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, Dec 10, 2020

## 1 Main Memory Organization and Interleaving

Consider the following piece of code:

```
for(i = 0; i < 8; ++i){
    for(j = 0; j < 8; ++j){
        sum = sum + A[i][j];
    }
}
```

The figure below shows an 8-way interleaved, byte-addressable memory. The total size of the memory is 4 KB . The elements of the 2-dimensional array, A, are 4-bytes in length and are stored in the memory in column-major order (i.e., columns of A are stored in consecutive memory locations) as shown. The width of the bus is 32 bits, and each memory access takes 10 cycles.

## Bank 7

## Bank 1

Bank 0


A more detailed picture of the memory chips in Bank 0 of Rank 0 is shown below.

| A[0][0] <31:24> | 3 | A $[0][0]<23: 16$ > | 2 | A[0][0] < 15:8> | 1 | $\mathrm{A}[0][0]<7: 0>$ | $-\begin{gathered} 0 \\ 32 \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| A[0][1] <31:24> |  | A $[0][1]<23: 16>$ |  | A[0][1] <15:8> |  | A[0][1]<7:0> |  |
| A[0][2] <31:24> |  | $\mathrm{A}[0][2]<23: 16$ > |  | $\mathrm{A}[0][2]<15: 8>$ |  | $\mathrm{A}[0][2]<7: 0>$ | 64 |
| . |  | . |  | . |  |  |  |
| . |  |  |  |  |  |  |  |
| A[0][7] <31:24> | 227 | A[0][7] <23:16> | 226 | A[0][7] < $15: 8$ > | 225 | $\mathrm{A}[0][7]<7: 0>$ | 224 |

(a) Since the address space of the memory is $4 \mathrm{~KB}, 12$ bits are needed to uniquely identify each memory location, i.e., Addr[11:0]. Specify which bits of the address will be used for:

- Byte on bus

Addr [ 1 : 0 ]

- Interleave/Bank bits

Addr [ 4 : 2 ]

- Chip address

Addr [ 7 : 5]

- Rank bits

```
Addr [ 11 : 8 ]
```

(b) How many cycles are spent accessing memory during the execution of the above code? Compare this with the number of memory access cycles it would take if the memory were not interleaved (i.e., a single 4 -byte wide array).

## Assumptions: Requests are scheduled in order one after another and any number of requests can be outstanding at a time (limited only by bank conflicts).

As consecutive array elements are stored in the same bank, they have to be accessed serially. However, the memory access in the last iteration of the inner for loop ( $\mathrm{A}[\mathrm{x}][7]$ ) and the memory access in the first iteration of the next time through the inner for loop $(\mathrm{A}[\mathrm{x}+1][0])$ go to different banks and can be accessed in parallel.
Therefore, number of cycles $=(10 \times 8 \times 8)-(9 \times 7)=577$ cycles.
(c) Can any change be made to the current interleaving scheme to optimize the number of cycles spent accessing memory? If yes, which bits of the address will be used to specify the byte on bus, interleaving, etc. (use the same format as in part a)? With the new interleaving scheme, how many cycles are spent accessing memory? Remember that the elements of A will still be stored in column-major order.

Elements along a column are accessed in consecutive cycles. So, if they were mapped to different banks, accesses to them can be interleaved.
This can be done by using the following address mapping:

- Byte on bus

Addr [ 1 : 0]

- Interleave/Bank bits

Addr [ 7 : 5]

- Chip address

Addr [ 4 : 2 ]

- Rank bits

Addr [ 11 : 8 ]
The first iteration of the outer loop starts at time 0 . The first access is issued at cycle 0 and ends at cycle 10. Accesses to consecutive banks are issued every cycle after that until cycle 7 and these accesses complete by cycle 17. The first access of the second iteration of the outer loop can start only in cycle 10 , as bank 0 is occupied until cycle 10 with the first access of the previous outer loop iteration. The second iteration's accesses complete at cycle 27. Extending this, the eighth iteration's first access starts at cycle 70 and completes by cycle 87 .
Therefore, number of cycles $=87$ cycles.
(d) Using the original interleaving scheme, what small changes can be made to the piece of code to optimize the number of cycles spent accessing memory? How many cycles are spent accessing memory using the modified code?

The inner and outer loop can be reordered as follows:

```
for(j = 0; j < 8; ++j){
    for(i = 0; i < 8; ++i){
        sum = sum + A[i][j];
    }
}
```

Thus, elements along a row, which are stored in consecutive banks (using the original interleaving scheme), will be accessed in consecutive cycles. The effect this has on the number of cycles is exactly the same as part (c).
Therefore, number of cycles $=87$ cycles.

## 2 Main Memory Potpourri

A machine has a 4 KB DRAM main memory system. Each row is refreshed every 64 ms .
Note: This question is open ended. We provide one possible set of solutions. There could be other possible right solutions.
(a) The machine's designer runs two applications A and B (each run alone) on the machine. Although applications A and B have a similar number of memory requests, application A spends a surprisingly larger fraction of cycles stalling for memory than application $B$ does. What might be the reasons for this?

A large number of application A's memory requests are row-buffer conflicts, whereas a large number of application B's memory requests are row-buffer hits.
Hence, application A's requests take longer to service and it spends more time stalling for memory.
(b) Application A also consumes a much larger amount of memory energy than application B does. What might be the reasons for this?

Row-buffer conflicts also consume more energy, as they require a precharge, activate and a read/write, whereas row-buffer hits only require a read/write.
Hence, application A consumes more memory energy.
(c) When applications A and B are run together on the machine, application A's performance degrades significantly, while application B's performance doesn't degrade as much. Why might this happen?

