

CMU 18-447 INTRODUCTION TO COMPUTER ARCHITECTURE, SPRING 2012  
HANDOUT 15/ SOLUTIONS TO HW 6: CACHES AND MAIN MEMORY

Prof. Onur Mutlu, Instructor  
Chris Fallin, Lavanya Subramanian, Abeer Agrawal, TAs

Given: Wednesday, Apr 4, 2012  
Due: **Monday, Apr 16, 2012**

## 1 Caches and Virtual Memory

A 2-way set associative write back cache with perfect LRU replacement requires  $15 \times 2^9$  bits of storage to implement its tag store (including bits for valid, dirty and LRU). The cache is virtually indexed, physically tagged. The virtual address space is 1 MB, page size is 2 KB, cache block size is 8 bytes.

- (a) What is the size of the data store of the cache in bytes?

**Solution:**

**8 KB**

The cache is 2-way set associative.

So, each set has 2 tags, each of size  $t$ , 2 valid bits, 2 dirty bits and 1 LRU bit (because a single bit is enough to implement perfect LRU for a 2-way set associative cache).

Let  $i$  be the number of index bits.

$$\text{Tag store size} = 2^i \times (2 \times t + 2 + 2 + 1) = 15 \times 2^9$$

Therefore,

$$2t = 10 \Rightarrow t = 5$$

$$i = 9$$

$$\text{Data store size} = 2^i \times (2 \times 8) \text{ bytes} = 2^9 \times (2 \times 8) \text{ bytes} = 8 \text{ KB.}$$

- (b) How many bits of the virtual index come from the virtual page number?

**Solution:**

**1 bit**

Page size is 2 KB. Hence, the page offset is 11 bits (bits 10:0).

The cache block offset is 3 bits (bits 2:0) and the virtual index is 9 bits (bits 11:3).

Therefore, one bit of the virtual index (bit 11) comes from the virtual page number.

- (c) What is the physical address space of this memory system?

**Solution:**

**64 KB**

The page offset is 11 bits.

The physical frame number, which is the same as the physical tag is 5 bits.

Therefore, the physical address space is  $2^{(11+5)} = 2^{16}$  bytes = 64 KB.

## 2 Interleaving and Row-Buffer Locality

A machine has a main memory of 4 KB, organized as 1 channel, 1 rank and  $N$  banks (where  $N > 1$ ). The system does not have virtual memory.

- Data is interleaved using a cache block interleaving policy, as described in lecture, where consecutive cache blocks are placed on consecutive banks.
- The size of a cache block is 32 bytes. Size of a row is 128 bytes.
- An open row policy is used, i.e., a row is retained in the row-buffer after an access, until an access to another row is made.
- A row-buffer hit is an access to a row that is present in the row-buffer. A row-buffer miss is an access to a row that is not present in the row-buffer.

- (a) For a program executing on the machine, accesses to the following bytes miss in the on-chip caches and go to memory.

0, 32, 320, 480, 4, 36, 324, 484, 8, 40, 328, 488, 12, 44, 332, 492

The row-buffer hit rate is 0%, i.e., all accesses miss in the row-buffer.

What is the minimum value of  $N$  - the number of banks?

**Solution:**

**2 banks**

Size of a cache block is 32 bytes. Therefore, the cache block access sequence corresponding to the given byte access sequence is 0, 1, 10, 15, 0, 1, 10, 15, 0, 1, 10, 15, 0, 1, 10, 15.

When the number of banks is 1, all cache blocks are mapped to the same bank. Cache block 0, cache block 1, cache block 10 and cache block 15 are mapped to rows 0, 0, 2 and 3 respectively. Therefore, when cache block 1 is accessed right after cache block 0, it would hit in the row-buffer as row 0 would already be in the row-buffer. Hence, row-buffer hit rate would not be 0%.

