18-447 Intro to Computer Architecture, Spring 2012
Midterm Exam II

Instructor: Onur Mutlu
Teaching Assistants: Chris Fallin, Lavanya Subramanian, Abeer Agrawal
Date: April 11, 2012

Name: SOLUTIONS

Problem I (60 Points)	:	
Problem II (50 Points)	:	
Problem III (60 Points)	:	
Problem IV (30 Points)	:	
Problem V (60 Points)	:	
Problem VI (50 Points)	:	
Bonus Problem VII (40 Points)	:	
Legibility and Name (5 Points)	:	
Total (315 + 40 Points)	:	

Instructions:

1. This is a closed book exam. You are allowed to have two letter-sized cheat sheets.
2. No electronic devices may be used.
3. This exam lasts 1 hour 50 minutes.
4. Clearly indicate your final answer.
5. Please show your work when needed.
6. Please write your initials on every odd page.
7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

- **Be cognizant of time.** Do not spend too much time on one question.
- **Be concise.** You will be penalized for verbosity.
- **Show work when needed.** You will receive partial credit at our discretion.
- **Write legibly.** Show your final answer.

I. Potpourri (60 Points)

1) Dataflow and Vector Computers (5 Points)

Dataflow computers exploit irregular / instruction-level parallelism; vector computers exploit regular / data-level parallelism.

2) Processing Paradigms (5 Points)

A/an array processor executes the same instruction with different data at the same time; a/an vector processor executes the same instructions with different data at different times.

3) True Multiporting (5 Points)

In class, we discussed that adding a true second port to a memory cell increases access latency. Explain why this is so, concisely.

Longer wires and higher capacitance (loading) on storage cells. Adding wordlines, bitlines and access transistors makes the memory array larger (giving longer wires) and adds capacitance (loading) to the memory cell storage element.

4) L1 Data Cache (5 Points)

A creative designer realizes one day that he can increase the L1 cache size of his design from 64KB to 2MB without affecting the number of cycles it takes to retrieve data from the cache and without reducing the processor frequency.

What can you definitively say about this design?

Either the critical path somewhere else in the processor design was already very long, or the clock frequency was already very low to begin with (because the cache design change will make the cache's critical path significantly longer).

5) Sectored Cache vs. Smaller Blocks (10 Points)

After mounds of coffee and a good number of high-level simulation cycles, you narrowed down the design choices for the L1 cache of the next-generation processor you are architecting to the following two:

Choice 1. A sectored 64KB cache with 64-byte blocks and 8-byte subblocks

Choice 2. A non-sectored 64KB cache with 8-byte blocks

Assume the associativity of the two caches are the same.

What is one definitive advantage of the sectored cache over the non-sectored one?

Smaller tag store size.

What is one definitive advantage of the non-sectored cache over the sectored one?

Potentially better hit-rate if program does not have spatial locality.

Which of the choices has the faster cache hit latency? Circle one:
Choice 1 Choice 2 **Not Enough Information**

Justify your choice, describing the tradeoffs involved if necessary.

The described sectored cache has fewer sets. Hence, the tag access time will be shorter, which may result in a shorter cache hit latency.

On the other hand, the described sectored cache must check more valid bits, and must mux out the appropriate data from a 64-byte-wide block, while the described non-sectored cache checks only one valid bit per tag and muxes data out of a smaller 8-byte block. Hence, the non-sectored cache may also have a shorter cache hit latency.

Without knowing exactly how long each of these components of the latency takes, we cannot know which choice of cache design has lower hit latency.

6) FastCaches, Inc. (10 Points)

Suppose a creative colleague of yours at FastCaches, Inc. proposed the following idea. Instead of directly using a number of bits to index into the cache, take those bits and form the index using a hash function implemented in hardware that randomizes the index.

What type of cache misses can this idea potentially reduce?

Conflict misses.

Explain briefly why.

