ETH 263-2210-00L Computer Architecture, Fall 2022
HW 2: Genome Analysis, RowHammer, Memory Refresh (SOLUTIONS)

Instructor: Prof. Onur Mutlu
TAs: Juan Gómez Luna, Mohammad Sadrosadati, Mohammed Alser, Rahul Bera, Nisa Bostanci, João Dinis Ferreira, Can Firtina, Nika Mansouri Ghiasi, Geraldo Francisco De Oliveira Junior, Konstantinos Kanellopoulos, Joël Lindegger, Rakesh Nadig, Ataberk Olgun, Abdullah Giray Yaglikci, Yahya Can Tugrul, Haocong Luo, Banu Cavlak, Aditya Manglik

Given: Saturday, October 22, 2022
Due: **Saturday, November 5, 2022**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture22/`. Please, check your inbox, you should have received an email with the password you should use to login. If you did not receive any email, contact comparch@lists.inf.ethz.ch. In the first page after login, you should click in "Computer Architecture Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-5).** You should upload your answers to the Moodle Platform (`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=816655`) as a single PDF file.
- If you have any questions regarding this homework, please ask them the Moodle forum (`https://moodle-app2.let.ethz.ch/mod/moodleoverflow/view.php?id=816657`).
- Please note that the handin questions have a hard deadline. However, you can submit your paper reviews till the end of the semester.

1. Critical Paper Reviews [1,000 points]

You will do at least 5 readings for this homework, out of which 4 are tagged as **REQUIRED** papers. You may access them by simply clicking on the QR codes below or scanning them.

Required 1   Required 2   Required 3   Required 4

Write an approximately one-page critical review for the readings (i.e., papers from #1 to #4 **and** at least 1 of the remaining papers, from #5 to #22). If you review a paper other than the 5 mandatory papers, you will receive 200 BONUS points on top of 1,000 points you may get from paper reviews (i.e., each additional submission is worth 200 BONUS points with a possibility to get up to 4400 points). Note that you will get **zero** points from the critical paper reviews if you do not submit the required paper reviews (i.e., papers from #1 to #4).

Please read the guideline slides for reviewing papers and watch Prof. Mutlu's guideline video on how to do a critical paper review. We also provide you with sample reviews which you can access using the QR code. 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. **We will give out extra credit that is worth 0.5% of your total course grade for each good review.**

