Computer System Architecture
6.823 Quiz #3
April 28th, 2017
Professors Daniel Sanchez and Joel Emer

Name: _____**Solutions**_____

This is a closed book, closed notes exam.
85 Minutes
15 Pages (+2 Scratch)

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.
- Pages 16 and 17 are scratch pages. Use them if you need more space to answer one of the questions, or for rough work.

Part A	_____	26 Points
Part B	_____	26 Points
Part C	_____	24 Points
Part D	_____	24 Points
		(+10 Bonus Points)
TOTAL	_____	**100 Points**

Part A: Cache Coherence (26 points)

We want to study the tradeoffs between the standard directory-based MSI and MESI coherence protocols. The figures below show their state-transition diagrams.

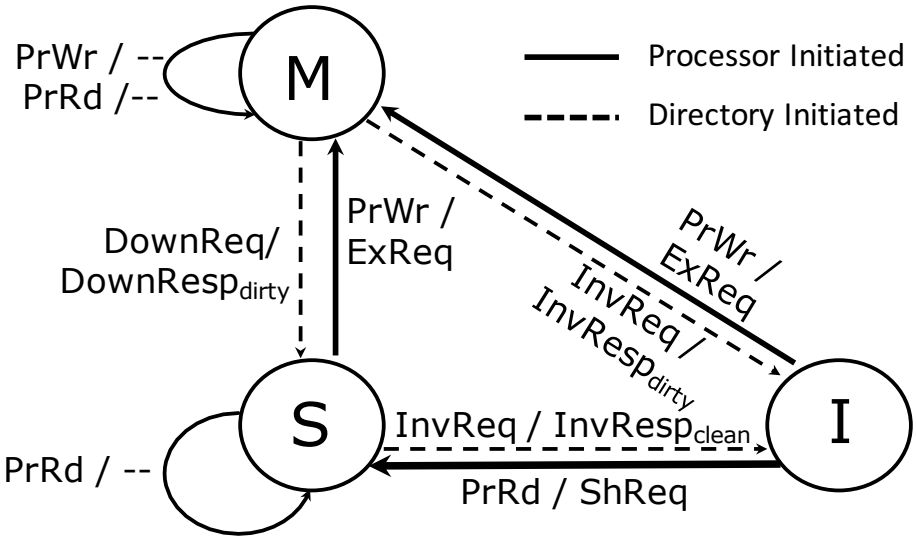


Figure A-1: MSI protocol state transition diagram.

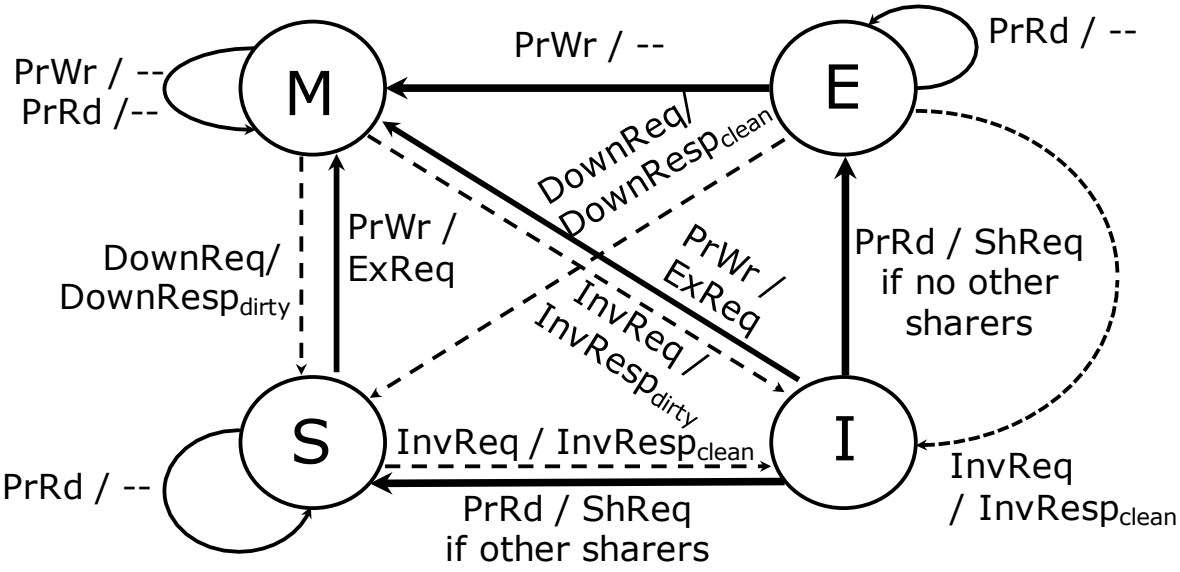
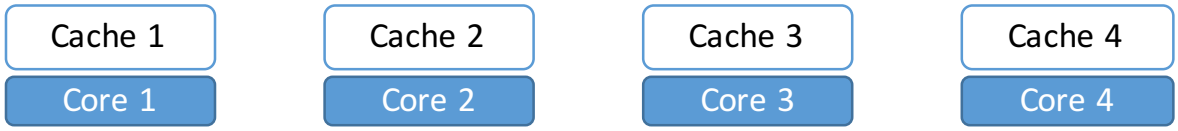


