ETH 263-2210-00L Computer Architecture, Fall 2020

HW 4: Memory Interference and QoS, Memory Scheduling, Memory Controller, Emerging Memory Technologies (SOLUTIONS)

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, João Dinis Ferreira, Rahul Bera, Geraldo Francisco De Oliveira Junior, Can Firtina, Juan Gómez Luna, Jawad Haj-Yahya, Hasan Hassan, Konstantinos Kanellopoulos, Jeremie Kim, Nika Mansouri Ghiasi, Haiyu Mao, Lois Orosa Nogueira, Jisung Park, Minesh Patel, Gagandeep Singh, Kosta Stojiljkovic, Abdullah Giray Yaglikci

Given: Thursday, Nov 20, 2020
Due: **Thursday, Dec 4, 2020**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture20/`. Please check your inbox. You should have received an email with the password you should use to login. If you did not receive any emails, contact comparch@lists.inf.ethz.ch. In the first page after login, you should click in "Computer Architecture Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-8).** You should upload your answers to the Moodle Platform (`https://moodle-app2.let.ethz.ch/course/view.php?id=13549`) as a single PDF file.

1. Critical Paper Reviews [500 points]

Please read the guidelines for reviewing papers and check the sample reviews. We also assign you a **required reading** for this homework. You may access them by *simply clicking on the QR codes below or scanning them.* We will give out extra credit that is worth 0.5% of your total grade for each good review. If you review a paper other than the REQUIRED papers, you will receive 250 BONUS points on top of 500 points you may get from paper reviews (i.e., each additional submission is worth 250 BONUS points with a possibility to get upto 3000 points).

Guidelines | Sample reviews | Required Reading 1 | Required Reading 2

Write an approximately one-page critical review for the following required readings (i.e., Paper #1, and Paper #2) and earn *bonus* points for the remaining 8 papers (i.e., papers from #3 to #10). A review with bullet point style is more appreciated. Try not to use very long sentences and paragraphs. Keep your writing and sentences simple. Make your points bullet by bullet, as much as possible.

1. (REQUIRED) B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "*Architecting Phase Change Memory as a Scalable DRAM Alternative*", in ISCA 2009. `https://people.inf.ethz.ch/omutlu/pub/pcm_isca09.pdf`
2. (REQUIRED) O. Mutlu and T. Moscibroda, "*Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems*", in ISCA 2008. `https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf`
3. E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "*Self Optimizing Memory Controllers: A Reinforcement Learning Approach*" in ISCA 2008. `https://people.inf.ethz.ch/omutlu/pub/rlmc_isca08.pdf`
4. Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "*ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers"* in HPCA 2010. `https://people.inf.ethz.ch/omutlu/pub/atlas_hpca10.pdf`.
5. L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "*BLISS: Balancing performance, fairness and complexity in memory access scheduling"* in TPDS 2016. `https://people.inf.ethz.ch/omutlu/pub/bliss-memory-scheduler_ieee-tpds16.pdf`

6. M. Patel, J. S. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "*Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics*", in MICRO, 2020 https://people.inf.ethz.ch/omutlu/pub/BEER-bit-exact-ECC-recovery_micro20.pdf
7. N. Hajinazar, P. Patel, M. Patel, K. Kanellopoulos, S. Ghose, R. Ausavarungnirun, G. F. de Oliveira Jr., J. Appavoo, V. Seshadri, and O. Mutlu, "*The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework"* in ISCA 2020. https://people.inf.ethz.ch/omutlu/pub/VBI-virtual-block-interface_isca20.pdf
8. Y. Kim, W. Yang, and O. Mutlu, "*Ramulator: A Fast and Extensible DRAM Simulator"* in CAL 2015. https://people.inf.ethz.ch/omutlu/pub/ramulator_dram_simulator-ieee-cal15.pdf
9. H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "*Row Buffer Locality Aware Caching Policies for Hybrid Memories"* in ICCD 2012. https://people.inf.ethz.ch/omutlu/pub/rowbuffer-aware-caching_iccd12.pdf
10. M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "*Scalable high performance main memory system using phase-change memory technology"* in ISCA 2009. https://dl.acm.org/doi/pdf/10.1145/1555754.1555760
11. Kültürsay E, Kandemir M, Sivasubramaniam A, Mutlu O, "*Evaluating STT-RAM as an energy-efficient main memory alternative"* in ISPASS 2013. https://people.inf.ethz.ch/omutlu/pub/sttram_ispass13.pdf
12. L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "*The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"* in MICRO 2015. https://people.inf.ethz.ch/omutlu/pub/application-slowdown-model_micro15.pdf
13. E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "*Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems,"* in ASPLOS 2010. https://people.inf.ethz.ch/omutlu/pub/fairness-via-throttling_acm_tocs12.pdf
14. S.P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, T. Moscibroda, "*Reducing Memory Interference in Multicore Systems via Application-aware Memory Channel Partitioning,"* MICRO 2011. https://people.inf.ethz.ch/omutlu/pub/memory-channel-partitioning-micro11.pdf

