ETH 263-2210-00L Computer Architecture, Fall 2019
HW 2: RowHammer, Genome Analysis, Computation in Memory

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, Rahul Bera, Geraldo Francisco De Oliveira Junior, Can Firtina, Juan Gomez Luna, Jawad Haj-Yahya, Hasan Hassan, Konstantinos Kanellopoulos, Jeremie Kim, Nika Mansouri Ghiasi, Lois Orosa Nogueira, Jisung Park, Minesh Hamenbhai Patel, Abdullah Giray Yaglikci

Given: Monday, Oct 14, 2019
Due: **Sunday, Oct 27, 2019**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture19/`. Please, check your inbox, you should have received an email with the password you should use to login. If you didn't receive any email, contact comparch@lists.ethz.ch. In the first page after login, you should click in "Architecture - Fall 2019 Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-5).** You should upload your answers to the Moodle Platform (`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=382814`) as a single PDF file.

1. Critical Paper Reviews [450 points]

Please read the guidelines for reviewing papers and check the sample reviews. You may access them by *simply clicking on the QR codes below or scanning them.* We will give out extra credit that is worth 0.5% of your total grade for each good review.

Guidelines

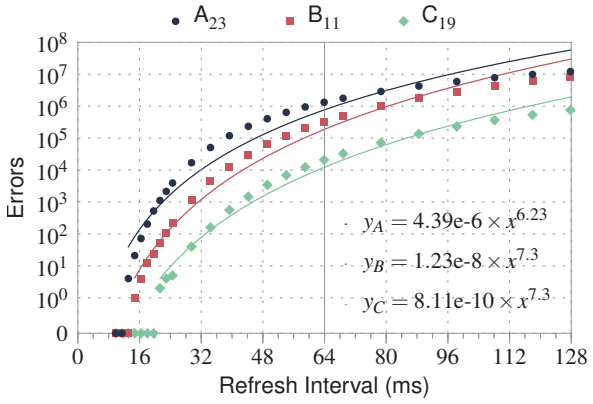
Sample reviews

Write an approximately one-page critical review for each of the following papers. A review with bullet point style is more appreciated. Try not to use very long sentences and paragraphs. Keep your writing and sentences simple. Make your points bullet by bullet, as much as possible.

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015. `https://people.inf.ethz.ch/omutlu/pub/tesseract-pim-architecture-for-graph-processing_isca15.pdf`
- Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Giboos, and Todd C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, 2017. `https://people.inf.ethz.ch/omutlu/pub/ambit-bulk-bitwise-dram_micro17.pdf`
- Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gómez-Luna, and Onur Mutlu, "Processing-in-Memory: A Workload-Driven Perspective," IBM Journal of Research & Development, 2019 (to appear).`https://arxiv.org/pdf/1907.12947.pdf` (Preliminary axXiv version)

2. RowHammer Mitigations [90 points]

One research group rigorously investigated the relationship between the DRAM refresh rate and RowHammer-induced errors using real DRAM modules from three major memory manufacturers: A, B, and C. The following figure shows the characterization results of the *most RowHammer-vulnerable* modules of each DRAM manufacturer A, B, and C (A_{23}, B_{11}, and C_{19}, respectively). As shown in the figure below, we can achieve an *effectively-zero* probability of RowHammer-induced errors when we reduce the refresh interval to 8 ms. This is because the probability of RowHammer-induced errors becomes less than that of a random DRAM circuit fault.



As a leading computer architect of a company, you are asked to design an efficient RowHammer mitigation technique based on the characterization results.

(a) [10 points] A junior engineer suggests to simply reduce the refresh interval from the default 64 ms to 8 ms. How will the bank utilization U and the DRAM energy consumption E of all refresh operations be changed over the current system?

(b) [10 points] A DRAM manufacturer releases a higher capacity version of the same DRAM module (4x the total capacity). After rigorously reverse-engineering, you determine that the manufacturer doubles both the (1) number of rows in a bank, and (2) number of banks in a module without changing any other aspects (e.g., row size, bus rate, and timing parameters). Your company considers upgrading the current system with the higher-capacity DRAM module and asks your opinion. In the current system, the bank utilization of refresh operations is 0.05 when the refresh interval is 64 ms. Would reducing the refresh interval to 8-ms refresh interval still be applicable (in terms of RowHammer protection capability and performance overheads) in a module with 2x the number of rows per bank and number of banks per module? How about in a module with 4x the number of rows per bank and the number of banks per module?