Figure A-2: MESI protocol state transition diagram.

Question 1 (6 points)

Consider the four-core system below. Each core has a private cache, and caches are kept coherent with a directory protocol. Each core runs a thread that issues a load followed by a store to a single address, as shown below. Each thread accesses a different address (core 1's thread accesses `A`, core 2's thread accesses `B`, etc.). These thread-private addresses are on different cache lines. The number in parenthesis indicates the global order of the accesses (i.e. `LD A` happens before `ST A`, which happens before `LD B`, etc.). Each access completes before the next one begins.



(1) LD A (2) ST A	(3) LD B (4) ST B	(5) LD C (6) ST C	(7) LD D (8) ST D

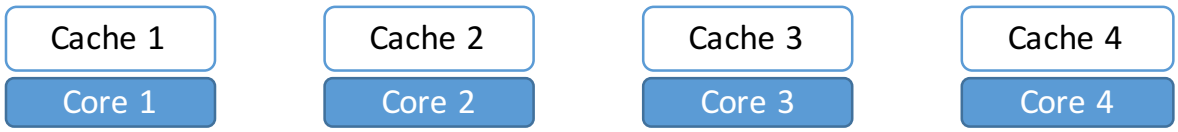
For this sequence of 8 accesses, provide in the table below the total number of share and exclusive requests from caches to directory (`ShReq` and `ExReq`), and the total number of invalidation and downgrade requests from the directory to caches (`InvReq` and `DownReq`) for the MSI and MESI protocols. If a single shared or exclusive request causes the directory to issue invalidation or downgrade requests to multiple caches, you should count each of those messages as a separate request. Ignore coherence responses. *Assume all caches are initially empty.*

	# of ShReq	# of ExReq	# of InvReq	# of DownReq
MSI	***4***	***4***	***0***	***0***
MESI	***4***	***0***	***0***	***0***

With MESI, each cache transitions from I->E, then silently upgrades E->M

Question 2 (6 points)

Consider a different program where each thread reads globally shared data, as shown below.



(1) LD A	(2) LD A	(3) LD A	(4) LD A

For this sequence of 4 accesses, fill in the table below for the MSI and MESI protocols. Ignore coherence responses. *Assume all caches are initially empty.*

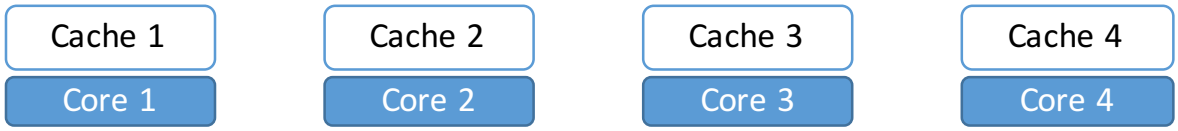
	# of ShReq	# of ExReq	# of InvReq	# of DownReq
MSI	*4*	*0*	*0*	*0*
MESI	*4*	*0*	*0*	*1*

With MESI, the directory responds to the first share request with E permission. When it receives the next share request, the directory must downgrade that first cache E->S.

