

## HW 3: GENOME ANALYSIS, ROWCLONE, AND LOW-LATENCY MEMORY(SOLUTIONS)

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: Saturday, Oct 24, 2020

Due: **Monday, Nov 09, 2020**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to <https://safari.ethz.ch/review/architecture20/>. 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.inf.ethz.ch](mailto: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-app2.1et.ethz.ch/course/view.php?id=13549>) as a single PDF file.

**1. Critical Paper Reviews [500 points]**

Please read the guidelines for reviewing papers and check the sample reviews. We also assign you two **required reading** for this homework. 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. If you review a paper other than the REQUIRED papers, you will receive 250 BONUS points on top of 500 points you may get from paper reviews (i.e., each additional submission is worth 250 BONUS points with a possibility to get upto 2000 points).



Guidelines



Sample reviews



Required Reading 1



Required Reading 2

Write an approximately one-page critical review for the following required readings (i.e., Paper #1, and Paper #2) **and** earn bonus points for the remaining 7 papers (i.e., papers from #3 to #9). 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.

1. (REQUIRED) Alser et al., "Accelerating Genome Analysis: A Primer on an Ongoing Journey," in Proceedings of the 53rd International Symposium on Microarchitecture (MICRO), 2020. [https://people.inf.ethz.ch/omutlu/pub/AcceleratingGenomeAnalysis\\_ieemicro20.pdf](https://people.inf.ethz.ch/omutlu/pub/AcceleratingGenomeAnalysis_ieemicro20.pdf)
2. (REQUIRED) Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), 2013. [https://people.inf.ethz.ch/omutlu/pub/tldram\\_hpca13.pdf](https://people.inf.ethz.ch/omutlu/pub/tldram_hpca13.pdf)
3. Kim et al., "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in Proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA), 2019. [https://people.inf.ethz.ch/omutlu/pub/drang-dram-latency-based-true-random-number-generator\\_hpca19.pdf](https://people.inf.ethz.ch/omutlu/pub/drang-dram-latency-based-true-random-number-generator_hpca19.pdf)
4. Hassan et al., "SoftMC: A flexible and practical open-source infrastructure for enabling experimental DRAM studies," in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017. [https://research.ece.cmu.edu/safari/pubs/softMC\\_hpca17.pdf](https://research.ece.cmu.edu/safari/pubs/softMC_hpca17.pdf)

5. Cali et al., “*GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis*,” in Proceedings of the 53rd International Symposium on Microarchitecture (MICRO), 2020. [http://users.ece.cmu.edu/~saugatag/papers/20micro\\_genasm.pdf](http://users.ece.cmu.edu/~saugatag/papers/20micro_genasm.pdf)
6. Kanellopoulos et al., “*SMASH: Co-designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations*,” in Proceedings of the 52nd International Symposium on Microarchitecture (MICRO), 2019. [https://people.inf.ethz.ch/omutlu/pub/SMASH-sparse-matrix-software-hardware-acceleration\\_micro19.pdf](https://people.inf.ethz.ch/omutlu/pub/SMASH-sparse-matrix-software-hardware-acceleration_micro19.pdf)
7. Koppula et al., “*EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM*,” in Proceedings of the 52nd International Symposium on Microarchitecture (MICRO), 2019. [https://people.inf.ethz.ch/omutlu/pub/EDEN-efficient-DNN-inference-with-approximate-memory\\_micro19.pdf](https://people.inf.ethz.ch/omutlu/pub/EDEN-efficient-DNN-inference-with-approximate-memory_micro19.pdf)
8. Alser et al., “*GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping*,” in Bioinformatics, 2017. [https://people.inf.ethz.ch/omutlu/pub/gatekeeper\\_FPGA-genome-prealignment-accelerator\\_bionformatics17.pdf](https://people.inf.ethz.ch/omutlu/pub/gatekeeper_FPGA-genome-prealignment-accelerator_bionformatics17.pdf)
9. Kim et al., “*A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM*,” in Proceedings of the 39th International Symposium on Computer Architecture (ISCA), 2017. [https://people.inf.ethz.ch/omutlu/pub/salp-dram\\_isca12.pdf](https://people.inf.ethz.ch/omutlu/pub/salp-dram_isca12.pdf)
10. Kang et al., “*Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling*,” in The Memory Forum, 2014. <https://safari.ethz.ch/architecture/fall12019/lib/exe/fetch.php?media=kang-memoryforum14.pdf>