Guideline Slides   Guideline Video   Sample Reviews

1. **(REQUIRED)** Alser et al., "Accelerating Genome Analysis: A Primer on an Ongoing Journey", IEEE MICRO, 2020. `https://people.inf.ethz.ch/omutlu/pub/AcceleratingGenomeAnalysis_ieeemicro20.pdf`
2. **(REQUIRED)** Kim et al., "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques," ISCA, 2020. `https://people.inf.ethz.ch/omutlu/pub/Revisiting-RowHammer_isca20.pdf`
3. **(REQUIRED)** Yaglikci et al., "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows", MICRO 2021, `https://people.inf.ethz.ch/omutlu/pub/BlockHammer_preventing-DRAM-rowhammer-at-low-cost_hpca21.pdf`
4. **(REQUIRED)** Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," in ISCA, 2012. `https://people.inf.ethz.ch/omutlu/pub/raidr-dram-refresh_isca12.pdf`
5. Liu et al., "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in ISCA, 2013. `https://people.inf.ethz.ch/omutlu/pub/dram-retention-time-characterization_isca13.pdf`
6. Chang et al., "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in HPCA, 2014. `https://people.inf.ethz.ch/omutlu/pub/dram-access-refresh-parallelization_hpca14.pdf`
7. Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in ISCA, 2014. `https://people.inf.ethz.ch/omutlu/pub/dram-row-hammer_isca14.pdf`
8. Kang et al., "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling", The Memory Forum, 2014, `https://safari.ethz.ch/architecture/fall2022/lib/exe/fetch.php?media=kang-memoryforum14.pdf`
9. Khan et al., "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study", SIGMETRICS, 2014, `https://people.inf.ethz.ch/omutlu/pub/error-mitigation-for-intermittent-dram-failures_sigmetrics14.pdf`
10. Qureshi et al., "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems", DSN 2015, `https://people.inf.ethz.ch/omutlu/pub/avatar-dram-refresh_dsn15.pdf`
11. Meza et al., "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field", DSN, 2015, `https://people.inf.ethz.ch/omutlu/pub/memory-errors-at-facebook_dsn15.pdf`
12. Patel et al., "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions", ISCA 2017, `https://people.inf.ethz.ch/omutlu/pub/reaper-dram-retention-profiling-lpddr4_isca17.pdf`
13. Mutlu et al., "RowHammer: A Retrospective", TCAD 2019, `https://people.inf.ethz.ch/omutlu/pub/RowHammer-Retrospective_ieee_tcad19.pdf`
14. Senol Cali et al., "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis", MICRO 2020, `https://people.inf.ethz.ch/omutlu/pub/GenASM-approximate-string-matching-framework-for-genome-analysis_micro20.pdf`
15. Alser et al., "SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs", Bioinformatics, 2020, `https://people.inf.ethz.ch/omutlu/pub/SneakySnake_UniversalGenomePrealignmentFilter_bioinformatics20.pdf`
16. Frigo et al., "TRRespass: Exploiting the Many Sides of Target Row Refresh", S&P, 2020. `https://people.inf.ethz.ch/omutlu/pub/rowhammer-TRRespass_ieee_security_privacy20.pdf`
17. Patel et al., "HARP: Practically and Effectively Identifying Uncorrectable Errors in Memory Chips That Use On-Die Error-Correcting Codes", MICRO 2021, `https://people.inf.ethz.ch/omutlu/pub/HARP-memory-error-profiling_micro21.pdf`
18. Orosa et al., "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses", MICRO 2021, `https://people.inf.ethz.ch/omutlu/pub/ADeeperLookIntoRowhammer_micro21.pdf`
19. Senol Cali et al., "SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping", ISCA, 2022, `https://people.inf.ethz.ch/omutlu/pub/SeGraM_genomic-sequence-mapping-universal-accelerator_isca22.pdf`
20. Yaglikci et al., "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices", DSN, 2022, `https://people.inf.ethz.ch/omutlu/pub/RowHammerUnderReducedWordlineVoltage_dsn22.pdf`
21. Hassan et al., "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications", MICRO, 2021 `https://people.inf.ethz.ch/omutlu/pub/U-TRR-uncovering-RowHammer-protection-mechanisms_micro21.pdf`
22. Hassan et al., "A Case for Self-Managing DRAM Chips: Improving Performance, Efficiency, Reliability, and Security via Autonomous in-DRAM Maintenance Operations", arXiv preprint, 2022, `https://arxiv.org/pdf/2207.13358.pdf`

2. Genome Analysis [150 points]

2.1. Edit Distance Computation

One of the most fundamental computational steps in most bioinformatics analyses is the detection of the differences between two DNA sequences or two 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) Montag & Donnerstag

Answer:
5 Edits.
1- Substitute character 'M' with 'D' from string "Montag" to be "Dontag"
2- Insert character 'n' into string "Dontag" to be "Donntag"
3- Insert character 'e' into string "Donntag" to be "Donnetag"
4- Insert character 'r' into string "Donnetag" to be "Donnertag"
5- Insert character 's' into string "Donnertag" to be "Donnerstag"

(b) Freitag & Samstag

Answer:
4 Edits.
1- Substitute character 'F' with 'S' from string "Freitag" to be "Sreitag"
2- Substitute character 'r' with 'a' from string "Sreitag" to be "Saeitag"
3- Substitute character 'e' with 'm' from string "Saeitag" to be "Samitag"
4- Substitute character 'i' with 's' from string "Samitag" to be "Samstag"