Let's try to improve on MSI by using a predictor. The idea is to predict whether a load that misses will be followed by a store to the same cache line. We leave the MSI protocol unchanged, but augment each private cache with a new `ShReq`-vs-`ExReq` predictor. On a store miss, the cache always sends an exclusive request to the directory, as in normal MSI. However, on a load miss, the cache can send either a shared request or an exclusive request, deferring the decision to the predictor. This way, on a load miss, the core may send an `ExReq` and acquire the line in M instead of S.

Question 3 (7 points)

Consider the private read/write access sequence in Question 1 (repeated below for convenience).



(1) LD A (2) ST A	(3) LD B (4) ST B	(5) LD C (6) ST C	(7) LD D (8) ST D

Assume all private caches use the MSI protocol with predictor, and make the same prediction.

a) What is the correct prediction (`ShReq` or `ExReq`) for this sequence of accesses?

`ExReq`, since the load will be followed by a store.

b) What is the incorrect prediction (`ShReq` or `ExReq`) for this sequence of accesses?

`ShReq`

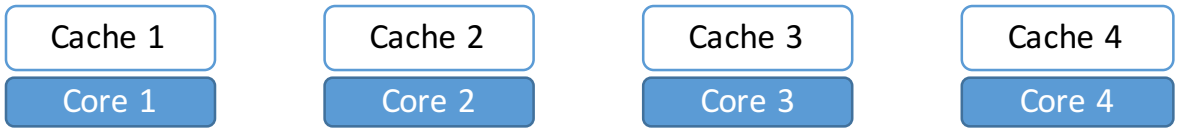
c) For the sequence of 8 accesses, fill in the table below for MSI with correct and incorrect predictions. Ignore coherence responses. *Assume all caches are initially empty.*

	# of ShReq	# of ExReq	# of InvReq	# of DownReq
MSI with Correct Prediction	***0***	***4***	***0***	***0***
MSI with Incorrect Prediction	***4***	***4***	***0***	***0***

Correct: each cache sends an ExReq to get their private data in M state.
Incorrect: same as MSI in Question 1.

Question 4 (7 points)

Consider the shared read-only memory access sequence in Question 2.



(1) LD A	(2) LD A	(3) LD A	(4) LD A

Assume all private caches use the MSI protocol with predictor, and make the same prediction.

a) What is the correct prediction (`ShReq` or `ExReq`) for this sequence of accesses?

`ShReq`

b) What is the incorrect prediction (`ShReq` or `ExReq`) for this sequence of accesses?

`ExReq`

c) For the sequence of 4 accesses, fill in the table below for MSI with correct and incorrect predictions. Ignore coherence responses. *Assume all caches are initially empty.*

	# of ShReq	# of ExReq	# of InvReq	# of DownReq
MSI with Correct Prediction	***4***	***0***	***0***	***0***
MSI with Incorrect Prediction	***0***	***4***	***3***	***0***

Correct: same as MSI in Question 2.
Incorrect: the directory serializes ExReq requests. It grants some cache M state, then must invalidate that cache, grant another cache M state, invalidate that cache, and so on.

Part B: Memory Consistency (26 points)

Consider two processes, P1 and P2, running on two different processors.
Assume that memory locations A, B, and C contain initial value 0.