The hash function randomizes the mapping of each cache block to a set so that set conflicts (e.g. due to unlucky data alignment or aliasing) are less likely.

What is a disadvantage of the idea other than the additional hardware cost of the hash function?

Longer latency to access the cache (due to the hash function).

7) Dataflow versus Out-of-Order Execution (10 Points)

In class, we covered several dataflow machines that implemented dataflow execution at the ISA level. These machines included a structure/unit called the "matching store." What is the function of the matching store (in less than 10 words)?

To determine if all operands of an instruction are ready.

What structure accomplishes a similar function in an out-of-order processor?

The reservation stations.

When does the fetch of an instruction happen in a dataflow processor?

When its operands are ready.

When does the fetch of an instruction happen in an out-of-order execution processor?

When the control flow reaches the instruction.

8) Page Table Bits (10 Points)

What is the purpose of the "reference" or "accessed" bit in a page table entry?

To aid page replacement.

Describe what you would do if you did not have a reference bit in the PTE. Justify your reasoning and/or design choice.

Pick a random page to replace when a page fault occurs.

What is the purpose of the dirty or modified bit in a page table entry?

To enable writeback of only dirty pages (rather than all pages) to disk.

Describe what you would do if you did not have a modified bit in the PTE. Justify your reasoning and/or design choice.

Write back all pages to disk. Alternative answer: the OS could map all pages read-only by default. On a page-fault due to a write, if the program has permission to change the page, the operating system remaps the page as read-write and also knows that the page has become dirty.

II. Vector Processing (50 Points)

You are studying a program that runs on a vector computer with the following latencies for various instructions:

- VLD and VST: 50 cycles for each vector element; fully interleaved and pipelined.
- VADD: 4 cycles for each vector element (fully pipelined).
- VMUL: 16 cycles for each vector element (fully pipelined).
- VDIV: 32 cycles for each vector element (fully pipelined).
- VRSHF (right shift): 1 cycle for each vector element (fully pipelined).

Assume that:

- The machine has an in-order pipeline.
- The machine supports chaining between vector functional units.
- In order to support 1-cycle memory access after the first element in a vector, the machine interleaves vector elements across memory banks. All vectors are stored in memory with the first element mapped to bank 0, the second element mapped to bank 1, etc.
- Each memory bank has an 8KB row buffer.
- Vector elements are 64 bits in size.
- Each memory bank has two ports (so that two loads/stores can be active simultaneously), and there are two load/store functional units available.

(a) What is the minimum power-of-two number of banks required in order for memory accesses to never stall? (Assume a vector stride of 1.)

64 banks, because memory latency is 50 cycles and the next power of two is 64.

There is another solution if one interprets "never stall" to mean that a single load will never stall rather than the memory accesses in the program below: in that case, 32 banks suffices since each bank has two ports. For those who answered this way on the test, we gave full credit.

(b) The machine (with as many banks as you found in part (a)) executes the following program (assume that the vector stride is set to 1):

```
VLD V1 <- A
VLD V2 <- B
VADD V3 <- V1, V2
VMUL V4 <- V3, V1
VRSHF V5 <- V4, 2
```

It takes 111 cycles to execute this program. What is the vector length?

40 elements

```
VLD     |----50------|---(VLEN-1)----|
VLD     |1|----50------|
VADD                  |-4-|
VMUL                      |-16-|
VRSHF                         |1|-----(VLEN-1)-----|
50+1+4+16+1 + (VLEN-1) = 71 + VLEN = 111 -> VLEN = 40
```

If the machine did not support chaining (but could still pipeline independent operations), how many cycles would be required to execute the same program? Show your work.