When the applications are run together, they interfere with each other. Hence, both applications' performance degrades when they run together. However, if a memory scheduler that favor row-buffer hits over row-buffer conflicts (like FR-FCFS) is used, it would favor application B's requests over application A's requests.
Therefore, application A's performance degrades more.
(d) The designer decides to use a smarter policy to refresh the memory. A row is refreshed only if it has not been accessed in the past 64 ms . Do you think this is a good idea? Why or why not?

This can reduce refresh energy significantly if a large fraction of the rows in memory contain data and are accessed (within the 64 ms refresh window), as these rows do not have to be refreshed explicitly. However, if only a small number of rows contain data and only these rows are accessed, this policy will not provide much reduction in refresh energy as a large fraction of rows are still refreshed at the 64 ms rate.
(e) The refresh energy consumption when application B is run, drops significantly when this new refresh policy is applied, while the refresh energy when application A is run reduces only slightly. Is this possible? Why or why not?

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

## 3 Banks

A processor's memory hierarchy consists of a small SRAM L1-cache and a large DRAM main memory, both of which are banked. The processor has a 24-bit physical address space and does not support virtual memory (i.e., all addresses are physical addresses). An application has just started running on this processor. The following figure shows the timeline of memory references made by that application and how they are served in the L1-cache or main memory.


For example, the first memory reference made by the application is to byte-address 0 xffbc 67 (assume that all references are byte-sized reads to byte-addresses). However, the memory reference misses in the L1-cache (assume that the L1-cache is intially empty). Immediately afterwards, the application accesses main memory, where it experiences a row-buffer miss (initially, assume that all banks in main memory each have a row opened that will never be accessed by the application). Eventually, the cache block (and only that cache block) that contains the byte-address $0 x f f b c 67$ is fetched from memory into the cache.

Subsequent memory references may experience bank-conflicts in the L1-cache and/or main memory, if there is a previous reference still being served at that particular bank. Bank-conflicts are denoted as hatched shapes in the timeline.
The following table shows the address of the memory references made by the application in both hexadecimal and binary representations.
(a) Memory address table

| Hexadecimal | Binary |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ffbc67 | 1111 | 1111 | 1011 | 1100 | 0110 | 0111 |
| ffbda7 | 1111 | 1111 | 1011 | 1101 | 1010 | 0111 |
| ffbdd7 | 1111 | 1111 | 1011 | 1101 | 1101 | 0111 |
| ff5e28 | 1111 | 1111 | 0101 | 1110 | 0010 | 1000 |
| ff5a28 | 1111 | 1111 | 0101 | 1010 | 0010 | 1000 |
| ff4e28 | 1111 | 1111 | 0100 | 1110 | 0010 | 1000 |
| ff 4 c 04 | 1111 | 1111 | 0100 | 1100 | 0000 | 0100 |
| ffbc6f | 1111 | 1111 | 1011 | 1100 | 0110 | 1111 |
| ffbc70 | 1111 | 1111 | 1011 | 1100 | 0111 | 0000 |

From the above timelines and the table, your job is to answer questions about the processor's cache and main memory organization. Here are some assumptions to help you along the way.

- Assumptions about the L1-cache
- Block size: ? (Power of two, greater than two)
- Associativity: ? (Power of two, greater than two)
- Total data-store size: ? (Power of two, greater than two)
- Number of banks: ? (Power of two, greater than two)
- Initially empty
- Assumptions about main memory
- Number of channels: 1
- Number of ranks per channel: 1
- Number of banks per rank: ? (Power of two, greater than two)
- Number of rows per bank: ? (Power of two, greater than two)
- Number of cache-blocks per row: ? (Power of two, greater than two)
- Contains the entire working set of the application
- Initially, all banks have their $0^{t h}$ row open, which is never accessed by the application


### 3.1 First, let's cover the basics

(a) Caches and main memory are sub-divided into multiple banks in order to allow parallel access. What is an alternative way of allowing parallel access?

## Multiporting, duplicating

(b) A cache that allows multiple cache misses to be outstanding to main memory at the same time is called what? (Two words or less. Hint: It's an adjective.)

Non-blocking (or lockup-free)
(c) While cache misses are outstanding to main memory, what is the structure that keeps bookkeeping information about the outstanding cache misses? This structure often augments the cache.

Miss status handling registers (MSHRs)
(d) Which is larger, an SRAM cell or a DRAM cell?

## SRAM cell

(e) What is the number of transistors and/or capacitors needed to implement each cell, including access transistor(s)?

SRAM: 6T
DRAM: 1T-1C

### 3.2 Cache and memory organization

NOTE: For the following questions, assume that all offsets and indexes come from contiguous address bits.
(a) What is the L1-cache's block size in bytes? Which bit positions in the 24-bit physical address correspond to the cache block offset? (The least-significant bit in the physical address has a bit position of 0 .)

Block size: 16 bytes
Bit positions of block offset: 0-3
(b) How many banks are there in the L1-cache? Which bit positions in the 24 -bit physical address correspond to the L1-cache bank index? (The least-significant bit in the physical address has a bit position of 0 .)

Number of L1-cache banks: 4
Bit positions of L1-cache bank index: 4-5
(c) How many banks are there in main memory? Which bit positions in the 24 -bit physical address correspond to the main memory bank index? (The least-significant bit in the physical address has a bit position of 0 .)

Number of main memory banks: 8
Bit positions of main memory bank index: 10-12
(d) What kind of interleaving is used to map physical addresses to main memory?
$\square$
(e) To fully support a 24 -bit physical address space, how many rows must each main memory bank have? Which bit positions in the 24 -bit physical address correspond to the main memory row index? (The least-significant bit in the physical address has a bit position of 0 .)