(c) [10 points] Due to significant overheads of reducing the refresh interval, your team decides to find other RowHammer mitigation techniques with lower overhead. The junior engineer proposes a counter-based approach as follows:

In this approach, the memory controller maintains a counter for each row R, which increments when an adjacent row is activated, and resets when Row R is activated or refreshed. If the value of a row R's counter exceeds a threshold value, T, the memory controller activates row R and resets its respective counter.

Here are some specifications on the current memory system:

- Interface: DDR3-1333
- $CL = 7$
- $tRCD = 13.5$ ns
- $tRAS = 35$ ns
- $tWR = 15$ ns
- $tRP = 13.5$ ns
- $tRFC = 120$ ns
- Configuration: Per-channel memory controller deals with 2 ranks, each of which has 8 banks. The number of rows per bank is 2^{15}. Each row in one bank is 8 KiB.

Are the given specifications enough to implement the proposed approach? If no, list the specifications additionally required.

(d) [20 points] Suppose that all the necessary specifications for implementing the proposed approach are known. Calculate the maximum value of T that can guarantee the same level of security against RowHammer attacks over when adopting 8-ms refresh interval?

(e) [10 points] Calculate the number of bits required for counters in each memory controller. How does it change when the number of rows per bank and the number of banks per chip are doubled?

(f) [10 points] Obviously, we can reduce the size of counters in each memory controller, if we use a smaller value for T. What are its drawbacks?

(g) [20 points] You recall *PARA (Probabilistic Adjacent Row Activation)* which is proposed in the first RowHammer paper. Here is how a PARA-enabled memory controller works (for more details, see Section 8.2 in the paper[1]:

Whenever a row is closed, the controller flips a biased coin with a probability p of turning up heads, where $p << 1$. If the coin turns up heads, the controller opens one of its adjacent rows where either of the two adjacent rows are chosen with equal probability ($p/2$). Due to its probabilistic nature, PARA does not guarantee that the adjacent will always be refreshed in time. Hence, PARA cannot prevent disturbance errors with absolute certainty. However, its parameter p can be set so that disturbance errors occur at an extremely low probability — many orders of magnitude lower than the failure rates of other system components (e.g., more than 1% of hard-disk drives fail every year.)

Suppose that the probability of experiencing an error for PARA in 64 ms is 1.9×10^{-22} when $p = 0.001$ at the *worst operating conditions*. How can we estimate the probability of experiencing an error in one year based on that value? Show your work. (If you just put an answer, you will get no points for this problem.)

[1] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu, "Flipping Bits in Memory without Accessing Them: an Experimental Study of DRAM Disturbance Errors," in Proceedings of the 41st Annual International Sympoisum on Computer Architecture, 2014. `https://people.inf.ethz.ch/omutlu/pub/dram-row-hammer_isca14.pdf`

3. Genome Analysis [60 points]

3.1. Edit Distance [15 points]

One of the most fundamental computational steps in most bioinformatics analyses is the detection of the differences between two DNA or protein sequences. *Edit distance* is one way to measure the differences between two genomic sequences. *Edit distance* algorithm calculates the minimum number of edit operations needed to convert one sequence into the other. Allowed edit operations are: (1) *substitution*, (2) *insertion*, and (3) *deletion of a character*. The notion of edit distance is also useful for spell checking and pattern recognition applications.

Compute the *Edit distance* for each of the following string pairs and provide the list of the edit operations (e.g., delete character 'F' from string "Friday" to be "riday") used to convert the first string into the second string.

