Name:

CS / EE 6810 Final Exam - Dec 10th 2013

Notes: This is an open notes and open book exam. If necessary, make reasonable assumptions and clearly state them. The only clarifications you may ask for during the exam are definitions of terms. You may use calculators. Laptops are allowed if you want to browse through class material (textbook CD, notes, etc.) or if you want to use a calculator, but you are not allowed to use search features. Complete your answers in the space provided (including the back-side of each page). Confirm that you have 9 questions on 6 pages, followed by a blank page. Turn in your answer sheets before 12:20pm.

1. **Cache Misses and Access Times.** Consider a processor and a program that would have an IPC of 1 with a perfect 1-cycle L1 cache. Assume that each additional cycle for cache/memory access causes program execution time to increase by one cycle. Assume the following MPKIs and latencies for the following caches for this program:

   - L1: 32KB: 1-cycle: 60 MPKI
   - L2: 512KB: 15-cycle: 30 MPKI
   - L3: 4MB: 30-cycle: 15 MPKI
   - Memory: 400 cycles

   Estimate the number of cycles required to execute 1000 instructions of this program. **(5 points)**

   *Solution: Execution time = 1000 + 60*15 + 30*30 + 15*400 = 8800 cycles.*

2. **Transactional Memory.** List five additions to the hardware that are required to support hardware transactional memory. Provide a one-sentence explanation (each) for why they are required. **(5 points)**

   Solution:

   (a) A register checkpoint so you can roll back the register state on an abort.

   (b) A read bit with every cache block so you can track the blocks read by the transaction.

   (c) A write bit with every cache block so you can track the blocks written by the transaction.

   (d) Logic to check for conflicts (on every snoop or every request from the directory).

   (e) Mechanisms to detect starvation; requires a count of the number of aborts or NACKs for a transaction.

   (f) Mechanisms to detect deadlock; in an eager-eager system, must see if a transaction has received a NACK from an older transaction and sent a NACK to an older transaction.

   (g) In a lazy-lazy system, an arbiter is required to grant permissions for commit.

3. **Memory SMBs.** What is the boost in memory capacity and memory rd/wr bandwidth provided by using an SMB? Assume that a DDR3 channel requires 64 data wires, 24 addr/cmd wires, and runs at a frequency of 800 MHz (DDR). Assume that the SMB connects to the processor with a 16-bit upstream link (to the processor) and a 12-bit downstream link (away from the processor). The links run at frequencies of 4.8 GHz (no DDR). Assume that two DDR3 channels can connect to an SMB. Assume that 40% of the downstream link's bandwidth is used for commands and addresses. **(10 points)**

Solution:

*The rd/wr bandwidth per pin of an SMB system = (16 + 0.6*12)*4.8G / 28 pins*

*The rd/wr bandwidth per pin of a DDR3 system = 64*0.8G*2 / 88 pins*

The ratio of SMB to DDR3 bandwidth is: 3.4

Assume that a memory capacity D can be connected to one DDR3 channel.

The memory capacity per pin of an SMB system = 2D/28.

The memory capacity per pin of a DDR3 system = D/88.

The ratio of SMB to DDR3 memory capacity is: 6.3

4. **Consistency Models.**

   Why should a hardware designer understand the concept of consistency models?

   Why should a compiler designer understand the concept of consistency models?

   Why should an application developer (programmer) understand the concept of consistency models?

   (10 points)

   Solution:

   A hardware designer must have primitives in place to support sequential consistency, e.g., having an ACK for coherence invalidations, not starting a coherence transaction unless the previous transaction on that block has finished, fence instructions. The hardware designer must articulate the hardware re-orderings possible so that a programmer can reason about what can happen when fences are not used in racy code.

   The compiler may be asked to introduce fence instructions at appropriate places, e.g., around lock and unlock statements. The compiler may also be asked to automatically identify racy code and insert the necessary fences. The compiler must be careful to not hoist instructions above a fence.

   The programmer must be able to reason about the behavior of a program under the sequentially consistent model. The programmer must know that bad things can happen if racy code is not surrounded by locks. If racy code is left unprotected by locks (for performance reasons), the programmer may have to dig up the consistency model for the system and reason about how it might affect program outputs.