228 cycles

```
VLD  |-----50-----|---(VLEN-1)---|
VLD  |1|-----50-----|---(VLEN-1)---|
VADD                              |-4-|--(VLEN-1)---|
VMUL                                                  |-16-|--(VLEN-1)---|
VRSHF                                                                       |1|--(VLEN-1)--|
50 + 1 + 4 + 16 + 1 + 4*(VLEN-1) = 68 + 4*VLEN = 228
```

(c) The architect of this machine decides that she needs to cut costs in the machine's memory system. She reduces the number of banks by a factor of 2 from the number of banks you found in part (a) above. Because loads and stores might stall due to bank contention, an arbiter is added to each bank so that pending loads from the oldest instruction are serviced first. How many cycles does the program take to execute on the machine with this reduced-cost memory system (but with chaining)?

129 cycles

```
VLD [0]  |----50----|   bank 0 (takes port 0)
    ...
    [31] |--31--|----50----| bank 31
    [32]            |---50---| bank 0 (takes port 0)
    ...
    [39]                     |--7--|  bank 7
VLD [0]  |1|----50----|   bank 0 (takes port 1)
    ...
    [31] |1|--31--|----50----| bank 31
    [32]              |---50----| bank 0 (takes port 1)
    ...
    [39]                        |--7--| bank 7
VADD                                  |--4--| (tracking last elements)
VMUL                                        |--16--|
VRSHF                                              |1|
(B[39]: 1 + 50 + 50 + 7) + 4 + 16 + 1 = 129 cyc
```

Now, the architect reduces cost further by reducing the number of memory banks (to a lower power of 2). The program executes in 279 cycles. How many banks are in the system?

8 banks

```
VLD  [0]    |---50---|
     ...
     [8]             |---50---|
     ...
     [16]                     |--50--|
     ...
     [24]                            |--50--|
     ...
     [32]                                   |--50--|
     ...
```

```
     [39]                                                |--7--|
VLD  [39]                                                    |1|
VADD                                                           |--4--|
VMUL                                                                 |--16--|
VRSHF                                                                       |1|
5*50 + 7 + 1 + 4 + 16 + 1 = 279 cyc
```

(d) Another architect is now designing the second generation of this vector computer. He wants to build a multicore machine in which 4 vector processors share the same memory system. He scales up the number of banks by 4 in order to match the memory system bandwidth to the new demand. However, when he simulates this new machine design with a separate vector program running on every core, he finds that the average execution time is longer than if each individual program ran on the original single-core system with 1/4 the banks. Why could this be (in less than 20 words)? Provide concrete reason(s).

Inter-application memory interference which leads to loss of row-buffer locality and bank-level parallelism. This occurs because all applications interleave their vectors across all banks in the system.

What change could this architect make to the system in order to alleviate this problem (in less than 20 words), while *only* changing the shared memory hierarchy?

Partition applications across channels, or use application-aware memory scheduling.

III. DRAM Refresh (60 Points)

A memory system has four channels, and each channel has two ranks of DRAM chips. Each memory channel is controlled by a separate memory controller. Each rank of DRAM contains eight banks. A bank contains 32K rows. Each row in one bank is 8KB. The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms. Every DRAM row refresh is initiated by a command from the memory controller which occupies the command bus on the associated memory channel for 5 ns and the associated bank for 40 ns. Let us consider a 1.024 second span of time.

We define *utilization* (of a resource such as a bus or a memory bank) as the fraction of total time for which a resource is occupied by a refresh command.

For each calculation in this section, you may leave your answer in *simplified* form in terms of powers of 2 and powers of 10.

(a) How many refreshes are performed by the memory controllers during the 1.024 second period in total across all four memory channels?

2^{25} or $32M$ refreshes.

There are 2^{23} refreshes per channel, because there are 16 refreshes/row in 1.024 seconds, and $2^{15} * 8 * 2 = 2^{21}$ rows/channel. With 4 channels, this yields $2^{25} = 32M$ refreshes. (Also accepted 32768000 refreshes if the assumption was made that $32K = 32000$ rather than 2^{15}.)

(b) What command bus utilization, across all memory channels, is directly caused by DRAM refreshes?