Number of rows per main memory bank: 2048
Bit positions of row index: 13-23
(f) Each cache block within a row is called a column. How many columns are there in a single row? Which bit positions in the 24-bit physical address correspond to the main memory column index? (The least-significant bit in the physical address has a bit position of 0 .)

Number of columns per row: 64
Bit positions of column index: 4-9

## 4 Memory Hierarchy

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 other than its density? (Please do not repeat what is stated in the above paragraph.) Explain.

Yes. 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? Explain.

Yes. MagicRAM has higher density and lower cost than SRAM.
(c) Assume you have a system that has a 64 KB L1 cache made of SRAM, a 12 MB L2 cache made of SRAM, and 4 GB 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).
(i) Does it makes sense to add MagicRAM anywhere in this memory hierarchy given that you can potentially reduce its shortcoming?

Yes.
(ii) If so, where would you place MagicRAM? Describe it in terms of the figure above and describe why you made this choice.

If not, why not? Explain below clearly and methodically

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.

(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.

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):


## 5 Prefetching

Suppose you have designed the next fancy hardware prefetcher for your system. You analyze its behavior and find the following:
(a) The prefetcher successfully prefetches block A into the cache before it is required by a load instruction. The prefetched block evicts a never-to-be-used block from the cache, so it does not cause cache pollution. Furthermore, you find that the prefetch request does not waste bus bandwidth needed by some other request.
(b) The prefetcher successfully prefetches block B into the cache before it is required by a load instruction. The prefetched block evicts a never-to-be-used block from the cache, so it does not cause cache pollution. Furthermore, you find that the prefetch request does not waste bus bandwidth needed by some other request.

Upon further analysis, you find that the prefetching of block A actually reduced execution time of the program whereas prefetching of block B did not reduce execution time significantly. Describe why this could happen. Draw two execution timelines, one with and one without the prefetcher, to illustrate the concept.


## 6 DRAM Refresh

A memory system is composed of eight banks, and each bank contains 32 K rows. The row size is 8 KB .
Every DRAM row refresh is initiated by a command from the memory controller, and it refreshes a single row. Each refresh command keeps the command bus busy for 5 ns .

We define command bus utilization as a fraction of the total time during which the command bus is busy due to refresh.

The retention time of each row depends on the temperature ( T ). The rows have different retention times, as shown in the following Table 1

| Retention Time | Number of rows |
| :--- | :--- |
| $(128-T) \mathrm{ms}, 0^{\circ} \mathrm{C} \leq T \leq 128^{\circ} \mathrm{C}$ | $2^{8}$ rows |
| $2 *(128-T) \mathrm{ms}, 0^{\circ} \mathrm{C} \leq T \leq 128^{\circ} \mathrm{C}$ | $2^{16}$ rows |
| $4 *(128-T) \mathrm{ms}, 0^{\circ} \mathrm{C} \leq T \leq 128^{\circ} \mathrm{C}$ | all other rows |
| $8 *(128-T) \mathrm{ms}, 0^{\circ} \mathrm{C} \leq T \leq 128^{\circ} \mathrm{C}$ | $2^{8}$ rows |

Table 1: Retention time

### 6.1 Refresh Policy A

Assume that the memory controller implements a refresh policy where all rows are refreshed with a fixed refresh interval, which covers the worst-case retention time (Table 1).
(a) What is the maximum temperature at which the DRAM can operate reliably with a refresh interval of 32 ms ?
$\mathrm{T}=96^{\circ} \mathrm{C}$.
Explanation. Looking at the first column of Table 1, it is obvious that the DRAM rows with the least retention time (i.e., the first row of the table) can operate at temperatures up to $96^{\circ} \mathrm{C}(128-96)$ when refreshed at fixed 32 ms refresh period.
(b) What command bus utilization is directly caused by DRAM refreshes (with refresh interval of 32 ms )?
$4.096 \%$.
Explanation. The time that the command bus spends on refresh commands in 32 ms is $2^{18 *} 5 \mathrm{~ns}$, where $2^{18}$ is the number of rows ( 8 banks, 32 K rows per banks), and 5 ns is the time that the bus is busy for each refresh command.
Utilization $=\frac{2^{18} * 5 n s}{32 m s}=4.096 \%$

### 6.2 Refresh Policy B

Now assume that the memory controller implements a refresh policy where all rows are refreshed only as frequently as required to correctly maintain their data (Table 1).
(a) How many refreshes are performed by the memory controller during a 1.024 second period? (with $\mathrm{T}=64^{\circ} \mathrm{C}$ )

The number of refreshes in 1.024 seconds is $\mathbf{1 3 1 3 2 8 0}$.
Explanation. $2^{8}$ rows are refreshed every $64 \mathrm{~ms} \Longrightarrow(1024 / 64) * 2^{8}=2^{12}$ refresh commands in 1.024 seconds.
$2^{16}$ rows are refreshed every $128 \mathrm{~ms} \Longrightarrow(1024 / 128) * 2^{16}=2^{19}$ refresh commands in 1.024 seconds.
$2^{8}$ rows are refreshed every $512 \mathrm{~ms} \Longrightarrow(1024 / 512) * 2^{8}=2^{9}$ refresh commands in 1.024 seconds.

The remaining rows are $2^{18}-\left(2^{8}+2^{16}+2^{8}\right)=2^{18}-2^{16}-2^{9}$, so: $2^{18}-2^{16}-2^{9}$ rows are refreshed every $256 \mathrm{~ms} \Longrightarrow(1024 / 256) *\left(2^{18}-2^{16}-2^{9}\right)=2^{20}-2^{18}-2^{11}$ refresh commands in 1.024 second.