5. **DRAM Scheduling.** For the following memory access pattern, estimate when each memory access completes for two different scheduling mechanisms: open-page policy and close-page policy. You are allowed to re-order requests already waiting in the memory controller. The access pattern only specifies the row being touched. All accesses are to the same bank. Assume that bus latencies are zero. Assume that the bank is already precharged at time 0. Assume that precharge takes 20 ns, loading a row buffer takes 20 ns, and cache line transfer to output pins also takes 20 ns. **(10 points)**

Row being accessed	Arrival time at memory controller	Open-Page	Close-Page
X	10 ns		
X	70 ns		
Y	90 ns		
X	100 ns		
Y	180 ns		

Solution:

Row being accessed	Arrival time at memory controller	Open-Page	Close-Page
X	10 ns	50 ns	50 ns
X	70 ns	90 ns	110 ns
Y	90 ns	150 ns	190 ns
X	100 ns	210 ns	130 ns
Y	180 ns	270 ns	210 ns

6. **Memory Channels.** Modern systems have processors with four memory channels. Assume that a memory channel has 64 bits for data and 24 bits for address and command. Assume that a memory channel is connected to eight x8 memory chips.

   Consider a new memory system that shrinks the data bus of the memory channel to 32 bits. Assume that such a channel would be connected to four x8 memory chips.

   What are the pros and cons of the new memory system? **(10 points)**

   Solution:

   Pros:

   (a) *The energy per cache line fetch is reduced (only 4 chips activated and precharged).*

   (b) *The parallelism in the system is higher (more ranks and hence more banks). This can lead to better bus utilization and lower queuing delays.*

   Cons:

   (a) *Fetching a cache line takes longer (16 transfers on the bus instead of 8).*

   (b) *More processor pins are required for a given memory capacity and memory rd/wr bandwidth because the overhead of the address/command signals goes up (24 addr/cmd for every 32 bits of data).*

7. **Coherence Protocols.** Consider a distributed shared memory multiprocessor that implements a write-invalidate directory-based cache coherence protocol, similar to the one assumed in HW9. Four processors are connected with a point-to-point interconnect. For each of the events below, explain the coherence protocol steps (does the cache flag a hit/miss, what request is placed on the network, what messages are sent, how is directory state updated, is a writeback required, etc.) and mention the state of the data block X in the caches of P1, P2, P3. Assume that the home nodes for memory locations X and Y are both associated with processor P4. Assume that X and Y are not in any of the caches at the start of the sequence, the caches are direct-mapped, each cache block only stores one word, and words X and Y map to the same set in each cache (X and Y cannot co-exist in a cache at any time). **(20 points)**

   - P1: Read X
   - P2: Read Y
   - P2: Write X
   - P3: Read X
   - P3: Write Y

   Solution:

   P1 has a read miss. A read-miss request is sent to the directory. The directory sees that X is not cached. It sets the state of X to S: P1. *The block is sent to P1. P1 keeps X in its cache in* Shared *state.*

   P2 has a read miss. A read-miss request is sent to the directory. The directory sees that Y is not cached. It sets the state of Y to S: P2. *The block is sent to P2. P2 keeps Y in its cache in* Shared *state.*

   P2 has a write miss. A write-miss request is sent to the directory. The directory sends an INV to P1. P1 sends an ACK to the directory. P1 now has X in invalid state in its cache. The directory state for X is changed to M: P2. *The block is sent to P2. P2 caches X in* Modified state. *P2 silently evicts Y.*

   P3 has a read miss. A read-miss request is sent to the directory. The directory forwards the request to P2. P2 downgrades its cache state for X to shared. P2 sends the block to the directory. A writeback is performed in memory. The directory state for X is changed to S: P2, P3. *The block is sent to P3. P3 caches X in* Shared *state.*

   P3 has a write miss. A write-miss request is sent to the directory. The directory for Y indicates that the blocks is in shared state in P2. So an invalidation is sent to P2. P2 doesn't really have the block. P2 sends an ACK to the directory. The directory state for Y is changed to M: P3. *The block is sent to P3. P3 caches Y in* Modified *state. P3 silently evicts X.*

