EECS 570 Midterm Exam
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]

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]

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

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]

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:

Exclusive:

Non-inclusive:

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]

Maximum speedup:

Speedup with 9 processors:

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]

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]

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 {
      ____________________________

      ____________________________

      ____________________________

      ____________________________

     ____________________________

      } while (_____________________________________________);
}
```

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

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

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: _____ _____

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: _____ _____

4. Coherence protocol optimizations [20 points]

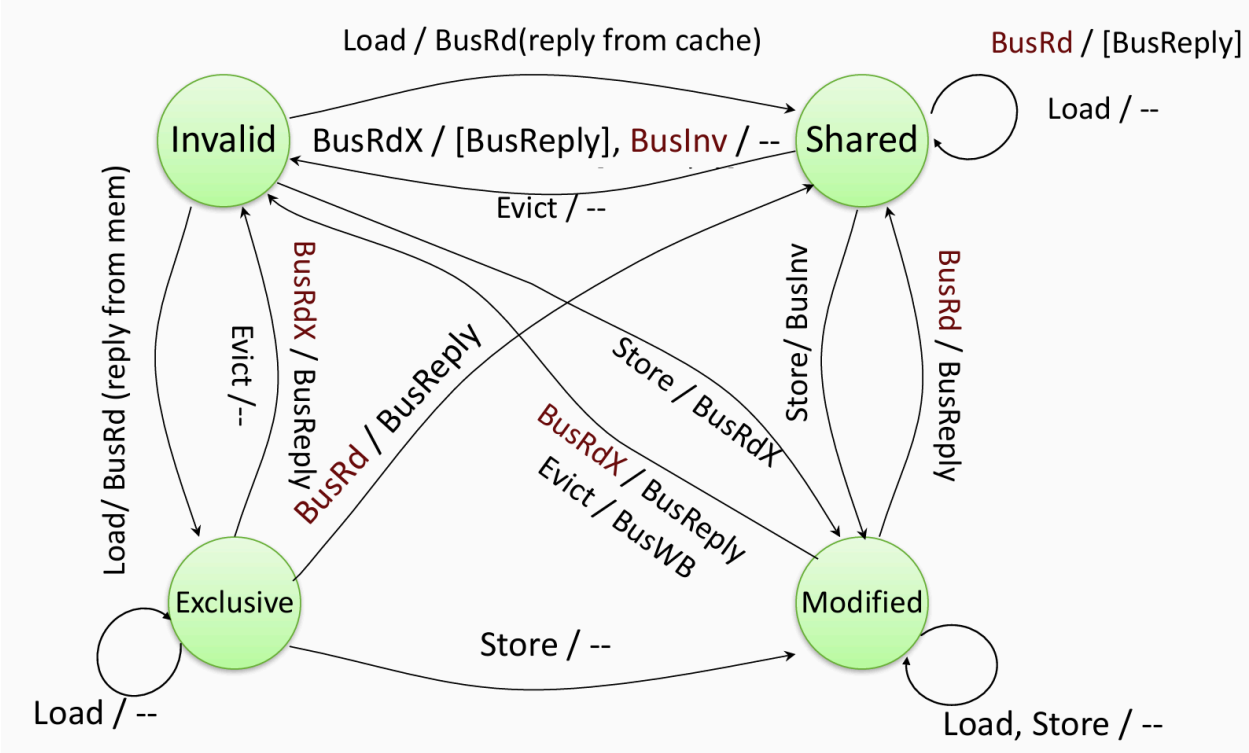
a. Identify a property of a cache block that obviates the need to maintain coherence. [3 pts]

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

c.

d. 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]

e. 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
```

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]

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]

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]

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]

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]

6. Transactional Memory (TM) [15 points]

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

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

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

Advantage:

Disadvantage:

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:

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

Feasible: Yes / No Serializable: Yes / No

Reason:

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

Feasible: Yes / No Serializable: Yes / No

Reason:

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

Feasible: Yes / No Serializable: Yes / No

Reason:

EMPTY