4.096% utilization.

Each refresh contributes $5ns$ on one command bus. Overall there are $2^{25} * 5$ command bus-nanoseconds used, but there are four command buses, hence $2^{25} * 5ns/4$ of time for which each command bus is occupied. Over $1.024s$ total, this gives $2^{25} * 5ns/(4 * 1.024s) = 4.096\%$.

(c) What data bus utilization, across all memory channels, is directly caused by DRAM refreshes?

0: refreshes use only the command bus. They do not transfer data.

(d) What bank utilization (on average across all banks) is directly caused by DRAM refreshes?

2.048% utilization.

In each bank, each of 2^{15} rows is refreshed 16 times, occupying the bank for a total of $2^{15} * 16 * 40ns$ of time, over $1.024s$ of total time, hence 2.048% utilization.

(e) The system designer wishes to reduce the overhead of DRAM refreshes in order to improve system performance and reduce the energy spent in DRAM. A key observation is that not all rows in the DRAM chips need to be refreshed every 64 ms. In fact, rows need to be refreshed only at the following intervals in this particular system:

Required Refresh Rate	Number of Rows (overall)
64 ms	2^5
128 ms	2^9
256 ms	all other rows

Given this distribution, if all rows are refreshed only as frequently as required to maintain their data, how many refreshes are performed by the memory controllers during the 1.024 second period in total across all four memory channels?

8391040 refreshes.

There are $(2^{21} - 2^9 - 2^5)$ rows that refresh 4 times (at 256ms intervals), 2^9 rows that refresh 8 times (at 128ms intervals), and 2^5 rows that refresh 16 times (at 64ms intervals). Hence, $4 * (2^{21} - 2^9 - 2^5) + 8 * (2^9) + 16 * (2^5) = 8391040$.
We also accepted answers based on the assumption that the row counts given were either per channel or per bank, as long as the assumption was clearly stated.

What command bus utilization (as a fraction of total time) is caused by DRAM refreshes in this case?

1.0243% utilization.

Each refresh occupies $5ns$ on one command bus, and we must take total command bus time occupied over the period of time on all command buses, hence $5ns * 8391040/(4 * 1.024) = 1.0243\%$.

(f) What DRAM data bus utilization is caused by DRAM refreshes in this case?

0

(g) What bank utilization (on average across all banks) is caused by DRAM refreshes in this case?

0.5121%.

We can take the total bank occupancy time (computed as $40ns$ multiplied by total number of refreshes) over total number of banks times the total period: $40ns * 8391040/(64 * 1.024) = 0.5121\%$.

(h) The system designer wants to achieve this reduction in refresh overhead by refreshing rows less frequently when they need less frequent refreshes. In order to implement this improvement, the system needs to track every row's required refresh rate. What is the minimum number of bits of storage required to track this information?

4 Mbit (2 bits per row). Also accepted simply "2 bits per row."

(i) Assume that the system designer implements an approximate mechanism to reduce refresh rate using Bloom filters, as we discussed in class. One Bloom filter is used to represent the set of all rows which require a 64 ms refresh rate, and another Bloom filter is used to track rows which require a 128 ms refresh rate. The system designer modifies the memory controller's refresh logic so that on every potential refresh of a row (every 64 ms), it probes both Bloom filters. If either of the Bloom filter probes results in a "hit" for the row address, and if the row has not been refreshed in the most recent length of time for the refresh rate associated with that Bloom filter, then the row is refreshed. (If a row address hits in both Bloom filters, the more frequent refresh rate wins.) Any row that does not hit in either Bloom filter is refreshed at the default rate of once per 256 ms.

The false-positive rates for the two Bloom filters are as follows:

Refresh Rate Bin	False Positive Rate
64 ms	2^{-20}
128 ms	2^{-8}

The distribution of required row refresh rates specified in part (e) still applies.

How many refreshes are performed by the memory controllers during the 1.024 second period in total across all four memory channels?

