Assignment IV – Rowhammer

Team quench-quokka
Mihai-George Licu
Changling Wang

Rowhammer - Overview

- DRAM is organized into a grid of rows and columns, grouped into multiple banks within a chip
- Repeatedly accessing (hammering) a row causes bit flips in adjacent rows
- Bit flips can change stored data without direct access

Task 1 & 2

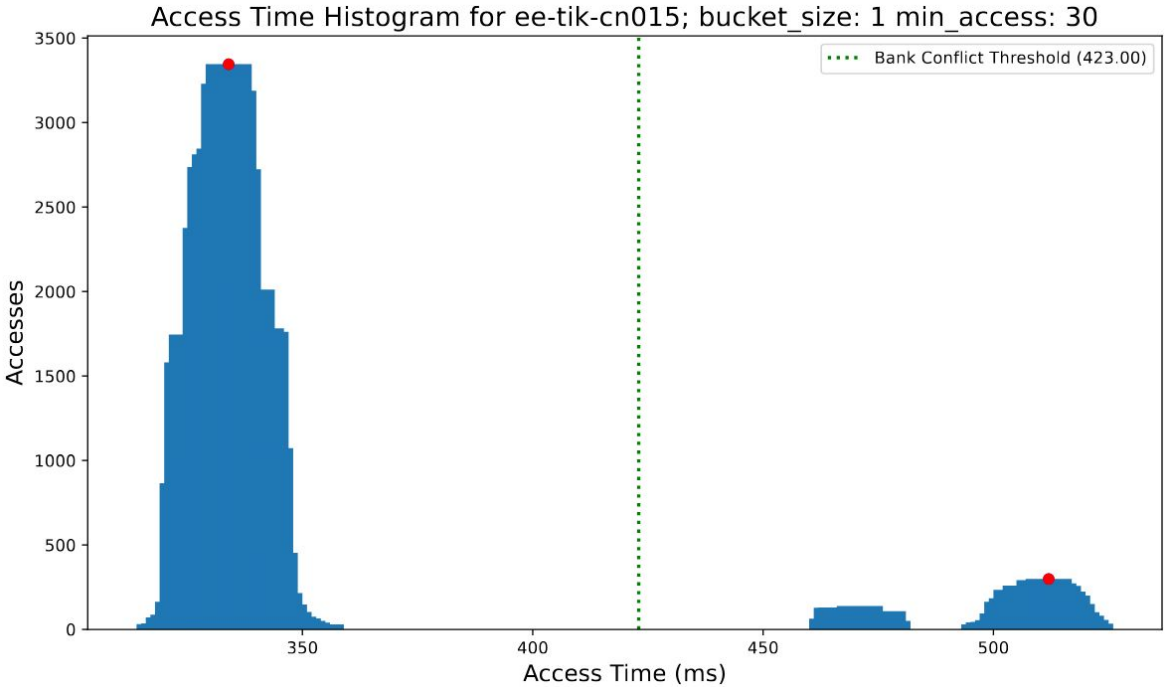
Identifying bank conflicts & #banks

Through accessing random memory addresses and timing we can determine if they are in the same bank (slower access, if not in the same row)

Location between these two peaks gives us a threshold

With this threshold we can generate a number of sets of addresses that are in the same bank

#sets generated == #banks



Task 3

Reverse Engineering Bank Selection Mask

```
conflict set s
a0 = s[0]
for bit in range(30):
    a1 = a0.flip(bit)
    if a0 no conflict a1:       # either column bit flipped, or to another bank (/rank etc.)
        a2 = set[1]             # a2 conflict a0
        if a1 conflict a2:      # a1 a2 in same bank, i.e. column bit flipped
            bit is column bit
        else:                   # a1 a2 not in same bank, i.e. bank (/rank etc.) bit flipped
            bit is bank bit
```

Task 4

Reverse Engineering Bank Selection Functions

Exhaustively try out all the (XOR) combinations of (pairs of) bits of bit masks found in Task 3.

Functions are those that yield same results for all addresses in the same conflict set, for all conflict sets found.

Task 5

Reverse Engineering Row Selection Mask

```
char *a0
for bit in range(0,30):
    a1 = a0.flip(bit)
    if a1 not in same bank as a0:
        flip corresponding bit of a1 based on bank mapping functions, so that a1 has same bank idx as a0
    # now a1 should be in the same bank as a0
    if a0 conflict a1:
        higher bit flipped is in row mask # as row bits are usually higher
```

Rowhammer - Results

Selected to run on nodes 10, 15, 17:

hostname,	threshold,	nbanks,	dram_addr_mask,	row_mask,	bank_functions
ee-tik-cn010,	414,	16,	0x0000000fe040,	0x3ffe0000,	0x2040, 0x24000, 0x48000, 0x90000
ee-tik-cn015,	423,	32,	0x0000003fe040,	0x3ffc0000,	0x2040, 0x44000, 0x88000, 0x110000, 0x220000
ee-tik-cn017,	349,	16,	0x0000000fe040,	0x3ffe0000,	0x2040, 0x24000, 0x48000, 0x90000

Task 6

Generating Blacksmith Fuzzing Patterns

1. Randomly pick a victim bank and total length of pattern
2. Randomly pick a victim row, fuzzing amplitude & frequency to fill the pattern. **Be careful with the overlapping bits of bank fns and row bits**.
3. Repeat step 2 until the pattern completely filled

Task 7 & 8

Blacksmith - Hammering

Implement the hammering function to access memory patterns that trigger bitflips

Ensure every aggressor access bypasses cache

Random data patterns to initialize memory, as bitflips depend on the data pattern in aggressor and victim rows

Measure the number of ACT commands between refresh commands and sync hammering patterns with REF cycles

Sweep best pattern over 32MiB of memory