When the number of banks is 2, the 4 cache blocks 0, 1, 10 and 15 map to different rows and banks - (bank 0, row 0), (bank 1, row 0), (bank 0, row 1) and (bank 1, row 1) respectively. Hence, the access sequence would be (bank 0, row 0), (bank 1, row 0), (bank 0, row 1), (bank 1, row 1) (repeated four times).

Therefore, rows 0 and 1 on each bank are accessed in an alternating fashion, resulting in a 0% row-buffer hit rate.

- (b) If the row-buffer hit rate for the same sequence were 75%, what would be minimum value of  $N$  - the number of banks?

**Solution:**

**4 banks**

When the number of banks is 1, cache blocks 0, 1, 10 and 15 map to rows 0, 0, 2 and 3 respectively (as we saw in part a). Thus, the row access sequence is 0, 0, 2, 3 (repeated four times). Three out of four accesses are row-buffer conflicts, hence the row-buffer hit rate is only 25%.

For all other number of banks, the four cache blocks 0, 1, 10 and 15 map to different rows.

Hence, assuming the rows containing the four cache blocks are not already open in the row-buffer, the maximum achievable hit rate is 75%, as the first access to each cache block would result in a (compulsory) row-buffer miss. This maximum hit rate of 75% can be achieved only if there are no row-buffer conflicts. Thus, rows containing the four cache blocks should map to different banks. Therefore, the minimum number of banks required to achieve this hit rate is 4.

- (c) i) Could the row-buffer hit rate for the sequence be 100%? Why or why not? Explain.

**Solution:**

The four cache blocks are mapped to different rows. Hence the row-buffer hit rate can be 100% only if the row containing each cache block is already open.

ii) If yes, what is the minimum number of banks required to achieve a row-buffer hit rate of 100%?

**Solution:**

4 banks is sufficient to achieve this, if the four rows containing the four cache blocks are already open (at each of the four banks).

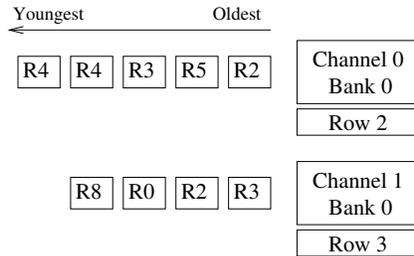
### 3 Memory Scheduling

A machine has a DRAM main memory of 4 KB, organized as 2 channels, 1 rank/channel and 4 banks/rank. The system does not have virtual memory.

An open row policy is used, i.e., a row is retained in the row-buffer after an access, until an access to another row is made.

Following are the commands issued to DRAM, to access data.

- **ACTIVATE:** Loads the row (that needs to be accessed) into the bank's row-buffer. This is called opening a row. (Latency: 15ns)
  - **PRECHARGE:** Restores the contents of the banks row-buffer back into the row. This is called closing a row. (Latency: 15ns)
  - **READ/WRITE:** Accesses data from the row-buffer. (Latency: 15ns)
- (a) Application A runs alone on this machine. The following figure shows a snapshot of the request buffers at time  $t$ . Each request is tagged with the index of the row it is destined to. Row 2 is currently open in bank 0 of channel 0 and row 3 is currently open in bank 0 of channel 1.



Application A is stalled until all of these memory requests are serviced and does not generate any more requests.

What is the stall time of application A using

i) an FCFS scheduling policy

**Solution:**

**165 ns**

At bank 0, channel 0, two row-buffer hits (request to row 2 and second request to row 4) and three row-buffer conflicts.

Stall time at channel 0 =  $(15 \times 2) + (45 \times 3) = 165$  ns.

At bank 0, channel 1, one row-buffer hit (request to row 3) and three row-buffer conflicts.

Stall time at channel 1 =  $(15 \times 1) + (45 \times 3) = 150$  ns.

The time for which application A stalls is the longer of these two stall times. Hence, the stall time of application A = 165 ns.

ii) an FR-FCFS scheduling policy?

