Computer Architecture (263-2210-00L), Fall 2018
HW 4: Memory Interference and QoS
SOLUTIONS

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, Can Firtina, Hasan Hassan, Jeremie Kim, Juan Gómez Luna, Geraldo Francisco de Oliveira, Minesh Patel, Giray Yaglikci

Assigned: Wednesday, Nov 21, 2018
Due: **Wednesday, Dec 5, 2018**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture18/`. Please check your inbox. You should have received an email with the password you can use to login to the paper review system. If you have not received any email, please contact comparch@lists.ethz.ch. In the first page after login, you should click in "Architecture - Fall 2018 Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-5).** Please upload your solution to the Moodle (`https://moodle-app2.let.ethz.ch/`) as a single PDF file. **Please use a typesetting software (e.g., LaTeX) or a word processor (e.g., MS Word, LibreOfficeWriter) to generate your PDF file. Feel free to draw your diagrams either using an appropriate software or by hand, and include the diagrams into your solutions PDF.**

1 Critical Paper Reviews [150 points]

Please read the following handout on how to write critical reviews. We will give out extra credit that is worth 0.5% of your total grade for each good review.

- Lecture slides on guidelines for reviewing papers. Please follow this format.
  `https://safari.ethz.ch/architecture/fall2018/lib/exe/fetch.php?media=onur-comparch-f18-how-to-do-the-paper-reviews.pdf`
- Some sample reviews can be found here: `https://safari.ethz.ch/architecture/fall2018/doku.php?id=readings`

(a) Write a one-page critical review for the following paper:

- Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," Proceedings of the IEEE, 2017. `https://arxiv.org/pdf/1706.08642.pdf`

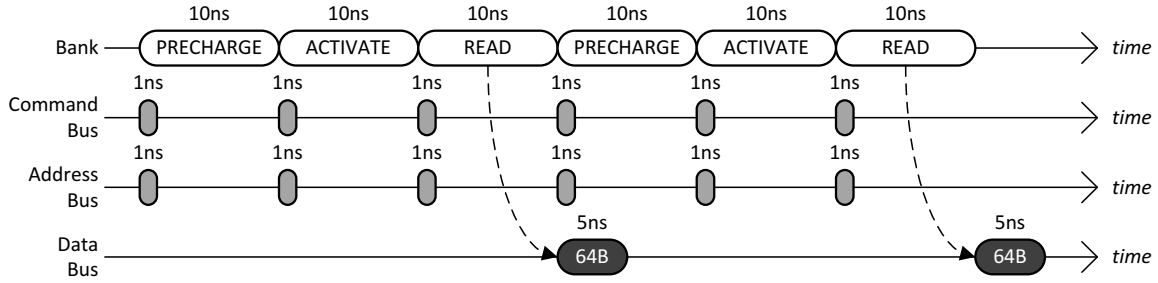
(b) Write a one-page critical review for at least **three** of the following papers:

- O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," ISCA 2008. `https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf`
- 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," ASPLOS 2010. `https://people.inf.ethz.ch/omutlu/pub/fst_asplos10.pdf`
- S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and 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`

- L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," TPDS 2016. https://people.inf.ethz.ch/omutlu/pub/bliss-memory-scheduler_ieee-tpds16.pdf
- A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS 2018. https://people.inf.ethz.ch/omutlu/pub/Google-consumer-workloads-data-movement-and-PIM_asplos18.pdf

2 Memory Interference and QoS [100 points]

Row-Buffer Conflicts. The following timing diagram shows the operation of a single DRAM channel and a single DRAM bank for two back-to-back reads that conflict in the row-buffer. Immediately after the bank has been busy for 10ns with a READ, data starts to be transferred over the data bus for 5ns.