Total: $2^{12}+2^{19}+2^{9}+2^{20}-2^{18}-2^{11}=1313280$ refresh commands in 1.024 second.

### 6.3 Refresh Policy C

Assume that the memory controller implements an even smarter policy to refresh the memory. In this policy, the refresh interval is fixed, and it covers the worst-case retention time ( 64 ms ), as the refresh policy in part 6.1. However, as an optimization, a row is refreshed only if it has not been accessed during the past refresh interval. For maintaining correctness, if a cell reaches its maximum retention time without being refreshed, the memory controller issues a refresh command.
(a) Why does a row not need to be refreshed if it was accessed in the past refresh interval?

The charge of a DRAM cell is restored when it is accessed.
(b) A program accesses all the rows repeatedly in the DRAM. The following table shows the access interval of the rows, the number of rows accessed with the corresponding access interval, and the retention times of the rows that are accessed with the corresponding interval.

| Access interval | Number of rows | Retention times |
| :--- | :--- | :--- |
| 1 ms | $2^{16}$ rows | $64 \mathrm{~ms}, 128 \mathrm{~ms}$ or 256 ms |
| 60 ms | $2^{16}$ rows | $64 \mathrm{~ms}, 128 \mathrm{~ms}$ or 256 ms |
| 128 ms | all other rows | 128 ms or 256 ms |

(c) What command bus utilization is directly caused by DRAM refreshes?

Command bus utilization: $0.512 \%$.
Explanation. The rows that are accessed every 1 ms and every 60 ms do not need to be refreshed.

The remaining rows $\left(2^{17}\right)$ are accessed once every two refresh intervals ( 128 ms ). So, the command bus utilization is $\frac{2^{17} * 5 n s * 100}{128 m s}=0.512 \%$

### 6.4 Refresh Policy D

Assume that the memory controller implements an approximate mechanism to reduce refresh rate using Bloom filters, as we discussed in class. For this question we assume the retention times in Table 1 with a constant temperature of $64^{\circ} \mathrm{C}$.

One Bloom filter is used to represent the set of all rows that require a 64 ms refresh rate.
The memory controller's refresh logic is modified so that on every potential refresh of a row (every 64 ms ), the refresh logic probes the Bloom filter. If the Bloom filter probe results in a "hit" for the row address, then the row is refreshed. Any row that does not hit in the Bloom filter is refreshed at the default rate of once per 128 ms .
(a) The memory controller performs 2107384 refreshes in total across the channel over a time interval of 1.024 seconds. What is the false positive rate of the Bloom filter? (NOTE: False positive rate $=$ Total number of false positives / Total number of accesses).

- Hint: $2107384=2^{3} *\left(2^{18}+2^{11}-2^{10}+2^{9}-2^{8}-1\right)$

False Positive rate $=\left(2^{10}-1\right) / 2^{18}$.
Explanation. First, there are $2^{8}$ rows that need to be refreshed every 64 ms . All the other rows $\left(2^{18}-2^{8}\right)$ will be refreshed every 128 ms .

At 64 ms , we have $2^{8}$ rows which actually require this rate, plus P out of the $2^{18}-2^{8}$ other rows which will be false-positives. In a period of 1.024 s , this bin is refreshed 16 times. The number of refreshes for this bin is $16 *\left(2^{8}+P *\left(2^{18}-2^{8}\right)\right)$.

At 128 ms we have the rest of the rows which are $2^{18}-2^{8}$ minus the false positives refreshed in the 64 ms bin. So, the number of rows are $2^{18}-2^{8}-P *\left(2^{18}-2^{8}\right)$. In a period of 1.024 s , this bin is refreshed 8 times. The number of refreshes for this bin is $8 *\left(\left(2^{18}-2^{8}\right)-\left(P *\left(2^{18}-2^{8}\right)\right)\right.$

Thus, in total we have $16 *\left(2^{8}+P *\left(2^{18}-2^{8}\right)\right)+8 *\left(\left(2^{18}-2^{8}\right)-P *\left(2^{18}-2^{8}\right)\right)=2107384$ . Solving the equation, $\mathrm{P}=2^{-8}$.

Finally, False positive rate $=$ Total number of false positives $/$ Total number of accesses $=$ $\left.P *\left(2^{18}-2^{8}\right)\right) / 2^{18}=2^{-8} *\left(2^{18}-2^{8}\right) / 2^{18}=\left(2^{10}-1\right) / 2^{18}$

## 7 DRAM Refresh - Utilization

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 32 K rows. Each row in one bank is 8 KB . 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?
$2^{23}$ refreshes per channel; $2^{25}=32 M$ across 4 channels.
(b) What command bus utilization, across all memory channels, is directly caused by DRAM refreshes?

$$
4.096 \%=(32 M \times 5 n s /(4 \times 1.024 s))
$$

(c) What data bus utilization, across all memory channels, is directly caused by DRAM refreshes?
$\square$
0
(d) What bank utilization (on average across all banks) is directly caused by DRAM refreshes?

$$
2.048 \%=(32 K \times 16 \times 40 \mathrm{~ns} / 1.024 \mathrm{~s})
$$

(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 |
| :--- | :--- |
| 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?

$$
(32 k \times 1)+\left(2^{5} \times\left(\frac{256 / 64}{-} 1\right)+2^{9} \times\left(\frac{256 / 128}{-} 1\right)\right) \text { row refreshes per } 256 \mathrm{~ms} .
$$

$$
\times 4 \text { for } 1.024 \mathrm{~s}
$$

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

$$
1.0243 \%=5 e-9 *\left(2^{5} \times 16+2^{9} \times 8+\left(2^{21}-2^{9}-2^{5}\right) \times 4\right) /(4 \times 1.024)
$$

(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?

$$
0.5121 \%=40 e-9 \times\left(2^{5} \times 16+2^{9} \times 8+\left(2^{21}-2^{9}-2^{5}\right) \times 4\right) /(64 \times 1.024)
$$

(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)
```

(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?
false positives: $8192+2(8194)$

What command bus utilization results from this refresh scheme?

$$
5 e-9 \times\left(\left(2^{5}+2\right) \times 16+\left(2^{9}+2^{13}\right) * 8+\left(2^{21}-2^{9}-2^{5}-2-2^{13}\right) \times 4\right) /(4 \times 1.024)
$$

What data bus utilization results from this refresh scheme?