2. Memory System [200 points]

A machine with a 4 GB DRAM main memory system has 4 channels, 1 rank per channel and 4 banks per rank. The cache block size is 64 bytes.

(a) You are given the following byte addresses and the channel and bank to which they are mapped:

Byte:	$0x0000$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0100$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0200$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0400$	$\Rightarrow$	Channel	1,	Bank	0
Byte:	$0x0800$	$\Rightarrow$	Channel	2,	Bank	0
Byte:	$0x0C00$	$\Rightarrow$	Channel	3,	Bank	0
Byte:	$0x1000$	$\Rightarrow$	Channel	0,	Bank	1
Byte:	$0x2000$	$\Rightarrow$	Channel	0,	Bank	2
Byte:	$0x3000$	$\Rightarrow$	Channel	0,	Bank	3

Determine which bits of the address are used for each of the following address components. Assume row bits are higher order than column bits:

- Byte on bus
  Addr [2 : 0]
- Channel bits (channel bits are contiguous)
  Addr [$\underline{11}$: $\underline{10}$]
- Bank bits (bank bits are contiguous)
  Addr [$\underline{13}$: $\underline{12}$]
- Column bits (column bits are contiguous)
  Addr [$\underline{9}$: $\underline{3}$]
- Row bits (row bits are contiguous)
  Addr [$\underline{31}$: $\underline{14}$]

(b) Two applications App 1 and App 2 share this memory system (using the address mapping scheme you determined in part (a)). The memory scheduling policy employed is FR-FCFS. The following requests are queued at the memory controller request buffer at time t. Assume the first request (A) is the oldest and the last one (A + 15) is the youngest.

A B A + 1 A + 2 A + 3 B + 10 A + 4 B + 12 A + 5 A + 6 A + 7
A + 8 A + 9 A + 10 A + 11 A + 12 A + 13 A + 14 A + 15

These are cache block addresses, not byte addresses. Note that requests to A + x are from App 1, while requests to B + x are from App 2. Addresses A and B are row-aligned (i.e., they are at the start of a row) and are at the same bank but are in different rows.

Assuming row-buffer hits take T time units to service and row-buffer conflicts/misses take 2T time units to service, what is the slowdown (compared to when run alone on the same system) of

- App 1?

1.

All requests A + x map to one row, and all requests B + x map to another row (both rows are in the same bank), because there are 16 cache blocks/row and in all requests above, x < 16. Since the request for cache block address A comes first, the row containing all requested cache blocks A+x will be opened and all of these requests, which are row-buffer hits come first (and will complete in time 1 2T + 15 1T = 17T). Thus, none of App 1s requests are ever delayed by requests from App 2, and so they all execute at exactly the same time as they would if App 2 were not running. This results in a slowdown of 1.

- App 2?

21/4.

When running alone, App 2s three requests are a row buffer-closed access (time 2T) and two row buffer hits (each taking time T); thus they complete in 1 2T + 2 1T = 4T time. When running with App 1, all of App 1s requests come first, in time 17T (see above). Then App 2s requests execute as in the alone case (row conflict, row hit, row hit) in 4T time. Hence App 2s requests are completed at time 21T. Slowdown is thus 21T /4T = 21/4.

(c) Which application slows down more?

App 2.

Why?

The high row-buffer locality of App 1 causes its requests to occupy the bank for a long period of time with the FR-FCFS scheduling policy, denying App 2 of service during that period.

(d) In class, we discussed memory channel partitioning and memory request scheduling as two solutions to mitigate interference and application slowdowns in multicore systems. Propose another solution to reduce the slowdown of the more-slowed-down application, without increasing the slowdown of the other application? Be concrete.