(c) Donnerstag & "" (where "" is an empty string)

Answer:
10 Edits.
1- Delete character 'D' from string "Donnerstag" to be "onnerstag"
2- Delete character '0' from string "onnerstag" to be "nnerstag"
3- Delete character 'n' from string "nnerstag" to be "nerstag"
4- Delete character 'n' from string "nerstag" to be "erstag"
5- Delete character 'e' from string "erstag" to be "rstag"
6- Delete character 'r' from string "rstag" to be "stag"
7- Delete character 's' from string "stag" to be "tag"
8- Delete character 't' from string "tag" to be "ag"
9- Delete character 'a' from string "ag" to be "g"
10- Delete character 'g' from string "g" to be ""

2.2. Hash Table

During a process called read mapping in genome analysis, each read (i.e., genomic subsequence) is mapped to one or more locations in the reference genome. Potential mapping locations are identified based on the presence of exact short segments (i.e., *k-mers* where *k* is the length of the short segment) from the read sequence, in the reference genome. The locations of the k-mers in the reference genome are usually determined using a *hash table*. Each entry of the hash table stores a key-value pair, where the key is a k-mer and the value is a list of locations at which the k-mer occurs in the reference genome. A challenge in designing such a hash table is deciding which k-mers to use as keys. In this question, you will be exploring two approaches to k-mer selection:

(1) **Non-overlapping 3-mers**: Every *non-overlapping 3-mers* in the reference genome is used as a key in the hash table. For example, the reference segment "ACTCTAAACATT" contains four non-overlapping 3-mers: ACT, CTA, AAC, and ATT. Thus, the hash table would have the following entries: {ACT} → {1}, {CTA} → {4}, {AAC} → {7}, and {ATT} → {10}, where 1, 4, 7 and 10 are the start locations of the non-overlapping 3-mers (keys) in the reference.

(2) **Non-overlapping 3-mer minimizers**: For every non-overlapping 3-mer in the reference genome, the lexicographically minimum 3-mer of it and the two subsequent non-overlapping 3-mers is used as a key in the hash table. For example, the reference segment "ACTCTAAACATT" contains only one non-overlapping 3-mer *minimizers*, AAC. This is because AAC is the lexicographically minimum k-mer among the first three consecutive 3-mers (i.e., ACT, CTA, and AAC), and the second three consecutive 3-mers (i.e., CTA, AAC, and ATT). Thus, the hash table would only have one entry: {AAC} → {7}, where 7 is the start location of the minimizer (key) in the reference.

Suppose that you would like to map a set of reads to the following reference genome. Note that the 3-mers are separated by '_' only to help you identify the 3-mers easily, so you should **not** count them when creating a list of locations for a key.

AGG_ATG_AGA_CAG_ATA_GTA_GAG_ATG_GAG_GTA_ATG_GTA_AAC_ATG_ATA

Answer the following questions based on the information given above:

(a) Please list all {key} → {value} entries in the hash table if we use all **non-overlapping 3-mers** as keys. The order of the entries is **not** important.

Answer:
{AGG} → {1},
{ATG} → {4, 22, 31, 40},
{AGA} → {7},
{CAG} → {10},
{ATA} → {13, 43},
{GTA} → {16, 28, 34},
{GAG} → {19, 25},
{AAC} → {37}.

(b) Please list all {key} $\rightarrow$ {value} entries in the hash table if we use all **non-overlapping 3-mer minimizers** as keys. Please list all the entries of this hash table. The order of the entries is **not** important.

Answer:
{AGA} $\rightarrow$ {7},
{ATA} $\rightarrow$ {13},
{ATG} $\rightarrow$ {22, 31},
{AAC} $\rightarrow$ {37}.

