Name ____________________________

Computer System Architecture
6.823 Quiz #4
May 14th, 2014
Professors Daniel Sanchez and Joel Emer

This is a closed book, closed notes exam.

80 Minutes
15 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Show your work to receive full credit.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

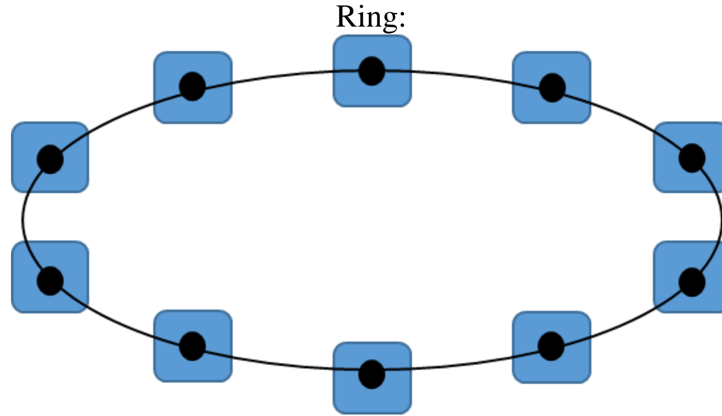
Part A	________	25 Points
Part B	________	55 Points
Part C	________	20 Points
TOTAL	________	**100 Points**

Name ______________________________

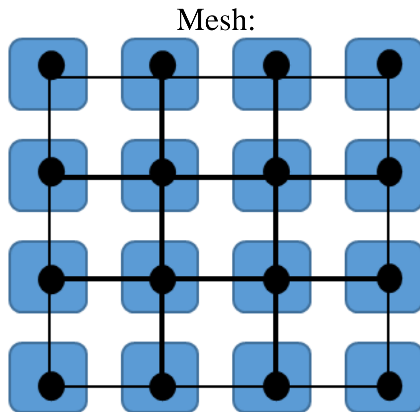
Part A: Network Effects (25 pts)

You are choosing between several network topologies for your on-chip network, shown below.

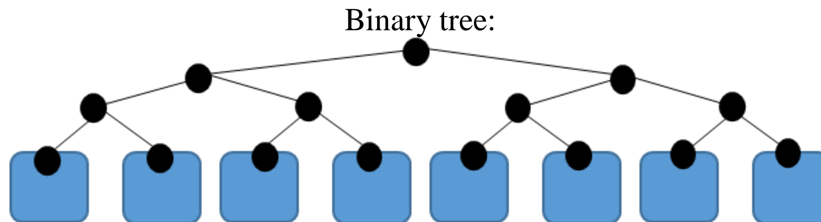
Ring:



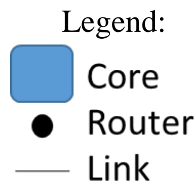
Mesh:



Binary tree:



Legend:



Question 1 (10 points):

Your first task is to evaluate these topologies along several important dimensions. Fill in the table below as a function of the number of nodes in the network, *N*. You can safely assume *N* is an even power of 2, giving a complete mesh and binary tree. *For partial credit, give the asymptotic growth instead.*

	Ring	**Mesh**	**Tree**
Number of links	N	$2(N - \sqrt{N})$	$2N - 2$
Diameter	$\frac{N}{2}$	$2(\sqrt{N} - 1)$	$2\log_2 N$
Average distance	$\frac{N}{4}$	$\frac{2(\sqrt{N} - 1)}{3}$	$\approx \log_2 N$
Bisection bandwidth	2	$\sqrt{N}$	1

To compute the number of links, consider how many outgoing channels leave each router, and divide by 2 since they are bidirectional links. Then compute how many routers you need for *N* cores. In all cases, the number of links grows proportional to *N* since we are considering topologies where the number of channels per router is constant with respect to N.

Ring: With a bidirectional ring, you have *N* routers and 2 outgoing channels for each, giving *N* bidirectional links. At worst you have to go half way around the ring to reach another node, for a diameter of $N/2$. On average you will need to do half of this, or $N/4$. To split the network in half, you must cut 2 links, for a bisection bandwidth of 2.

Mesh: You have 4 outgoing channels for each router, except those on the perimeter of the mesh. So you expect the answer to be $2N$ minus those on the perimeter, or the number of channels on the edge divided by 2, $2\sqrt{N}$. The diameter of a mesh is the distance from the top-left to bottom-right, which is twice the number of links in each dimension. The average distance is complicated because, unlike the ring, the starting position makes a difference. Without going into details, this gives you length/3, and the length here is the length of a dimension. You have to traverse this for each dimension, so we multiply by two. Finally the bisection bandwidth is just the bandwidth across an *even* split of nodes in the network, or the length of a dimension, $\sqrt{N}$.