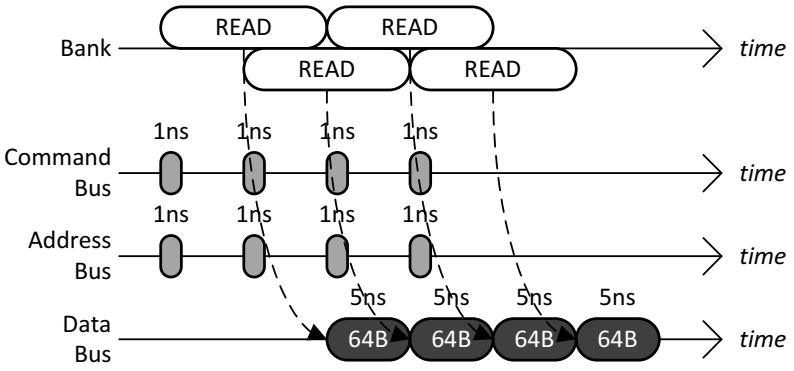
(a) Given a long sequence of back-to-back reads that always conflict in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

64B/30ns = 32B/15ns = 32GB/15s = 2.13 GB/s

(b) To increase the data throughput, the main memory designer is considering adding more DRAM banks to the single DRAM channel. Given a long sequence of back-to-back reads to all banks that always conflict in the row-buffers, what is the minimum number of banks that is required to achieve the maximum data throughput of the main memory system?

30ns/5ns = 6

Row-Buffer Hits. The following timing diagram shows the operation of the single DRAM channel and the single DRAM bank for four back-to-back reads that hit in the row-buffer. It is important to note that row-buffer hits to the same DRAM bank are pipelined: while each READ keeps the DRAM bank busy for 10ns, up to at most **half** of this latency (5ns) can be overlapped with another read that hits in the row-buffer.



(c) Given a long sequence of back-to-back reads that always hits in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

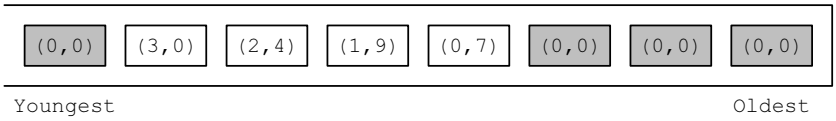
64B/5ns = 64GB/5s = 12.8GB/s

(d) When the maximum data throughput is achieved for a main memory system that has a single DRAM channel and a single DRAM bank, what is the bottleneck that prevents the data through- put from becoming even larger? **Circle** all that apply.

BANK **COMMAND BUS** **ADDRESS BUS** **DATA BUS**

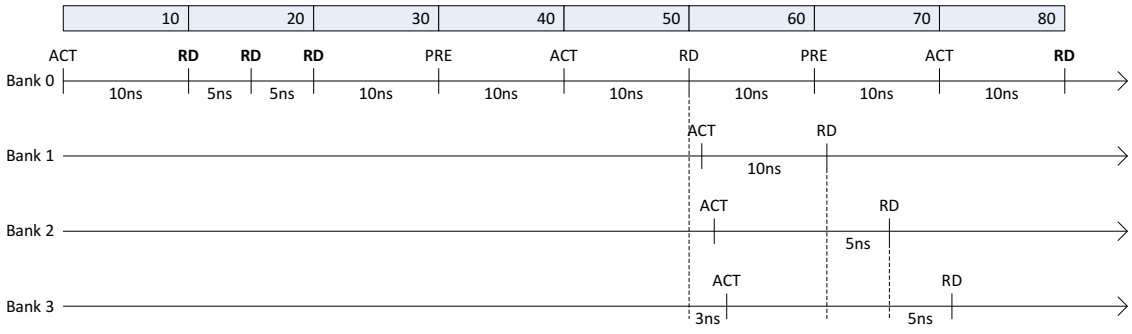
Memory Scheduling Policies. The diagram below shows the memory controllers *request queue* at time 0. The shaded rectangles are read requests generated by thread *T0*, whereas the unshaded rectangles are read requests generated by thread *T1*. Within each rectangle, there is a pair of numbers that denotes the requests (*BankAddress, RowAddress*). Assume that the memory system has a **single** DRAM channel and four DRAM banks. Further assume the following.