(c) Assume that we calculate the size of the hash table allocated in memory as: $2^{\lceil \log_2 e \rceil} + p$ bytes, where e is the total number of hash table entries and p is the total number of locations stored across all values. Calculate the memory footprint (in bytes) of each of the two hash tables you designed in part (a) and part (b). Show your work. In your opinion what are the advantages and disadvantages of using the non-overlapping 3-mers approach or the non-overlapping 3-mer minimizers approach?

Answer: Based on the content of the hash tables that we constructed in part a, and part b of this question, we find that the size of the hash tables for:
1) Non-overlapping 3-mers: $2^{\lceil \log_2 8 \rceil} + 15 = 23$ bytes
2) Non-overlapping 3-mer minimizers: $2^{\lceil \log_2 4 \rceil} + 5 = 9$ bytes
Therefore the hash table with the non-overlapping 3-mers approach requires $\sim 2.5\times$ more memory than using the non-overlapping 3-mer minimizers approach.
Based on these calculations in the given reference segment, non-overlapping 3-mer minimizers approach consumes less memory as the e and p values cannot be larger than that of the non-overlapping 3-mers approach. However, it is not possible to add either 1) more key values (i.e., k-mers) or 2) more locations in the hash table using non-overlapping 3-mer minimizers approach than using the non-overlapping 3-mers approach as the latter already stores all possible non-overlapping {key} $\rightarrow$ {value} pairs in the hash table.

3. RowHammer [150 points]

3.1. RowHammer Attacks

In order to characterize the vulnerability of your DRAM device to RowHammer attacks, you must be able to induce RowHammer errors. Assume the following about the target system:

- The CPU has a single in-order processor, and does not implement virtual memory.
- The physical memory address is 16 bits.
- The DRAM subsystem consists of two channels, four banks per channel, and 64 rows per bank.
- The memory controller employs open-page policy.
- The DRAM modules and the memory controller do not employ any remapping or scrambling schemes for the physical address.
- All the cells in the DRAM subsystem are equally vulnerable to RowHammer-induced errors.

You implement codes based on instructions shown in Table 1.

Instruction	Description	Functionality
`B LABEL`	Unconditional Branch	`PC = LABEL`
`STORE IMM, Rs`	Store word to memory	`MEM[IMM] = Rs`
`CLFLUSH IMM`	Cache line flush	`Flush cache line containing IMM`

Table 1. Instruction Descriptions.

(a) You run Code 1 below, but you cannot observe any errors in the target system. You figured out that the number of activations is much lower than your expectation. Give reason(s) as to why Code 1 cannot introduce a sufficient amount of activations.

Code 1

```
LOOP:
  STORE 0x8732, R0
  CLFLUSH 0x8732
  B LOOP
```

All of reads in Code 1 are to the same row in DRAM, and the memory controller minimizes the number of DRAM commands by opening and closing the row just once, while issuing many column reads.

(b) You try Codes 2a, 2b, and 2c, but find that *only one of them can induce RowHammer errors* in your DRAM subsystem. Which code segment is the one that can induce RowHammer errors? Justify your answer.

Code 2a

```
LOOP:
   STORE 0x8732, R0
   STORE 0x98CD, R1
   CLFLUSH 0x8732
   CLFLUSH 0x98CD
   B LOOP
```

Code 2b

```
LOOP:
   STORE 0xF1AB, R0
   STORE 0x0054, R1
   CLFLUSH 0xF1AB
   CLFLUSH 0x0054
   B LOOP
```

Code 2c

```
LOOP:
   STORE 0x2B97, R0
   STORE 0xDA68, R1
   CLFLUSH 0x2B97
   CLFLUSH 0xDA68
   B LOOP
```

(a) In order to introduce enough activations, two STORE instructions should access *different rows in the same bank*.
(b) Three code segments are identical except for the memory addresses, so we can assume that only one code segment has two STORE instructions whose destination addresses are assigned to the same bank (but different rows).
(c) Since the DRAM subsystem 1) consists of 8 banks and 2) employs no address remapping/scrambling schemes, two addresses assigned the same bank should satisfy a condition C: they have at least three same bit values at the same position.

