CMU 18-447 Introduction to Computer Architecture, Spring 2015
HW 7: Coherence, Consistency, Multiprocessors and Interconnect

Instructor: Prof. Onur Mutlu
TAs: Rachata Ausavarungnirun, Kevin Chang, Albert Cho, Jeremie Kim, Clement Loh

Assigned: Wed., 4/8, 2015
Due: **Wed., 4/29, 2015**

1 Cache Coherence: MESI

Assume you are designing a MESI snoopy bus cache coherence protocol for write-back private caches in a multi-core processor. Each private cache is connected to its own processor core as well as a common bus that is shared among other private caches.

There are 4 input commands a private cache may get for a cacheline. Assume that bus commands and core commands will not arrive at the same time:

```
CoreRd: Read request from the caches core
CoreWr: Write request from the caches core
BusGet: Read request from the bus
BusGetI: Read and Invalidate request from the bus (invalidates shared data in
the cache that receives the request)
```

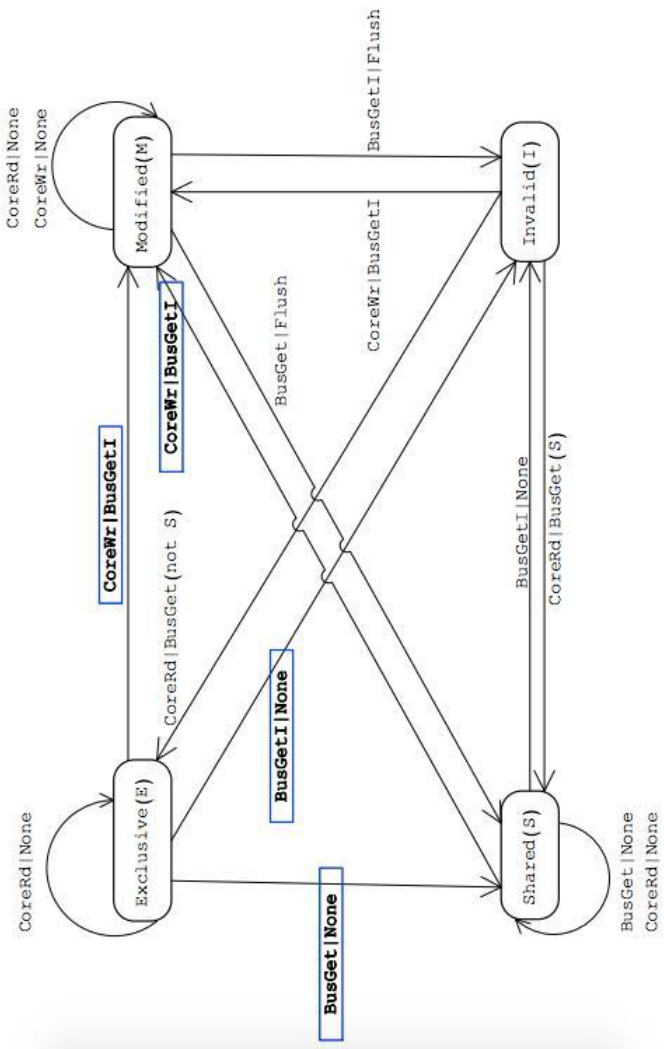
There are 4 actions a cache can take while transferring to another state:

```
Flush: Write back the cacheline to lower-level cache
BusGet: Issue a BusGet request to the bus
BusGetI: Issue a BusGetI request to the bus
None: Take no action
```

There is also an is shared (S) signal on the bus. S is asserted upon a BusGet request when at least one of the private caches shares a copy of the data (BusGet (S)). Otherwise S is deasserted (BusGet (not S)).

Assume upon a BusGet or BusGetI request, the inclusive lower-level cache will eventually supply the data and there are no private cache to private cache transfers.

On the next page, Hongyi drew a MESI state diagram. There are 4 mistakes in his diagram. Please show the mistakes and correct them. You may want to practice on the scratch paper first before finalizing your answer. If you made a mess, clearly write down the mistakes and the changes below.


Modified(M)
Invalid(I)
Exclusive(E)
Shared(S)
CoreRd|None
CoreWr|None
BusGetI|Flush
CoreWr|BusGetI
CoreWr|BusGetI
BusGet|Flush
CoreWr|BusGetI
CoreRd|BusGet(not S)
BusGetI|None
BusGetI|None
CoreRd|BusGet(S)
CoreRd|None
BusGet|None
BusGet|None
CoreRd|None

2 Coherence Protocols

Suppose we have a multiprocessor system with 512 processors. Each processor has a 1 Megabyte private writeback cache with 64-byte cache blocks. The main memory size is 1 Gigabyte.