- All the row-buffers are **closed** at time 0.
- Both threads start to stall at time 0 because of memory.
- A thread continues to stall until it receives the data for all of its requests.
- Neither thread generates more requests.



We provide two sets of answers. The correct way to solve the problem is to model contention in the banks as well as in all of the buses (address/command/data). The answer that is given in the answer boxes is for the case you modeled contention in only the banks.

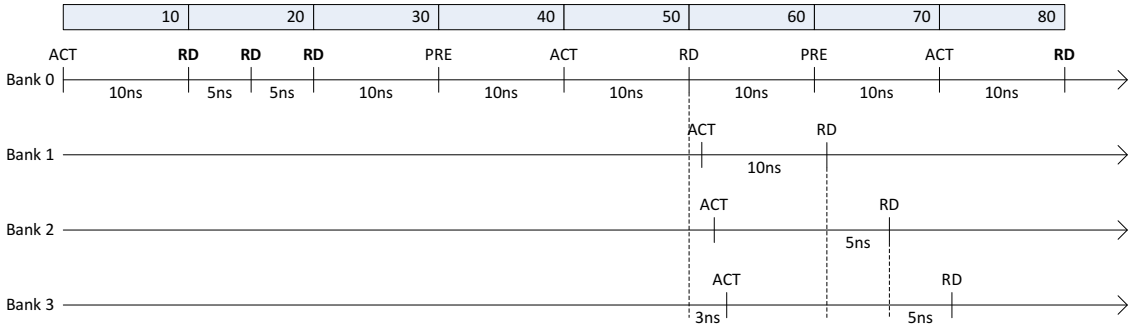
(f) For the $FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 10 + 10 + 10) + 10 + 10 + 10 + 10 + 5 = 95ns

T1: (10 + 5 + 5 + 10 + 10 + 10) + 1 + 10 + 5 + 5 + 10 + 5 = 86ns

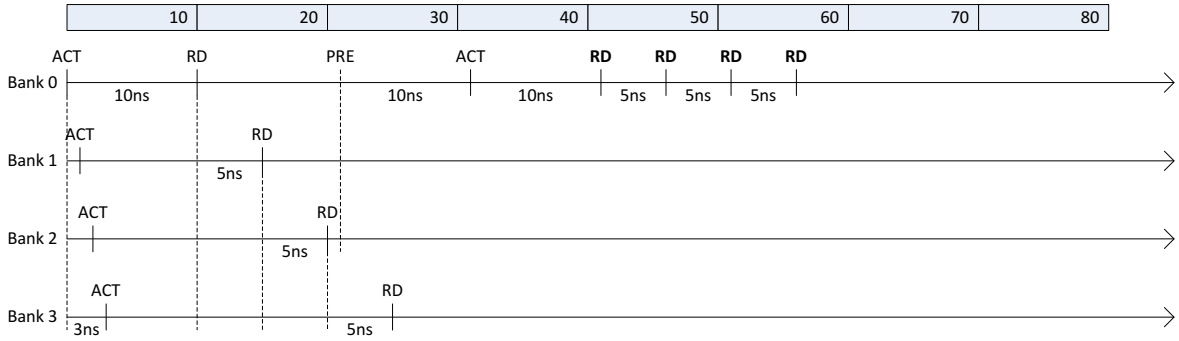
(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 10 + 10 + 10) + 10 + 10 + 10 + 10 + 5 = 95ns

T1: (10 + 5 + 5 + 10 + 10 + 10) + 1 + 10 + 5 + 5 + 10 + 5 = 86ns

(h) For the $PAR-BS$ scheduling policy, calculate the memory stall time of *T0* and *T1*. Assume that all eight requests are included in the same batch.



T0: (10 + 5 + 5) + 1 + 10 + 10 + 5 + 5 + 5 + 10 + 5 = 71ns

T1: (10 + 5 + 5) + 5 + 10 + 5 = 40ns

(f) For the $FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.

T0:

Bank 0 is the critical path for both threads.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 30ns + 5ns
= 95ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 5ns
= 65ns

(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.