8. **On-Chip Networks.** Consider a processor and application where 40% of the application's execution time can be attributed to delays in an on-chip network. The processor has 64 tiles connected with a ring network and the communication pattern among the tiles is random. What is the improvement in application performance if the ring is replaced with a grid topology (no wraparound)? Show your answer for two cases. Case 1: at least one link on the network is saturated in both topologies. Case 2: no link on the network is saturated in either topology. **(15 points)**

   Solution:

   Case 1: Since some links are saturated, the application is limited by bandwidth. Therefore, the bisection bandwidth (BB) is a good measure of how much faster a network can process a given set of messages (for random traffic). The BB for the ring is 2 links. The BB for the 8x8 grid is 8 links. So the grid should be able to deliver the messages in 4x less time than the ring. So the application takes

60 + (40/4) = 70 units of time to finish. Performance is 1/exectime, so if the initial performance was 1, the new performance is 1/0.7 = 1.41, i.e., a 41% improvement.

*Case 2. Since the links are not saturated, the measure of performance is the average network delay for a message. In a 64-node ring, the longest message takes 32 hops and the average message takes 16 hops. In an 8x8 mesh, the the longest message takes 14 hops and the average message takes 7 hops. So the application takes 60 + (40*7/16) = 77.5 units of time to finish. Performance improvement is 29%.*

In the above analysis, I assumed that a link in the ring and grid have the same bandwidth and a hop on either network takes the same time (even though a router in the grid is more complex).

9. **Loop unrolling and Software Pipelining.** Consider the loop below, running on an in-order processor with bypassing. The processor has four pipelines: one to handle FP operations (DADD) and three others that can handle everything else. The processor does not have branch delay slots, but is equipped with a very good branch predictor (assume perfect prediction). The compiler produces VLIW packets with one instruction for each of the four pipelines in every packet (some of these instructions may be NOPs). Show the highest performing code that the compiler can generate for this source code. The code must fit in 10 packets or less. Assume the following delays between dependent instructions:

   (a) Load feeding any instruction: 1 stall cycle

   (b) FP ADD feeding store: 4 stall cycles

   (c) Int add feeding any instruction, including branch: 0 stall cycles

   Below is the source code and default assembly code for the loop.

```
    Assembly code                      Source code
Loop: L.D     F1, 0(R1)            for (i=1000;i>0;i--) {
      DADD    F3, F1, F2             x[i] = s + x[i];
      S.D     F3, 0(R1)            }
      DADDUI  R1, R1, #-8
      BNE     R1, R2, Loop
```

   (15 points)

   Solution: For such a multi-pipeline model, SWP can achieve higher performance. SWP can be combined with unrolling to avoid all stall cycles. Consider the following packets of code. Each packet has 4 instructions, one for each pipeline. The terms "i" and "i+1", etc. specify the corresponding iteration in the original code. The fifth packet does the LD DADD and SD for three consecutive original iterations. This is the SWP that we've seen before in class. But I have to ensure that the DADD is separated from its consuming store by 4 cycles. So I unroll this new loop and do 4 more such packets before looping back. The fourth pipeline can be used to do the DADDUI and BNE in the last two packets. Further unrolling doesn't help maximize the throughput rate.

```
LD i+4  DADD  i+3   SD i+2     NOP
LD i+3  DADD  i+2   SD i+1     NOP
LD i+2  DADD  i+1   SD i       NOP
LD i+1  DADD  i     SD i-1     DADDUI
LD i    DADD  i-1   SD i-2     BNE
```

If I had instead tried to do this with unrolling (no SWP), the code would have looked like this (max 10 packets). You'll see lots of NOPs.

```
LD i     NOP          NOP         NOP
LD i+1   NOP          NOP         NOP
LD i+2   DADD i       NOP         NOP
NOP      DADD i+1     NOP         NOP
NOP      DADD i+2     NOP         NOP
NOP      NOP          NOP         NOP
NOP      NOP          NOP         NOP
NOP      NOP          SD i        NOP
NOP      NOP          SD i+1      DADDUI
NOP      NOP          SD i+2      BNE
```