CSE 560 – Practice Problem Set 9 Solution

1. Suppose there are 10 processors on a bus that each try to lock a variable simultaneously. Assume that the primitive instructions available are a load-lock (`ll`) and a store-conditional (`sc`). Further, assume that each bus transaction (read miss or write miss) is 100 clock cycles long. You can ignore the time of the actual read or write of a lock held in the cache, as well as the time the lock is held (they won't matter much!).

   Determine the number of bus transactions required for all 10 processors to each acquire and release the lock, assuming they are all spinning when the lock is released at time 0. About how long will it take to process the 10 requests? Assume that the bus is totally fair so that every pending request is serviced before a new request and that the processors are equally fast.

   When *i* processes are contending for the lock, they perform the following sequence of actions, each of which generates a bus transaction:

   - *i* bus transactions to access the lock (execute the `ll` instruction)
   - *i* store conditional operations (`sc` instruction) to try to lock the lock
   - 1 store to release the lock

   Thus for *i* processes, there are a total of $2i + 1$ bus transactions each time that *i* processes are contending for the lock. Note that this assumes the critical section is negligible, so that the lock is released before any other processes whose store conditional failed attempt another load linked.

   The above time results in one of the *i* contending processes to successfully acquire the lock, the remaining $i - 1$ processes are unsuccessful.

   For each of *n* processes to successfully acquire the lock, the total number of bus operations is:

$$\sum_{i=1}^{n} (2i + 1) = n(n + 1) + n = n^2 + 2n$$

   For 10 processes there are 120 bus transactions requiring 12,000 clock cycles.

2. Assume that variables `x1` and `x2` are in the same cache block, which starts in the shared (S) state in the caches of processor 1 (P1) and processor 2 (P2). I.e., at some point in the past, both variables were read by both processors. The system uses a traditional MSI (3-state) cache coherence protocol.

   Assuming the following sequence of events, identify each access as a true sharing miss, a false sharing miss, or a hit. Any miss that would occur if the block size were one word (the size of `x1` or `x2`) is designated a true sharing miss.

Time	**P1**	**P2**
1	Write `x1`	
2		Read `x2`
3	Write `x1`	
4		Write `x2`
5	Read `x2`	

   Here are the classifications by time step:

   (1) This event is a true sharing miss, since `x1` was read previously by P2 and needs to be invalidated from P2.
   (2) This event is a false sharing miss, since `x2` was invalidated by the write of `x1` by P1 at time step 1, but that value of `x1` is not used in P2.
   (3) This event is a false sharing miss, since the block containing x1 is marked shared due to the read in P2 (at time step 2), but P2 did not read x1. The cache block containing x1 will be in the shared state after the read by P2; a write miss (upgrade) is required to obtain exclusive access to the block.
   (4) This event is a false sharing miss for the same reason as time step 3.
   (5) This event is a true sharing miss, since the value being read was written by P2 in time step 4.

3. For this problem, assume that we are dealing with a bus-based shared memory multiprocessor using the MESI protocol.

   Each processor core in the system is identical, and the following information applies to each processor core in the system individually. Instructions are 32 bits wide. 50% of the instructions executed are loads or stores. Of these loads and stores, on average 70% are reads to private data, 20% are writes to private data, 8% are reads to shared data, and 2% are writes to shared data.

   Each processor has a single-level split instruction/data cache. The instruction cache is 16 KB, two-way associative, and has 16 byte lines. The data cache is 16 KB, direct mapped, and also has 16 byte lines. The hit rates in the caches are as follows: 97% for private data, 95% for shared data, and 98.5% for instructions. Cache hit time is one cycle.

   The system bus has 64 data lines and 32 address lines. The bus is atomic. The bus is clocked at one-half the speed for the processor. For reads, memory responds with data 12 bus cycles after being presented the address, and supplies one block of data per bus cycle after that. For writes, both address and data are presented to memory at the same time. Thus a single-word write consumes 1 bus cycle, while a 16-byte write consumes two cycles. Assume all requests are satisfied by the memory system, not by other caches.