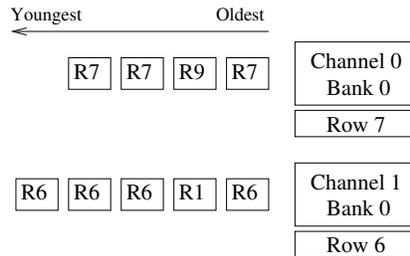
**Solution:**

**165 ns**

At both banks (and channels), the scheduling order is the exactly the same as when using FCFS, as the same requests hit in the row-buffer in both cases. Hence, the stall time of application A = 165 ns.

- (b) Now, application B runs alone on this machine. The following figure shows a snapshot of the request buffers at time t. Each request is tagged with the index of the row it is destined to. Row 7 is currently open in bank 0 of channel 0 and row 6 is currently open in bank 0 of channel 1.

Application B is stalled until all of these memory requests are serviced and does not generate any more requests.



What is the stall time of application B using

i) an FCFS scheduling policy

**Solution:**

**135 ns**

At bank 0, channel 0, two row-buffer hits and two row-buffer conflicts.

Stall time at channel 0 =  $(15 \times 2) + (45 \times 2) = 120$  ns.

At bank 0, channel 1, three row-buffer hits and two row-buffer conflicts.

Stall time at channel 1 =  $(15 \times 3) + (45 \times 2) = 135$  ns.

Hence, stall time of application B is 135 ns.

ii) an FR-FCFS scheduling policy?

**Solution:**

**105 ns**

At bank 0, channel 0, all three requests to row 7 are row-buffer hits, while only the request to row 9 is a row-buffer conflict.

Stall time at channel 0 =  $(15 \times 3) + (45 \times 1) = 90$  ns.

At bank 0, channel 1, all four requests to the open row, row 6, are row-buffer hits and only the request to row 1 is a row-buffer conflict.

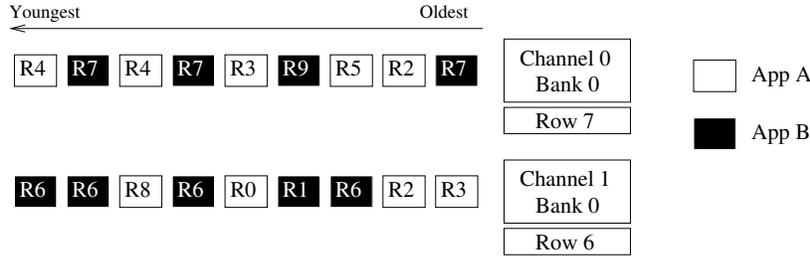
Stall time at channel 1 =  $(15 \times 4) + (45 \times 1) = 105$  ns.

Hence, stall time of application B is 105 ns.

Application B has higher row-buffer locality and hence benefits from FR-FCFS memory scheduling.

- (c) Applications A and B are run on the same machine. The following is a snapshot of the request buffers at time t. Requests are tagged with the index of the row they are destined to. Additionally, requests of applications A and B are indicated with different colors. Row 7 is currently open in bank 0 of channel 0 and row 6 is currently open in bank 0 of channel 1.

An application is stalled until all of its memory requests are serviced and does not generate any more requests.



What is the stall time of application A using

i) an FCFS scheduling policy

**Solution:**

**375 ns**

At bank 0, channel 0, application A stalls for 1 row-buffer hit latency and 8 row-buffer conflict latencies.

Stall time at channel 0 =  $(15 \times 1) + (45 \times 8) = 375\text{ns}$ .

At bank 0, channel 1, it stalls for 7 row-buffer conflict latencies.

Stall time at channel 1 =  $45 \times 7 = 315\text{ns}$ .

Hence, stall time of application A is 375 ns.

ii) an FR-FCFS scheduling policy?

**Solution:**

**285 ns**

At bank 0, channel 0, application A stalls for 4 row-buffer hit latencies and 5 row-buffer conflict latencies.