(a) If we implement a snoopy bus based MESI cache coherence protocol, how many bits of state do we need in the entire system for the coherence protocol implementation? Where do these bits reside?

(b) If we instead implement a directory based cache coherence protocol *as we discussed in class*, how many bits of state do we need in the entire system for the coherence protocol implementation? Where do these bits reside?

(c) Which of the above two protocols would you choose for this system? Why?

3 Parallel Speedup

You are a programmer at a large corporation, and you have been asked to parallelize an old program so that it runs faster on modern multicore processors.

(a) You parallelize the program and discover that its speedup over the single-threaded version of the same program is significantly less than the number of processors. You find that many cache invalidations are occuring in each core's data cache. What program behavior could be causing these invalidations (in 20 words or less)?

cache ping-ponging and performance degradation

(b) You modify the program to fix this first performance issue. However, now you find that the program is slowed down by a global state update that must happen in only a single thread after every parallel computation. In particular, your program performs 90% of its work (measured as processor-seconds) in the parallel portion and 10% of its work in this serial portion. The parallel portion is perfectly parallelizable. What is the maximum speedup of the program if the multicore processor had an infinite number of cores?

10x speedup (Amdahl)

(c) How many processors would be required to attain a speedup of 4?

6 processors (10% serial; 15% wall-clock time for parallel, from 90% -¿ split by 6)

(d) In order to execute your program with parallel and serial portions more efficiently, your corporation decides to design a custom heterogeneous processor.

- This processor will have one large core (which executes code more quickly but also takes greater die area on-chip) and multiple small cores (which execute code more slowly but also consume less area), all sharing one processor die.
- When your program is in its parallel portion, all of its threads execute **only** on small cores.
- When your program is in its serial portion, the one active thread executes on the large core.
- Performance (execution speed) of a core is proportional to the square root of its area.
- Assume that there are 16 units of die area available. A small core must take 1 unit of die area. The large core may take any number of units of die area n^2, where n is a positive integer.
- Assume that any area not used by the large core will be filled with small cores.

(i) How large would you make the large core for the fastest possible execution of your program?

speedup = 1 / (10% / sqrt(L) + 90% / (16-L))

maximize speedup -¿ minimize 10/sqrt(L) + 90/(16-L)

L = k^2, k integer

minimize 10/k + 90/(16-k^2) k = 2: 5 + 90/12 = 5 + 7.5 = 12.5 k = 3: 3.33 + 90/7 = 3.33 + 12.... = 15... k = 1: 10 + 90/15 = 10 + 6 = 16

k = 2 -¿ L = 4 (large core uses 4 units of area) speedup is 8

(ii) What would the same program's speedup be if all 16 units of die area were used to build a homogeneous system with 16 small cores, the serial portion ran on one of the small cores, and the parallel portion ran on all 16 small cores?

speedup is 1 / (.1 + .9/16) = 6.4

(iii) Does it make sense to use a heterogeneous system for this program which has 10% of its work in serial sections?
Why or why not?

(e) Now you optimize the serial portion of your program and it becomes only 4% of total work (the parallel portion is the remaining 96%).

(i) What is the best choice for the size of the large core in this case?

L = 1 (make it the same size as the small cores)

(ii) What is the program's speedup for this choice of large core size?

1 / (0.02 / 1 + 0.98 / 15) = 9.61

(iii) What would the same program's speedup be for this 4%/96% serial/parallel split if all 16 units of die area were used to build a homogeneous system with 16 small cores, the serial portion ran on one of the small cores, and the parallel portion ran on all 16 small cores?

1 / (0.04 + 0.96 / 16) = 1 / (0.1) = 10

(iv) Does it make sense to use a heterogeneous system for this program which has 4% of its work in serial sections?
Why or why not?

4 Topologies

Suppose you would like to connect 625 processors, and you are considering three different topologies: bus, point-to-point network, and mesh.

Describe one disadvantage of each:

(i) A Single Bus. **Very high bus contention (with 625 processors)**

(ii) A Point-to-Point Network. **Many individual wires (between every pair of nodes)**

(iii) A 25x25 Mesh. **High complexity**

Which one would you choose? Why?

A 25x25 Mesh: it is performance-scalable to 625 processors and not cost-prohibitive.

5 Building Multicore Processors

You are hired by Amdahl's Nano Devices (AND) to design their newest multicore processor.
Ggl, one of AND's largest customers, has found that the following program can predict people's happiness.

```
for (i = 12; i < 2985984; i++) {
  past = A[i-12]
  current = A[i]
  past *= 0.37
  current *= 0.63
  A[i] = past + current
}
```

`A` is a large array of 4-byte floating point numbers, gathered by Ggl over the years by harvesting people's private messages. Your job is to create a processor that runs this program as fast as possible.