   The processor CPI is 2.0 before considering memory penalties.

   (a) We want to place as many processors as possible on the bus. What is the bus utilization of a single processor if the caches are write-through with write-allocate strategy? How many of these processors can the bus support before it saturates?

   For this part of the problem, assume that a write to the bus automatically invalidates any other existing copies of the data being written. Ignore bus contention and coherence messages received from other processors (e.g., remote snoops), but do consider coherence traffic generated by the processor.

   The key to approaching this problem is to recognize that the true CPI (including memory penalties) includes three components: the 2.0 base CPI (CPI_{base}), plus a component due to bus stalls on instruction cache misses (CPI_{busI}), plus a component due to bus stalls on data cache misses (CPI_{busD}). Once we have computed these bus CPI components, we can compute the bus utilization as the average fraction of the per-instruction time that the bus is busy, or

$$Bus\ Utilization = \frac{CPI_{busI}+CPI_{busD}}{CPI_{base}+CPI_{busI}+CPI_{busD}}.$$

   Perform all of your calculations in processor cycles.

   It takes 26 processor cycles to fetch a cache line from the memory system, 24 to get the first 8 bytes and 2 to get the remaining 8 bytes. Since the bus is atomic, it is busy during that entire time. Call this t_{missF}, miss time for a memory fetch.

First consider the instruction cache. 1.5% of instructions miss the cache, and each miss requires t_{missF} cycles.

$CPI_{busI} = 0.015 \times t_{missF} = 0.015 \times 26 = 0.39$

Next consider the data cache, which is a bit trickier. First, there is an overall factor of 0.5, since only half of the instructions are loads or stores:

$CPI_{busD} = 0.5 \times (\ldots)$

Of these, 70% are reads to private data, of which 3% miss in the cache, each using tmissF cycles of bus time to fetch a cache line:

$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missF}) + \ldots]$

Next, 20% of the memory operations are writes to private data. 97% of these hit in the cache, but because the cache is write-through, each hit still consumes 2 processor cycles worth of bus activity to write the word through to memory. 3% of the private writes miss, causing tmissF cycles of bus time to fetch the cache line accessed (since the cache is write-allocate) plus two more cycles to write-through the new value:

$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missF}) + 0.2 \times (0.97 \times 2 + 0.03(t_{missF} + 2)) + \ldots]$

Next, 8% of the memory operations are reads to shared data. These are identical to reads to private data, except that the miss rate is now 0.05:

$$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missF}) + 0.2 \times (0.97 \times 2 + 0.03(t_{missF} + 2)) + (0.08 \times 0.05 \times t_{missF}) + \ldots]$$

Finally, 2% of accesses are writes to shared data, which again behave identically to writes to private data, since the cache is write-through and the bus write serves as an invalidation to other processors:

$$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missF}) + 0.2 \times (0.97 \times 2 + 0.03 \times (t_{missF} + 2)) + (0.08 \times 0.05 \times t_{missF}) + 0.02 \times (0.95 \times 2 + 0.05 \times (t_{missF} + 2))]$$

$CPI_{busD} = 0.636$

We can now put it all together:

$$Bus\ Utilization = \frac{(0.39 + 0.636)}{(2.0 + 0.36 + 0.636)} = 0.34$$

Therefore, the bus is 34% saturated by a single processor core. This means it can support at most $\left\lfloor \frac{1}{0.34} \right\rfloor = 2$ processors.

(b) How many processors can the bus support without saturating if the caches are write-back (and write-allocate)? Assume that the probability of having to replace a dirty block in the cache on a miss that fetches a new block is 0.3. Also assume a MESI protocol, and again ignore bus contention and coherence messages received from other processors (e.g., remote snoops), but do consider coherence traffic generated by the processor. Assume that the cache protocol supports upgrades, so a write hit to a shared block causes an invalidate transaction only, taking one bus cycle.

You may make the following two assumptions: (1) writeback is not overlapped with reading the new data on the bus, and (2) all write hits to shared data require an upgrade transaction. As in the previous part, $CPI_{base} = 2.0$, and all calculations should be in processor cycles.