Tree: The tree is the most complicated to compute the number of links. To form a tree of size N, we combine two trees of size $N/2$. This gives a recurrence relation: the number of links for a tree of size N is twice the number of links for $N/2$ nodes, plus 2. Or in other

words, $L(N) = 2L(N/2) + 2$. This yields the solution $L(N) = 2N - 2$. The diameter is the distance traversing up to the top of the tree and back down again—twice the logarithm of the number of nodes. The average distance is similar, since *half the nodes are contained in the other side of the tree*, so the distance is roughly proportional to $\log N$. The exact answer for this is very complicated and we were generous in awarding points. Finally, the bisection bandwidth is just a single link, since cutting any link at the root divides the network.

Another question we could have asked, and is worth thinking about, is scenarios when each of these topologies could be better than the others.

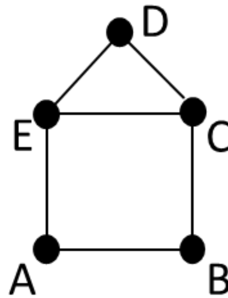
The ring requires the fewest links and simplest routers of the three topologies here. For small systems, it is probably the best choice, since the difference between diameter and average distance only becomes substantial as N grows large. (At small N, rings can have lower distances actually.)

A mesh is probably best for large systems since distances grow slowly—with the square root of N—and bisection bandwidth also grows with N—although not proportionally. The mesh requires the most complicated routers, however, so there are non-trivial costs in adopting a mesh topology.

Finally the tree has the best asymptotic growth of distance and so might be beneficial for large systems, but only if bandwidth requirements are low since the bisection bandwidth is so low. Also note that we have computed average distance assuming uniform traffic. A well-designed parallel algorithm should account for locality, in which case the picture gets a lot murkier.

Name ___________________________

In a sudden flash of inspiration, you decide to use the following topology:



Having decided upon a topology, you now want to make sure your system works properly. All links are bidirectional.

Question 2 (5 points):

Show how deadlock could arise in the network by drawing an example on the graph above. Explain your answer in one or two sentences.

If B is sending a message to C through BAEC, and E is sending a message to A through ECBA, then B can grab BA and E can grab EC and the system is deadlocked.

Question 3 (10 points):

Draw the channel dependency graph (CDG) for your topology.

The CDG connects *links* on the network to each other if one link can route to another. Links can be grouped into four categories on this topology based on their incoming/outgoing links.

Incoming/Outgoing	Links
1/1	AB, BA
1/2	AE, BC, DE, DC
2/1	EA, CB, ED, CD
2/2	EC, CE

The adjacency list for this CDG is below. (Just think: from this link, where can I go?) Note the symmetry.

AB: BC
BA: AE
AE: ED, EC
BC: DC, CE
DE: EC, EA
DC: CE, CB
EA: AB
CB: BA
ED: DC
CD: DE
EC: CD, CB
CE: ED, EA

Show an example of how to eliminate routes to prevent deadlock on the CDG.

There are many possible answers to this. You must eliminate edges until the CDG is a DAG.

Name ____________________________

Part Deux: Synchronicity (55 points)

(Same as Handout #14.) You are writing a queue to be used in a multi-producer/single-consumer application. (Producer threads write messages that are read by one consumer.) We assume here a queue with infinite space. The basic code is shown below.

`TST rs, Imm(rt)` is the test-and-set instruction, which *atomically* loads the value at `Imm(rt)` into `rs`, and if the value is zero, updates the memory location at `Imm(rt)` to 1. This atomic instruction is useful for implementing locks: a value of 1 at the memory location indicates that someone holds the lock, and a value of 0 means the lock is free.

Producer pushes a message onto queue: (memory operations in bold)

```
void push(int** tail_ptr, int* tail_write_lock, int message) {
      while (lock_try(tail_write_lock) == false);
      **tail_ptr = message;
      *tail_ptr++;
      lock_release(tail_write_lock);
}

# R1 - contains address of data to enqueue
# R2 - contains the address of the tail pointer of queue
# R3 - address of tail pointer write lock
P1 SpinLock:TST R4, 0(R3)          # try to acquire tail write lock
P2          BNEZ R4, R4, SpinLock
P3          LD R4, 0(R2)            # get tail pointer
P4          ST R1, 0(R4)            # write message to tail
P5          ADD R4, R4, 4           # update tail pointer
P6          ST R4, 0(R2)
P7          ST R0, 0(R3)            # release lock
```