Two addresses in each code are:

- **Code 2a**
  $\underline{100}$0 0111 0011 0010 (0x8732)
  $\underline{100}$1 1000 1100 1101 (0x98CD)
- **Code 2b**
  1111 $\underline{000}$1 1010 1011 (0xF1AB)
  0000 $\underline{000}$0 0101 0100 (0x0054)
- **Code 2c**
  0010 $\underline{101}$1 1001 0111 (0x2B97)
  1101 $\underline{101}$0 0110 1000 (0xDA68)

We can observe
(a) Two paired addresses in every code segment have only three same bit values at the same position, i.e., satisfy C.
(b) The position of the same bit values in Code 2b and Code 2c is the same, but different from Code 1. Therefore, if Code 2b can induce RowHammer errors, Code 2c should also be able toinduce errors and Code 2a should not. On the other hand, if Code 2a can induce RowHammer errors, neither Code 2b or Code 2c can induce errors.

Since only one code segment can induce RowHammer errors, Code 2a is the one able to induce RowHammer errors.

3.2. RowHammer Mitigation Mechanisms

To identify a viable RowHammer mitigation mechanism for your system, you compare the two following mitigation mechanisms:

- **Mechanism A.** The memory controller maintains a counter for every row, which increments every time the corresponding row is activated. If the counter value for a row exceeds a threshold value T, the memory controller activates the row's two adjacent rows and resets the counter.
- **Mechanism B.** Each time a row is closed (or precharged), the memory controller flips a biased coin with a probability p of turning up heads, where $p << 1$. If the coin turns up heads, the memory controller activates one of its adjacent rows where either of the two adjacent rows are selected with equal probability ($p/2$).

(a) You set T for Mechanism A to 164 K based on the value of the Maximum Activation Count (MAC, i.e., the maximum number of times a row can be activated without inducing RowHammer errors in its adjacent rows) reported by the DRAM manufacturer. Calculate the number of bits required for counters in a memory controller which manages a single channel, 2 ranks per channel, 8 banks per rank, and 2^{15} rows per bank.

To count values up to 164 K, we need at least 18 bits ($2^{18} > 164K$) per counter.
$\therefore 18\ [\frac{bit}{row}] \times (2^{15})\ [\frac{row}{bank}] \times 16\ [bank] = 9 \times 2^{20}$ bits $= 9$ Mib

(b) How does the answer to (a) change when both the number of rows per bank and the number of banks per chip are doubled?

It will increase to 36 Mib with 2x rows and 2x banks.

(c) You profile the memory access pattern of the target system, and observe that the same pattern repeats exactly every 64 ms (the current refresh interval). Table 2 shows the number of activations for each row within a 64-ms time interval in a descending order. Given values $T = 164$ K for Mechanism A and $p = 0.001$ for Mechanism B, calculate the expected number of additional activations within a 64-ms time interval under each technique.

Row Index	# of ACTs
0x7332F	73 K
0x1802C	64 K
0x03F05	32 K
0x5FF02	10 K
...	...
Total	480 K

Table 2. Number of Activations for Each Row.

Mechanism A introduces no additional row activation, since no row is activated more than the threshold.

On the other hand, for Mechanism B, the number of additional activations can be modeled as a random variable X that is binomially-distributed with parameters $B(480,000,\ p)$.
$\therefore$ # of additional activations $= E(X) = 480,000 \times 0.001 = 480$

(d) How does the answer to (c) change when both the number of rows per bank and the number of banks per chip are doubled? Assume that the memory access pattern does not change.

The performance overhead only depends on the access pattern, so it will not change.

(e) What is the *common challenge* to implement the above mechanisms in the commodity systems?

This question is open ended. There could be other possible right answers.