## 2. Genome Analysis I [150 points]

### 2.1. Edit Distance

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) 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 'o' 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. Read Mapping

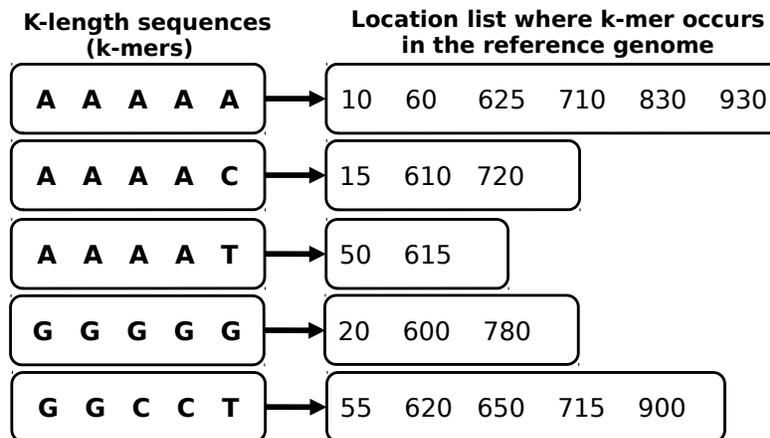
During a process called read mapping in genome analysis, each genomic read is mapped onto one or more possible locations in the reference genome based on the similarity between the read and the reference genome segment at that location. Suppose that you would like to map the following reads to the human 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) How many times in total the edit distance function,  $edit\_distance()$ , will be invoked using the 3-step hash-based mapping method described above?

**Answer:** 51 times.

**Explanation:** the number of times that the  $edit\_distance()$  function will be invoked is equal to the total number of occurrences of all k-mers that exist in both the hash table and the queried read. The frequency of each k-mer is given below:

read 1 = AAAAA_AAAAC_GGGGG	6 + 3 + 3
read 2 = AAAAT_GGCCT_AAAAA	2 + 5 + 6
read 3 = AAAAT_AGCGG_GCGCT	2 + 0 + 0
read 4 = AAAAC_AAAAT_GGCCT	3 + 2 + 5
read 5 = AAAAA_GGCCT_AAAAC	6 + 5 + 3

- (b) 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?

**Answer:** 5 times.

**Explanation:** If all 3 5-mers are adjacent in the reference genome then the  $edit\_distance()$  function will be invoked once for these three locations. The locations of the adjacent k-mers for each read are listed below:

read 1 = AAAAA_AAAAC_GGGGG	10, 15, 20
read 2 = AAAAT_GGCCT_AAAAA	50, 55, 60 Also 615, 620, 625
read 3 = AAAAT_AGCGG_GCGCT	2nd and 3rd k-mers do not exist
read 4 = AAAAC_AAAAT_GGCCT	610, 615, 620
read 5 = AAAAA_GGCCT_AAAAC	710, 715, 720.

- (c) 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?

**Answer:** 18 times.

**Explanation:** the  $edit\_distance()$  function will be invoked for each k-mer that exists no more than three times in the hash table. Below are the frequency of each k-mer that exists no more than 3 times in the hash table:

read 1 = AAAAA_AAAAC_GGGGG	0 + 3 + 3
read 2 = AAAAT_GGCCT_AAAAA	2 + 0 + 0
read 3 = AAAAT_AGCGG_GCGCT	2 + 0 + 0
read 4 = AAAAC_AAAAT_GGCCT	3 + 2 + 0
read 5 = AAAAA_GGCCT_AAAAC	0 + 0 + 3

### 3. Genome Analysis II [150 points]

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 based on the similarity between the read and the reference genome segment at that location. 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, as it affects the size of the hash table and the number of potential mapping locations, which affect the execution time of read mapping. In this question, you will be exploring the trade-offs between two strategies of *k*-mer selection:

- (1) **Non-overlapping 4-mers:** Every *non-overlapping 4-mers* in the reference genome is used as a key in the hash table. For example, the reference AAAATTCA contains only two non-overlapping 4-mers: AAAA and TTCA. Thus, the hash table would have the following entries: {AAAA} → {1} and {TTCA} → {5}, where 1 and 5 are the start locations of the non-overlapping 4-mers (keys) in the reference.
- (2) **Non-overlapping 4-mer minimizers:** For every non-overlapping 4-mer in the reference genome, the lexicographically minimum 4-mer of it and the two subsequent non-overlapping 4-mers is used as a key in the hash table. For example, the segment AAAATTCAACGGGCAG contains only two non-overlapping 4-mer minimizers, AAAA and ACGG. This is because AAAA is the lexicographically minimum *k*-mer among the first three consecutive *k*-mers (i.e., AAAA, TTCA, ACGG), and ACGG is the lexicographically minimum *k*-mer among the next three consecutive *k*-mers (i.e., TTCA, ACGG, and GCAG). Thus, the hash table would have the following entries: {AAAA} → {1} and {ACGG} → {9}, where 1 and 9 are the start locations of the minimizers (keys) in the reference.

Suppose that you would like to map a set of reads to the following reference genome. Note that the 4-mers are separated by ‘\_’ *only* to help you identify the 4-mers easily, so you should **not** count them when creating a list of locations for a key.

AAAA\_ATAC\_TGAT\_CCTT\_ATAC\_GTTG\_TAAG\_GTTT\_CAAA\_GTTG\_ATAC\_TAAG\_TGAT

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 4-mers** as keys? The order of the entries is **not** important.

{AAAA} → {1},  
{ATAC} → {5, 17, 41},  
{TGAT} → {9, 49},  
{CCTT} → {13},  
{GTTG} → {21, 37},  
{TAAG} → {25, 45},  
{GTTT} → {29},  
{CAAA} → {33}

- (b) Please list all  $\{\text{key}\} \rightarrow \{\text{value}\}$  entries in the hash table if we use all **non-overlapping 4-mer minimizers** as keys? Please list all the entries of this hash table. The order of the entries is **not** important.

$\{\text{AAAA}\} \rightarrow \{1\},$   
 $\{\text{ATAC}\} \rightarrow \{5, 17, 41\},$   
 $\{\text{GTTG}\} \rightarrow \{21\},$   
 $\{\text{CAAA}\} \rightarrow \{33\}$

- (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 (a) and (b). Show your work.

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 4-mers:  $2^{\lceil \log_2 8 \rceil} + 13 = 21$  bytes

2) Non-overlapping 4-mer minimizers:  $2^{\lceil \log_2 4 \rceil} + 6 = 10$  bytes

Therefore the hash table with the non-overlapping 4-mers strategy requires  $\sim 2.1\times$  more memory than using the non-overlapping 4-mer minimizers. We also make the observation that it is **not** possible to add either 1) more key values (i.e.,  $k$ -mers) or 2) more locations in the hash table using the non-overlapping 4-mer minimizers strategy than using the non-overlapping 4-mers as the latter already stores all possible non-overlapping  $\{\text{key}\} \rightarrow \{\text{value}\}$  pairs in the hash table.

Based on these observations and calculations, the non-overlapping 4-mer minimizers strategy consumes less memory as the  $k$  and  $p$  values cannot be larger than that of the non-overlapping 4-mers strategy.

- (d) Now assume we can query the hash table in  $\log_2 e$  cycles, where  $e$  is the total number of entries in the hash table. A read mapper queries the hash table using the *first 4-mer of the read* and calculates the *edit distance* between the read and the reference segment at *each* location returned by the hash table. Calculating the edit distance takes  $l^2$  cycles where  $l$  is the length of the read. If the edit distance between the read and a segment in the reference is higher than a certain threshold, the read mapper discards the location. We refer to cycles spent calculating the edit distance for segments at discarded locations as *wasted cycles*. When we profile read mapping with *non-overlapping 4-mers* and *non-overlapping 4-mer minimizers* strategies, we find the  $\frac{\text{wasted cycles}}{\text{total cycles}}$  ratios to be 0.9 and 0.8, respectively. Assume that we want to align the read: GTTGACCAATGA to the reference genome above. What are the *wasted cycles* when aligning the read using 1) *non-overlapping 4-mers* and 2) *non-overlapping 4-mer minimizers* strategies? Please show your work.