P1	P2
P1.1: ST (A) ← 1	P2.1: ST (B) ← 1
P1.2: LD RB ← (B)	P2.2: LD RC ← (C)
P1.3: ST (C) ← 1	P2.3: LD RA ← (A)

Question 1 (10 points)

Out of the following possible final values of (RA, RB, RC), circle the ones that could occur if the system is **Sequentially Consistent (SC)**.

(0,0,0) **(0,0,1)** **(0,1,0)** **(0,1,1)**

(1,0,0) **(1,0,1)** **(1,1,0)** **(1,1,1)**

Question 2 (5 points)

Out of the following possible final values of (RA, RB, RC), circle the ones that could occur if the system enforces **Total Store Order (TSO)**. TSO is a weak memory model where *stores may be reordered after loads*, i.e., a load may complete before a store that is earlier in program order if they access different addresses and there are no data dependencies.

(0,0,0) **(0,0,1)** **(0,1,0)** **(0,1,1)**

(1,0,0) **(1,0,1)** **(1,1,0)** **(1,1,1)**

Question 3 (5 points)

Out of the following possible final values of (RA, RB, RC), circle the ones that could occur if the system enforces **RMO**. RMO is a weak memory model where *loads or stores may be reordered after loads or stores*, i.e., a load or a store may complete before a load or a store that is earlier in program order if they access different addresses and there are no data dependencies.

(0,0,0) **(0,0,1)** **(0,1,0)** **(0,1,1)**

(1,0,0) **(1,0,1)** **(1,1,0)** **(1,1,1)**

Question 4 (6 points)

Add the **minimum number** of memory barrier instructions (i.e., fences) to **P1** and **P2** such that the **TSO machine** produces the same register values as the **SC machine** for the given code.

The following fine-grained barrier instructions are available:

- **MEMBAR$_{RR}$** guarantees that all reads initiated before MEMBAR$_{RR}$ will be performed before any read that follows the barrier.
- **MEMBAR$_{RW}$** guarantees that all reads initiated before MEMBAR$_{RW}$ will be performed before any write that follows the barrier.
- **MEMBAR$_{WR}$** guarantees that all writes initiated before MEMBAR$_{WR}$ will be performed before any read that follows the barrier.
- **MEMBAR$_{WW}$** guarantees that all writes initiated before MEMBAR$_{WW}$ will be performed before any write that follows the barrier.

P1	P2
P1.1: ST (A) ← 1	P2.1: ST (B) ← 1
MEMBAR$_{WR}$	**MEMBAR$_{WR}$**
P1.2: LD RB ← (B)	P2.2: LD RC ← (C)
P1.3: ST (C) ← 1	P2.3: LD RA ← (A)

A **MEMBAR$_{WR}$** between P2.2 and P2.3 instead of between P2.1 and P2.2 is also correct.

Part C: Reliability (24 points)

Consider the ACEness of different fields in an out-of-order machine. The machine uses a Data-in-ROB design. Each instruction is allocated an entry in the reorder buffer (ROB) until commit. Assume the following:

- An ROB entry's `src1` and `src2` fields either hold the tag of the entry that will produce the instruction's source operand(s), or the actual data when it becomes available.
- An ROB entry's `dest` field holds the ID of the register the instruction writes, if any.
- When an instruction is issued to execute, its operands are latched by the functional unit or load/store unit.
- After execute, the output from a functional unit (or a load/store unit), if any, is written back to the `data` field in the ROB entry. This data, if any, is written to the `dest` register in the register file at commit.
- All instructions are ACE (i.e. no wrong-path execution).

For reference, here is a sample ROB entry for a subtract instruction that is waiting on data for its first operand.