8423823 refreshes.

First find the number of rows refreshed at each rate. At 64 ms, we have 2^5 rows which actually require this rate, plus 2^{-20} out of the $2^{21} - 2^5$ other rows which will be false-positives. At 128 ms, we have 2^9 rows which actually require this rate, plus 2^{-8} out of the $2^{21} - 2^9 - 2^5$ other rows. At 256 ms we have the rest of the rows. During the 1.024s, each bin is refreshed respectively 16, 8, and 4 times. Thus we have $16 * (2^5 + (2^{21} - 2^5) * 2^{-20}) + 8 * (2^9 + (2^{21} - 2^9 - 2^5) * 2^{-8}) + 4 * (2^{21} - 2^9 - 2^5 - (2^{21} - 2^9) * 2^{-20} - (2^{21} - 2^9 - 2^5) * 2^{-8}) = 8423823$ total refreshes.

What command bus utilization results from this refresh scheme?

1.028% utilization.

$5ns * 8423823/(4 * 1.024s) = 1.028\%$.

What data bus utilization results from this refresh scheme?

0

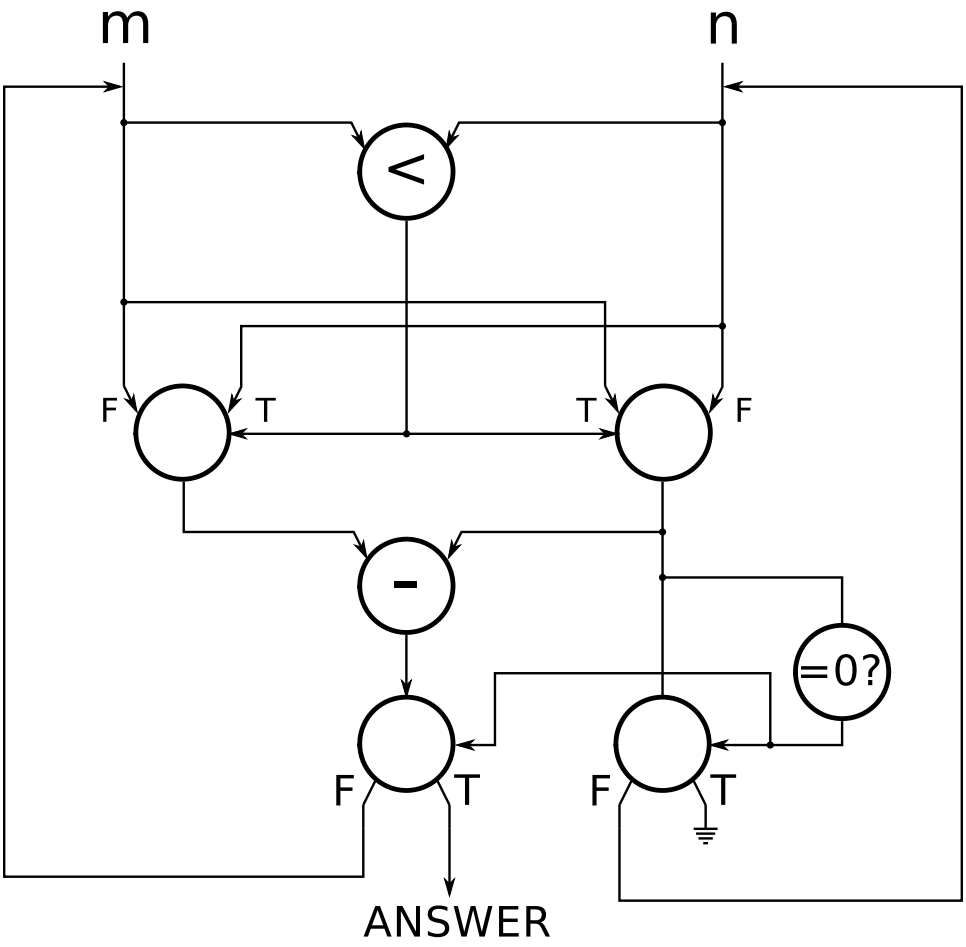
What bank utilization (on average across all banks) results from this refresh scheme?