T0:

Bank 0 is the critical path for both threads. First, we serve all four shaded requests since they are row-buffer hits. Lastly, we serve the unshaded request.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 5ns + 5ns + 5ns + 5ns
= 40ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 5ns + 30ns + 5ns
= 70ns

(h) For the $PAR-BS$ scheduling policy, calculate the memory stall time of *T0* and *T1*. Assume that all eight requests are included in the same batch.

T0:

First, we serve all four unshaded requests in parallel across the four banks. Then, we serve all four shaded requests in serial.

T0 = Closed + Conflict + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(PRE+ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 30ns + 5ns + 5ns + 5ns + 5ns
= 70ns

T1:

T1 = Closed + Data
= (ACT+RD)+DATA
= 20ns + 5ns
= 25ns

3 Memory Scheduling [50 points]

In class, we covered "parallelism-aware batch scheduling," which is a memory scheduling algorithm that aims to reduce interference between threads in a multi-core system.

(a) What benefit does request batching provide in this algorithm?

Request batching allows PAR-BS to avoid starvation as requests of older batches are always prioritized over requests of younger batches.

(b) How does the algorithm preserve intra-thread bank parallelism?

Threads are ranked based on the number of requests they have at all the banks. All banks service requests based on this ranking. Hence, requests from the same thread will likely be serviced in parallel in different banks, which preserves the threads bank-level parallelism.

(c) If thread ranking was formed in a "random manner" (i.e., threads were assigned a random rank), would each thread's parallelism be preserved? Why or why not? Explain.

It depends on the applications. Assume there are two banks and there is one application which has a lot of requests to one bank and no requests to the other. A random ranking can prioritize this application over others and thereby preventing the other applications from exploiting their bank-level parallelism.

4 Memory Request Scheduling [100 points]

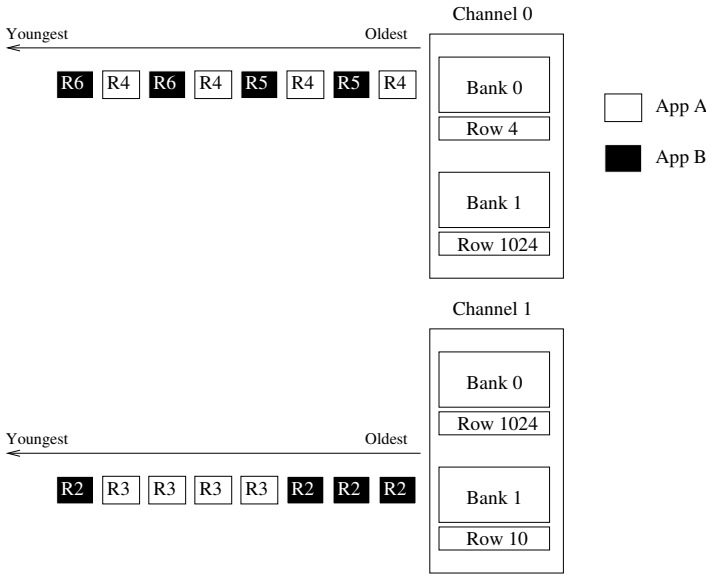
A machine has a DRAM main memory organized as 2 channels, 1 rank and 2 banks/channel. An open row policy is used, i.e., a row is retained in the row-buffer after an access until an access to another row is made. The following commands (defined as we discussed in class) can be issued to DRAM with the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

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

Assume the bus latency is 0 cycles.

(a) Two applications A and B are run on the machine. The following is a snapshot of the request buffers at time t0 . 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 4 is initially open in bank 0 of channel 0 and row 10 is initially open in bank 1 of channel 1.

Each 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 an FR-FCFS scheduling policy?

180 ns

At channel 0, bank 0, row 4 is open. As the scheduling policy is FR-FCFS, application A's four requests to row 4 (which are row-buffer hits) are serviced and take 15 ns (read/write time alone) each. Thus, application A's stall time at channel 0 is 15 4 = 60ns.

