OSwrite: Improving the lifetime of MLC STT-RAM with One-Step write

***Wei Zhao*[1]**, Wei Tong[1], Dan Feng[1], Jingning Liu[1], Jie Xu[1], Xueliang Wei[1], Bing Wu[1], Chengning Wang[1], Weilin Zhu[1], Bo Liu[2]

[1]Huazhong University of Science and Technology,
[2]Hikstor Technology Co., LTD

36th International Conference on Massive Storage Systems and Technology (MSST 2020)

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Background

Big Data

Cloud Storage

Machine Learing

- Huge demand of larger cache
- Leading to large energy consumption

SRAM based cache: high leakage power, low density and scalability.

Background

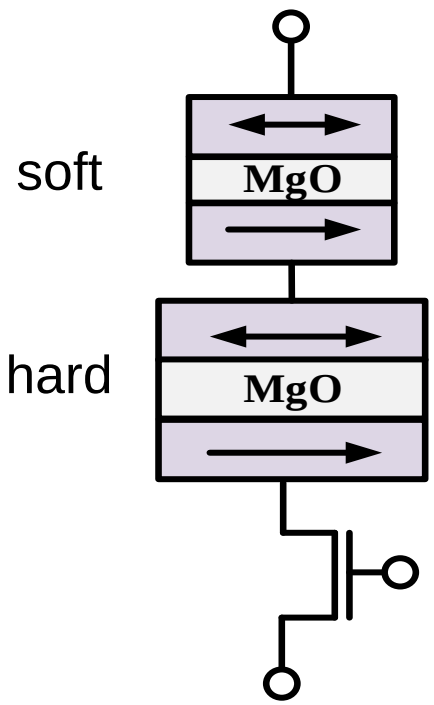
Spin-Transfer Torque Random Access Memory (STT-RAM):

- High density ✓
- Low leakage power ✓
- Compatibility with CMOS ✓
- High write energy and latency × leading to low performance improvement compared with SRAM

Multi-Level Cell STT-RAM can be used:

- Nearly 2X density of (Single-Level Cell) STT-RAM
  Better performance
- Two-Step write
  Less lifetime, higher write energy and write latency

Background



Structure: Two Magneto resistive cells and one transistor.

Write *hard* bit need a larger current than *soft* bit.

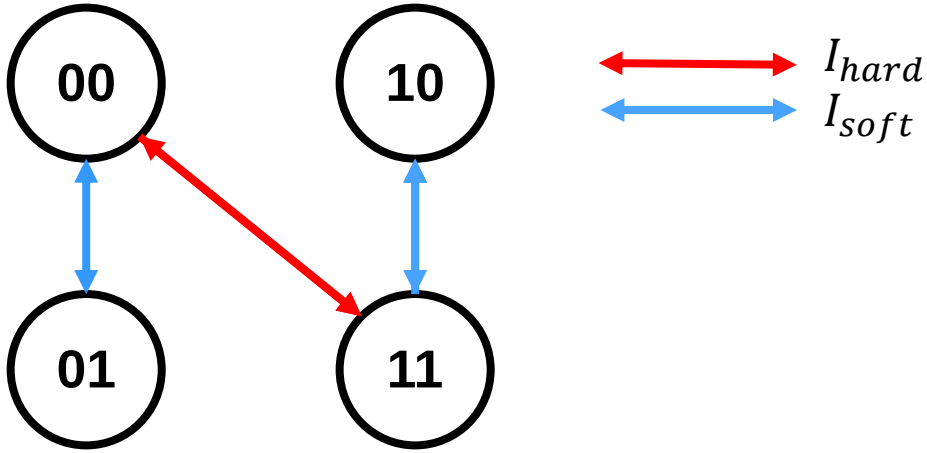
$$I_{hard} > I_{soft}$$

Write hard damages the soft value, and one extra write is needed to restore soft. *i.e. two-step write*

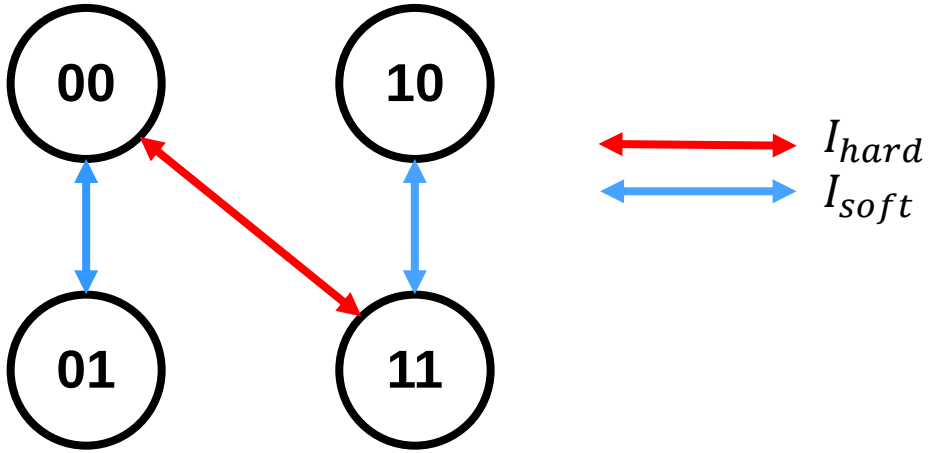
Data format:

