7 In-DRAM Bitmap Indices [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 ands 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?

$TotalRows = \lceil \frac{u}{8 \times 8k} \rceil \times w.$

Explanation:
The u users are spread across a number of subarrays:
$NumSubarrays = \lceil \frac{u}{8 \times 8k} \rceil.$

Thus, the total number of rows is:
$TotalRows = \lceil \frac{u}{8 \times 8k} \rceil \times w.$

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

$Thr_{Ambit} = \frac{u}{\lceil \frac{u}{8 \times 8k} \rceil \times w \times (t_{rc} + t_{and}) + \frac{u}{X \times 8}}$ users/second.

Explanation:
First, let us calculate the total time for all bulk and operations. We should add t_{rc} and t_{and} for all rows:
$t_{and-total} = \lceil \frac{u}{8 \times 8k} \rceil \times w \times (t_{rc} + t_{and})$ seconds.

Then, we calculate the time needed to compute the bit count on CPU:
$t_{bitcount} = \frac{\frac{u}{8}}{X} = \frac{u}{X \times 8}$ seconds.

Thus, the throughput in users/s is:
$Thr_{Ambit} = \frac{u}{t_{and-total} + t_{bitcount}}$ users/second.

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

$Thr_{CPU} = \frac{X \times 8}{w}$ users/second.

Explanation:
We calculate the time needed to bring all users and weeks to the CPU:
$t_{CPU} = \frac{\frac{u \times w}{8}}{X} = \frac{u \times w}{X \times 8}$ seconds.

Thus, the throughput in users/s is:
$Thr_{CPU} = \frac{u}{t_{CPU}} = \frac{X \times 8}{w}$ users/second.

(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.

$w < \frac{1}{1 - \frac{X}{8k} \times (t_{rc} + t_{and})}$.

Explanation:
We compare t_{CPU} with $t_{and-total} + t_{bitcount}$:
$t_{CPU} < t_{and-total} + t_{bitcount}$;

$\frac{u \times w}{X \times 8} < \frac{u}{8 \times 8k} \times w \times (t_{rc} + t_{and}) + \frac{u}{X \times 8}$;

$w < \frac{1}{1 - \frac{X}{8k} \times (t_{rc} + t_{and})}$.