Interleaving data at a sub-row or cache line granularity could reduce the slowdown of App 2 by reducing the row-buffer locality of App 1 which causes the interference.

One possible interleaving scheme that achieves this is shown below:

- Byte on bus Addr [2 : 0]
- Lower Column bits Addr [7 : 3]
- Channel bits Addr [9 : 8]
- Bank bits Addr [11 : 10]
- Higher Column bits Addr [13 : 12]
- Row bits Addr [31 : 14]

This address interleaving scheme interleaves 256 KB chunks across channels. Thus, the longest row hit streak would be 4, as compared to 16 in the original interleaving scheme in part (a), preventing App 2s requests from being queued behind 16 of App 1s requests.

3. DRAM Scheduling and Latency [200 points]

You would like to understand the configuration of the DRAM subsystem of a computer using reverse engineering techniques. Your current knowledge of the particular DRAM subsystem is limited to the following information:

- The physical memory address is 16 bits.
- The DRAM subsystem consists of a single channel, 2 banks, and 64 rows per bank.
- The DRAM is byte-addressable.
- The most-significant bit of the physical memory address determines the bank. The following 6 bits of the physical address determine the row.
- The DRAM command bus operates at 1 GHz frequency.
- The memory controller issues commands to the DRAM in such a way that *no command* for servicing a *later* request is issued before issuing a READ command for the current request, which is the oldest request in the request buffer. For example, if there are requests A and B in the request buffer, where A is the older request and the two requests are to different banks, the memory controller does *not* issue an ACTIVATE command to the bank that B is going to access *before* issuing a READ command to the bank that A is accessing.
- The memory controller services requests in order with respect to each bank. In other words, for a given bank, the memory controller first services the oldest request in the *request buffer* that targets the same bank. If all banks are ready to service a request, the memory controller first services the oldest request in the request buffer.

You realize that you can observe the memory requests that are waiting to be serviced in the request buffer. At a particular point in time, you take the snapshot of the request buffer and you observe the following requests in the request buffer (in descending order of request age, where the oldest request is on the top):

time ↓

```
Read 0xD780
Read 0x280C
Read 0xE4D0
Read 0x2838
```

At the same time you take the snapshot of the request buffer, you start probing the DRAM command bus. You observe the DRAM command type and the cycle (relative to the first command) at which the command is seen on the DRAM command bus. The following are the DRAM commands you observe on the DRAM bus while the requests above are serviced.

```
Cycle 0  --- READ
Cycle 1  --- PRECHARGE
Cycle 8  --- PRECHARGE
Cycle 13 --- ACTIVATE
Cycle 18 --- READ
Cycle 20 --- ACTIVATE
Cycle 22 --- READ
Cycle 25 --- READ
```

Answer the following questions using the information provided above.

(a) What are the following DRAM timing parameters used by the memory controller, in terms of nanoseconds? If there is not enough information to infer the value of a timing parameter, write *unknown*.

i) ACTIVATE-to-READ latency:

5 ns.

Explanation. After issuing the ACTIVATE command at cycle 13, the memory controller waits until cycle 18, which indicates that the ACTIVATE-to-READ latency is 5 cycles. The command bus operates at 1 GHz, so it has 1 ns clock period. Thus, the ACTIVATE-to-READ is $5 * 1 = 5$ ns.

ii) ACTIVATE-to-PRECHARGE latency:

Unknown.

Explanation. In the command sequence above, there is not a PRECHARGE command that follows an ACTIVATE command with a known issue cycle. Thus, we cannot determine the ACTIVATE-to-PRECHARGE latency.

iii) PRECHARGE-to-ACTIVATE latency:

12 ns.

Explanation. The PRECHARGE-to-ACTIVATE latency can be easily seen in the first two commands at cycles 1 and 13. The PRECHARGE-to-ACTIVATE latency is 12 cycles = 12 ns.

iv) READ-to-PRECHARGE latency:

8 ns.

Explanation. The READ command at cycle 0 is followed by a PRECHARGE command to the same bank at cycle 8. There are idle cycles before cycle 8, which indicates that the memory controller delayed the PRECHARGE command until cycle 8 because the timing constaints but not because the command bus was busy. Thus, the READ-to-PRECHARGE is 8 cycles, which is $8 * 1 = 8$ ns for the 1 GHz DRAM command bus.

v) READ-to-READ latency:

4 ns.