$[hard, soft]$

00 → 10



Background



Data write can be summarized as four types:

- Zero Transition (ZT) : The old and new data are the same.
- Soft Transition (ST) : Only need I_{soft}.
- Hard Transition (HT) : Only need I_{hard}.
- Two-step Transition (TT): Need both I_{hard} and I_{soft}.

Motivation

For HT, a large current I_{hard} flows through both the soft and hard domain, resulting in one wear to both the hard and soft domain.

As for TT, soft suffers from two wears due to one extra ST to restore data.

From \ To	00	01	10	11
00	ZT(0, 0)	ST(0, 1)	TT(1, 2)	HT(1, 1)
01	ST(0, 1)	ZT(0, 0)	TT(1, 2)	HT(1, 1)
10	HT(1, 1)	TT(1, 2)	ZT(0, 0)	ST(0, 1)
11	HT(1, 1)	TT(1, 2)	ST(0, 1)	ZT(0, 0)

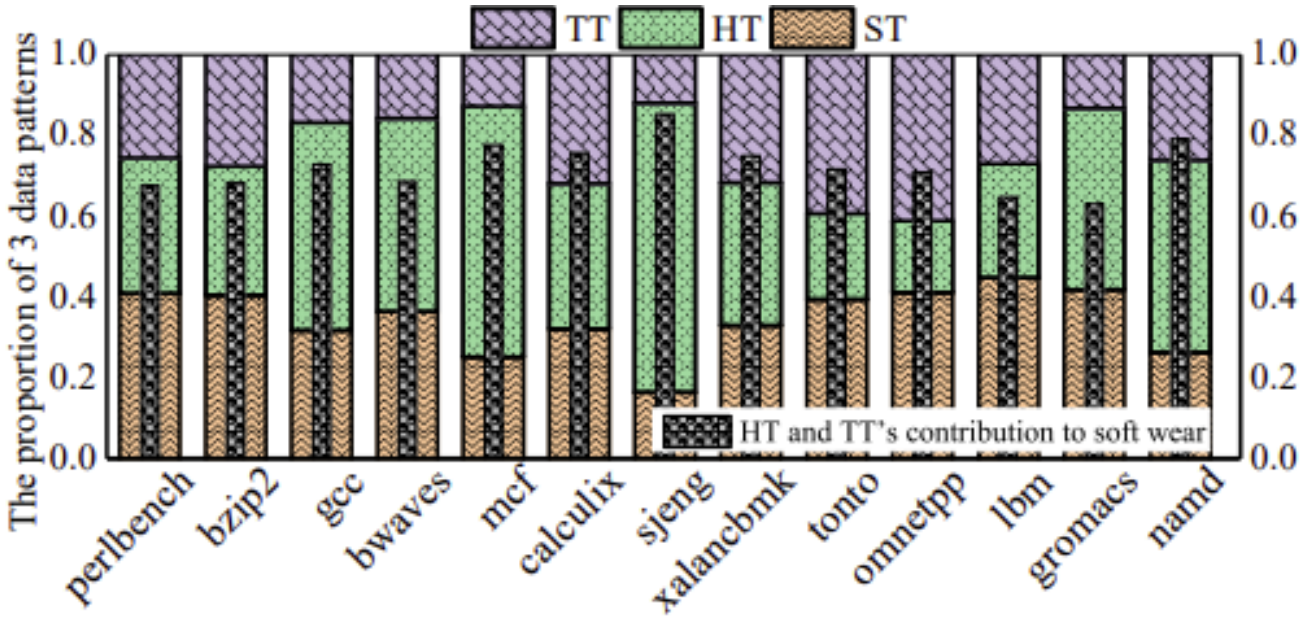
(Wear to hard, wear to soft)

Soft bit suffers more wears thus leading to less lifetime than hard.