We find from the question that each edit distance calculation takes  $(\log_2 e) * n * (l^2)$  cycles where  $n$  is the number of locations that the queried 4-mer exists in the reference genome,  $(\log_2 e)$  is the cost to look up for the presence of the first 4 characters (i.e., 4-mer) of a read from the hash table, and  $l^2$  is the cost of calculating the edit distance of a read.

Based on the content of the hash tables that we constructed in part *a* and part *b* of this question, we find that:

- The number of keys in the hash tables are  $e = 8$  and  $e = 4$  when using 1) *non-overlapping 4-mers* and 2) *non-overlapping 4-mer minimizers* strategies, respectively.
- The size of the reads are 12 characters (i.e.,  $l = 12$ )
- The first four characters of the read are: "GTTG". Thus, we find  $n = 2$  locations (i.e.,  $\{\text{GTTG}\} \rightarrow \{21, 37\}$ ) and  $n = 1$  location (i.e.,  $\{\text{GTTG}\} \rightarrow \{21\}$ ) when we query the hash tables that store 1) *non-overlapping 4-mers* and 2) *non-overlapping 4-mer minimizers* with "GTTG", respectively.

Based on these values we calculate the *overall* and *wasted cycles* cycles of both approaches as follows:

1) Non-overlapping 4-mers:

$$\text{Total cycles} = (\log_2 8) + 2 * (12^2) = 291$$

$$\text{Wasted cycles} = \text{Total cycles} * 0.9 = 291 * 0.9 = 261.9$$

2) Non-overlapping 4-mer minimizers:

$$\text{Total cycles} = (\log_2 4) + (12^2) = 146$$

$$\text{Wasted cycles} = \text{Total cycles} * 0.8 = 146 * 0.8 = 116.8$$

#### 4. RowClone [150 points]

Recall that the RowClone<sup>1</sup> idea presented in lecture performs the bulk copy or initialization of a page (or any arbitrary sized memory region) completely within DRAM, with support from the memory controller.

Suppose we have a cache-coherent system with six cores, a 32 KB L1 cache in each core, a 1MB private L2 cache per core, a 16MB shared L3 cache across all cores, and a single shared DRAM controller across all cores. The system supports RowClone as described in class. Physical page size is 8KB. MESI protocol is employed to keep the caches coherent.

Suppose Core 1 sends a request to the memory controller to Copy Physical Page 10 to Physical Page 12 via RowClone. Assume the two pages reside in the same DRAM subarray and the copy can be done with two consecutive ACTivate commands as a result.

- (a) To maintain correctness of data in such a system, what should the memory controller do before performing the RowClone request? Explain step by step. Be precise.

Step 1:

The memory controller writes back any dirty cache line from the source region (Page 10).

Step 2:

The memory controller invalidates any cache line (clean or dirty) from the destination region (Page 12) that is cached in the on-chip caches.

- (b) How much data transfer is eliminated from the main memory data bus with this particular copy operation? Justify your answer and state your assumptions.

Amount of data eliminated:

16 KB

Justification:

---

<sup>1</sup>Seshadri et al., "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization." In Proceedings of the 46th International Symposium on Microarchitecture (MICRO), 2013.

## 5. Tiered-difficulty [150 points]

Recall from your required reading on Tiered-Latency DRAM that there is a near and far segment, each containing some number of rows. Assume a very simplified memory model where there is just one bank and there are two rows in the near segment and four rows in the far segment. The time to activate and precharge a row is 25ns in the near segment and 50ns in the far segment. The time from start of activation to reading data is 10ns in the near segment and 15ns in the far segment. All other timings are negligible for this problem. Given the following memory request stream, determine the optimal assignment (minimize average latency of requests) of rows in the near and far segment (assume a fixed mapping where rows cannot migrate, a closed-row policy, and the far segment is inclusive).

```
time 0ns : row 0 read
time 10ns : row 1 read
time 100ns: row 2 read
time 105ns: row 1 read
time 200ns: row 3 read
time 300ns: row 1 read
```

(a) What rows would you place in near segment? Hint: draw a timeline.