Both Mechanisms require the information about exact mappings between row address and physical row, which is unlikely disclosed by manufacturers.

(f) How can you address the common challenge?

This question is open ended. There could be other possible right answers.

Possible solutions

- Reverse engineering
- Implementing the techniques inside DRAM modules where the mapping function is managed.
- For counter-based approach, we can block future activations on a row (instead of refreshing its adjacent rows), if the counter value of a row exceeds the threshold value.

4. VRT Mitigation Techniques [150 points]

DRAM cells are susceptible to retention errors, which come in two varieties:

1. `Static`: cells that have one fixed retention time, beyond which they will fail.
2. `Variable Retention Time (VRT)`: cells that have a dynamically changing retention time.

`VRT` cells are particularly problematic because they change between different retention times unpredictably at runtime.

4.1. Part I: Per-Row Techniques

You will implement an idealized version of RAIDR[1] that extends the fixed 64 ms DRAM refresh window in order to improve DRAM performance and save energy. Your design provides three retention time "buckets" for refreshing DRAM rows, labeled `A`, `B`, and `C`. Each bucket refreshes its corresponding rows with the refresh windows shown in Table 3.

Bucket Label	Refresh Window for Rows in the Bucket
`A`	64 ms
`B`	128 ms
`C`	256 ms

Table 3. RAIDR retention time buckets.

To assign DRAM rows to buckets, you measure the retention time of each row as the worst-case cell in the row. The profile in Figure 1 shows how rows are split into buckets. Letters `a`..`f` represent the **proportion** of all DRAM rows with retention times within each range. For example, if `a` = 0.1, 10% of all DRAM rows have worst-case retention times within the range shown by `a`. Note that `static` rows are fixed within a bucket, but `VRT` rows span more than one bucket.

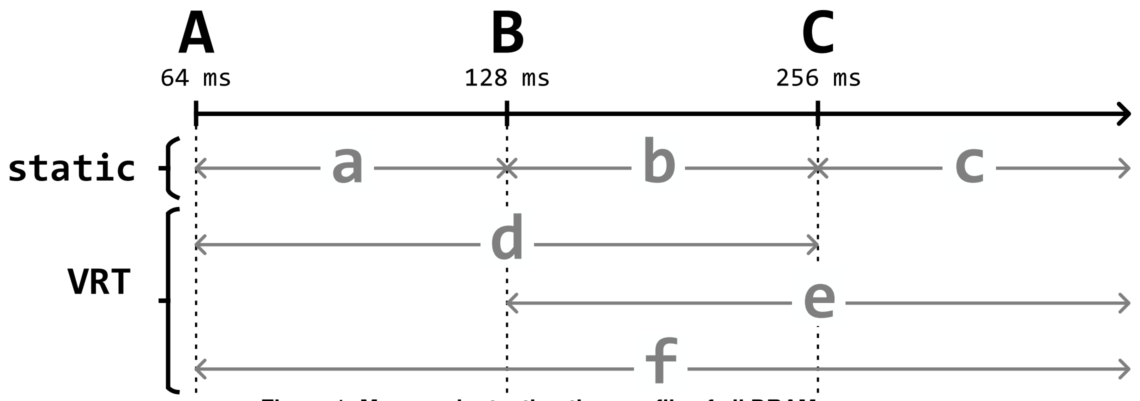

Figure 1. Measured retention time profile of all DRAM rows.

[1]Liu, J., Jaiyen, B., Veras, R. and Mutlu, O., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA, 2012.

(a) Compute the proportion of all DRAM rows that fall in each of the four RAIDR buckets. Minimize the total number of refresh operations required. Leave your answer in terms of the variables `a`, `b`, etc.

Ignoring VRT rows
i.e., assume `d`= 0, `e`= 0, and `f`= 0

Bucket	Proportion of all rows
`A`	a
`B`	b
`C`	c

Including VRT rows

Bucket	Proportion of all rows
`A`	a+d+f
`B`	b+e
`C`	c

(b) Considering the VRT rows, when do unnecessary refresh operations occur?