Consumer pops a message off queue: (memory operations in bold)

```
int pop(int** head_ptr, int** tail_ptr) {
      while (*head_ptr == *tail_ptr);
      int message = **head_ptr;
      *head_ptr++;
      return message;
}

# R1 - will receive address contained in message
# R2 - contains the address of the head pointer of queue
# R3 - contains the address of the tail pointer of the queue
C1 Retry:   LD R4, 0(R2)            # get head pointer
C2          LD R5, 0(R3)            # get tail pointer
C3          SUB R5, R4, R5          # is there a message?
C4          BNEZ R5, Pop
C5          JMP Retry
C6 Pop:     LD R1, 0(R4)            # read message from queue
C7          ADD R4, R4, 4           # update head pointer
C8          ST R4, 0(R2)
```

Name ____________________________

Question 1 (10 points):

You are trying to port this code to an architecture that does not have the `TST` instruction (but, happily, the rest of the ISA is unchanged). Instead the new architecture has load-reserve/store-conditional instructions. Implement `TST rs, 0(rt)` using load-reserve/store-conditional:

```
LR rs, Imm(rt):
     rs ← Memory[(rt) + Imm]
     Track address (rt) + Imm

SC rs, Imm(rt):
     If (rt) + Imm modified:
          rs ← 0                            # Fail
     Else:
          Memory[(rt) + Imm] = (rs)         # Succeed
          rs ← 1
```

```
TST rs, 0(rt):
     LR rs, 0(rt)    # test: is 0(rt) 1?
     BNEZ rs, skip
     ADD rs, rs, 1   # set: try to store 1
     SC rs, 0(rt)
     NOR rs, rs, rs # invert result to match TST
skip: NOP
```

Question 2 (10 points):

This new architecture is also *not* sequentially consistent. Give an example of memory orderings between the producer and consumer that would result in incorrect behavior. *Explain your answer fully or you will not receive credit.*

Your answer should look something like:
`P1, P3, P4, C1, C2, P6, P7, C1, C2, C6, C8`
(Except that this is a sequentially consistent ordering, so it is not a correct answer.)

If the tail write is visible to the consumer before the message write, then we have a problem. Thus any sequence that contains the subsequence:

P6 C6 P4

Will read an invalid message. There are many other invalid sequences.

Name ____________________________

Question 3 (10 points):

Show where memory fences should be added to the producer and consumer code to ensure correctness with a weak consistency model. Explain your answer fully.

```
P1 SpinLock:TST R4, 0(R3)          # try to acquire tail write lock
            FENCE_WR # don't read tail ptr before getting lock

P2          BNEZ R4, R4, SpinLock

P3          LD R4, 0(R2)           # get tail pointer

P4          ST R1, 0(R4)           # write message to tail
            FENCE_WW # don't update tail before writing message

P5          ADD R4, R4, 4          # update tail pointer

P6          ST R4, 0(R2)
            FENCE_WW # don't release lock before updating tail

P7          ST R0, 0(R3)           # release lock




C1 Retry:   LD R4, 0(R2)           # get head pointer

C2          LD R5, 0(R3)           # get tail pointer

C3          SUB R5, R4, R5         # is there a message?

C4          BNEZ R5, Pop

C5          JMP Retry
            FENCE_RR # don't read message before tail is updated

C6 Pop:     LD R1, 0(R4)           # read message from queue

C7          ADD R4, R4, 4          # update head pointer

C8          ST R4, 0(R2)
```

Question 4 (5 points):

Let's next consider performance with a single producer thread and consumer thread. The following happens repeatedly:

1. The producer executes all instructions to push a message on the queue.
2. The consumer executes all instructions to pop a message off the queue.

Assume data, head, and tail pointers all lie in different, non-conflicting cache blocks.

First, after a few messages have been sent through the queue, will the consumer ever miss reading the head pointer? Will the producer ever miss reading the tail write lock, or fail to acquire the tail write lock? Explain in one or two sentences.

No, the head pointer belongs exclusively to the consumer. Likewise with a single producer, the tail write lock belongs exclusively to the producer. The consumer and producer will ping-pong on the tail pointer, however, since each uses it.

Question 5 (5 points):

We'll now focus on the tail pointer only. Assuming a MSI invalidate coherence protocol, show the state of the tail pointer in the producer and consumer cache after each operation in the sequence below. Show any data or permissions transfers, e.g. "Memory→C" or "C invalidates P".

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr	S		P ← Memory
P4 ST message			
P6 ST new_tail	M		
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	S	S	C ← P
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail	M	I	P invalidates C
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	S	S	C ← P; Memory ← P
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state? 2 (second half of table)

Name ______________________________

Question 6 (5 points):

Stay focused on the tail pointer only. Assume an update coherence protocol where the state of each line is either valid (V) or invalid (I). Show the state of the tail pointer in the producer and consumer cache after each operation in the sequence below in the steady state. Show any data or permissions transfers, e.g. "Memory→C" or "C invalidates P".