Rows 0 and 2.

**Explanation.** If you were to map 0 and 2 (this is the answer) to near segment:

```
row 0: activated at time = 0
row 0: read at time = 10 (10ns latency)
row 1: activated at time = 25
row 1: read at time = 40 (30ns latency)
row 2: activated at time = 100
row 2: read at time = 110 (10ns latency)
row 1: activated at time = 125
row 1: read at time = 140 (35ns latency)
row 3: activated at time = 200
row 3: read at time = 215 (15ns latency)
row 1: activated at time = 300
row 1: read at time = 315 (15 ns latency)
total latency is 115ns.
```

If you were to map 1 and 2 (an example incorrect answer) to near segment:

```
row 0: activated at time = 0
row 0: read at time = 15 (15ns latency)
row 1: activated at time = 50
row 1: read at time = 60 (50ns latency)
row 2: activated at time = 100
row 2: read at time = 110 (10ns latency)
row 1: activated at time = 125
row 1: read at time = 135 (30ns latency)
row 3: activated at time = 200
row 3: read at time = 215 (15ns latency)
row 1: activated at time = 300
row 1: read at time = 310 (10 ns latency)
total latency is 130ns.
```

(b) What rows would you place in far segment?

Rows 1 and 3 (also rows 0 and 2 since inclusive).

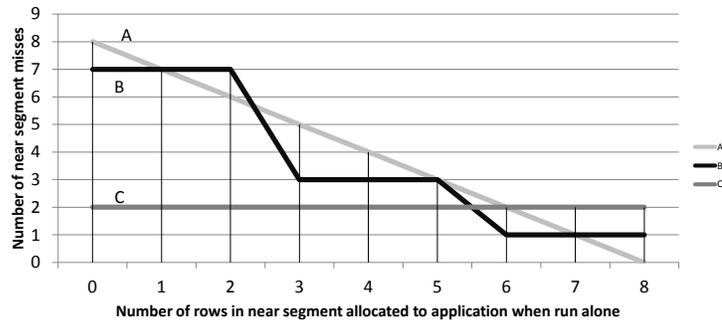
(c) In 15 words or less, describe the insight in your mapping?

See TL-DRAM's WMC policy – the first access in near simultaneous requests causes the second to wait activation + precharge time. minimizing this wait by caching first row in near segment is better than caching second row in near segment (this decreases only time to read from start of activation), even if second row is accessed more frequently (see example above)

(d) Assume now that the mapping is dynamic. What are the tradeoffs of an exclusive design vs. an inclusive design? Name one advantage and one disadvantage for each.

Exclusive requires swapping, but can use nearly full capacity of DRAM. Inclusive, the opposite.

(e) Assume now that there are eight (8) rows in the near segment. Below is a plot showing the number of misses to the near segment for three applications (A, B, and C) when run alone with the specified number of rows allocated to the application in the near segment. This is similar to the plots you saw in your Utility-Based Cache Partitioning reading except for TL-DRAM instead of a cache. Determine the optimal static partitioning of the near segment when all three of these applications are run together on the system. In other words, how many rows would you allocate for each application? Hint: this should sum to eight. Optimal for this problem is defined as minimizing total misses across all applications.



(1) How many near segment rows would you allocate to A?

5

(2) How many near segment rows would you allocate to B?

3

(3) How many near segment rows would you allocate to C?

0

## 6. Low-Latency DRAM [150 points]

In class, we have seen the idea of Tiered-Latency DRAM (TL-DRAM). Recall that in TL-DRAM, each bitline of a subarray is segmented into two portions by adding isolation transistors in between, creating two segments on the bitline: the *near segment* and the *far segment*. The near segment is close to the sense amplifiers whereas the far segment is far away.

- (a) Why is accessing a row in the near segment faster in TL-DRAM compared to a commodity DRAM chip?

The bitline is shorter (thus lower capacitance) than commodity DRAM.

- (b) Why is accessing a row in the far segment slower in TL-DRAM compared to a commodity DRAM chip?

There is an additional capacitance because of the isolation transistor.

