16.482 / 16.561: Computer Architecture and Design

Spring 2015

Homework #9 Solution

1. <u>RAID</u> *(40 points) You are working with a 5-disk RAID array that contains a total of 15 sectors; the exact sector configuration depends on the RAID level used. In all cases, twelve of the fifteen sectors (S0-S11) will hold data, while the remaining three sectors (P0-P2) hold parity information. Large reads and writes (reads/writes that access an entire stripe in the array) take 300 ms, small reads (reads involving only a single disk) take 150 ms, and small writes (writes involving 1 data disk + 1 parity disk) take 200 ms.*

Given the following sequence of sector reads and writes, determine the time required if the array is configured with RAID 3, RAID 4, and RAID 5. Assume the following:

- *Requests are queued in such a manner that two consecutive operations may proceed simultaneously if they do not share any disk within the array.*
- *If two disks, D_x and D_y, are in use, and the access to D_x finishes before the access to D_y, a new operation may start immediately assuming it does not involve D_y.*
- *Multiple accesses to the same stripe may overlap if they don't use the same disk.*

1. *read S0*
2. *write S5*
3. *write S8*
4. *read S9*
5. *read S3*
6. *write S7*
7. *read S11*
8. *write S1*

In each case, show the organization of the array to support your answer.

<u>Solution:</u>

- In RAID 3, only large reads and writes are allowed. Since each operation involves every disk, no operations may be overlapped, and the total time for 8 large reads and writes is 8 x 300 ms = **2400 ms = 2.4 s**.
- In RAID 4, you may perform small reads, but only large writes. Therefore, any two consecutive reads that do not use the same disk can proceed in parallel. Nothing may be overlapped with a write. We assume the following organization:

Disk 1	**Disk 2**	**Disk 3**	**Disk 4**	**Disk 5**
S0	S1	S2	S3	P0
S4	S5	S6	S7	P1
S8	S9	S10	S11	P2

Operation	Start time	End time	Notes
read S0	1	150	Small read followed by write—no overlap
write S5	151	450	
write S8	451	750	
read S9	751	900	Sectors are on different disks, so small reads can be overlapped
read S3	751	900	
write S7	901	1200	
read S11	1201	1350	Small read followed by write—no overlap
write S1	1351	1650	

In RAID 4, this sequence takes **1650 ms = 1.65 s.**

- In RAID 5, both small reads and writes are allowed; we can therefore overlap any two consecutive operations that do not share the same disk. Remember RAID 5 also involves interleaved parity to make small writes possible, so the organization changes as follows:

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
S0	S1	S2	S3	P0
S4	S5	S6	P1	S7
S8	S9	P2	S10	S11

Note also that the problem states we can start a new transaction any time an existing transaction ends, provided the new transaction does not use the same disk as a currently executing transaction. Note that we must be careful. Although RAID 5 does enable small writes, these operations use two disks—the disk being written and the parity disk for that stripe. The solution to this part of the problem therefore becomes more complex and can best be described by tracking start and end times for each operation:

Operation	Start time	End time	Notes
read S0	1	150	Different disks—overlap allowed.
write S5	1	200	
write S8	151	350	Can start when read to S0 finishes.
read S9	201	350	Can start when write to S5 finishes. Allow multiple accesses in same stripe if different disks involved.
read S3	351	500	Cannot overlap with next write—both use Disk 4.
write S7	501	700	Cannot overlap with next read—both use Disk 5.
read S11	701	850	Cannot overlap with next write—both use Disk 5.
write S1	851	1050	

In RAID 5, this sequence takes **1050 ms = 1.05 s.**

2. Snooping protocols *(30 points) You are given a four-processor system that uses a write-invalidate, snooping coherence protocol. Each direct-mapped, write-back cache has four lines, each of which holds eight bytes; in the diagram below, only the least-significant byte of each word is shown. The cache states are I (invalid), S (shared), and M (modified/exclusive). The caches and memory have the following initial state:*

P0

	State	Tag	Data	
B0	I	0x100	01	23
B1	S	0x108	00	88
B2	M	0x110	00	30
B3	I	0x118	00	10

P1

	State	Tag	Data	
B0	I	0x100	01	23
B1	M	0x128	AB	CD
B2	S	0x130	14	12
B3	S	0x118	14	92

P2

	State	Tag	Data	
B0	S	0x120	13	31
B1	S	0x108	00	88
B2	I	0x130	51	55
B3	I	0x138	01	38

P3

	State	Tag	Data	
B0	S	0x120	13	31
B1	S	0x108	00	88
B2	I	0x110	00	30
B3	S	0x118	14	92

Memory

Address	Data	
0x100	00	00
0x108	00	88
0x110	20	08
0x118	14	92
0x120	13	31
0x128	FF	FE
0x130	14	12
0x138	AB	BA

