Optimizing Systems for Byte-Addressable NVM by Reducing Bit Flipping

Daniel Bittman    Peter Alvaro    Darrell D. E. Long    Ethan L. Miller

Center for Research In Storage Systems
University of California, Santa Cruz

FAST '19
2019-02-26
Byte-addressable Non-volatile Memory

Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here!
by Ian Cutress & Billy Tallis on May 30, 2016 2:13 PM EST

BNVM is coming, and with it, new optimization targets
Byte-addressable Non-volatile Memory

Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here!

by Ian Cutress & Billy Tallis on May 30, 2018 2:18 PM EST

It’s not just writes...

...it’s the bits flipped by those writes
BNVM power usage

PCM power consumption scales with bits flipped

DRAM has refresh cost that PCM does not have
Can we take advantage of this?

Software vs. hardware?
Can we take advantage of this?

Software vs. hardware?

How hard is it to reason about bit flips?
Can we take advantage of this?

Software vs. hardware?

How hard is it to reason about bit flips?

How do we design data structures to reduce bit flips?
This talk & this work

Future Research
Reducing Bit Flips in Software

- XOR linked lists
- Red black trees
- Hash tables
XOR linked lists

Traditional doubly linked list

XOR linked list

xptr = next ⊕ prev
Pointers!

Some actual pointers

\[ A = 0x000055b7bda8f260 \]
\[ B = 0x000055b7bda8f6a0 \]
\[ A \oplus B = 0x4C0 = 0b10011000000 \]
Using XOR in hash tables

```
key
value
xnext
...
key
value
xnext
...
key
value
xnext
```
Using XOR in hash tables

Both indicate “entry is empty”

Saves bitflips during overwrites
From XOR linked lists to Red Black Trees

Standard 3-pointer red-black tree design
From XOR linked lists to Red Black Trees

Now 2-pointer, and XOR pointers

\[ R_x = R \oplus P \]
\[ L_x = L \oplus P \]
Evaluation

- Determine bit flip characteristics
- Measure performance impact
Experimental framework

Full system simulator, patched to support bitflip counting at the memory controller

Test different data structures, with different cache parameters, over a varying number of operations
Bit flips: calling malloc()

Trends often become linear

Cross-over point between 40 and 48 bytes
Bit flips: XOR Linked Lists

XOR linked lists reduce bit flips dramatically.
Bit flips: Hash table

- Hash table already had few flips to save: chains should be short

<table>
<thead>
<tr>
<th>Memory events per insert operation</th>
<th>Single-linked</th>
<th>Node XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits Flipped</td>
<td>13.4</td>
<td>11.9</td>
</tr>
<tr>
<td>Bytes Written</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bit flips: Red-black Trees

Write/bitflip inversion!
Significant savings

better
Bit flips: L2 Cache Behavior

L2 cache has ultimately little effect!
Bit flips: L2 Cache Behavior

L2 cache has ultimately little effect!
...even when increasing in size
Performance: RBT insert

Performance is not significantly affected!

a) Performance cost of XORs
b) Performance benefit of smaller node size
Performance: hash table

Almost no effect on performance
Conclusions

Savings are significant with little performance impact.

*We can* design around bit flips, and *we should.*

Bit flip/write inversion
This talk & this work

- System simulation
- Data structures
- Power and wear estimates
- Caching effects
- Pointer distance

Full systems

- Additional techniques
- Bitflips from algorithms

Real hardware

- More data structures

Future Research
Thank You! Questions?

Daniel Bittman  
@danielbittman  
dbittman@ucsc.edu

Peter Alvaro  
palvaro@ucsc.edu

Darrell Long  
darrell@ucsc.edu

Ethan Miller  
elm@ucsc.edu

https://gitlab.soe.ucsc.edu/gitlab/crss/opensource-bitflipping-fast19
Backup slides
Bit flips: instrumentation

![Graph showing the number of bits flipped versus the number of insert operations. The graph includes data points for xlist, sim., dlist, sim., xlist, instr., and dlist, instr., each marked with different symbols.]
Performance: RBT lookup

- xrbt
- xrbt-big
- rbt

Nanoseconds per lookup

Seq. Lookup

Rand. Lookup

better
## Performance: XOR Linked List

<table>
<thead>
<tr>
<th>Operation</th>
<th>XOR Linked List</th>
<th>Doubly Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert (ns)</td>
<td>45 +/- 1</td>
<td>45 +/- 1</td>
</tr>
<tr>
<td>Pop (ns)</td>
<td>27 +/- 1</td>
<td>28 +/- 1</td>
</tr>
<tr>
<td>Traverse (ns/node)</td>
<td>2.6 +/- 0.1</td>
<td>2.2 +/- 0.1</td>
</tr>
</tbody>
</table>
Stack frames

```
fn0
arg0
pc
sp
csr0

fn1
arg0
pc
sp
csr1
```
Stack frames

```
fn0
arg0
pc
sp
csr0
csr1

fn1
arg0
pc
sp
csr1
```

"Wasted space"
Bit flips: stack frames

Bits flipped per iteration

Bytes written per iteration

x-x  x-s  x-y

24    22    90.6

x-x  x-s  x-y

500
250
0