Tag	Inum	PC	Use	Ex	Op	p1	src1	p2	src2	pd	dest	data
T8	I8	0xb4	1		sub		T6	1	2		R5	

Question 1 (8 points)

An ROB entry is allocated to instruction **`add R3,R1,R2`**, which adds the contents of registers `R1` and `R2`, and writes the result to register `R3`. Indicate whether the following intervals in the lifetime of the specified ROB fields are ACE, unACE, or unknown. Explain any assumptions you make.

	src1	src2	dest	data
Entry Allocation to Operands Available	**ACE**	**ACE**	**ACE**	**unACE**
Operands Available to Issue	**ACE**	**ACE**	**ACE**	**unACE**
Issue to Write-Back (i.e. executing)	**unACE**	**unACE**	**ACE**	**unACE**
Write-Back to Commit	**unACE**	**unACE**	**ACE**	**ACE**

The operands are latched at the adder at issue, so the src1/src2 fields are unACE after issue. The dest field is read at commit to update architectural state of R3, so remains ACE through its lifetime. The data field value is produced at write-back and is not read until commit.

Question 2 (8 points)

An ROB entry is allocated to instruction **`st R4,0(R7)`**, which stores the contents of register `R4` into the effective memory address contained in register `R7`. Indicate whether the following intervals in the lifetime of the specified ROB fields are ACE, unACE, or unknown. Explain any assumptions you make.

	src1	src2	dest	data
Entry Allocation to Operands Available	**ACE**	**ACE**	**unACE**	**unACE**
Operands Available to Issue	**ACE**	**ACE**	**unACE**	**unACE**
Issue to Write-Back (i.e. executing)	**unACE**	**unACE**	**unACE**	**unACE**
Write-Back to Commit	**unACE**	**unACE**	**unACE**	**unACE**

The effective address (from R7) and data (from R4) are latched at the load/store unit at issue, so the src1/src2 fields are unACE after issue. This instruction type never updates architectural state of a register, so the dest and data fields are always unACE.

Question 3 (8 points)

Consider a 16-entry load buffer that holds the virtual addresses of issued loads. On average, a load is issued every 5 cycles, i.e., an address is allocated a new entry once every 5 cycles. The cache hit rate of this access stream is 90%, and accesses that hit in the cache hold their entry for 10 cycles on average. Those accesses that miss take 110 cycles on average to release their entry.

a) In steady state, how many load buffer entries are in use on average?

T = 1 allocation / 5 cycles = 0.2 allocations/cycle
L = 0.9 * 10 cycles + 0.1 * 110 cycles = 9 + 11 = 20 cycles
Little's law:
N = T * L = 4 entries

b) You profile the memory access stream. Of those loads that **miss** in the cache, you find that 5% are issued from wrong-path execution, 15% are dynamically dead, and the rest are used by ACE instructions. Of those addresses that **hit** in the cache, 2% are issued from wrong-path execution, 18% are dynamically dead, and the rest are used by ACE instructions. For the given access stream, what is the Architectural Vulnerability Factor (AVF) of this load buffer?

AVF is the fraction of cycles an entry contains ACE state.

AVF = Nace / Ntotal
Ntotal = 16 entries

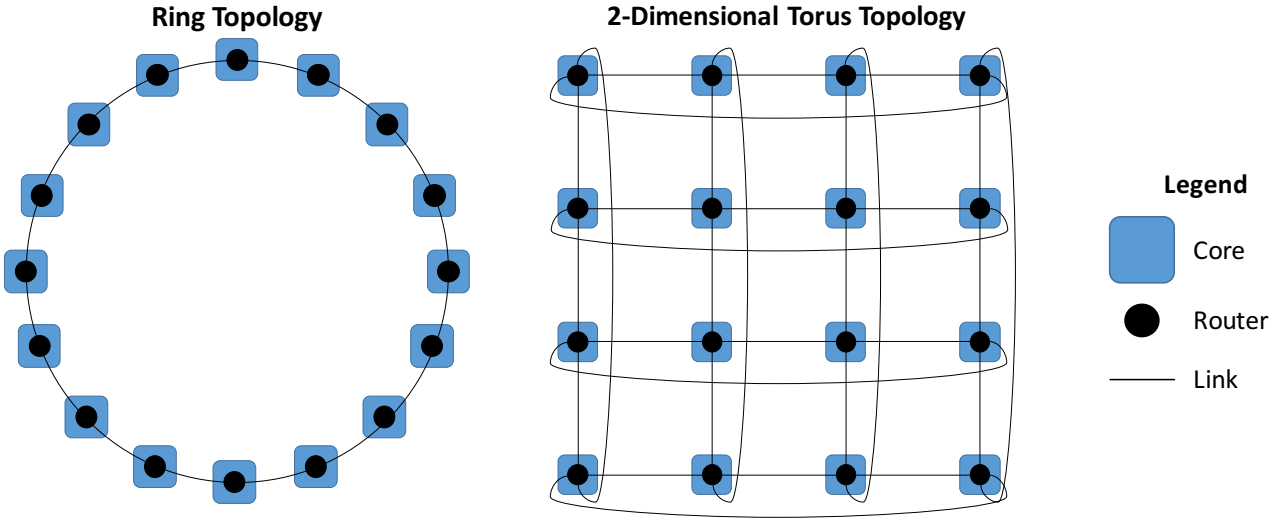
We can use Little's Law to find the average number of ACE entries in a cycle.
The throughput of allocated entries remains the same.
T = 0.2 allocations/cycle (from part a)