Motivation

The contributions to the lifetime degradation of three data transitions:

$$Contribution_{HT,TT} = \frac{wear_{HT,TT}}{wear_{HT,TT,ST}}$$



The result indicates that nearly 70% lifetime degradation is caused hard bit flips. ST leads to 30% reduction.

Design

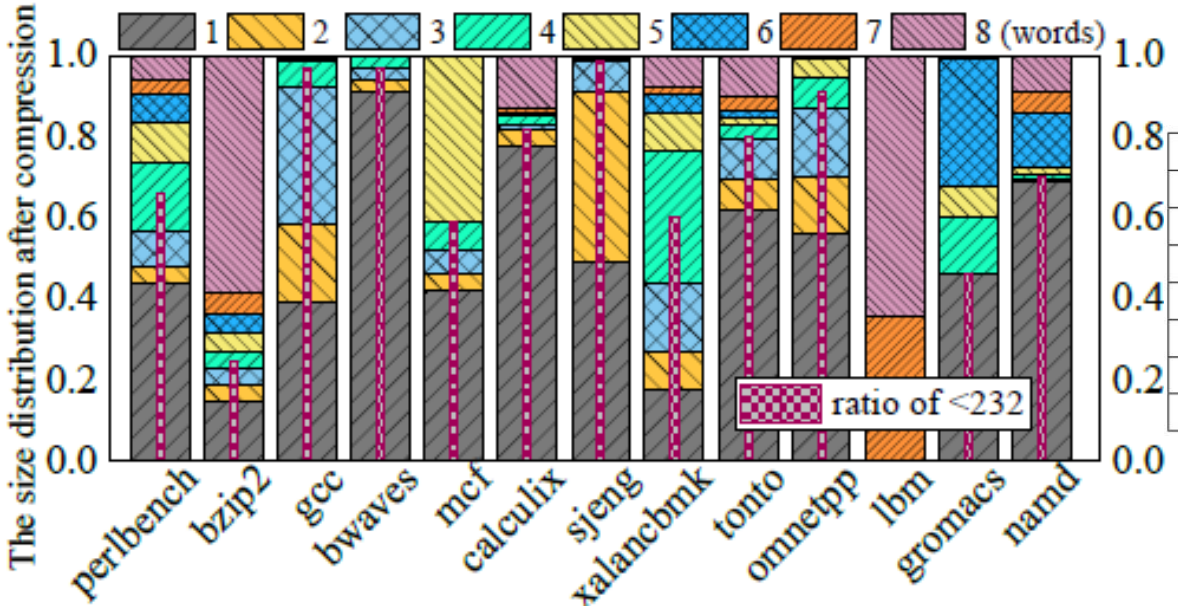
We mainly reduce HT and TT, and decreasing ST at the same time.

We propose One-Step write (OSwrite) to write data with only one step, the main contributions are as follows:

- We propose Half-Sized Compression (HSC).
- We propose Hard Transition Removal Encoding scheme (HTRE).
- The implementation of OSwrite.
- System level evaluation.

Design

1. Half-Sized Compression (HSC)



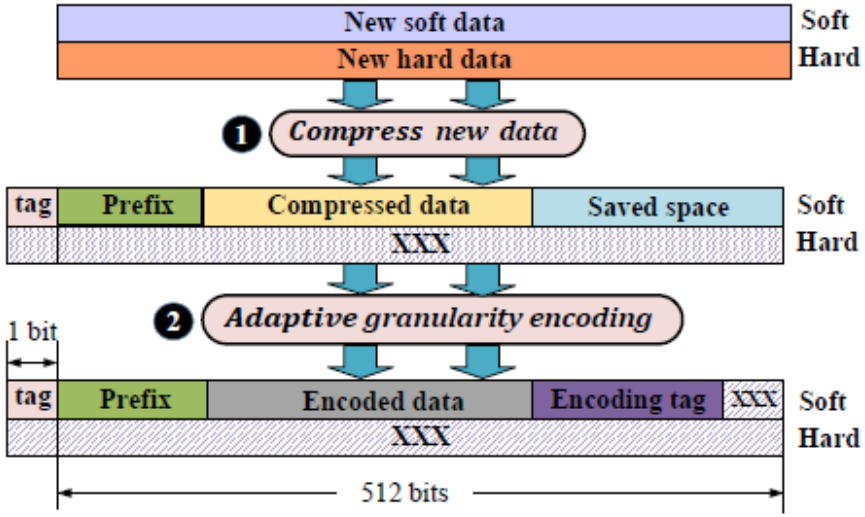
Frequent Pattern Compression

Prefix	Pattern encoded	Example	Compressed example	Encoded size
000	Zero run	0x0000000000000000	0x0	3 bits
001	8-bits sign-extended	0x000000000000007F	0x17F	11 bits
010	16-bits sign-extended	0xFFFFFFFFFFFFB6B6	0x2B6B6	19 bits
011	Half-word sign-extended	0x0000000076543210	0x376543210	35 bits
100	Half-word, padded with a zero half-word	0x7654321000000000	0x476543210	35 bits
101	Two half-words, each two bytes sign-extended	0xFFFFBEEF00003CAB	0x5BEEF3CAB	35 bits
110	Consisting of four repeated double bytes	0xCAFECAFECAFECAFE	0x6CAFE	19 bits

We observe that many LLC cache lines can be compressed (Frequent Pattern Compression) to *half-size*, and *the saved space is varied*.

