EECS 570 *Midterm Exam* *(Ans Key)*

Winter 2024

Name: ______________________________ unique name: ____________

Sign the honor code:

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

____

Scores:

#	Points
1	**/ 30**
2	**/ 30**
3	**/ 20**
4	**/ 20**
Total	***/ 100***

NOTES:

- Closed book. One 8.5" x 11" page of notes is allowed.
- Calculators are allowed
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam (avg. 30 minutes per problem).
- There are 10 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 Answers [30 points]

a. Dennard Scaling [3 points]

Moore's law is an observation that the number of transistors on an integrated circuit will double every two years with minimal rise in cost. This is possible because of Dennard scaling, where when the wire width scales down from W to W/α and the gate width scales from G to G/α, voltage decreases from V to V/α.

(i) transistor density scales from D to ____$D*\alpha^2$________

(ii) Power per transistor scales from P to ___P/α^2__________

(iii) Power density scales from p to _______p__________

b. Post Dennard Scaling [4 points]

As the voltage keeps decreasing, the leak current increases as Vdd gets closer to Vth. In this post-Dennard scaling scenario, when the wire width scales down from W to W/α and the gate width scales from G to G/α, voltage stays constant at V.

(i) transistor density scales from D to ____$D*\alpha^2$_________

(ii) Power per transistor scales from P to ____P_________

(iii) Power density scales from p to ______$p*\alpha^2$___________

(iv) Post-Dennard scaling creates a [✓ Power] / ILP / frequency (circle one) wall that kills Moore's Law.

c. Message Passing vs. Shared Memory [11 points]:

Compare a message-passing model against a shared memory model for multiple core systems. Please circle the model or models that fit the description

Programming model and HW-SW abstraction

Different threads/cores see unified address space. [✓ Shared mem] / message passing

Requires explicit communication defined by SW. Shared mem / [✓ message passing]

Implementation

Implementation of synchronization is complex. [✓ Shared mem] / message passing

High API overhead Shared mem / [✓ message passing]

Flexibility

Different cores can run different OS. shared mem / [✓ message passing]

Easy to optimize for specific workloads. Ambiguous Question, all marked correct

Give an example use case of the message passing model and an example use case of the shared memory model. Explain why each model is used by connecting their pros and cons. [5 points]

Examples of MP:

Distributed machines connected via Ethernet / IO / sockets
cluster where each node runs a different OS / on a different dataset.
Heterogeneous systems

Why MP?

More flexible -> support different OS / heterogeneous memory models
better isolation between nodes because of the explicit communication (if the example requires different nodes run on different data)
can tolerate high API overhead (if the example presents a case where communication is not frequent)
HW implementation is easier and can scale to a larger cluster

Examples of shared memory:

Multi-core CPU on the same die
GPU

Why shared mem?

Better programmability / easier for the SW implementation / offers shared address space
Finer-grained communication
Lower latency / no API overhead

The pros/cons of MP/shared memory should be relevant in your example.

Example Answer

MP example: distributed systems where devices are connected via Ethernets and work on a map-reduce workload.
Why MP: devices only need to communicate when each device finishes working on its own piece of data. High API overhead can be tolerated since the communication is coarse-grained.
Shared memory example: Multi-core processor running HPC workload.
Why shared mem: The shared memory model allows finer-grained communication so that different cores can access the same data structure in parallel.

d. GPU Architecture [12 points]

SIMT (single instruction multiple threads) vs. SIMD (single instruction multiple data)

(circle the model or models that fit the description [5 points])

Programming model used on:

CPU	SIMD / SIMT
GPU	SIMD / SIMT
Provides better support for conditional control flow	SIMD / SIMT
Executes in locked steps.	SIMD / SIMT

In SIMD, each thread processes a vector of data in locked steps.

In SIMT, 32 threads in the same warp are executed in locked steps.

Need explicit synchronization	SIMD / SIMT

b. CUDA memory model: circle which scopes the following memory space is defined in [4 points]

Register	per instruction	/	per block	/	per grid
Local mem	per instruction	/	per block	/	per grid
Shared mem	per instruction	/	per block	/	per grid
Global mem	per instruction	/	per block	/	per grid

c. Warp Scheduling [3 points]:

GPU SMs perform a context switch to schedule a different warp. True / False

Register values of (the warp currently being executed / all the warps) stay in register file. (select one that makes the sentence correct)

Conditional branches are handled with _predication masks / serializing both paths____

2. Cache Coherence [30 points]
(a) It is common in shared memory programs for one processor to do a single write to a cache block read by many processors reading frequently. In the base MSI protocol this causes every processor to invalidate the cache line, flood the bus with load / BusRd messages, and wait for the writer in the M state to downgrade itself and share the updated cache block.