```
0%
```

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

$$
0.5141 \%=40 e-9 \times\left(\left(2^{5}+2\right) \times 16+\left(2^{9}+2^{13}\right) * 8+\left(2^{21}-2^{9}-2^{5}-2-2^{13}\right) \times 4\right) /(64 \times 1.024)
$$

## 8 DRAM Scheduling and Latency

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 and 4 banks.
- The DRAM is byte-addressable.
- The most-significant 2 bits of the physical memory address determine the bank.
- The DRAM command bus operates at 500 MHz 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.

You realize that you can observe the memory requests that are waiting to be serviced in the request buffer. At a particular point of time, you take the snapshot of the request buffer and you observe the following requests in the request buffer.
Requests in the request buffer (in descending order of request age, where the oldest request is on the top):
Read 0x4C80
Read 0x0140
time
Read 0x4EC0
Read 0x8000
Read 0xF000
Read 0x803F
Read 0x4E80
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 --- PRECHARGE
Cycle 6 --- ACTIVATE
Cycle 10 --- READ
Cycle 11 --- READ
Cycle 21 --- PRECHARGE
Cycle 27 --- ACTIVATE
Cycle 31 --- READ
Cycle 32 --- ACTIVATE
Cycle 36 --- READ
Cycle 37 --- READ
Cycle 38 --- READ
Cycle 42 --- PRECHARGE
Cycle 48 --- ACTIVATE
Cycle 52 --- 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?
i) ACTIVATE-to-READ latency

8 ns.
Explanation. After issuing the ACTIVATE command at cycle 6, the memory controller waits until cycle 10 , which indicates that the ACTIVATE-to-READ latency is 4 cycles. The command bus operates at 500 MHz , so it has 2 ns clock period. Thus, the ACTIVATE-to-READ is $4 * 2=8 \mathrm{~ns}$.
ii) ACTIVATE-to-PRECHARGE latency

30 ns .

Explanation. The bank activated at cycle 6 is precharged at cycle 21. Although the memory controller is idle after cycle 11, it waits until cycle 21, which means the ACTIVATE-to-PRECHARGE latency restricts the memory controller from issuing the PRECHARGE command earlier. Thus, the ACTIVATE-to-PRECHARGE latency is 15 cycles $=30 \mathrm{~ns}$.
iii) PRECHARGE-to-ACTIVATE latency

12 ns.

Explanation. The PRECHARGE-to-ACTIVATE latency can be easily seen in the first two commands at cycles 0 and 6 . The PRECHARGE-to-ACTIVATE latency is 6 cycles $=12 \mathrm{~ns}$.
(b) What is the row size in bytes? Explain your answer.

## 64 bytes.

Explanation. The Read request to address 0 x 803 F (to Bank 2) does not require an ACTIVATE command, which means there is a row hit for that access. The open row was activated by the command issued for the request to the address $0 \times 8000$. That means the target rows of both requests should be the same. When we look at the binary form of those addresses, we see that the least significant 6 bits are different ( 000000 for $0 \times 8000$ and 111111 for $0 x 803 F$ ). That means at least 6 of the least significant bits should be the column bits.
Later, when we look at the commands issued for the requests to 0 x 4 ECO and 0 x 4 E 80 , we see that for both of those requests, the memory controller has opened a new row. Thus, the target rows of those requests should be different. Since only the 6 th bit (assuming the least significant bit is the 0th bit) is different between the two addresses, the 6 th bit should be part of the row address. So, of the 6 th bit is part of the row address, the number of columns bits should be 6 or less. As we previously found from the requests to $0 \times 8000$ and $0 \times 803 \mathrm{~F}$ that the number of column bits should be at least 6 , combining those two findings we can say that the number of column bits should be exactly 6 . Thus, the row size is $2^{6}=64$ bytes.
(c) What is the status of the banks prior to the execution of any of the 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 | 5 |
| Bank 1 | Open | Unknown |
| Bank 2 | Closed | - |
| Bank 3 | Open | 192 |

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).

(d) 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 2 cycles and the ACTIVATE-to-PRECHARGE latency reduces by 5 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.

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 $M-1$ are the far-segment rows.

If the above DRAM commands are issued 5 cycles faster with TL-DRAM compared to the baseline (the last command is issued in cycle 47), how many rows are in the near-segment? Show your work.

## 59 rows have to be in the near segment.

Explanation. There should 59 rows in the near-segment (rows 0 to 58 ) since rows until row id 58 need to be accessed with low latency to get 5 cycle reduction. Rows 59 and 192 are in the far-segment, thus latency for accessing them increases slightly.
Here is the new command trace:

```
Cycle 0 --- PRECHARGE - Bank 1
Cycle 6 --- ACTIVATE - Bank 1, Row 50, near segment
Cycle 8 --- READ - Bank 1
Cycle 9 --- READ - Bank 0
Cycle 16 --- PRECHARGE - Bank 1
Cycle 22 --- ACTIVATE - Bank 1, Row 59, far segment
Cycle 27 --- READ - Bank 1
Cycle 28 --- ACTIVATE - Bank 2, Row 0
Cycle 30 --- READ - Bank 2
Cycle 31 --- READ - Bank 3
Cycle 32 --- READ - Bank 2
Cycle 39 --- PRECHARGE - Bank 1
Cycle 45 --- ACTIVATE - Bank 1, Row 58, near segment
Cycle 47 --- READ - Bank 1
```


## 9 Memory Scheduling

In lectures, we introduced a variety of ways to tackle memory interference. In this problem, we will look at the Blacklisting Memory Scheduler (BLISS) to reduce unfairness. There are two key aspects of BLISS that you need to know.

- When the memory controller services $\eta$ consecutive requests from a particular application, this application is blacklisted. We name this non-negative integer $\eta$ the Blacklisting Threshold.
- The blacklist is cleared periodically every $\mathbf{1 0 0 0 0}$ cycles starting at $t=0$.

To reduce unfairness, memory requests in BLISS are prioritized in the following order:

- Non-blacklisted applications' requests
- Row buffer hit requests
- Older requests

The memory system for this problem consists of 2 channels with 2 banks each. Tables 1 and 2 show the memory request stream in the same bank for both applications at different times $(t=0$ and $t=10)$. For both tables, a request on the left-hand side is older than a request on the right-hand side in the same table. The applications do not generate more requests than those shown in Tables 1 and 2. The memory requests are labeled with numbers that represent the row position of the data within the accessed bank. Assume the following for all questions:

- A row buffer hit takes $\mathbf{1 0 0}$ cycles.
- A row buffer miss (i.e., opening a row in a bank with a closed row buffer) takes 200 cycles.
- A row buffer conflict (i.e., closing the currently open row and opening another one) takes 250 cycles.
- All row buffers are closed at time $t=0$

| Application A (Channel 0, Bank 0) |  |  |  |  |  |  |  |  |
| :---: | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Application B (Channel 0, Bank 0) | Row 2 | Row 2 | Row 2 | Row 2 | Row 2 | Row 3 | Row 3 | Row 4 |

Table 2: Memory requests of the two applications at $t=0$

| Application A (Channel 0, Bank 0) | Row 3 | Row 7 | Row 2 | Row 0 | Row 5 | Row 3 | Row 8 | Row 9 |
| :---: | :---: | :---: | :---: | :---: | :---: | :--- | :--- | :--- |
| Application B (Channel 0, Bank 0) | Row 2 | Row 2 | Row 2 | Row 2 | Row 2 | Row 3 | Row 3 | Row 4 |

Table 3: Memory requests of the two applications at $t=10$. Note that none of the Application B's existing requests are serviced yet.
(a) Compute the slowdown of each application using the FR-FCFS scheduling policy after both threads ran to completion. We define:

$$
\text { slowdown }=\frac{\text { memory latency of the application when run together with other applications }}{\text { memory latency of the application when run alone }}
$$

Show your work.
slowdown $_{A}=\sim 1.53$
slowdown $_{B}=1.25$

## Explanation:

For both applications, the first request will incur row buffer miss penalty, and the rest of the requests will either be hits or conflicts.
Application A (alone) $=200+100+250 * 6=1800$ cycles
Application $B($ alone $)=200+100 * 4+250+100+250=1200$ cycles
Applications A (with B, FR-FCFS) $=200+100 * 4+100+250+100+100 * 2+250+250 * 5=$ 2750 cycles
Applications B (with A, FR-FCFS) $=200+100 * 4+100+250+100+100 * 2+250=1500$ cycles
From the two tables above we know that all requests of application $B$ were issued before any of the application A's requests were issued. Thus, all requests of $B$ are prioritized unless there is a row hit for A's requests.
slowdown $_{A}=\frac{2750}{1800}=\sim 1.53$
slowdown $_{B}=\frac{1500}{1200}=1.25$
(b) If we use the BLISS scheduler, for what value(s) of $\eta$ (the Blacklisting Threshold) will the slowdowns of both applications be equal to those obtained with FR-FCFS?

For $\eta \geq 6$ or $\eta=0$.

## Explanation:

We want both A and B to complete without blacklisting or to complete both blacklisted, thus $\eta \geq 6$ and $\eta=0$, respectively.
(c) For what value(s) of $\eta$ (the Blacklisting Threshold) will the slowdown of A be $<1.5$ ?

Impossible. Slowdown for A will always be $\geq 1.5$
Explanation: For the give memory requests, it is not possible to find $\eta$ that blacklists B but not A. Thus, the smallest slowdown for A is the case explained in the solution of part (b).
(d) For what value(s) of $\eta$ (the Blacklisting Threshold) will B experience the maximum slowdown it can possibly experience with the Blacklisting Scheduler?

For $\eta=5$.

Explanation: We already know that the slowdowns will be equal to the slowdown with FRFCFS when $\eta \geq 6$ or $\eta=0$. If we execute the memory requests for the rest of possible $\eta$ values, we find that $\eta=5$ causes application B to complete after 2150 cycles, which is the largest.
