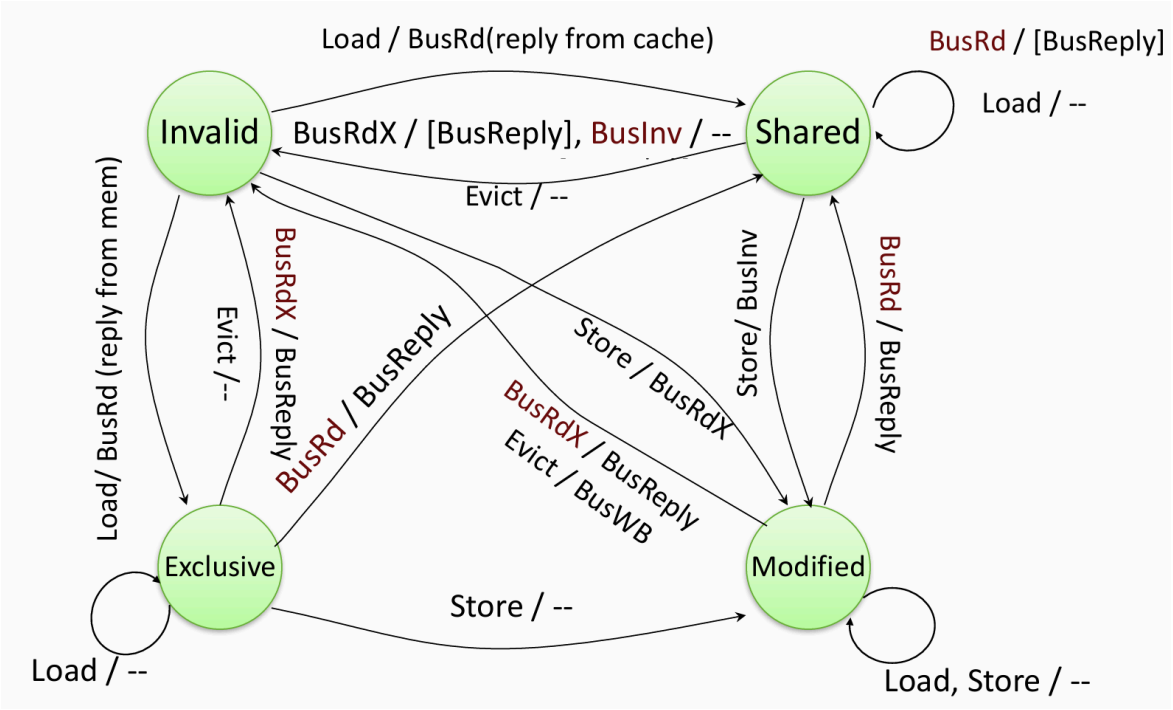
Bypass coherence (e.g., no need to check home directory in the case of directory protocol), and fetch data directly from memory.

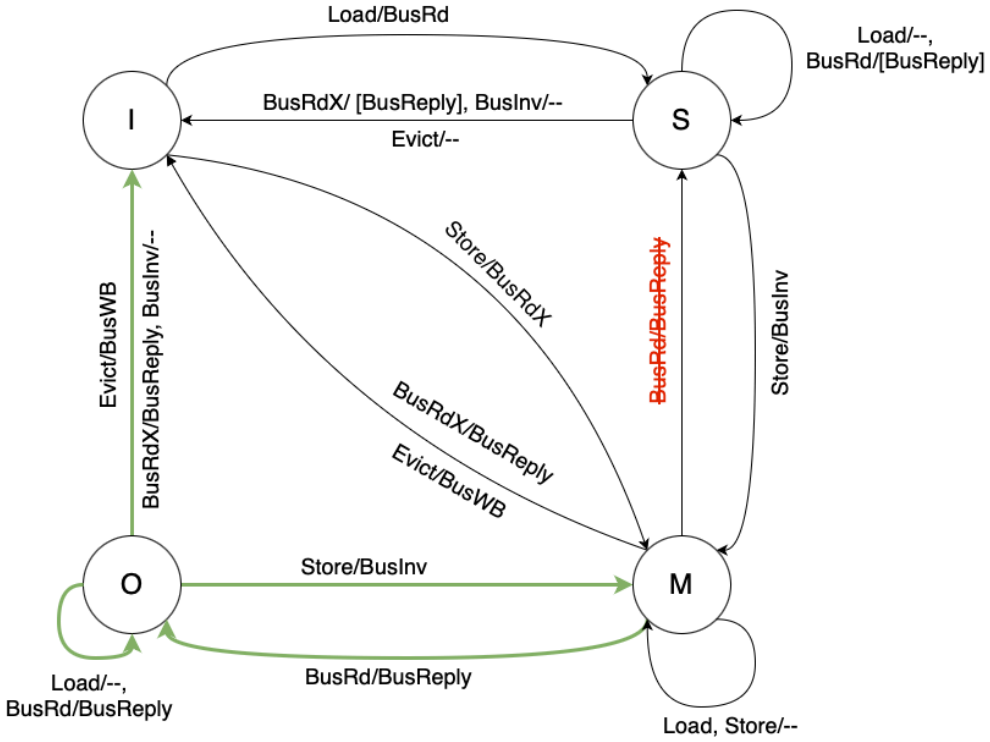
On a store-miss (or upgrade miss) OS handler is invoked to safely transition the cache block to be read-write cache block. After a transition, all future accesses to this block would go through conventional coherence protocol.