Explanation. Bank 0 receives back-to-back reads at cycles 18 and 22. The READ-to-READ latency is 4 cycles, which is $4 * 1 = 4$ ns for the 1 GHz DRAM command bus.

(b) What is the status of the banks *prior* to the execution of any of the above requests? In other words, which rows from which banks were open immediately prior to issuing the DRAM commands listed above? Fill in the table below indicating whether a bank has an open row, and if there is an open row, specify its address. If there is not enough information to infer the open row address, write *unknown*.

	Open or Closed?	**Open Row Address**
Bank 0	Open	Unknown
Bank 1	Open	43

Explanation. By decoding the accessed addresses we can find which bank and row each access targets. Looking at the commands issued for those requests, we can determine which requests needed PRECHARGE (row buffer conflict, the initially open row is unknown in this case), ACTIVATE (the bank is initially closed), or directly READ (the bank is initially open and the open row is the same as the one that the request targets).

time ↓

```
0xD780 → Bank:  1, Row:  43 (Row hit, so Bank 1 must have row 43 open.)
0x280C → Bank:  0, Row:  20 (PRECHARGE first. Any row other than 20 might have been open.)
0xE4D0 → Bank:  1, Row:  50
0x2838 → Bank:  0, Row:  20
```

(c) To improve performance, you decide to implement the idea of Tiered-Latency DRAM (TL-DRAM) in the DRAM chip. Assume that a bank consists of a single subarray. With TL-DRAM, an entire bank is divided into a near segment and far segment. When accessing a row in the near segment, the ACTIVATE-to-READ latency *reduces* by 1 cycle and the ACTIVATE-to-PRECHARGE latency reduces by 3 cycles. When precharging a row in the near segment, the PRECHARGE-to-ACTIVATE latency reduces by 3 cycles. When accessing a row in the far segment, the ACTIVATE-to-READ latency *increases* by 1 cycle and the ACTIVATE-to-PRECHARGE latency increases by 2 cycles. When precharging a row in the far segment, the PRECHARGE-to-ACTIVATE latency increases by 2 cycles. The following table summarizes the changes in the affected latency parameters.

Timing Parameter	**Near Segment Latency**	**Far Segment Latency**
ACTIVATE-to-READ	-1	$+1$
ACTIVATE-to-PRECHARGE	-3	$+2$
PRECHARGE-to-ACTIVATE	-3	$+2$

Assume that the rows in the near segment have smaller row ids compared to the rows in the far segment. In other words, physical memory row addresses 0 through $N-1$ are the near-segment rows, and physical memory row addresses N through 63 are the far-segment rows.
If the above DRAM commands are issued 2 cycles faster with TL-DRAM compared to the baseline (the last command is issued in cycle 23), how many rows are in the near segment, i.e., what is N? Show your work.

The rows in the range of [0-43] should definitely be in the near segment. Row 50 should definitely be in the far segment. Thus, N is a number between [44-50].

Explanation. There should be at least 44 rows in the near segment (rows 0 to 43) since rows until row id 43 need to be accessed with low latency to get 2 cycle reduction. The unknown open row in bank 0 should be in the near segment to get the 2 cycle improvement. Row 50 is in the far segment because if it was in the near segment, the command would have been finished in cycle 21, i.e., 4 cycles sooner instead of 2 cycles sooner. Thus, the number of rows in the near segment N is a number between 44 and 50.
Here is the new command trace:

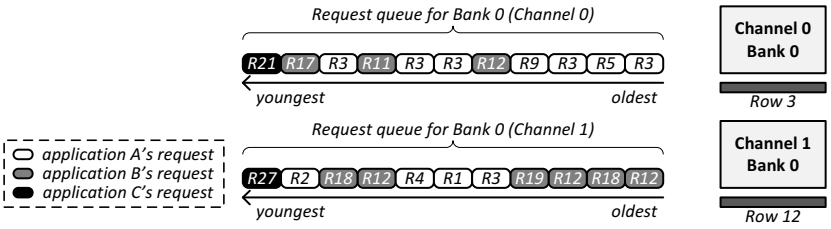
```
Cycle 0 -- READ - Bank 1
Cycle 1 -- PRECHARGE - Bank 0, an unknown row in the near segment
Cycle 8 -- PRECHARGE - Bank 1, row 43, which is in the near segment
Cycle 10 -- ACT - Bank 0, row 20, which is in the near segment
Cycle 14 -- READ - Bank 0
Cycle 17 -- ACTIVATE - Bank 1, Row 50, which is in the far segment
Cycle 18 -- READ - Bank 0
Cycle 23 -- READ - Bank 1, Row 0
```