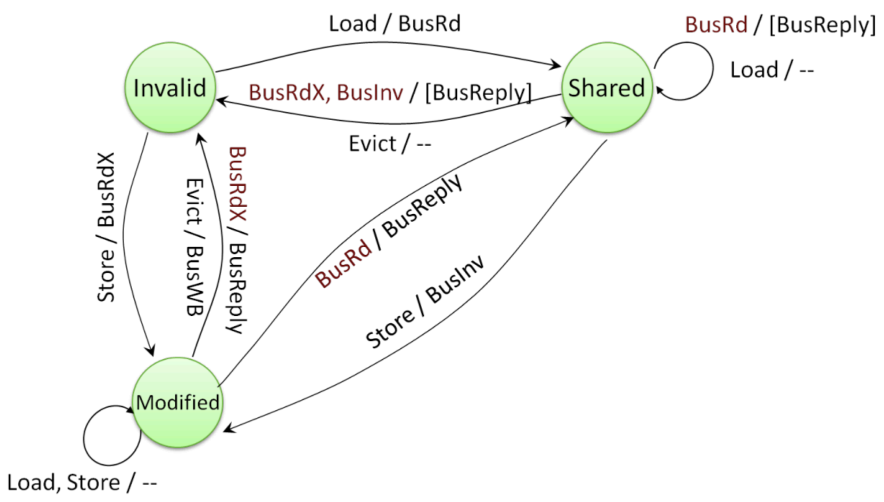
We propose an optimization where a processor that intends on doing a single write to a cache block can warn all the readers ahead of time, and broadcast the update when it is finished. This will reduce the storm of BusRd messages after invalidations in the MSI protocol. This requires two new states to be added to the MSI protocol (a standard MSI diagram is included below for reference.)

The two new states are:

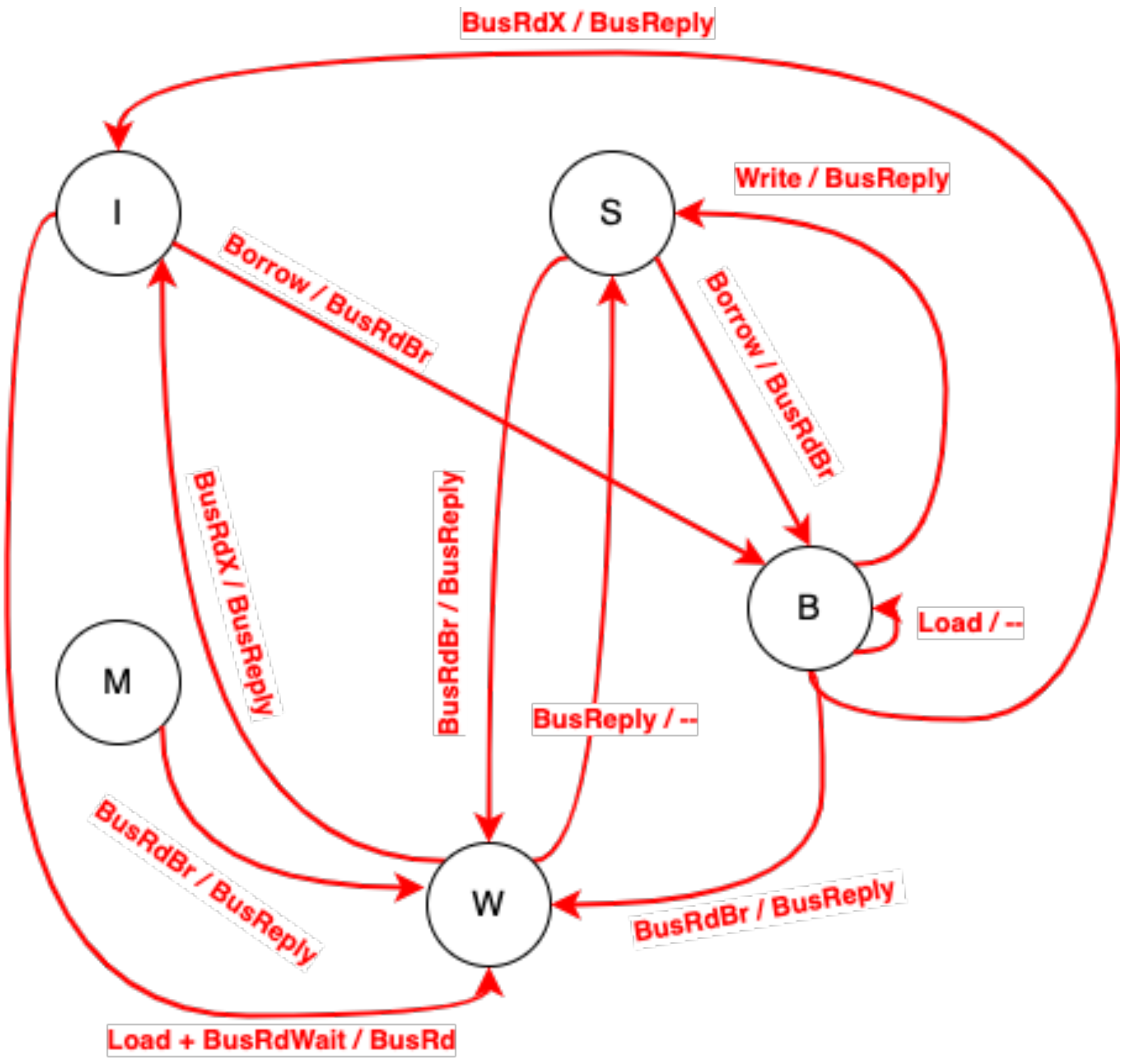
Borrowed (B) - the processor intends on performing a single write operation so it "borrows" the cacheline from processors in the modified or shared state. After a single write to the cache block the borrowed state downgrades itself to the shared state and sends a BusReply message to other processors waiting for the new cacheline.