Assume the following:

- You have magically fast DRAM that allows infinitely many cores to access data in parallel. We will relax this strong assumption in parts (d), (e), (f).
- Each floating point instruction (addition and multiplication) takes 10 cycles.
- Each memory read and write takes 10 cycles.
- No caches are used.
- Integer operations and branches are fast enough that they can be ignored.

(a) Assuming infinitely many cores, what is the maximum steady state speedup you can achieve for this program? Please show all your computations.

The cycle counts are extra information that are not necessary. Instead, the loop body is sequential, while 12 consecutive iterations are parallel. Notice that the 13th iteration is dependent on the 1st iteration. Therefore 1/12 of the program is serial. Using Amdahls law, this solves to a maximum speedup of 12.

(b) What is the minimum number of cores you need to achieve this speedup?

12 cores are needed.

(c) Briefly describe how you would assign work to each core to achieve this speedup.

Assign iteration i to processor i%12.

It turns out magic DRAM does not exist except in Macondo[1]. As a result, you have to use cheap, slow, low-bandwidth DRAM. To compensate for this, you decide to use a private L1 cache for each processor. The new specifications for the DRAM and the L1 cache are:

- DRAM is shared by all processors. DRAM may only process one request (read or write) at a time.
- DRAM takes 100 cycles to process any request.
- DRAM prioritizes accesses of smaller addresses and write requests. (Assume no virtual memory)
- The cache is direct-mapped. Each cache block is 16 bytes.
- It takes 10 cycles to access the cache. Therefore, a cache hit is processed in 10 cycles and a cache miss is processed in 110 cycles.

All other latencies remain the same as specified earlier. Answer parts (d), (e), (f) assuming this new system.

[1] An imaginary town featured in *One Hundred Years of Solitude* by the late Colombian author Gabriel García Márquez (1927-2014).

(d) Can you still achieve the same steady state speedup as before? Circle one: YES NO

Please explain.

No. Notice that the serial bottleneck is now caused by the DRAM. In steady state, only one memory access is performed for every four iterations. So four iterations take 100 + 200 = 300 cycles. 12 iterations requires 900 cycles. In order to achieve 12x speedup, each core needs to complete each iteration in 75 cycles. However, fetching from memory alone requires 100 cycles. Therefore, it is not possible. By just saying that it is now slower than before is not enough, since the single core baseline is also slower.

(e) What is the minimum number of cores your processor needs to provide the maximum speedup?

3 cores are needed to take the maximum advantage of each cache hit, and a maximum speed up of almost 3 can be achieved. Core 1 calculates iterations i+0, i+1, i+2, i+3, core 2 calculates iteration i+4, i+5, i+6, i+7, and core 3 calculates iteration i+8, i+9, i+10, i+11.

(f) Briefly describe how you would assign work to each core to achieve this speedup.

3 cores are needed to take the maximum advantage of each cache hit, and a maximum speed up of almost 3 can be achieved. Core 1 calculates iterations i+0, i+1, i+2, i+3, core 2 calculates iteration i+4, i+5, i+6, i+7, and core 3 calculates iteration i+8, i+9, i+10, i+11.

6 Sequential Consistency

Two threads (A and B) are concurrently running on a dual-core processor that implements a *sequentially consistent* memory model. Assume that the value at address 0x1000 is initialized to 0.

Thread A

```
X1: st 0x1, (0x1000)
X2: ld $r1, (0x1000)
X3: st 0x2, (0x1000)
X4: ld $r2, (0x1000)
```

Thread B

```
Y1: st 0x3, (0x1000)
Y2: ld $r3, (0x1000)
Y3: st 0x4, (0x1000)
Y4: ld $r4, (0x1000)
```

(a) List all possible values that can be stored in `$r3` after both threads have finished executing.

0x1, 0x2, 0x3

(b) After both threads have finished executing, you find that (`$r1`, `$r2`, `$r3`, `$r4`) = (1, 2, 3, 4). How many different *instruction interleavings* of the two threads produce this result?

X1/X2 have to execute back-to-back. The same goes for X3/X4, Y1/Y2 and Y3/Y4. Let us call these instruction pairs as P, Q, R, and S. Six possible interleavings: P -¿ Q -¿ R -¿ S P -¿ R -¿ Q -¿ S R -¿ P -¿ Q -¿ S R -¿ P -¿ S -¿ Q R -¿ S -¿ P -¿ Q P -¿ R -¿ S -¿ Q

(c) What is the total number of all possible instruction interleavings? You need not expand factorials.

Eight instructions in total are executed. Among them, you must choose the four that are from Thread A. 8-Choose-4 = 8!/4!/4!.

(d) On a *non-sequentially consistent* processor, is the total number of all possible instruction interleavings less than, equal to, or greater than your answer to question (c)?