Design

1. Half-Sized Compression (HSC)



① Compress data to half-size to *reduce HT and TT*, then organizing the new data layout.

$$G = \begin{cases} 2, & S_{comp} \in [0, 154] \\ 4, & S_{comp} \in [155, 185] \\ 8, & S_{comp} \in [186, 206] \\ 16, & S_{comp} \in [207, 218] \\ no\ encoding, & S_{comp} \in [219, 256] \end{cases}$$

G is the compressed data size.

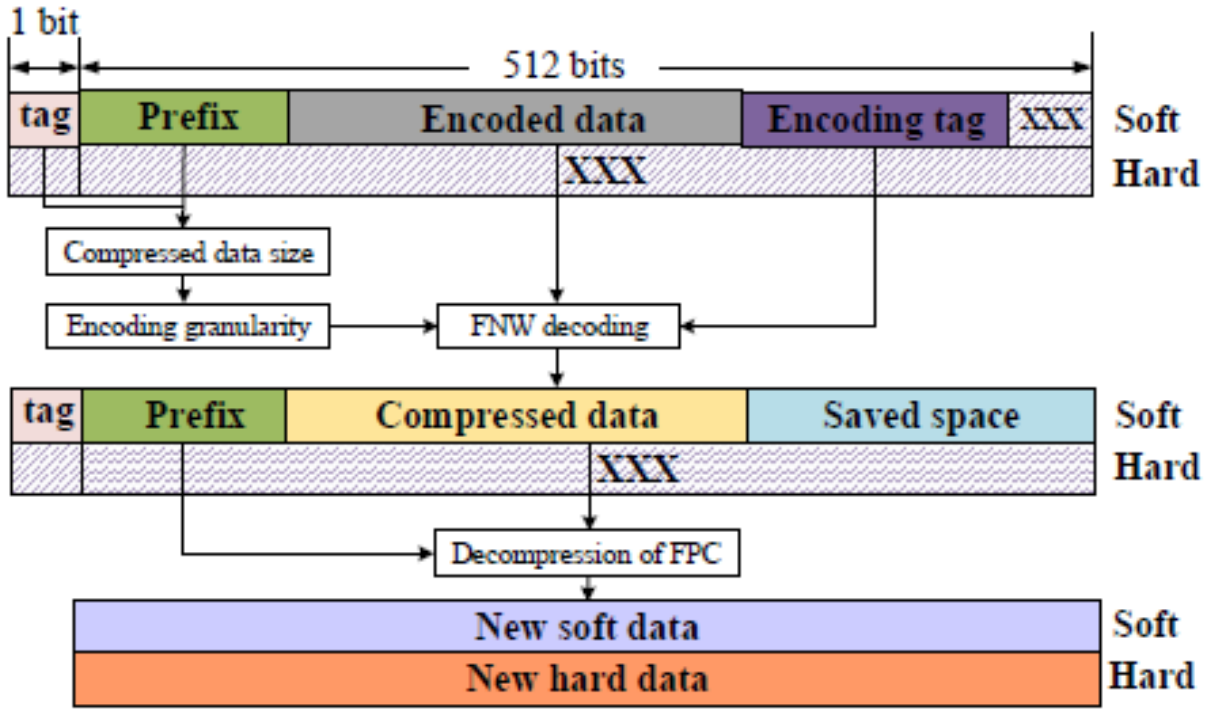
② Encoding the compressed data with adaptive granularity of Flip-N-Write (FNW) to *reduce ST*.

Writing data on soft line can be finished by one-step.