Waiting (W) - processors that are "lending" the cache block to the processor in the borrowed state will stay in the waiting state until they receive a BusReply or BusRdX message.

More information:
- To transition to the Borrowed state a processor issues a **borrow** command which sends a **BusRdBr** message over the bus.
- A processor cannot transition to the Borrowed state from the Modified state.
- There cannot be multiple processors in the Borrowed state, and if a processor is in the Borrowed state all other processors must be either Invalid or Waiting.
- A processor with an invalid cache block that attempts a *load* to enter the shared state may receive a **BusRdWait** message on the bus telling it to enter the waiting state.
- A processor with an invalid cache block that attempts a *store* or **borrow** will always succeed.
- Lastly, processors in the Waiting state will not perform loads, stores, or borrows.



Given the information above, please complete the new arrows and labels of the state diagram with the same level of detail as the reference MSI protocol for the new MSI+BW protocol. [24 Points]



Add arrows and labels to the diagram that are needed for the new protocol. **You do not need to redraw any arrows or labels from the base MSI protocol.**

\

(b) What is a potential downside of this optimization, explain in several sentences giving an example sequence of operations? [6 points]

Lots of things, the most "obvious" one to me is that every processor in the Waiting state cannot make progress until they receive a BusReply. In standard MSI this would not happen because doing a load / BusRd would force a Modifier to downgrade, but the Borrowed state is essentially a "promise" that the processor will issue a write quickly. If that promise is not kept it's detrimental for performance.

3) Transactional Memory [20 points]

(a) Can transactions be used to replace barrier synchronizations? Justify. [2 points]

Ambiguous depending on how you interpret “replace”.

Replace strictly interpreted, this means “end transaction” at the place of a barrier, which won’t work.

Replace less strictly defined as “create a library of barriers” is possible because transactions can be used to create locks, and locks can be used to create sense reversing barriers.

(b) Provide an example where two transactions deadlock. [3 points]

```
        X = Y = 0;
begin                 begin
  Y = 1;                while(!Y);
  while (!X);           X = 1;
end                   end
```

(c) Consider two versions of a program, one written using transactions, and another using locks. Assume that both are carefully optimized by expert programmers. Which of the two versions do you expect to perform better, and why? Assume that the runtime overhead of conflict check and versioning is similar to the overhead of lock operations. [3 points]

Lock version is likely to perform better.

Using fine-grained locks, a programmer should be able to expose all available parallelism. Transactions cannot expose more parallelism than a version with hand-crafted fine-grained locks. However, it could suffer from additional performance loss due to rollbacks when the runtime optimistically executes two conflicting transactions in parallel.

But well optimized fine-grained locks are hard to engineer in practice. Transactions can attain performance of fine-grained lock implementation, but they are as easy to use as a single coarse-grained lock.

(d) Two transactions are said to conflict if they execute memory accesses to the same location, and at least one of those accesses is a write. Two transactions are serializable if they appear to have executed one after the other.

Is it true that if two concurrent transactions conflict, then they are not serializable? Justify your answer. Use examples if necessary. [3 points]

No, it Is false. If two concurrent transactions conflict, they will then be run serial. So they must in fact appear to have executed one after the other.