0.5141% utilization.

$40ns * 8423823/(64 * 1.024s) = 0.5141\%$.

IV. Dataflow (30 Points)

What does the following dataflow program do? Specify clearly in less than 10 words (one could specify this function in three words).



Greatest Common Divisor (this is Euclid's Algorithm)

V. Memory Request Scheduling (60 Points)

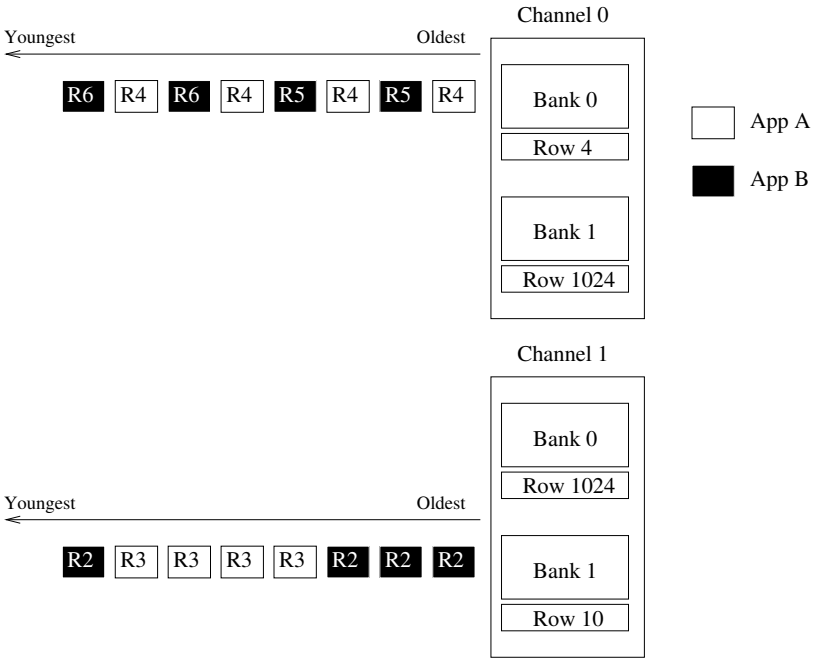
A machine has a DRAM main memory organized as 2 channels, 1 rank and 2 banks/channel. An open row policy is used, i.e., a row is retained in the row-buffer after an access until an access to another row is made. The following commands (defined as we discussed in class) can be issued to DRAM with the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

Command	Latency
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 t_0. 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 $15 \times 4 = 60ns$.

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 $= 45 \times 2 + 15 \times 6 = 180ns$.

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 $= 15 \times 6 + 45 \times 2 = 180ns$.

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 $= 45 \times 1 + 15 \times 3 = 90ns$.