(a) [5 points] Montag & Donnerstag

(b) [5 points] Freitag & Samstag

(c) [5 points] Donnerstag & "" (where "" is an empty string)

3.2. Read Mapping [45 points]

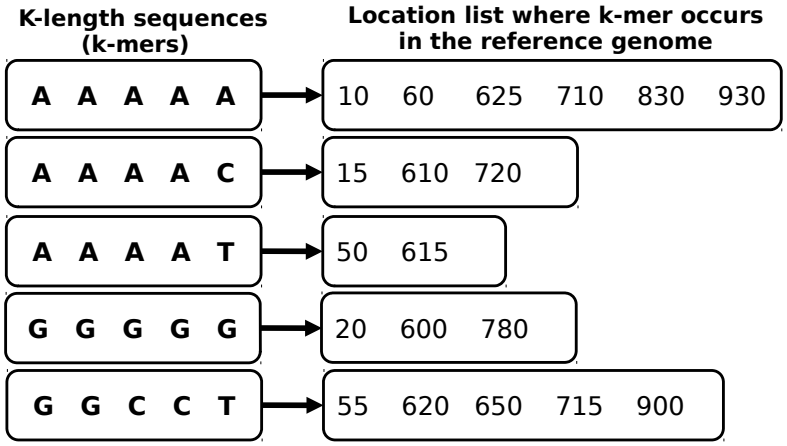
During a process called read mapping in genome analysis, each genomic read is mapped onto one or more possible locations in the reference genome based on the similarity between the read and the reference genome segment at that location. Suppose that you would like to map the following reads to the human reference genome sequence.

read 1 = AAAAA_AAAAC_GGGGG
read 2 = AAAAT_GGCCT_AAAAA
read 3 = AAAAT_AGCGG_GCGCT
read 4 = AAAAC_AAAAT_GGCCT
read 5 = AAAAA_GGCCT_AAAAC

And suppose that you need to use the following 3-step hash-based mapping method:

(1) Hash-based read mapper first constructs hash table to rapidly examine whether or not a short segment (called k-mers, where k is length of the segment) exists in the reference sequence.
(2) The mapper extracts 3 consecutive non-overlapping 5-mers from a read (5-mers are separated by '_' sign in each read) and uses them to query the hash table. The hash table returns all the occurrence hits of each k-mer in the reference genome.
(3) For each hit, the mapper examines the differences between the entire read that includes the k-mer and the extracted reference segment (starting from the returned location of the k-mer from the hash table) using dynamic-programming edit distance function (*edit_ distance()*).

The hash table (Step 1) is provided below, which includes a list of 5-mers extracted from the human reference genome and their location list (each number represents the starting location of that k-mer in the reference genome sequence). Answer the following questions:



(a) [15 points] How many times in total the edit distance function, *edit_ distance()*, will be invoked using the 3-step hash-based mapping method described above?

(b) [15 points] Suppose you want to change Step 3 to include "*Adjacency Filtering*". This means that all k-mers extracted from a read should be adjacent in the reference genome before calling the *edit_ distance()* function once for that read. For example, if the first k-mer extracted from the read exists in the reference genome at location x, then the second k-mer and the third k-mer should also exist in the reference genome at location $x+k$ and $x+2k$, respectively.
How many times in total the edit distance function, *edit_ distance()*, will be invoked?

(c) [15 points] Suppose now you want to change Step 3 to include "*Cheap K-mer Selection*", where the threshold is 4. This means that the *edit_ distance()* function will be invoked for each k-mer that occurs less frequently (less than a threshold) in the reference genome.
How many times in total the edit distance function, *edit_ distance()*, will be invoked?

4. Processing in Memory: Ambit [135 points]

4.1. In-DRAM Bitmap Indices I [70 points]

Recall that in class we discussed Ambit, which is a DRAM design that can greatly accelerate *bulk bitwise operations* by providing the ability to perform bitwise AND/OR of two rows in a subarray.

One real-world application that can benefit from Ambit's in-DRAM bulk bitwise operations is the database *bitmap index*, as we also discussed in the lecture. By using bitmap indices, we want to run the following query on a database that keeps track of user actions: "How many unique users were active every week for the past w weeks?" Every week, each user is represented by a single bit. If the user was active a given week, the corresponding bit is set to 1. The total number of users is u.

We assume the bits corresponding to one week are all in the same row. If u is greater than the total number of bits in one row (the row size is 8 kilobytes), more rows in different subarrays are used for the same week. We assume that all weeks corresponding to the users in one subarray fit in that subarray.