(e) A hardware transactional memory (HTM) system can support eager conflict detection by extending the coherence protocol. Would a lazy conflict detection have any advantages in a HTM? Justify your answer. Use examples if necessary. [3 points]

No it would not. The added cache coherence protocols in Eager will allow the detection of a conflict earlier without the need/overhead of searching through all other transactions that software TM must do. Lazy has advantages in SW TM by waiting until the end to avoid this overhead, but it is removed by the cache coherence protocol.

(f) Describe at least three unique challenges that may arise in supporting HTM in a GPU architecture, when compared to a multi-core CPU. [6 points]

1. Scalable conflict detection is a challenge as a GPU executes thousands of threads at any given time.

2. A GPU may not have coherence support, which makes it challenging to support conflict detection

3. Several warps are mapped to a core. Each warp has several threads. It is harder to track working set in a core's private cache. In a CPU where only one thread is mapped to a core, however, the working set of a transaction is easily maintained using one pair of read/write bits per cache line.

4. Register files are larger. Taking a full snap-shot of entire register file at the beginning of a transaction may be expensive.

5. If a transaction in a thread that belongs a wrap needs to be rollbacked, but other threads do not have a conflict, then it can lead to additional control-flow divergences between threads in a warp.

6. Caches have generally poorer temporal locality in GPUs than CPUs, and GPU threads have a
larger data footprint. Therefore, unbounded transactions (transactions whose working set does not fit within a cache) may be more common than in CPUs

4) Directory Protocols [20 Points]

a. What are the 2 major bottlenecks to scaling a snooping bus-based coherence protocol? [2 points]

Bus bandwidth
Snooping Broadcast Spam

b. What is the main difference between a 4-hop and 3-hop read transaction? [2 points]

In a 3-hop transaction, the remote node responds to the local node directly in parallel with the response to the home node.

c. Cruise missle invalidations are used to improve the performance of directory protocols. Describe how a cruise missle invalidation works and what it improves in the system. [4 points]

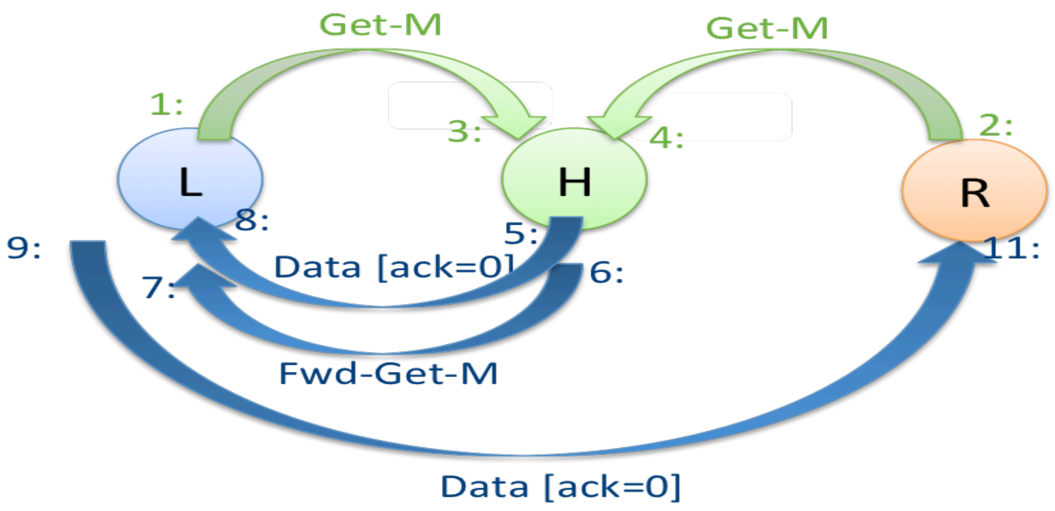
Invalidate messages are sent from sharer-to-sharer-to-sharer in a linked list fashion, rather than all invalidate messages originating from the home node in parallel. This helps to reduce the congestion near the home node, and bounds the number of invalidates and reduces the interconnect traffic.

d. When comparing a centralized versus distributed directory, what are the pros and cons of the design tradeoff. [4 points]

Centralized – Provides a serialization point (helpful for consistency) but creates a bottleneck in the interconnect

Distributed – Harder to serialize access to addresses in a different home node but helps distribute the home node traffic more evenly throughout the network

e. Below is the state diagram for a store-store race condition in a directory protocol. Each edge is labeled with the order the edge entered and exited the network. Below the diagram please fill out the state of the block at each node after the timestamp completes. You may leave any blocks blank if it doesn't change. [8 points]



After Cycle	State in L	State in H	State in R
0	I	I	I
1	IM^{AD}		
2			IM^{AD}
3		M, L	
4		M, R	
5			
6			
7	IMI^{AD}		
8	I		
9			
10			M

Would also take IM^{A} for the home state, indicating it was waiting for an ack.