Stall time at channel 0 =  $(15 \times 4) + (45 \times 5) = 285\text{ns}$ .

At bank 0, channel 1, it stalls for 4 row-buffer hit latencies and 5 row-buffer conflict latencies.

Stall time at channel 1 =  $(15 \times 4) + (45 \times 5) = 285\text{ns}$ .

Hence, stall time of application A is 285 ns.

What is the stall time of application B using

i) an FCFS scheduling policy

**Solution:**

**375 ns**

At bank 0, channel 0, application B stalls for 1 row-buffer hit latency and 7 row-buffer conflict latencies.

Stall time at channel 0 =  $(15 \times 1) + (45 \times 7) = 330\text{ns}$ .

At bank 0, channel 1, 1 row-buffer hit latency and 8 row-buffer conflict latencies.

Stall time at channel 1 =  $(15 \times 1) + (45 \times 8) = 375\text{ns}$ .

Hence, stall time of application B is 375 ns.

ii) an FR-FCFS scheduling policy?

**Solution:**

**195 ns**

At bank 0, channel 0, 3 row-buffer hit latencies and 3 row-buffer conflict latencies.

Stall time at channel 0 =  $(15 \times 3) + (45 \times 3) = 180\text{ns}$ .

At bank 0, channel 1, 4 row-buffer hit latencies and 3 row-buffer conflict latencies.

Stall time at channel 1 =  $(15 \times 4) + (45 \times 3) = 195\text{ns}$ .

Hence, stall time of application B is 195 ns.

As can be observed again, application B has higher row-buffer locality and hence benefits more from FR-FCFS scheduling.

## 4 Main Memory Potpourri

A machine has a 4 KB DRAM main memory system. Each row is refreshed every 64 ms.

**Note: This question is open ended. We provide one possible set of solutions. There could be other possible right solutions.**

- (a) The machine's designer runs two applications A and B (each run alone) on the machine. Although applications A and B have a similar number of memory requests, application A spends a surprisingly larger fraction of cycles stalling for memory than application B does? What might be the reasons for this?

**Solution:**

A large number of application A's memory requests are row-buffer conflicts, whereas a large number of application B's memory requests are row-buffer hits.

Hence, application A's requests take longer to service and it spends more time stalling for memory.

- (b) Application A also consumes a much larger amount of memory energy than application B does. What might be the reasons for this?

**Solution:**

Row-buffer conflicts also consume more energy, as they require a precharge, activate and a read/write, whereas row-buffer hits only require a read/write.

Hence, application A consumes more memory energy.

- (c) When applications A and B are run together on the machine, application A's performance degrades significantly, while application B's performance doesn't degrade as much. Why might this happen?

**Solution:**

When the applications are run together, they interfere with each other. Hence, both applications' performance degrades when they run together. However, if a memory scheduler that favor row-buffer hits over row-buffer conflicts (like FR-FCFS) is used, it would favor application B's requests over application A's requests.

Therefore, application A's performance degrades more.

- (d) The designer decides to use a smarter policy to refresh the memory. A row is refreshed only if it has not been accessed in the past 64 ms. Do you think this is a good idea? Why or why not?

**Solution:**

This can reduce refresh energy significantly if a large fraction of the rows in memory contain data and are accessed (within the 64 ms refresh window), as these rows do not have to be refreshed explicitly. However, if only a small number of rows contain data and only these rows are accessed, this policy will not provide much reduction in refresh energy as a large fraction of rows are still refreshed at the 64 ms rate.

- (e) The refresh energy consumption when application B is run, drops significantly when this new refresh policy is applied, while the refresh energy when application A is run reduces only slightly. Is this possible? Why or why not?

**Solution:**

This is possible. If application B has a large working set, it could access a large fraction of the memory rows (within the 64 ms refresh window) and hence these rows do not have to be refreshed explicitly. Application A on the other could have a much smaller working set and hence a large fraction of rows still have to be refreshed at the 64 ms rate.