We need to determine the average latency for ACE loads.
Lace = 0.9 * (1 – 0.02 – 0.18) * 10 cycles + 0.1 * (1 – 0.05 – 0.15) * 110 cycles
= 7.2 + 8.8 = 16 cycles

AVF = T * Lace / Ntotal = (0.2 * 16) / 16 = 0.2 (unitless), between 0 and 1.0

Part D: On-Chip Networks (24 points + 10 bonus)

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



For a system of N nodes, the 2-D Torus topology consists of $k = \sqrt{N}$ rows and columns, where each row or column is a ring.

Question 1 (10 points)

Your first task is to compare the topologies along key metrics. Fill in the table below as a function of the number of nodes in the network, N. The units for each cell are hops or links. For average distance, assume uniform random traffic (where each node sends $1/N^{th}$ of the traffic to each destination, including itself).

To ease your derivations, you can define a variable $k = \sqrt{N}$, and assume k is an even integer. *For partial credit, give the asymptotic growth instead.*

	Ring	2-D Torus
Number of links	N	$2N$
Diameter	$\frac{N}{2}$	$\sqrt{N}$
Average distance	$\frac{N}{4}$	$\frac{\sqrt{N}}{2}$
Bisection bandwidth	2	$2\sqrt{N}$

Question 2 (8 points)

As discussed in lecture, higher-dimensional topologies have smaller diameter, but the routers are more expensive. For what range of number of nodes, *N*, is the **average latency** in cycles of a 2-D torus topology lower than the **average latency** in cycles of a ring topology? (Provide an inequality on *N*.)

Assume the following:

- Traversing one link requires 1 cycle
- Traversing one router requires 1 cycles in the ring topology
- Traversing one router requires 4 cycles in the 2-D torus ring topology
- Entering and exiting the network at a router counts as traversing the router
- Traffic is uniform random

(For example, in a 4-node ring, the latency to send a flit from a node to a node two hops away is 5 cycles: 1 cycle to enter the network at the first router, 2 cycles for two link traversals, 1 cycle for one router traversal, and 1 cycle to exit the network at the last router.)

Consider the Ring topology.
Every hop takes 2 cycles (1 for link, 1 for router, including the exit router).
The average distance in hops for uniform random traffic is N/4.
Add 1 cycle to enter the network.
Lring = $N/4 * 2 + 1 = N/2 + 1$

Consider the 2-D Torus topology.
Every hop takes 5 cycles (1 for link, 4 for router)
Average distance in hops is $\sqrt{N}/2$.
Add 4 cycles to enter the network.
Ltorus = $\sqrt{N}/2 * 5 + 4 = 5/2\sqrt{N} + 4$