4. Memory Scheduling [200 points]

To serve a memory request, the memory controller issues one or multiple DRAM commands to access data from a bank. There are four different DRAM commands.

- `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 bank's row-buffer back into the row. This is called *closing* a row. **(Latency: 15ns)**
- `READ/WRITE`: Accesses data from the row-buffer. **(Latency: 15ns)**

The following figure shows the snapshot of the memory request buffers (in the memory controller) at t_0. Each request is color-coded to denote the application to which it belongs (assume that all applications are running on separate cores). Additionally, each request is annotated with the address (or index) of the row that the request needs to access (e.g., *R3* means that the request is to the 3^{rd} row). Furthermore, assume that all requests are read requests.



A memory request is considered to be *served* when the `READ` command is complete (i.e., 15ns after the request's `READ` command has been issued). In addition, each application (A, B, or C) is considered to be **stalled** until *all* of its memory requests (across all the request buffers) have been served.

Assume that, initially (at t_0) each bank has the 3^{rd} and the 12^{th} row loaded in the row-buffer, respectively. Furthermore, no additional requests from any of the applications arrive at the memory controller.

4.1. Application-Unaware Scheduling Policies

(a) Using the **FCFS** scheduling policy, what is the **stall time** of each application?

App A: MAX(2H+7M, H+9M) = H+9M = 15+405 = 420ns

App B: MAX(2H+8M, H+8M) = 2H+8M = 30+360 = 390ns

App C: MAX(2H+9M, H+10M) = H+10M = 15+450 = 465ns

(b) Using the **FR-FCFS** scheduling policy, what is the stall time of each application?

App A: MAX(5H+2M, (4H+2M)+4M) = 4H+6M = 60+270 = 330ns

App B: MAX((5H+2M)+3M, 4H+2M) = 5H+5M = 75+225 = 300ns

App C: MAX(((5H+2M)+3M)+M, ((4H+2M)+4M)+M) = 4H+7M = 60 + 315 = 375ns

(c) What property of memory references does the *FR-FCFS* scheduling policy exploit? (Three words or less.)

Row-buffer locality

(d) Briefly describe the scheduling policy that would **maximize** the *request throughput* at any given bank, where request throughput is defined as the number of requests served per unit amount of time. (Less than 10 words.)

FR-FCFS

4.2. Application-Aware Scheduling Policies

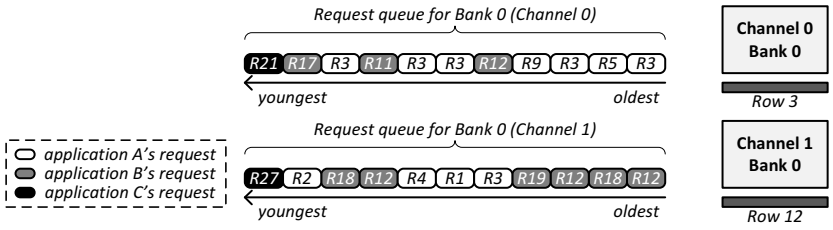
Of the three applications, application C is the least memory-intensive (i.e., has the lowest number of outstanding requests). However, it experiences the largest stall time since its requests are served only after the numerous requests from other applications are first served. To ensure the shortest stall time for application C, one can assign its requests with the highest priority, while assigning the same low priority to the other two applications (A and B).

(a) **Scheduling Policy X:** When application C is assigned a high priority and the other two applications are assigned the same low priority, what is the stall time of each application? (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

App A: MAX(M+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405ns

App B: MAX(M+(4H+3M)+3M, M+(3H+3M)) = 4H+7M = 60+315 = 375ns

App C: MAX(M, M) = M = 45ns



Can you design an even better scheduling policy? While application C now experiences low stall time, you notice that the other two applications (A and B) are still delaying each other.