Now, assume that:

- We have a system that uses the near segment as a cache to the far segment, and the far segment contains main memory locations.
- The near segment is not visible to software and the rows that are cached in it are completely managed by the memory controller.
- The far segment is inclusive of the near segment.
- In each subarray, the far segment contains 496 rows whereas the total number of rows in the subarray is 512.

- (c) What is the capacity loss in main memory size when we use TL-DRAM as opposed to commodity DRAM with the same number of total DRAM rows? Express this as a fraction of total memory capacity lost (no need to simplify the fraction).

$\frac{16}{512}$

- (d) What is the tag store size that needs to be maintained on a per subarray basis in the DRAM controller if the near segment is used as a fully-associative write-back cache? Assume the replacement algorithm is *Most Recently Used* (MRU) and use the **minimum number of bits** possible to implement the replacement algorithm. Show your work.

180 bits

**Explanation:** 1 valid bit, 1 dirty bit, 9 bits tags, 16 different entries (one for each row)  
Then we need 4 bits to point where the MRU entry is  
Total size =  $16 \cdot 11 + 4 = 176 + 4 = 180$  bits

Now assume near segment and far segment are *exclusive*. In other words, both contain memory rows, and a memory row can only be in one of the segments. When a memory row in the far segment is referenced, it is brought into the near segment by exchanging the MRU row in the near segment with the row in the far segment. Note that a row can end up in a *different* location in the far segment after being moved to the near segment and then back to the far segment.

- (e) When the near segment is used as an *exclusive* cache to the far segment, what is the capacity loss in main memory size when we use TL-DRAM as opposed to commodity DRAM with the same number of total DRAM rows? Express this as a fraction of total memory capacity lost (no need to simplify the fraction).

1/512 (or 0%)

**Explanation:** In this question, we accept 0% as an answer, but the correct answer is 1/512  
Because you need a dummy row to perform the swap.

- (f) What is the tag store size that needs to be maintained on a per subarray basis in the DRAM controller if the near segment is used as an *exclusive* fully-associative write-back cache? Assume the replacement algorithm is MRU and use the **minimum number of bits** possible to implement the replacement algorithm. Show your work.

4612 bits

**Explanation:** Near segment: (9 bits tags) \* 16row = 144 + 4 MRU bits = 148 bits  
A row swap can lead to any physical pages in the subarray getting mapped to any row in the subarray. This means we need the tag bits in each of the far segment rows.  
Far segment: 9 bits tags \* 496 rows = 4464 bits  
Total size = 4612 bits

(g) Assume the near and far segments are visible to the operating system (OS) and the OS can allocate physical pages in either of the segments. Answer the following questions as *True* or *False* and provide explanation to your answer.

- The OS *cannot* manage the near segment as a cache at granularity smaller than the physical page granularity.

True.

**Explanation:**

The statement is true as the smallest size that the OS can allocate from the physical memory is the page size of the system. Mapping virtual addresses to physical addresses at smaller chunks would require additional metadata to be stored in the page table, which is essentially equivalent of having a system with small pages.

- If the near segment is used as an OS-managed cache to store frequently-accessed pages, the cache can *only* be exclusive.

False.

**Explanation:**

The OS-managed cache can either be exclusive or inclusive. For the exclusive option, the OS can *swap* a frequently-accessed page in the far segment with a page from the near segment. In contrast, to use the near segment as inclusive cache, the OS can copy a frequently-accessed page in the far segment to the near segment.

- There is zero memory capacity loss when the near segment is used as OS-managed cache.

False.

**Explanation:**

The statement is incorrect when the OS uses the near segment as an inclusive cache since a virtual page can be mapped to a physical page from both near and far segments at the same time. In addition, the OS needs to reserve some memory capacity to maintain metadata for keeping track of the physical page from the far segment that a virtual page was mapped to before being copied to the near segment.

Using the near segment as exclusive cache would also cause memory capacity loss due to 1) metadata that will be potentially needed for the page replacement policy and 2) an empty page that is needed to swap the data of a page from far and near segments.

- An OS-managed near segment cache does *not* incur *any* tag store overhead in the memory controller.

True.

**Explanation:** This statement is true assuming purely OS-managed cache design. However, it is also possible that the memory controller provides architectural support to the OS to store tags efficiently when using the near segment as an inclusive cache.