We would like to compare two possible implementations of the database query:

- *CPU-based implementation*: This implementation reads the bits of all u users for the w weeks. For each user, it `and`s the bits corresponding to the past w weeks. Then, it performs a bit-count operation to compute the final result.
  Since this operation is very memory-bound, we simplify the estimation of the execution time as the time needed to read all bits for the u users in the last w weeks. The memory bandwidth that the CPU can exploit is X bytes/s.
- *Ambit-based implementation*: This implementation takes advantage of bulk `and` operations of Ambit. In each subarray, we reserve one *Accumulation* row and one *Operand* row (besides the control rows that are needed for the regular operation of Ambit). Initially, all bits in the *Accumulation* row are set to 1. Any row can be moved to the *Operand* row by using RowClone (recall that RowClone is a mechanism that enables very fast copying of a row to another row in the same subarray). t_{rc} and t_{and} are the latencies (in seconds) of RowClone's `copy` and Ambit's `and` respectively.
  Since Ambit does *not* support bit-count operations inside DRAM, the final bit-count is still executed on the CPU. We consider that the execution time of the bit-count operation is negligible compared to the time needed to read all bits from the *Accumulation* rows by the CPU.

(a) [15 points] What is the total number of DRAM rows that are occupied by u users and w weeks?

(b) [20 points] What is the throughput in users/second of the Ambit-based implementation?

(c) [20 points] What is the throughput in users/second of the CPU implementation?

(d) [15 points] What is the maximum w for the CPU implementation to be faster than the Ambit-based implementation? Assume u is a multiple of the row size.

4.2. In-DRAM Bitmap Indices II [65 points]

You have been hired to accelerate ETH's student database. After profiling the system for a while, you found out that one of the most executed queries is to *"select the hometown of the students that are from Switzerland and speak German"*. The attributes *hometown*, *country*, and *language* are encoded using a four-byte binary representation. The database has 32768 (2^{15}) entries, and each attribute is stored contiguously in memory. The database management system executes the following query:

```
bool position_hometown[entries];
for(int i = 0; i < entries; i++){
    if(students.country[i] == "Switzerland" && students.language[i] == "German"){
        position_hometown[i] = true;
    }
    else{
        position_hometown[i] = false;
    }
}
```

(a) [25 points] You are running the above code on a single-core processor. Assume that:

- Your processor has an 8 MiB direct-mapped cache, with a cache line of 64 bytes.
- A hit in this cache takes one cycle and a miss takes 100 cycles for both load and store operations.
- All load/store operations are serialized, i.e., the latency of multiple memory requests cannot be overlapped.
- The starting addresses of *students.country*, *students.language*, and *position_hometown* are 0x05000000, 0x06000000, 0x07000000, respectively.
- The execution time of a non-memory instruction is zero (i.e., we ignore its execution time).

How many cycles are required to run the query? Show your work.

(b) Recall that in class we discussed Ambit, which is a DRAM design that can greatly accelerate *bulk bitwise operations* by providing the ability to perform bitwise AND/OR/XOR of two rows in a subarray. Ambit works by issuing back-to-back ACTIVATE (A) and PRECHARGE (P) operations. For example, to compute AND, OR, and XOR operations, Ambit issues the sequence of commands described in the table below (e.g., $AAP(X, Y)$ represents double row activation of rows X and Y followed by a precharge operation, $AAAP(X, Y, Z)$ represents triple row activation of rows X, Y, and Z followed by a precharge operation). In those instructions, Ambit copies the source rows D_i and D_j to auxiliary rows (B_i). Control rows C_i dictate which operation (AND/OR) Ambit executes. The DRAM rows with dual-contact cells (i.e., rows DCC_i) are used to perform the bitwise NOT operation on the data stored in the row. Basically, copying a source row to DCC_i flips all bits in the source row and stores the result in both the source row and DCC_i. Assume that:

- The DRAM row size is **8 Kbytes**.
- An ACTIVATE command takes 50 cycles to execute.
- A PRECHARGE command takes 20 cycles to execute.
- DRAM has a single memory bank.
- The syntax of an Ambit operation is: *bbop_* [and/or/xor] *destination, source_ 1, source_ 2.*
- Addresses 0x08000000 and 0x09000000 are used to store partial results.
- The rows at addresses 0x0A000000 and 0x0B00000 store the codes for *"Switzerland"* and *"German"*, respectively, in each four bytes throughout the entire row.