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr	V		P ← Memory
P4 ST message			
P6 ST new_tail	V		
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	V	V	C ← P
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr	V	V	
P4 ST message			
P6 ST new_tail	V	V	C ← P
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	V	V	
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state? Zero, but one data transfer (P6). Memory may also be updated at P6, depending on the protocol (if not, V→I must writeback).

Name ______________________________

Question 7 (10 points):

Your new architecture supports “remote access” for cached lines. This lets you assign a “home cache” for lines so that all memory operations will be sent *over the network* to operate remotely on the line *without allocating it in the requesting cache*.

For example, if line 0x100 is homed to processor A, and processor B writes 0x100, then *processor A’s cache will be updated* and processor B’s will be unchanged.

Assume the tail pointer is mapped to the producer’s cache, and the cache uses an MSI invalidate protocol (similar to Question 5). Once again, show the state of the tail pointer for the sequence of operations in the steady state and data/permission transfers:

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr	S		P ← Memory
P4 ST message			
P6 ST new_tail	M		
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			C *processor* ← P
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			C *processor* ← P
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state? Zero, but one data transfer. The difference is that in this case the transfer is on demand—which may or may not be an improvement, depending on consumer behavior.

Name ____________________________

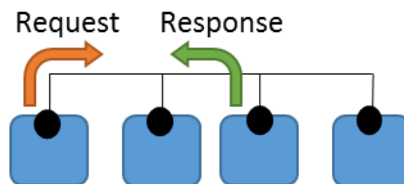
Part III: The Truth Will Set You Free (20 points)

Question 1 – Peanuts (5 points):

Snoopy coherence protocols rely on broadcast communication to detect sharing and updates. These are conventionally implemented using bus networks that allow for one message to be sent at a time to all nodes on the network.

Ben Bitdiddle is implementing a bus-based snoopy coherence protocol. One fifth of instructions access memory, and one quarter of these miss in the core's local cache (either because the line is invalid or doesn't have necessary permissions). Assuming each memory operation consists of a request and acknowledgement, the network traffic per core is therefore: $\frac{1}{5} \times \frac{1}{4} \times 2 = \frac{1}{10} \frac{\text{messages}}{\text{instruction}}$. Assume all messages fit within a single network flit.

Assuming a fixed IPC of 1, perfect bus arbitration, and infinite buffers, how many cores can the bus support?



A bus has an aggregate throughput of 1 message per cycle.

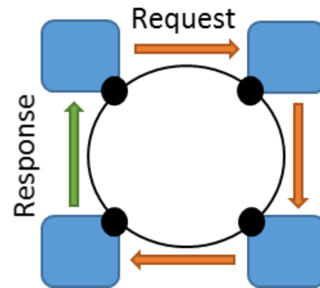
A memory operation requires 2 messages on 1/20 of instructions, or 1/10 messages per cycle.

The number of cores this system can support is $1 = N/10$ so $N = 10$.

Name ____________________________

Question 2 – …To rule them all (10 points):

Ben needs to build a larger system than the bus network will allow, so he changes the system to use a unidirectional ring network. In this design, the core issuing the memory operation sends the request around the ring, and each node along the way either forwards the request or replaces it with its response. Assuming fixed IPC of 1 and a single-cycle per hop in the network, at how many cores will this design saturate?



The ring with N cores has an aggregate throughput of N messages per cycle. (It is a unidirectional ring.)

Each memory operation requires one circuit around the ring, or N messages. Each core produces one request every 20 instructions, so the messages generated per core is $N/20$.

Thus the number of cores is $N = N/(N/20) = 20$.

Maybe a simpler way to see this is that with the bus, each memory operation required global communication twice (for the request and response). In the ring, each memory request requires global communication only once—since ~half the nodes see the request and the rest simply forward the response. Since we are placing half the demand on the network, we can support twice as many cores.

Question 3 – Matryoshka (5 points):

Ben next explores the tradeoffs in cache design between an inclusive cache, where the parent always has a copy of every line in the child's cache, and non-inclusive caches, where this isn't guaranteed.

Give one advantage and one disadvantage of a non-inclusive cache design.

Non-inclusive caches allow the parent cache to get rid of a copy of a line without invalidating the child's copy. This essentially increases the capacity of the parent cache, since it can use the space to store new lines instead of copies of the child's contents.

The downside of this is that the parent no longer knows from looking at its own contents whether or not a child has a line. This makes coherence more expensive since the parent must now check with the children to see if they have a line, for example to process an invalidate when the line is written elsewhere.

One way to try to get the best of both worlds is to separate coherence tracking and data storage into separate structures. So rather than having the directory in the cache tags, the directory is a separate structure. This directory can be kept inclusive with the cache non-inclusive.