(b) Assign priorities to the other two applications such that you minimize the average stall time across all applications. Specifically, list all **three** applications in the order of highest to lowest priority. (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

C > B > A

(c) **Scheduling Policy Y:** Using your new scheduling policy, what is the stall time of each application? (Among requests with the same priority, assume that *FR-FCFS* is used to break ties.)

App A: MAX(M+(3M)+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405ns

App B: MAX(M+(3M), M+(3H+3M)) = 3H+4M = 45+180 = 225ns

App C: MAX(M, M) = M = 45ns

(d) Order the four scheduling policies (FCFS, FR-FCFS, X, Y) in the order of lowest to highest average stall time.

Y<X<FR-FCFS<FCFS

5. Memory Request Scheduling [200 points]

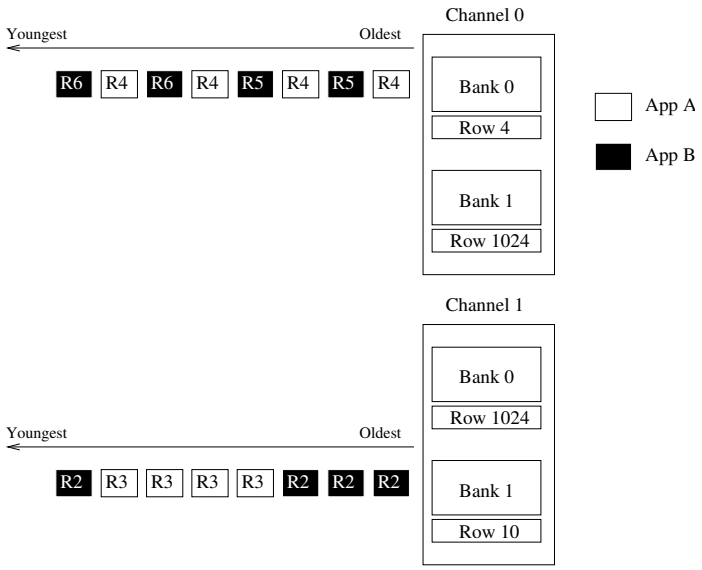
A machine has a DRAM main memory organized as 2 channels, 1 rank and 2 banks/channel. An open row policy is used, i.e., a row is retained in the row-buffer after an access until an access to another row is made. The following commands (defined as we discussed in class) can be issued to DRAM with the given latencies:

- ACTIVATE: 15 ns
- PRECHARGE: 15 ns
- READ/WRITE: 15 ns

Assume the bus latency is 0 cycles.

(a) We studied Parallelism Aware Batch Scheduling (PAR-BS) in class. We will use a PAR-BS-like scheduler that we will call X. This scheduler operates as follows:

- The scheduler forms request batches consisting of the 4 oldest requests from each application at each bank.
- At each bank, the scheduler ranks applications based on the number of requests they have outstanding at that bank. Applications with a smaller number of requests outstanding are assigned a higher rank.
- The scheduler always ranks the application with the oldest request higher in the event of a tie between applications.
- The scheduler prioritizes the requests of applications based on this ranking (higher ranked applications requests are prioritized over lower ranked applications requests).
- The scheduler repeats the above steps once all requests in a batch are serviced at all banks.



For the same above buffer state:
What is the stall time of application A using this scheduler?

180 ns

First, at each bank, scheduler X groups four oldest requests from each application into a batch. In this case, the batch happens to contain all of the outstanding requests.

Next, since both applications have an equal number of outstanding requests at all banks, the application with the oldest request at a bank is prioritized. This is application A at channel 0, bank 0 and application B at channel 1, bank 1. The resulting scheduling order is exactly the same as when using an FR-FCFS scheduler. Therefore, the stall time of application A is **180 ns.**

What is the stall time of application B using this scheduler?

180 ns

As the request scheduling order is exactly the same as when FR-FCFS is used (see above), the stall time of application B is 180 ns.

Explain why or why not. Provide the fundamental reason why X does or does not improve system performance over an FR-FCFS scheduler.

Scheduler X prioritizes the requests of the application with the oldest request at each bank. Hence, different applications might be prioritized at each bank. Specifically, application A is prioritized at channel 0, bank 0, while application B is prioritized at channel 1, bank 1. This mismatch in prioritization decisions means that neither application will have a shorter stall time because each application waits for its requests at the bank at which it was not prioritized.