At channel 1, there are no requests to the open row (row 10). Hence the oldest request, which is application B's request to row 2, is serviced first (row-buffer conflict). The next three requests to row 2 are row-buffer hits and hence are serviced next. Following this, application A's requests to row 3 are serviced. The first request to row 3 is a row-buffer conflict, while the remaining three are row-buffer hits. Thus, application A's stall time at channel 0 is 2 row-buffer conflicts and 6 row-buffer hits = 452+ 156 = 180ns.

The application stalls for the longer of the stall times at either channel, which is **180 ns**.

What is the stall time of application B using an FR-FCFS scheduling policy?

180 ns

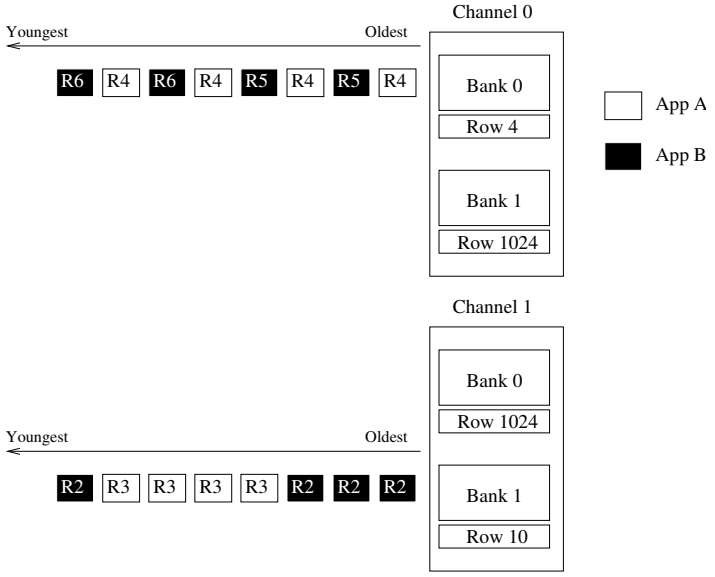
At channel 0, bank 0, application A's row-buffer hits are serviced first, incurring 45 ns. Following this, application B's requests to rows 5 and 6 are serviced. The first request to each row (rows 5 and 6) is a row-buffer conflict and the subsequent request to each row is a row-buffer hit. Hence, application B's stall time at channel 0 is 6 row-buffer hits and 2 row-buffer conflicts = 15 6 + 45 2 = 180ns.

At channel 1, bank 1, application B's first request to row 2 is serviced first because it is the oldest. The next three requests to row 2 are serviced next because row 2 is now open. Thus, application B's stall time at channel 1 is 1 row-buffer conflict and 3 row-buffer hits = 45 1 + 15 3 = 90ns.

The time for which application B stalls is the longer of the two stall times. Hence, the total stall time is **180 ns.**

(b) 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 request buffer state as in Part (a) (replicated above for your benefit):
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.

The PAR-BS scheduler we studied in lecture provided better system performance than the FR-FCFS scheduler. Is this true for X also? (i.e., does X provide better system performance than FR-FCFS?)

Circle one: **YES** *NO*

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.

(c) 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 154 = 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 1 + 15 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.**

(d) 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 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 1 + 15 3 = 90 ns.

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 2 + 15 2 = 120 ns.

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

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.

Is channel partitioning and FR-FCFS memory scheduling always strictly better or strictly worse than FR-FCFS memory scheduling alone? Explain.

Channel partitioning and FR-FCFS memory scheduling used together are not always better or worse than FR-FCFS memory scheduling alone.

If an application has requests outstanding to several banks across different channels, channel partitioning can reduce the amount of bandwidth that this single application gets and hence degrade its performance.

On the other hand, in cases where applications are severely interfering with each other (as we saw above), then using channel partitioning reduces this interference and improves performance.

5 Memory System [80 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 [11 : 10]
- Bank bits (bank bits are contiguous)
  Addr [13 : 12]
- Column bits (column bits are contiguous)
  Addr [9 : 3]
- Row bits (row bits are contiguous)
  Addr [31 : 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.