The same. Since all the memory accesses are to the same address, they cannot be reordered more aggressively without affecting the correctness within even a single thread.

7 Cache Compression

Cache compression is a promising technique to increase cache capacity and to decrease on-chip and off-chip bandwidth usage.

(a) For many cache lines, the values within the cache line have a low dynamic range i.e., the differences between values stored within the cache line are small. The following 16-byte cache line (which could be storing pointers to random entries in an array) is an example:

Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0x80000004	0x800000FF	0x8000001C	0x80000094

How would you compress this 16-byte cache line? Your compression method should also work on similar cache lines of low dynamic range. (The compressed cache line should be at most half the size of the uncompressed cache line.)

How would you decompress this cache line?

How many bytes is your compressed cache line?

Using the compression algorithm you designed above, compress the following cache line. Show your work and report the size of your compressed cache line.

Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0x80000004	0x8000FFFF	0x80008F12	0x80000444

List one advantage and one disadvantage of cache compression over no cache compression.

(b) Other cache lines may not be as straightforward to compress. However, they are still compressible using the compression algorithm from above with some small tricks. How would you compress the following cache line? (The compressed cache line should be at most half the size of the uncompressed cache line.)

Bytes 0-3	Bytes 4-7	Bytes 8-11	Bytes 12-15
0x19000001	0x16000001	0x17000001	0x18000001

How would you decompress this cache line?

How many bytes is your compressed cache line?

List one advantage and one disadvantage of using this compression method as opposed to the compression method from part (a).

8 RowClone

Recall that the RowClone idea presented in lecture performs the bulk copy or initialization of a page (or any arbitrary sized memory region) completely within DRAM, with support from the memory controller.

Suppose we have a cache-coherent system with six cores, a 32 KB L1 cache in each core, a 1MB private L2 cache per core, a 16MB shared L3 cache across all cores, and a single shared DRAM controller across all cores. The system supports RowClone as described in class. Physical page size is 8KB. MESI protocol is employed to keep the caches coherent.

Suppose Core 1 sends a request to the memory controller to Copy Physical Page 10 to Physical Page 12 via RowClone. Assume the two pages reside in the same DRAM subarray and the copy can be done with two consecutive ACTivate commands as a result.

a) To maintain correctness of data in such a system, what should the memory controller do before performing the RowClone request? Explain step by step. Be precise.

Step 1:

Step 2:

b) How much data transfer is eliminated from the main memory data bus with this particular copy operation? Justify your answer and state your assumptions.

Amount of data eliminated:

Justification:

9 BossMem

A researcher has developed a new type of nonvolatile memory, BossMem. He is considering BossMem as a replacement for DRAM. BossMem is 10x faster (all memory timings are 10x faster) than DRAM, but since BossMem is so fast, it has to frequently power-off to cool down. Overheating is only a function of time, not a function of activityan idle stick of BossMem has to power-off just as frequently as an active stick. When powered-off, BossMem retains its data, but cant service requests. Both DRAM and BossMem are banked and otherwise architecturally similar. To the researchers dismay, he finds that a system with 1GB of DRAM performs considerably better than the same system with 1GB of BossMem.

(i) What can the researcher change or improve in the core (he cant change BossMem or anything beyond the memory controller) that will make his BossMem perform more favorably compared to DRAM, realizing that he will have to be fair and evaluate DRAM with his enhanced core as well? (15 words or less)

Prefetcher degree or other speculation techniques so that misses can be serviced before memory powered off

(ii) A colleague proposes he build a hybrid memory system, with both DRAM and BossMem. He decides to place data that exhibits low row buffer locality in DRAM and data that exhibits high row buffer locality in BossMem. Assume 50% of requests are row buffer hits. Is this a good or bad idea? Show your work.

No, it may be better idea to place data with high row buffer locality in DRAM and low row buffer locality data in BossMem since row buffer misses are less costly

(iii) Now a colleague suggests trying to improve the last-level cache replacement policy in the system with the hybrid memory system. Like before, he wants to improve the performance of this system relative to one that uses just DRAM and he will have to be fair in his evaluation. Can he design a cache replacement policy that makes the hybrid memory system look more favorable?

In 15 words or less, justify NO or describe a cache replacement policy that would improve the performance of the hybrid memory system more than it would DRAM.

Yes, this is possible. Cost-based replacement where cost to replace is dependent on data allocation between DRAM and BossMem

(iv) In class we talked about another nonvolatile memory technology, phase-change memory (PCM). Which technology, PCM, BossMem, or DRAM requires the greatest attention to security?

What is the vulnerability?

PCM is nonvolatile and has potential endurance attacks.

(v) Which is likely of least concern to a security researcher?

DRAM is likely least vulnerable, as BossMem also has nonvolatility concerns.