(e) What is a simple mechanism (that we discussed in lectures) that we can use instead of BLISS to make the slowdowns of both A and B equal to 1.00?

Memory Channel Partitioning (MCP)
Explanation: With MCP, each application will operate on an independent channel, without any interference with the other application.

## 10 Emerging Memory Technologies

Researchers at Lindtel developed a new memory technology, L-RAM, which is non-volatile. The access latency of L-RAM is close to that of DRAM while it provides higher density compared to the latest DRAM technologies. L-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) Lindtel markets a new computer system with L-RAM to have a lifetime of 2 years and the following specifications:

- 4 GBs of L-RAM as main memory with a perfect wear-leveling mechanism, i.e., writes are equally distributed over all the cells of L-RAM.
- The processor is in-order and there is no memory-level parallelism.
- It takes 4 ns to send a memory request from the processor to the memory controller and it takes 20 ns to send the request from the memory controller to L-RAM. The write latency of L-RAM is 40 ns .
- L-RAM is word-addressable. Thus, each write request writes 8 bytes to memory.

A student at ETH tests the lifetime of the system and finds that this new computer system cannot guarantee a lifetime of 2 years. She writes a program to wear out the entire L-RAM device as quickly as possible. How fast is she able to wear out the device? Show all work.
$t_{\text {wear_out }}=\frac{2^{32}}{2^{3}} \times 10^{6} \times(40+4+20)$
$t_{\text {wear_out }}=2^{35} \times 10^{6} \mathrm{~ns}$
$t_{\text {wear_out }} \approx 397.68$ days

## Explanation:

- Each memory cell should receive $10^{6}$ writes.
- Since ETH-RAM is word addressable, the required number of writes is equal to $\frac{2^{32}}{2^{3}} \times 10^{6}$.
- The processor is in-order and there is no memory-level parallelism, so the total latency of each memory access is equal to $40+4+20$.
(b) L-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 L-RAM cells by using the single-level cell (SLC) mode. When L-RAM is used in SLC mode, the lifetime of each cell improves by a factor of 10 and the write latency decreases by $75 \%$. What is the lifetime of the system using the SLC mode, if we repeat the experiment in part (a), with all else remaining the same in the system? Show your work.
$t_{\text {wear_out }}=\frac{2^{31}}{2^{3}} \times 10^{7} \times(10+4+20) \times 10^{-9}$
$t_{\text {wear_out }}=91268055.04 \mathrm{~s} \approx 1056.34$ days


## 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 $=2^{32} / 2=2^{31}$
- The required number of writes is equal to $\frac{2^{31}}{2^{3}} \times 10^{7}$.
- The SLC write latency is $0.25 \times t_{\text {write_MLC }}: t_{\text {write_SLC }}=0.25 \times 40=10 \mathrm{~ns}$
(c) Provide a mechanism that would increase the guaranteed lifetime of the computer system without changing the physical circuitry of L-RAM. From the baseline computer system in part (a), describe the changes required to guarantee a computer system lifetime of 2 years, with your mechanism. Be concrete and precise.

Artificially increase the time to either (1) send a memory request from the memory controller to L-RAM or (2) send a request from the processor to the memory controller by 54 ns . $730 * 3600 * 24<\frac{2^{32}}{2^{3}} \times 10^{6} \times(40+4+20+x)$ $x>53.4 \mathrm{~ns}$

## 11 Prefetching II

An ETH student writes two programs A and B and runs them on 3 different toy machines, M1, M2, and M3, to determine the type of the prefetching mechanism used in each of these 3 machines. She observes programs A and B to have the following access patterns to cache blocks. Note that the addresses are cache block addresses, not byte addresses.

Program A: 27 accesses
$a, a+1, a+2, a+3, a+4, a+8, a+16, a+32, a+64$,
$a, a+1, a+2, a+3, a+4, a+8, a+16, a+32, a+64$,
$a, a+1, a+2, a+3, a+4, a+8, a+16, a+32, a+64$
Program B: 501 accesses
$b, b+2, b+4, \ldots, b+998, b+1000$
The student is able to measure the accuracy and coverage of the prefetching mechanism in each of the machines. The following table shows her measurement results:

|  | Machine M1 |  | Machine M2 |  | Machine M3 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Coverage | Accuracy | Coverage | Accuracy | Coverage | Accuracy |
| Program A | $6 / 27$ | $6 / 27$ | 0 | 0 | $1 / 3$ | $9 / 26$ |
| Program B | $499 / 501$ | $499 / 501$ | 0 | 0 | $499 / 501$ | $499 / 500$ |

The student knows the following facts about M1, M2, and M3 machines:

- The prefetcher prefetches into a fully-associative cache whose size is 8 cache blocks. The cache employs the FIFO (First-In First-Out) replacement policy.
- The prefetchers have large enough resources to detect and store access patterns.
- Each cache block access is separated long enough in time such that all prefetches issued can complete before the next access happens.
- There are 5 different possible choices for the prefetching mechanism:

1) Markov prefetcher with a correlation table of 4 entries