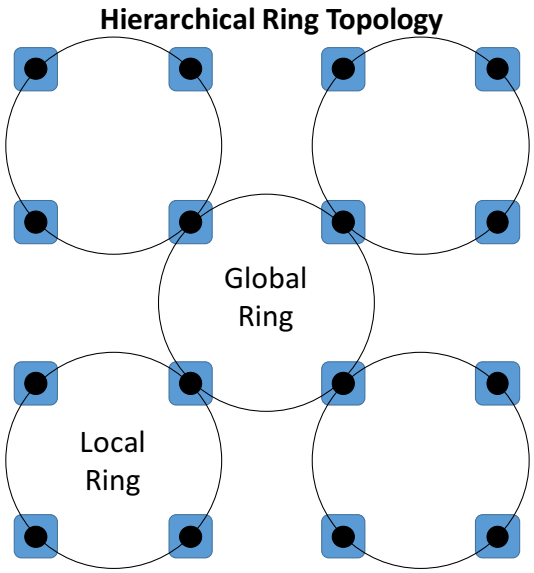
When is Ltorus < Lring?
=> $5/2\sqrt{N} + 4 < N/2 + 1$
=> $0 < N - 5\sqrt{N} - 6$
=> $0 < (\sqrt{N} - 6)(\sqrt{N} + 1)$
=> $N > 36$

Since N is an even square, N >= 64.

Question 3 (6 points)

In a stroke of genius, you realize that a Hierarchical Ring topology (shown below) can bridge the low-latency routing of a Ring topology with the lower diameter of a 2-D Torus topology. For a system of N nodes, the Hierarchical Ring topology consists of one *global ring* surrounded by $k = \sqrt{N}$ *local rings*. Each local ring has one node that also bridges with the global ring.

Fill in the table below as a function of the number of nodes in the network, N. The units for each cell are hops or links. To ease your derivations, you can define a variable $k = \sqrt{N}$, and assume k is an even integer. *For partial credit, give the asymptotic growth instead.*



	Hierarchical Ring
Number of links	$N + \sqrt{N}$
Diameter	$\frac{3}{2}\sqrt{N}$
Bisection bandwidth	*2*

Each of the $\sqrt{N}$ local rings contains $\sqrt{N}$ links. The single global ring contains $\sqrt{N}$ links.

The diameter within a ring is $\frac{\sqrt{N}}{2}$. In the worst case, such a diameter is crossed for three rings: one end of a local ring to the bridge to enter the global ring, to the most distant bridge to exit the global ring, then to the end of that local ring.

Bonus Question (10 points)

Warning: This question is harder than others, so we recommend finishing as much of the quiz as you can before attempting it!

Derive the average distance (in hops or links) for a Hierarchical Ring topology as a function of the number of nodes in the network, *N*. Assume uniform random traffic (where each node sends *1/N*th of the traffic to each destination, including itself). To ease your derivations, you can define a variable $k = \sqrt{N}$, and assume *k* is an even integer. *Only exact answers are accepted, no partial credit.*

With probability $\frac{1}{k}$ the source and destination are in the same ring.
The average distance in this case is $\frac{k}{4}$.

With probability $\frac{k-1}{k}$ the source and destination are in different rings.
The average distance to enter the global ring is $\frac{k}{4}$ (this includes when the source is a bridge node).
The average distance to exit the global ring is $\frac{k}{4}$ (this includes when the destination is a bridge).
The average distance to traverse the global ring is **not** $\frac{k}{4}$, because we have assumed the source and destination are in different rings; we scale the average distance for a ring by $\frac{k}{k-1}$.

The average distance is therefore:

$$\frac{1}{k}\cdot\frac{k}{4}+\frac{k-1}{k}\left(\frac{k}{4}+\frac{k}{4}+\frac{k}{4}\cdot\frac{k}{k-1}\right)=\frac{3\sqrt{N}-1}{4}$$

Scratch Space

Use these extra pages if you run out of space or for your own personal notes. We will not grade this unless you tell us explicitly in the earlier pages.