Since the I and D caches are split, and instructions are never modified, instruction misses cannot cause writebacks. Thus, CPI_{busI} is the same as in part (a):

$CPI_{busI} = 0.39$

We will define the data cache miss time, t_{missD}, to be the average time spent holding the bus when servicing a data cache miss. This includes the fetch time from above, t_{missF}, plus a term to account for the potential of writeback. We are told that on average 30% of cache misses cause a writeback, and a writeback takes four processor cycles on the bus (two for the address and first 8 bytes of the line, and two for the remaining 8 bytes), so

$t_{missD} = t_{missF} + 0.3 \times 4 = 26 + 0.3 \times 4 = 27.2$

Now, let's compute CPI_{busD}. First consider the cache misses. In all cases, a cache miss behaves the same (read and write misses both need to first read a line of data, and this read encapsulates all coherence actions), taking t_{missD} cycles, so for cache misses we get:

$$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missD}) + (0.2 \times 0.03 \times t_{missD}) + (0.08 \times 0.05 \times t_{missD}) + (0.02 \times 0.05 \times t_{missD}) + \ldots]$$

We haven't considered cache hits yet. All read hits are absorbed entirely by the cache. Because the cache is writeback, write hits to private data behave the same as read hits. Only write hits to shared data propagate to the bus, and they only require a single bus transaction (2 processor cycles) to send out an upgrade transaction. Thus we can complete CPI_{busD} as:

$$CPI_{busD} = 0.5 \times [(0.7 \times 0.03 \times t_{missD}) + (0.2 \times 0.03 \times t_{missD}) + (0.08 \times 0.05 \times t_{missD}) + (0.02 \times 0.05 \times t_{missD}) + (0.02 \times 0.95 \times 2)]$$

$CPI_{busD} = 0.454$

Use the same construction as above to get bus utilization:

$$Bus\ Utilization = \frac{(0.39 + 0.454)}{(2.0 + 0.36 + 0.454)} = 0.297$$

So we can support 3 processors in this configuration before the bus saturates.

(c) Assume we add a unified write-back, write-allocate second-level cache to each processor (assuming a write-through, *no-write-allocate* first level cache with the same parameters as before). The L2 line size is 32 bytes. The local miss rate for all traffic to the L2 cache is 10%. The hit time to the L2 cache (which includes the time to transfer data to the L1 cache, but not the L1 hit time) is 6 cycles. The L2 cache is clocked at the same speed as the processor. Inclusion is maintained between the caches. Again, assume that the probability of having to replace a dirty block in the L2 cache on a miss that fetches a new block is 0.3.

What is the bus utilization of a single processor now? How many of these processors can the bus support without saturating?

Start by computing the time associated with a miss in L2, t_{missL2}, which takes t_{missF} + 4 cycles (since the L2 cache line is twice as long as the L1 cache line) plus the time associated with a writeback (8 cycles):

$t_{missL2} = (t_{missF} + 4) + 0.3 \times 8 = 32.4$

CPI_{memI} (which incorporates the entire memory hierarchy for misses in the I cache) must now take into account a possible writeback from L2 (since it is unified) and the 6 cycles spent on a hit in L2 for a miss in L1:

$CPI_{memI} = 0.015 \times (6 + 0.1 \times t_{missL2}) = 0.1386$

For CPI_{memD} we begin with the cache misses in L1, which will have the following value for t_{missD}:

$t_{missD} = 6 + 0.1 \times t_{missL2} = 9.24$

We now have the same expression as before, using the new value of t_{missD}:

$$CPI_{memD} = 0.5 \times [(0.7 \times 0.03 \times t_{missD}) + (0.2 \times 0.03 \times t_{missD}) + (0.08 \times 0.05 \times t_{missD}) + (0.02 \times 0.05 \times t_{missD}) + \ldots]$$

Write hits in the L2 cache to shared data are a bit tricky. We know that 95% of them hit data already in the L1 cache, requiring 2 processor cycles worth of bus activity for the

upgrade, as before. However, there is the possibility that the write may miss in the L1 but hit in the L2; this case also requires 2 cycles for an upgrade transaction. Thus we add in the term 0.02×(0.95+0.05×0.9)×2, to get:

$CPI_{memD} = 0.5 \times [(0.7 \times 0.03 \times t_{missD}) + (0.2 \times 0.03 \times t_{missD}) +$
$(0.08 \times 0.05 \times t_{missD}) + (0.02 \times 0.05 \times t_{missD}) + 0.02 \times (0.95 + 0.05 \times 0.9) \times 2]$
$CPI_{memD} = 0.16774$

We might be tempted to use the following equation for bus utilization:

$$Bus\ Utilization = \frac{CPI_{memI} + CPI_{memD}}{CPI_{base} + CPI_{memI} + CPI_{memD}}$$

But we would be wrong, because the bus is not occupied for the entire time of an L1 cache miss. We still need values for CPI_{busI} and CPI_{busD}. Fortunately, all of the reasoning above for CPI_{memI} and CPI_{memD} still works, as long as we remove the 6 cycles needed for an L2 hit on an L1 miss (i.e., use $0.1 \times t_{missL2}$ instead of t_{missD}):

$CPI_{busI} = 0.015 \times 0.1 \times t_{missL2} = 0.0486$

$CPI_{memD} = 0.5 \times [(0.7 \times 0.03 \times 0.1 \times t_{missL2}) + (0.2 \times 0.03 \times 0.1 \times t_{missL2}) +$
$(0.08 \times 0.05 \times 0.1 \times t_{missL2}) + (0.02 \times 0.05 \times 0.1 \times t_{missL2}) + 0.02 \times (0.95 + 0.05 \times 0.9) \times 2]$
$CPI_{memD} = 0.07174$

We use the following formula for bus utilization:

$$Bus\ Utilization = \frac{CPI_{busI} + CPI_{busD}}{CPI_{base} + CPI_{memI} + CPI_{memD}}$$

Which gives

$$Bus\ Utilization = \frac{(0.0486 + 0.07174)}{(2.0 + 0.1386 + 0.16774)} = 0.052$$

So we can support 19 processors in this configuration before the bus saturates.

(d) Finally, compute the average memory access time, *in processor cycles*, for the processor described in part (c). Be sure to include both instruction and data references. First show the equation, and then the numerical results.

For a two-level cache of the type in this problem, the general formula for average memory access time, t_{mem}, is:

$$t_{mem} = t_{hitL1} + MissRate_{L1} \times (t_{hitL2} + MissRate_{L2} \times t_{missL2})$$

To simplify the calculation, we will separately calculate the instruction memory access time, t_{memI}, and data memory access time, t_{memD}. The total tmem is simply the weighted average taking into account the fact that only half the instructions are data accesses:

$$t_{mem} = \frac{t_{memI} + 0.5 \times t_{memD}}{1.5}$$

For the instructions:

$t_{memI} = 1 + 0.015 \times (6 + 0.1 \times t_{missL2})$

For the data, we can split up t_{memD} into four components corresponding to the different types of memory access:

$t_{memD\text{-}read\text{-}private} = 1 + 0.03 \times (6 + 0.1 \times t_{missL2})$
$t_{memD\text{-}read\text{-}shared} = 1 + 0.05 \times (6 + 0.1 \times t_{missL2})$
$t_{memD\text{-}write\text{-}private} = 1 + 0.03 \times (6 + 0.1 \times t_{missL2})$
$t_{memD\text{-}write\text{-}shared} = 1 + 0.05 \times (6 + 0.1 \times t_{missL2}) + (0.95 + 0.05 \times 0.9) \times 2$

Then

$t_{memD} = 0.7 \times t_{memD\text{-}read\text{-}private} + 0.08 \times t_{memD\text{-}read\text{-}shared} + 0.2 \times t_{memD\text{-}write\text{-}private} + 0.02 \times t_{memD\text{-}write\text{-}shared}$

Plugging in all the numbers gives

t_{memI} = 1.1386 cycles
t_{memD} = 1.3355 cycles

and

t_{mem} = 1.204 cycles