(b) Can you design a better memory scheduler (i.e., one that provides higher system performance) than X? Circle one: **YES** **NO**

If yes, answer the questions below. What modifications would you make to scheduler X to design this better scheduler Y? Explain clearly.

Prioritize the same applications requests at both channels/banks (i.e., ensure that all banks prioritize the same application over the other, instead of one bank prioritizing one application whereas the other prioritizing the other). Pick the application with the shortest stall time to prioritize in all banks. For the set of requests given, the application with the shortest stall time is application A because it has the same number of requests in each bank as B but some of its requests are to already-open rows.

What is the stall time of application A using this scheduler Y?

90 ns

Application A's requests are serviced first at both banks (and channels). At channel 0, bank 0, all four requests to row 4 are row-buffer hits and incur $15 \times 4 = 60ns$. At channel 1, bank 1, application A's first request to row 3 is a row-buffer conflict, while application A's next three requests to row 3 are row-buffer hits. Therefore, stall time is $45 \times 1 + 15 \times 3 =$ **90 ns.**

What is the stall time of application B using this scheduler Y?

180 ns

Application B's requests are scheduled after application A's requests at both banks (and channels). The stall time of application B is **180 ns.**

(c) Consider a simple channel partitioning scheme where application A's data is mapped to channel 0 and application B's data is mapped to channel 1. When data is mapped to a different channel, only the channel number changes; the bank number does not change. For instance, requests of application A that were mapped to bank 1 of channel 1 would now be mapped to bank 1 of channel 0. What is the stall time of application A using this channel partitioning mechanism and an FR-FCFS memory scheduler?

90 ns

All of application A's data are mapped to channel 0. The four requests to row 4 are still mapped to channel 0, bank 0 and are all row-buffer hits. application A's stall time at bank 0 is $15 \times 4 = 60ns$.

The four requests to row 3 go to channel 1, bank 1. One of them is a row-buffer conflict, while the remaining three are row-buffer hits. application A's stall time at bank 1 is $45 \times 1 + 15 \times 3 = 90ns$.

Therefore, application A's stall time is **90 ns**.

What is the stall time of application B using this channel partitioning mechanism and an FR-FCFS memory scheduler?

120 ns

application B's requests that went to channel 0, bank 0 now go to channel 1, bank 0. Two of these are row-buffer conflicts, while two are row-buffer hits. Stall time at bank 0 is $45 \times 2 + 15 \times 2 = 120ns$.

At bank 1, application B has one row-buffer conflict and three row-buffer hits. Stall time at bank 1 is $45 \times 1 + 15 \times 3 = 90ns$.

Therefore, application B's stall time is the longer of the stall times at the two banks, **120 ns.**

Explain why channel partitioning does better or worse than scheduler Y.

Because it eliminates interference by mapping application A and Bs request streams to different channels.

6. Emerging Memory Technologies [100 points]

Computer scientists at ETH developed a new memory technology, ETH-RAM, which is non-volatile. The access latency of ETH-RAM is close to that of DRAM while it provides higher density compared to the latest DRAM technologies. ETH-RAM has one shortcoming, however: it has limited endurance, i.e., a memory cell stops functioning after 10^6 writes are performed to the cell (known as cell wear-out).

A bright ETH student has built a computer system using 1 GB of ETH-RAM as main memory. ETH-RAM exploits a perfect wear-leveling mechanism, i.e., a mechanism that equally distributes the writes over all of the cells of the main memory.

(a) This student is worried about the lifetime of the computer system she has built. She executes a test program that runs special instructions to bypass the cache hierarchy and repeatedly writes data into different words until **all** the ETH-RAM cells are worn-out (stop functioning) and the system becomes useless. The student's measurements show that ETH-RAM stops functioning (i.e., all its cells are worn-out) in one year (365 days). Assume the following:

- The processor is in-order and there is no memory-level parallelism.
- It takes 5 ns to send a memory request from the processor to the memory controller and it takes 28 ns to send the request from the memory controller to ETH-RAM.
- ETH-RAM is word-addressable. Thus, each write request writes 4 bytes to memory.

What is the write latency of ETH-RAM? Show your work.