c. In the MOESI coherence protocol, a cache block in the **Owner (O)** state must be written back to memory when evicted. How can the MOESI protocol be extended to avoid this writeback when it's not necessary? [4 pts]
[2 pts]

When a cache block in O state is evicted, check if another processor has it in the shared state. If so, make that processor the new owner, instead of writing back the cache block.

d. Make changes to the state diagram of the MESI protocol to convert it into an M**O**SI protocol. You can directly edit the picture or redraw it. [10 pts]





5. Memory Consistency Models [15 points]

a. Assume a TSO (Total Store Order) processor that guarantees write atomicity. Insert one instruction (I2) to guarantee memory ordering between I1 and I3. I2 **cannot** be a fence or a synchronization operation. Justify. [3 pts]

```
I1:  Store A = register1

I2:  _________________

I3:  Load register2 = B
```

Load register3 = A, Store B = register3

b. You are asked to extend an existing C++ compiler to support a new language standard, **SC-C++,** which guarantees sequential consistency (SC) to the programmers. However, you only have TSO hardware to run your programs on. Assume TSO guarantees write atomicity.

i. Given an example optimization that SC-C++ compiler cannot do? [2 pts]

1) Common sub-expression elimination where X, Z1 and Z2 are shared variables. It would appear as if you reordered the second read to X ahead of read to Y.

tmp1 = (X * 5) + Z1 tmp2 = Y * Y tmp3 = (X * 5) + Z2	tmp0 = X * 5 tmp1 = tmp0 + Z1 tmp2 = Y * Y tmp3 = tmp0 + Z2

A high-level description of this answer is sufficient. No need for code example.

2) Load, Store Hoisting

ii. How can the SC-C++ compiler guarantee that its output binary's execution is sequentially consistent when it runs on a TSO processor? Make sure the constraints you specify are as lenient as possible. [3 pts]

TSO relaxes Store->Load memory ordering. To prevent hardware from performing this reordering, compiler inserts a fence between any pair of store-load that appear in the program order.

c. DrMagic has developed a powerful new compiler analysis tool called RacerX that can determine if a load or store instruction in the TSO binary is data-race-free or not.

i. Explain how you can use RacerX to reduce the overhead that SC-on-All compiler introduced in question (b.ii) to guarantee SC on TSO processor? [3 pts]

Use RacerX to detect data-races.

Insert fences between a store-load pair, only if either the store or the load is racy. This reduces the number of fences that need to be executed.

ii. If DrMagic has false positives (i.e. memory accesses that can never participate in a data-race are reported as racy), can you still use it for the above optimization? Why, or why not? [2 pts]

Yes. False positives would result in unnecessary fences being inserted. While this may reduce performance, it will not violate sequential consistency memory ordering constraints.

iii. If DrMagic has false negatives (i.e. memory accesses that can race are not reported), can you still use it for the above optimization? Why, or why not? [2 pts]

No. A false negative would cause the compiler to not insert a fence between store and load that may participate in a data-race in some execution. In those executions, sequential consistency may be violated.

6. Transactional Memory (TM) [15 points]

a) What is the benefit of TM over locks? [1 point]

Programming becomes easier. You can reason about it as if you are using a single coarse-grained lock for the entire program, while still achieving the performance benefits of a fine-grained lock implementation.

b) State one advantage and one disadvantage of **eager** conflict detection as compared to **lazy** conflict detection in transactional memory systems. [2 points]

Advantage: conflicting transactions abort immediately, do not waste resources for work that is later invalidated.

Disadvantage: overhead of detection on every memory access, unnecessary abort on temporally silent writes.

c) State one advantage and one disadvantage of hardware transactional memory as compared to software transactional memory. [2 points]

Advantage: higher performance

Disadvantage: difficult to support unbounded transaction sizes

d) Consider the following two transactions that are executed concurrently in two processors. [10 points]

```
Initial state:  X = 0 ; Y = 0;

T1                      T2

begin                   begin
     M1: X = 1               N1: Y = 1
     M2: Y = 2               N2: X = 2
end                     end
```

Consider the following memory states after speculatively executing the two transactions concurrently, ***before*** the transactions are committed. For each state, argue whether or not the execution is **feasible** in a modern out-of-order processor. If it is feasible, determine whether or not the execution of transactions is **serializable** (no need to roll-back).

State the reason(s) for your choice. You may want to consider explaining using the execution order of transactions and/or individual operations.

i) **Memory state:** **X = 2** **Y = 1**

Feasible: Yes / No Serializable: Yes / No

Reason: T2 executes after T1

ii) **Memory state:** **X = 2** **Y = 2**

Feasible: Yes / No Serializable: Yes / No

Reason:

M1; N1; M2; N2 T1 and T2 conflict; the problem asks for *feasible* intermediate state, not committed state

iii) **Memory state:** **X = 0** **Y = 0**

Feasible: Yes / No Serializable: Yes / No

Reason: T1 and T2 must execute

iv) **Memory state:** **X = 1** **Y = 1**

Feasible: Yes / No Serializable: Yes / No

Reason: M2; N2; M1; N1 M2 and N2 are executed first out-of-order

EMPTY