For each of the transactions listed below, list all cache blocks modified and their final state, as well as all memory blocks modified and their final state. Assume each set of transactions starts with the same initial state—in other words, your answer to part (b) does not depend on your answer to part (a). However, you should track the state transitions of each block throughout the problem.

a. *P1: read 0x128*
P2: read 0x128
P3: read 0x128

b. *P2: write 0x110 $\leftarrow$14*
P2: write 0x114 $\leftarrow$DF
P2: write 0x130 $\leftarrow$20

c. *P3: write 0x128$\leftarrow$45*
P0: write 0x128$\leftarrow$67
P3: read 0x12C

Solution: Solutions for each part are shown in the table below:

Transaction(s)	Cache blocks modified	Memory blocks modified
a. (i) P1: read 0x128 (ii) P2: read 0x128 (iii) P3: read 0x128	(i) No changes (ii) P1.B1: (S, 0x128, AB CD) P2.B1: (S, 0x128, AB CD) (iii) P3.B1: (S, 0x128, AB CD)	(ii) M[0x128]: (AB CD)
b. (i) P2: write 0x110 ←14 (ii) P2: write 0x114 ←DF (iii) P2: write 0x130 ←20	(i) P0.B2: (I, 0x110, 00 30) P2.B2: (M, 0x110, 14 30) (ii) P2.B2: (M, 0x110, 14 DF) (iii) P1.B2: (I, 0x130, 14 12) P2.B2: (M, 0x130, 20 12)	(i) M[0x110]: (00 30) (iii) M[0x110]: (14 DF)
c. (i) P3: write 0x128←45 (ii) P0: write 0x128←67 (iii) P3: read 0x12C	(i) P1.B1: (I, 0x128, AB CD) P3.B1: (M, 0x128, 45 CD) (ii) P0.B1: (M, 0x128, 67 CD) P3.B1: (I, 0x128, 45 CD) (iii) P0.B1: (S, 0x128, 67 CD) P3.B1: (S, 0x128, 67 CD)	(i) M[0x128]: (AB CD) (ii) M[0x128]: (45 CD) (iii) M[0x128]: (67 CD)

3. *Directory protocols (30 points) Say we have a four-processor system that uses a write-invalidate, directory coherence protocol. The system contains a total of 8 memory blocks, as shown in the initial directory state below:*

Block #	*P0*	*P1*	*P2*	*P3*	*Dirty*
0	*1*	*1*	*0*	*0*	*0*
1	*0*	*0*	*1*	*0*	*1*
2	*1*	*1*	*1*	*1*	*0*
3	*0*	*1*	*1*	*0*	*0*
4	*0*	*0*	*0*	*1*	*1*
5	*0*	*0*	*0*	*0*	*0*
6	*0*	*1*	*0*	*0*	*0*
7	*1*	*0*	*0*	*0*	*1*
8	*0*	*0*	*0*	*0*	*0*

a. *(5 points) What state is each block in?*

Solution: Remember that blocks in a directory protocol can be in one of three states:

- Uncached: presence bits are all 0 → **blocks 5 and 8**
- Shared: 1 or more presence bit is 1, and the dirty bit is 0 → **blocks 0, 2, 3, and 6**
- Exclusive: 1 presence bit is 1, and the dirty bit is 1 → **blocks 1, 4, and 7**

b. *(25 points) What messages are sent between the nodes and directory for each of the following transactions to ensure the processor has the most up-to-date block copy, the appropriate invalidations are made, and the directory holds the appropriate state?*

i. *P1: write block 0*

Solution: This transaction is a write to a shared block. Before P1 can write the block, it must ensure P0's copy is invalidated:

1. WriteMiss(P1, 0) sent to directory
2. Invalidate(0) sent from directory to P0

New directory state:

Block #	P0	P1	P2	P3	Dirty
0	0	1	0	0	1

ii. *P2: read block 7*

Solution: This transaction is a read to an exclusive block. P2 must therefore get the most up-to-date value from P0's cache:

1. ReadMiss(P2, 7) sent to directory
2. Fetch(7) sent to P0
3. DataWriteBack(0, mem[7]) sent from P0 to directory
4. DataValueReply(mem[7]) sent from directory to P2

New directory state:

Block #	P0	P1	P2	P3	Dirty
0	1	0	1	0	0

iii. *P3: write block 4*

Solution: This block is already in the exclusive state and held only by P3, so no messages are needed.

iv. P0: write block 2

<u>Solution:</u> Block 2 is shared by all processors in the system. P0 therefore has the most up-to-date value, but must invalidate all other copies. The last message really just serves as an acknowledgement that P0 now has the block in the exclusive state.

1. WriteMiss(P0, 2) sent to directory
2. Invalidate(2) sent from directory to {P1,P2,P3}
3. DataValueReply(mem[2]) sent from directory to P0

New directory state:

Block #	P0	P1	P2	P3	Dirty
2	1	0	0	0	1

v. P1: read block 6

<u>Solution:</u> This block is already in the shared state in P1's cache, so no messages are necessary.