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

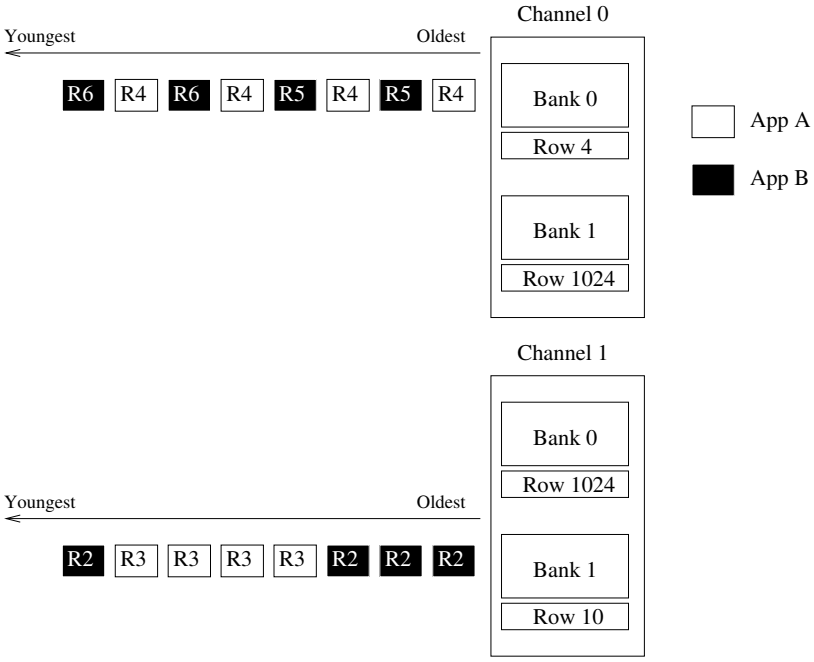
(b) We studied Parallelism Aware Batch Scheduling (PAR-BS) in class. We will use a PAR-BS-like scheduler that we will call X.

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?
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 application's 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**.

(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 $15 \times 4 = 60ns$.

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 $45 \times 1 + 15 \times 3 = 90$ 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 $45 \times 2 + 15 \times 2 = 120$ ns.

At bank 1, application B has one row-buffer conflict and three row-buffer hits. Stall time at bank 1 is $45 \times 1 + 15 \times 3 = 90$ 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 B's 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 intefering with each other (as we saw above), then using channel partitioning reduces this interference and improves performance.

VI. Caches and Virtual Memory (50 Points)

A four-way set-associative writeback cache has a $2^{11} \cdot 89$-bit tag store. The cache uses a custom replacement policy that requires 9 bits per set. The cache block size is 64 bytes. The cache is virtually-indexed and physically-tagged. Data from a given physical address can be present in up to eight different sets in the cache. The system uses hierarchical page tables with two levels. Each level of the page table contains 1024 entries. A page table may be larger or smaller than one page. The TLB contains 64 entries.

(a) How many bits of the virtual address are used to choose a set in the cache?

11 bits

(b) What is the size of the cache data store?

512 KB (2^{19} bytes)

(c) How many bits in the Physical Frame Number must overlap with the set index bits in the virtual address?

3 bits

(d) On the following blank figure representing a virtual address, draw in bitfields and label bit positions for "cache block offset" and "set number." Be complete, showing the beginning and ending bits of each field.

Virtual Address:

	set index (11 bits)	block offset (6 bits)

(e) On the following blank figure representing a physical address, draw in bitfields and label bit positions for "physical frame number" and "page offset." Be complete, showing the beginning and ending bits of each field.

Physical Address:

PFN (18 bits)	page offset (14 bits)

(f) What is the page size?

16 KB (2^{14} bytes)

(g) What is the size of the virtual address space?

16 GB (2^{34} bytes)

(h) What is the size of the physical address space?

4 GB (2^{32} bytes)

VII. BONUS: Memory Hierarchy (40 Points)

Assume you developed the next greatest memory technology, MagicRAM. A MagicRAM cell is non-volatile. The access latency of a MagicRAM cell is 2 times that of an SRAM cell but the same as that of a DRAM cell. The read/write energy of MagicRAM is similar to the read/write energy of DRAM. The cost of MagicRAM is similar to that of DRAM. MagicRAM has higher density than DRAM. MagicRAM has one shortcoming, however: a MagicRAM cell stops functioning after 2000 writes are performed to the cell.

(a) Is there an advantage of MagicRAM over DRAM? Circle one: **YES** **NO**

Explain.