Whenever a VRT row is in a higher retention time state relative to its assigned bucket.

Specifically, rows being refreshed unnecessarily:

- `d`: whenever it is in bucket B
- `e`: whenever it is in bucket C
- `f`: whenever it is in bucket B or C

(c) For simplicity, assume that VRT rows can <u>only</u> transition between buckets every 256 ms, perfectly aligned with refresh operations of bucket `C`. Then, assume that, on average, all VRT rows spend equal time in each bucket that their retention time spans (e.g., rows `d` spend equal time in buckets `A` and `B`). On average, what is the proportion of all refresh commands that are unnecessary? <u>Hint: consider the number of refreshes issued in $3 \times 512 = 1536$ ms.</u>

Without loss of generality, let us assume a time period of 3×512 ms. Within this period, per-row refresh counts for each row range are:

- `a`: $3 \times \frac{512\text{ ms}}{64\text{ ms}} = 3 \times 8$
- `b`: $3 \times \frac{512\text{ ms}}{128\text{ ms}} = 3 \times 4$
- `c`: $3 \times \frac{512\text{ ms}}{256\text{ ms}} = 3 \times 2$
- `d`: $3 \times \frac{512\text{ ms}}{64\text{ ms}} = 3 \times 8$
- `e`: $3 \times \frac{512\text{ ms}}{128\text{ ms}} = 3 \times 4$
- `f`: $3 \times \frac{512\text{ ms}}{64\text{ ms}} = 3 \times 8$

Let R be the total number of row refresh operations in 3×512 ms, normalized by the number of DRAM rows:
$\frac{R}{3} = 8a + 4b + 2c + 8d + 4e + 8f$
$R = 24a + 12b + 6c + 24d + 12e + 24f$

Based on spending equal time in each bucket, <u>necessary</u> refresh counts are:

- `a`: all
- `b`: all
- `c`: all
- `d`: 3×256 ms in each of A and B: $3 \times (4 + 2) = 3 \times 6$
- `e`: 3×256 ms in each of B and C: $3 \times (2 + 1) = 3 \times 3$
- `f`: 1×512 ms in each of A, B, and C: $1 \times (8 + 4 + 2) = 1 \times 14$

Therefore, average unnecessary refresh counts U in 3×512 ms are:
$U = R - (24a + 12b + 6c + 3 \times 6d + 3 \times 3e + 1 \times 14f)$
$U = R - (24a + 12b + 6c + 18d + 9e + 14f)$
$U = 6d + 3e + 10f$

The final proportion of unnecessary refresh commands is:

$$\boxed{\frac{U}{R} = \frac{6d + 3e + 10f}{24a + 12b + 6c + 24d + 12e + 24f}}.$$

4.2. Part II: Per-Cell Techniques

A different approach to handling VRT cells is to treat them at the cell granularity rather than the row granularity.

Assume that proportion p of all DRAM cells are susceptible to VRT and might fail. If read operations target random addresses, we can interpret p as the probability that a single cell might fail due to VRT. To prevent errors, we implement a single-error correcting error-correcting code (ECC) with 136-bit ECC words. This code corrects one error within every 136 bits (128 data bits + 8 metadata bits), so a single read operation provides 128 bits of data.

(d) What is the probability that a 136-bit word contains no errors after ECC operation? Express your answer in terms of p.