2) Markov prefetcher with a correlation table of 10 entries
3) 1st-next-block prefetcher (degree $=1$ ) - prefetches block $\mathrm{N}+1$ after seeing block N
4) 4th-next-block prefetcher (degree $=1$ ) - prefetches block $\mathrm{N}+4$ after seeing block N
5) stride prefetcher

- None of the above-mentioned prefetchers employ confidence bits.
- The prefetchers start out with an empty table when each program A and B start execution.
- The prefetcher sends only one prefetch request after a program access (i.e., prefetch degree $=1$ ).

Determine what type of prefetching mechanism each of the above-mentioned machines use:

| Machine M1: 4th-next-block prefetcher |
| :--- |
| Machine M2: Markov prefetcher with a correlation table of 4 entries |
| Machine M3: Stride prefetcher |

Extra space for explanation:

We calculate the accuracy and coverage for all 5 types of prefetchers, and then we can answer what prefetcher each machine is using:

The 5 prefetechers work in the following ways when running Application A:
Markov, table size $=4$ : Coverage: 0 , Accuracy: 0
$\mathrm{a}, \mathrm{a}+1, \mathrm{a}+2, \mathrm{a}+3, \mathrm{a}+4, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$,
$a, a+1, a+2, a+3, a+4, a+8, a+16, a+32, a+64$,
$\mathrm{a}, \mathrm{a}+1, \mathrm{a}+2, \mathrm{a}+3, \mathrm{a}+4, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$,

Markov, table size=10: Coverage: 17/27, Accuracy: 17/18
$\mathrm{a}, \mathrm{a}+1, \mathrm{a}+2, \mathrm{a}+3, \mathrm{a}+4, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$,
$\mathrm{a}, \frac{\mathrm{a}+1}{}, \underline{\mathrm{a}+2}, \underline{\mathrm{a}+3}, \underline{\mathrm{a}+4}, \underline{\mathrm{a}+8}, \underline{\mathrm{a}+16}, \underline{a+32}, \underline{a+64}$,
$\underline{a}, \underline{a+1}, \underline{a+2}, \underline{a+3}, \underline{a+4}, \underline{a+8}, \underline{a+16}, \underline{a+32}, \underline{a+64}$-unused: $a$

1st-next-block: Coverage: 4/9, Accuracy: 4/9
$\mathrm{a}, \underline{\mathrm{a}+1}, \underline{\mathrm{a}+2}, \underline{\mathrm{a}+3}, \underline{\mathrm{a}+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64,-$ unused: $\mathrm{a}+5, \mathrm{a}+9, \mathrm{a}+17, \mathrm{a}+33, \mathrm{a}+65$, $\mathrm{a}, \overline{\mathrm{a}+1}, \overline{\mathrm{a}+2}, \overline{\mathrm{a}+3}, \overline{\mathrm{a}+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64,-$ unused: $\mathrm{a}+5, \mathrm{a}+9, \mathrm{a}+17, \mathrm{a}+33, \mathrm{a}+65$,
$\mathrm{a}, \overline{\mathrm{a}+1}, \overline{\mathrm{a}+2}, \overline{\mathrm{a}+3}, \overline{\mathrm{a}+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$-unused: $\mathrm{a}+5, \mathrm{a}+9, \mathrm{a}+17, \mathrm{a}+33, \mathrm{a}+65$

4th-next-block: Coverage: 6/27, Accuracy: 6/27
$a, a+1, a+2, a+3, \underline{a+4}, \underline{a+8}, a+16, a+32, a+64,-$ unused: $a+5, a+6, a+7, a+12, a+20, a+$ $36, a+68$,
$a, a+1, a+2, a+3, \underline{a+4}, \underline{a+8}, a+16, a+32, a+64,-$ unused: $a+5, a+6, a+7, a+12, a+20, a+$ $36, a+68$,
$a, a+1, a+2, a+3, a+4, a+8, a+16, a+32, a+64$-unused: $a+5, a+6, a+7, a+12, a+20, a+$ $36, a+68$

Stride: Coverage: 1/3, Accuracy: 9/26
$\mathrm{a}, \mathrm{a}+1, \underline{a+2}, \underline{a+3}, \underline{a+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$, unused: $\mathrm{a}+5, \mathrm{a}+12, \mathrm{a}+24, \mathrm{a}+48, \mathrm{a}+96$, $a, \mathrm{a}+1, \overline{\mathrm{a}+2}, \overline{\mathrm{a}+3}, \overline{\mathrm{a}+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64,-$ unused: $\mathrm{a}-64, \mathrm{a}+5, \mathrm{a}+12, \mathrm{a}+24, \mathrm{a}+48, \mathrm{a}+$ 96,
$\mathrm{a}, \mathrm{a}+1, \underline{\mathrm{a}+2}, \underline{\mathrm{a}+3}, \underline{\mathrm{a}+4}, \mathrm{a}+8, \mathrm{a}+16, \mathrm{a}+32, \mathrm{a}+64$-unused: $\mathrm{a}-64, \mathrm{a}+5, \mathrm{a}+12, \mathrm{a}+24, \mathrm{a}+48, \mathrm{a}+$ 96

The 5 prefetechers work in the following ways when running Application B:
Markov, table size=4: Coverage: 0 , Accuracy: 0
$\mathrm{b}, \mathrm{b}+2, \mathrm{~b}+4, \mathrm{~b}+6, \mathrm{~b}+8, \mathrm{~b}+10, \ldots, \mathrm{~b}+998, \mathrm{~b}+1000$
Markov, table size $=10$ : Coverage: 0 , Accuracy: 0
$\mathrm{b}, \mathrm{b}+2, \mathrm{~b}+4, \mathrm{~b}+6, \mathrm{~b}+8, \mathrm{~b}+10, \ldots, \mathrm{~b}+998, \mathrm{~b}+1000$
1st-next-block: Coverage: 0, Accuracy: 0
$\mathrm{b}, \mathrm{b}+2, \mathrm{~b}+4, \mathrm{~b}+6, \mathrm{~b}+8, \mathrm{~b}+10, \ldots, \mathrm{~b}+998, \mathrm{~b}+1000$-unused: $\mathrm{b}+1, \mathrm{~b}+3, \ldots, \mathrm{~b}+999, \mathrm{~b}+1001$
4th-next-block: Coverage: 499/501, Accuracy: 499/501
$\mathrm{b}, \mathrm{b}+2, \underline{\mathrm{~b}+4}, \underline{\mathrm{~b}+6}, \underline{\mathrm{~b}+8}, \underline{\mathrm{~b}+10}, \ldots, \underline{\mathrm{~b}+998}, \underline{\mathrm{~b}+1000}$-unused: $\mathrm{b}+1002, \mathrm{~b}+1004$
Stride: Coverage: 499/501, Accuracy: 499/500
$b, b+2, \underline{b+4}, \underline{b+6}, \underline{b+8}, \underline{b+10}, \ldots, \underline{b+998}, \underline{b+1000}$-unused: $b+1002$