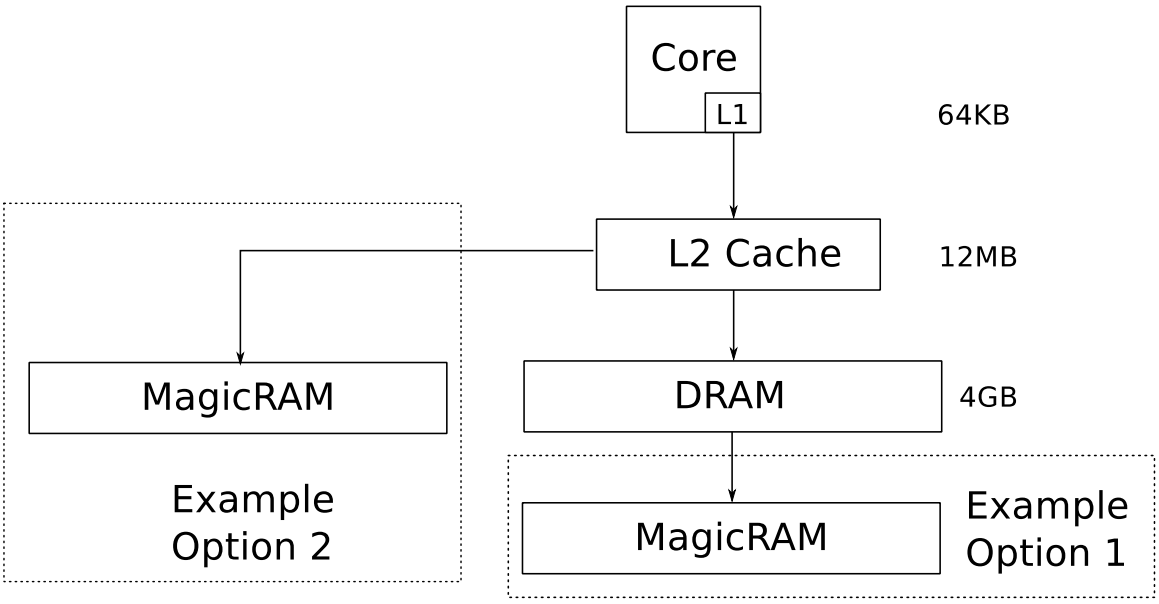
MagicRAM does not need refreshes, since it is nonvolatile. This can reduce dynamic power, bus utilization, and bank contention. MagicRAM is also nonvolatile, which can enable new uses or programming models.

(b) Is there an advantage of MagicRAM over SRAM? Circle one: **YES** **NO**

Explain.

MagicRAM has higher density and lower cost than SRAM.

(c) Assume you have a system that has a 64KB L1 cache made of SRAM, a 12MB L2 cache made of SRAM, and 4GB main memory made of DRAM.



Assume you have complete design freedom and add structures to overcome the shortcoming of MagicRAM. You will be able to propose a way to reduce/overcome the shortcoming of MagicRAM (note that you can design the hierarchy in any way you like, but cannot change MagicRAM itself).

Does it makes sense to add MagicRAM anywhere in this memory hierarchy given that you can potentially reduce its shortcoming? Circle one: **YES** **NO**

If so, where would you place MagicRAM? **Depict in the figure above clearly** and describe why you made this choice.

Many answers are possible. One option: Place MagicRAM below DRAM in the hierarchy, and use DRAM as a cache to MagicRAM. This way, DRAM performs write coalescing so that MagicRAM does not wear out as quickly. Another option could be to place MagicRAM "next to" DRAM (on the same or another channel), and use MagicRAM explicitly for read-only data.

If not, why not? Explain below clearly and methodically.

(d) Propose a way to reduce/overcome the shortcoming of MagicRAM by modifying the given memory hierarchy. Be clear in your explanations and illustrate with drawings to aid understanding.

Explanation:

A write-combining/coalescing buffer could be added. The memory hierarchy could also perform wear-leveling, or it could predict which data is less likely to be modified and place only that data in MagicRAM.

Figure(s):