$D_k = D_i$ **AND** D_j	$D_k = D_i$ **OR** D_j	$D_k = D_i$ **XOR** D_j
		AAP (D_i, B_0)
		AAP (D_j, B_1)
		AAP (D_i, DCC_0)
AAP (D_i, B_0)	AAP (D_i, B_0)	AAP (D_j, DCC_1)
AAP (D_j, B_1)	AAP (D_j, B_1)	AAP (C_0, B_2)
AAP (C_0, B_2)	AAP (C_1, B_2)	AAAP (B_0, DCC_1, B_2)
AAAP (B_0, B_1, B_2)	AAAP (B_0, B_1, B_2)	AAP (C_0, B_2)
AAP B_0, D_k	AAP B_0, D_k	AAAP (B_1, DCC_0, B_2)
		AAP (C_1, B_2)
		AAAP (B_0, B_1, B_2)
		AAP (B_0, D_k)

i) [20 points] The following code aims to execute the query *"select the hometown of the students that are from Switzerland and speak German"* in terms of Boolean operations to make use of Ambit. Fill in the blank boxes such that the algorithm produces the correct result. Show your work.

```
1 for(int i = 0; i < [          ] ; i++){
2
3   bbop_[          ] 0x08000000, 0x05000000 + i*8192, 0x0A000000;
4
5   bbop_[          ] 0x09000000, 0x06000000 + i*8192, 0x0B000000;
6
7   bbop_[          ] 0x07000000, 0x08000000, 0x09000000;
8 }
```

ii) [20 points] How much speedup does Ambit provide over the baseline processor when executing the same query? Show your work.