The probability of no error is the probability of 0 or 1 errors in the 136-bit word:

$$\begin{aligned} P[0 \text{ error}] &= P[\text{no error in bit}[0]] \wedge P[\text{no error in bit}[1]] \wedge ... \wedge P[\text{no error in bit}[135]] && (1)\\ &= (1-p)^{136} && (2) \end{aligned}$$

$$\begin{aligned} P[1 \text{ error}] &= \binom{136}{1} P[\text{error in bit}[0]] \wedge P[\text{no error in bit}[1]] \wedge ... \wedge P[\text{no error in bit}[135]] && (3)\\ &= 136p(1-p)^{135} && (4) \end{aligned}$$

$P[0 \text{ or } 1 \text{ error}] = P[0 \text{ error}] + P[1 \text{ error}] = \boxed{(1-p)^{136} + 136p(1-p)^{135}}$

Note this is the first two terms of the binomial distribution $B(136, 0|p) + B(136, 1|p)$.

(e) Given a DRAM module with bandwidth 16 GB/s, what is the worst-case probability of observing at least one uncorrectable error after 2^{25} seconds ($\approx$ 1 year) of operation? You may leave your answer in terms of your answer for part (d).

Let d be the response to part (d), i.e., the probability of no error in a 136-bit word after ECC operation. Over 2^{25} seconds, we read $16 * 2^{25}$ GB $= 2^{29}$ GB $= 2^{62}$ bits $= 2^{55}$ ECC words.

The probability of observing at least one error is then $1 - P[\text{noerror}] = \boxed{1 - d^{2^{55}}}$

5. SARP: Subarray Access-Refresh Parallelization [150 points]

A DRAM bank consists of multiple subarrays, each of which has a single local row buffer. However, only one of these local row buffers may be connected to the global row buffer at any given time. This means that only one subarray may be active at a time.

Chang et al. explore several ways to reduce the performance overhead of refresh operations in their paper.[2]

They propose (SARP), which refreshes idle subarrays while servicing the active one in order to overlap the refresh latency with accesses. In this question, you are asked to evaluate a system that uses SARP. Consider a baseline, where:

- Each bank has 8 subarrays.
- Each row needs to be refreshed every 64 ms, and each bank is unavailable for 0.2 ms during refresh.
- Your application accesses each subarray sequentially in a round-robin fashion such that each subarray receives its next request only after all other subarrays in the same bank have received exactly one request.
- Each memory request takes 50 ns to serve, and each bank receives a request every 500 ns.

Is it possible to completely hide the overhead of refresh? Explain. If no, can you enable achieve this by changing the subarray-bank organization? How?

Note: Assume that the access pattern argument always holds, independent from the subarray organization.

No!

Each subarray will receive a request every $500ns \times 8 = 4us$.
As each request takes 50ns to finish, each subarray will be idle for 3.95us.
Refreshing a bank takes 200us. Then refreshing a subarray costs $200us/8 = 25us$.
As the idle time of a subarray is not as large as the refresh latency, the refresh time cannot be completely hidden.

How to make it possible?
There are two obstacles:

- A subarray receives requests too frequently ($0.5 \times N_{subarray}$).
- Refreshing a subarray takes too long ($200/N_{subarray}$).

If we reduce the subarray size and have more subarrays in a bank, then change the decode logic accordingly to sustain the round-robin access pattern across subarrays, we can make these parameters to match.

We want to satisfy:
$0.5 \times N_{subarray} - 0.05 \geq 200/N_{subarray}$
$0.5 \times N_{subarray} - 0.05 - 200/N_{subarray} \geq 0$
$0.5 \times N_{subarray}^2 - 0.05 \times N_{subarray} - 200 \geq 0$
$N_{subarray}^2 - 0.1 \times N_{subarray} - 400 \geq 0$

Say $N_{subarray} = 16$: $256 - 1.6 - 400 \geq 0$ is false.
Say $N_{subarray} = 32$: $1024 - 3.2 - 400 \geq 0$ is true.

Increasing the number of subarrays to 32 enables completely overlapping the refresh latency.

[2]Chang, K. et al., *"Improving DRAM performance by Parallelizing Refreshes with Accesses."* In Proceedings of International Symposium on High-Performance Computer Architecture (HPCA), 2014.