$t_{wear\_out} = \frac{2^{30}}{2^2} \times 10^6 \times (t_{write\_MLC} + 5 + 28)$
$365 \times 24 \times 3600 \times 10^9 \text{ns} = 2^{28} \times 10^6 \times (t_{write\_MLC} + 33)$
$t_{write\_MLC} = \frac{365 \times 24 \times 3600 \times 10^3}{2^{28}} - 33 = 84.5\text{ns}$

Explanation:

- Each memory cell should receive 10^6 writes.
- Since ETH-RAM is word addressable, the required amount of writes is equal to $\frac{2^{30}}{2^2} \times 10^6$ (there is no problem if 1 GB is assumed to be equal to 10^9 bytes).
- The processor is in-order and there is no memory-level parallelism, so the total latency of each memory access is equal to $t_{write\_MLC} + 5 + 28$.

(b) ETH-RAM works in the multi-level cell (MLC) mode in which each memory cell stores 2 bits. The student decides to improve the lifetime of ETH-RAM cells by using the single-level cell (SLC) mode. When ETH-RAM is used in SLC mode, the lifetime of each cell improves by a factor of 10 and the write latency decreases by 70%. What is the lifetime of the system using the SLC mode, if we repeat the experiment in part (a), with everything else remaining the same in the system? Show your work.

$t_{wear\_out} = \frac{2^{29}}{2^2} \times 10^7 \times (25.35 + 5 + 28) \times 10^{-9}$
$t_{wear\_out} = 78579686.3\text{s} = 2.49 \text{ year}$

Explanation:

- Each memory cell should receive $10 \times 10^6 = 10^7$ writes.
- The memory capacity is reduced by 50% since we are using SLC: $Capacity = 2^{30}/2 = 2^{29}$
- The required amount of writes is equal to $\frac{2^{29}}{2^2} \times 10^7$.
- The SLC write latency is $0.3 \times t_{write\_MLC}$: $t_{write\_SLC} = 0.3 \times 84.5 = 25.35\,\text{ns}$

7. BossMem [100 points]

A researcher has developed a new type of nonvolatile memory, BossMem. He is considering BossMem as a replacement for DRAM. BossMem is 10x faster (all memory timings are 10x faster) than DRAM, but since BossMem is so fast, it has to frequently power-off to cool down. Overheating is only a function of time, not a function of activity. An idle stick of BossMem has to power-off just as frequently as an active stick. When powered-off, BossMem retains its data, but cannot service requests. Both DRAM and BossMem are banked and otherwise architecturally similar. To the researcher's dismay, he finds that a system with 1GB of DRAM performs considerably better than the same system with 1GB of BossMem.

(i) What can the researcher change or improve in the core (he can't change BossMem or anything beyond the memory controller) that will make his BossMem perform more favorably compared to DRAM, realizing that he will have to be fair and evaluate DRAM with his enhanced core as well? (15 words or less)

Solution: Prefetcher degree or other speculation techniques so that misses can be serviced before memory powered off.

(ii) A colleague proposes he builds a hybrid memory system, with both DRAM and BossMem. He decides to place data that exhibits low row buffer locality in DRAM and data that exhibits high row buffer locality in BossMem. Assume 50% of requests are row buffer hits. Is this a good or bad idea? Show your work.

Solution: No, it may be better idea to place data with high row buffer locality in DRAM and low row buffer locality data in BossMem since row buffer misses are less costly.

(iii) Now a colleague suggests trying to improve the last-level cache replacement policy in the system with the hybrid memory system. Like before, he wants to improve the performance of this system relative to one that uses just DRAM and he will have to be fair in his evaluation. Can he design a cache replacement policy that makes the hybrid memory system look more favorable? In 15 words or less, justify NO or describe a cache replacement policy that would improve the performance of the hybrid memory system more than it would DRAM.

Solution: Yes, this is possible. Cost-based replacement where cost to replace is dependent on data allocation between DRAM and BossMem.

(iv) In class we talked about another nonvolatile memory technology, phase-change memory (PCM). Which technology, PCM, BossMem, or DRAM requires the greatest attention to security? What is the vulnerability?

Solution: PCM is nonvolatile and has potential endurance attacks.

(v) Which is likely of least concern to a security researcher?

Solution: DRAM is likely least vulnerable, as BossMem also has nonvolatility concerns.