5. Caching vs. Processing-in-Memory [90 points]

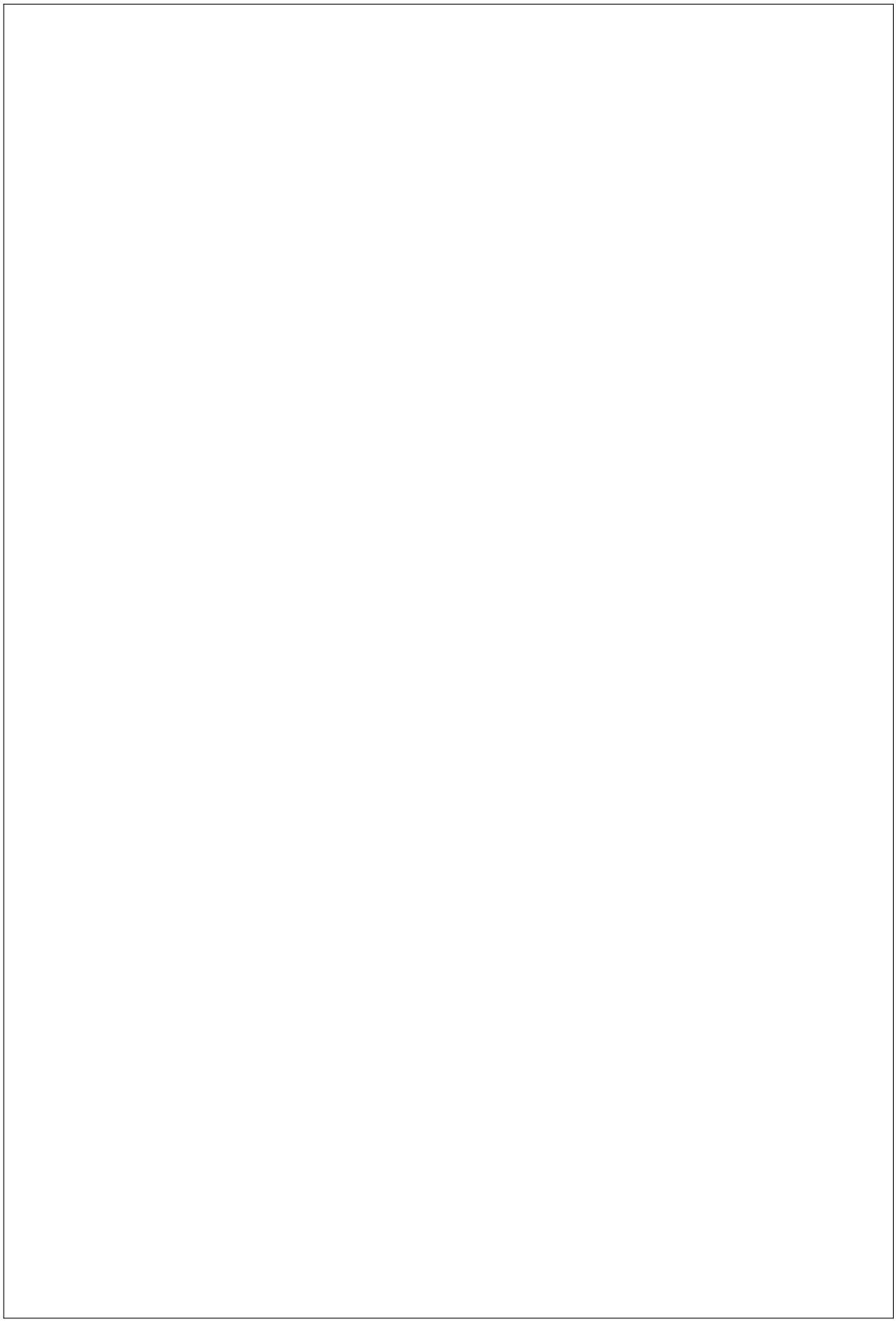
We are given the following piece of code that makes accesses to integer arrays A and B. The size of each element in both A and B is 4 bytes. The base address of array A is 0x00001000, and the base address of B is 0x00008000.

```
movi R1, #0x1000 // Store the base address of A in R1
movi R2, #0x8000 // Store the base address of B in R2
movi R3, #0

Outer_Loop:
    movi R4, #0
    movi R7, #0
    Inner_Loop:
        add R5, R3, R4  // R5 = R3 + R4
        // load 4 bytes from memory address R1+R5
        ld R5, [R1, R5] // R5 = Memory[R1 + R5],
        ld R6, [R2, R4] // R6 = Memory[R2 + R4]
        mul R5, R5, R6  // R5 = R5 * R6
        add R7, R7, R5  // R7 += R5
        inc R4          // R4++
        bne R4, #2, Inner_Loop // If R4 != 2, jump to Inner_Loop

    //store the data of R7 in memory address R1+R3
    st [R1, R3], R7     // Memory[R1 + R3] = R7,
    inc R3              // R3++
    bne R3, #16, Outer_Loop // If R3 != 16, jump to Outer_Loop
```

You are running the above code on a single-core processor. For now, assume that the processor *does not* have caches. Therefore, all load/store instructions access the main memory, which has a fixed 50-cycle latency, for both read and write operations. Assume that all load/store operations are serialized, i.e., the latency of multiple memory requests *cannot* be overlapped. Also assume that the execution time of a non-memory-access instruction is zero (i.e., we ignore its execution time).

(a) [15 points] What is the execution time of the above piece of code in cycles? Show your work.

(b) [30 points] Assume that a 128-byte private cache is added to the processor core in the next-generation processor. The cache block size is 8-byte. The cache is direct-mapped. On a hit, the cache services both read and write requests in 5 cycles. On a miss, the main memory is accessed and the access fills an 8-byte cache line in 50 cycles. Assuming that the cache is initially empty, what is the new execution time on this processor with the described cache? Show your work.



(c) [15 points]You are not satisfied with the performance after implementing the described cache. To do better, you consider utilizing a processing unit that is available *close to the main memory*. This processing unit can directly interface to the main memory with a *10-cycle* latency, for both read and write operations. How many cycles does it take to execute the same program using the in-memory processing units? Show your work. (Assume that the in-memory processing unit does not have a cache, and the memory accesses are serialized like in the processor core. The latency of the non-memory-access operations is ignored.)

(d) [15 points] You friend now suggests that, by changing the cache capacity of the single-core processor (in part (b)), she could provide as good performance as the system that utilizes the memory processing unit (in part (c)). Is she correct? What is the minimum capacity required for the cache of the single-core processor to match the performance of the program running on the memory processing unit? Show your work.

(e) [15 points] What other changes could be made to the cache design to improve the performance of the single-core processor on this program?