Design

1. Half-Sized Compression (HSC)

Decoding procedure:



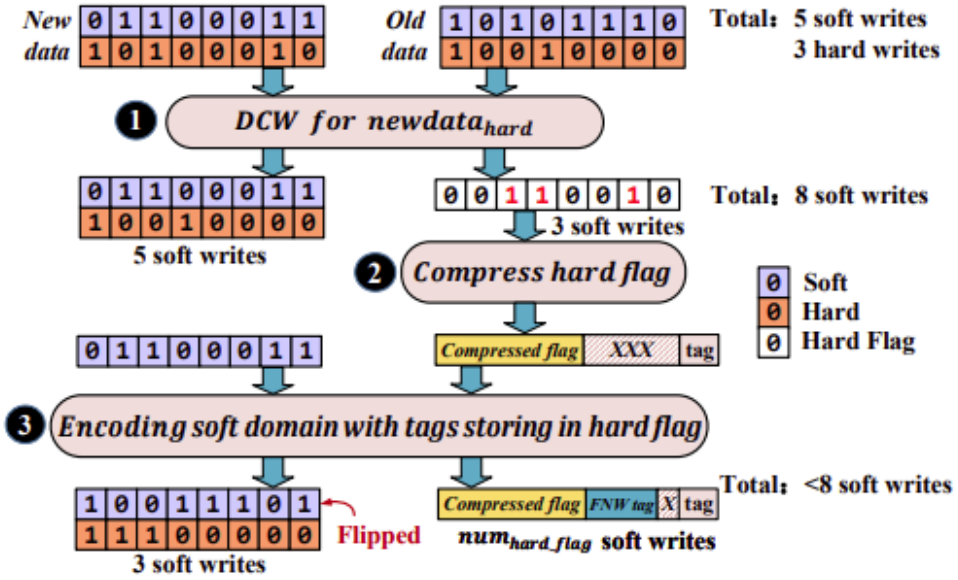
① The encoded data are decoded by FNW decoder.

② The decoded data are then decompressed by FPC decompressor.

Design

2. Hard Transition Removal Encoding (HTRE) scheme

Not all cache lines can be compressed to less than half-size.



$$G = \begin{cases} 2,\ S_{comp} \in [1, 76],\ encode\ soft\ and\ flag \\ 2,\ S_{comp} \in [77, 116],\ only\ encode\ soft \\ 4,\ S_{comp} \in [117, 180],\ only\ encode\ soft \\ 8,\ S_{comp} \in [181, 212],\ only\ encode\ soft \\ 16,\ S_{comp} \in [213, 228],\ only\ encode\ soft \\ no\ encoding,\ S_{comp} \in [229, 256] \end{cases}$$

① We use XOR logic to get the hard flag (SLC STT-RAM) data of the new and old data. *(Remove HT and TT)*

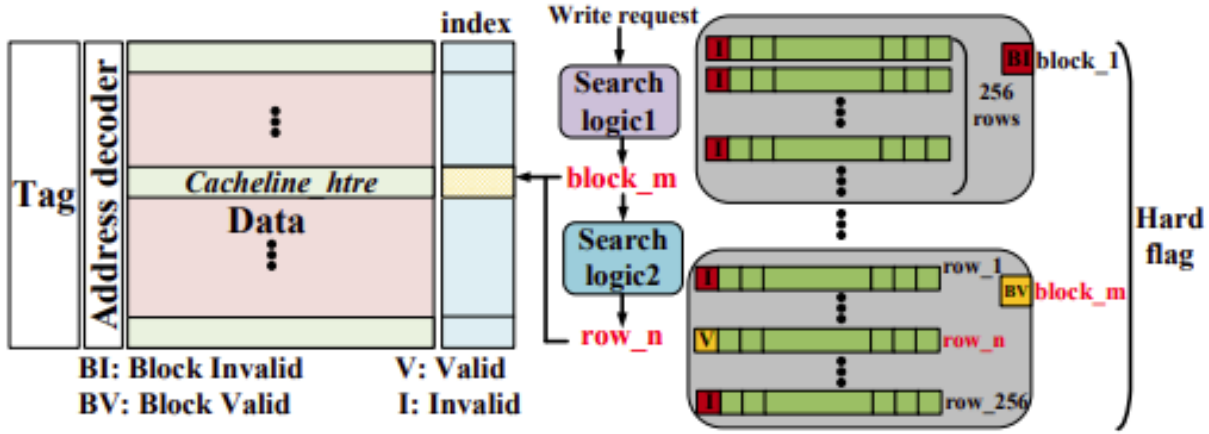
② Due to the many existing '0' data, hard flag can be easily compressed. *(Reduce the writes to hard flag)*

③ Encoding the soft line and hard flag to *reduce ST* and the write energy of hard flag. Then, they are write simultaneously to ensure *one-step write*.

Design

2. Hard Transition Removal Encoding (HTRE) scheme

Not all cache lines equip hard flags. We dynamically allocate it for cache line.



When a cache line will be encoded by HTRE, we use the two-level search logic to *find the empty hard flag*.

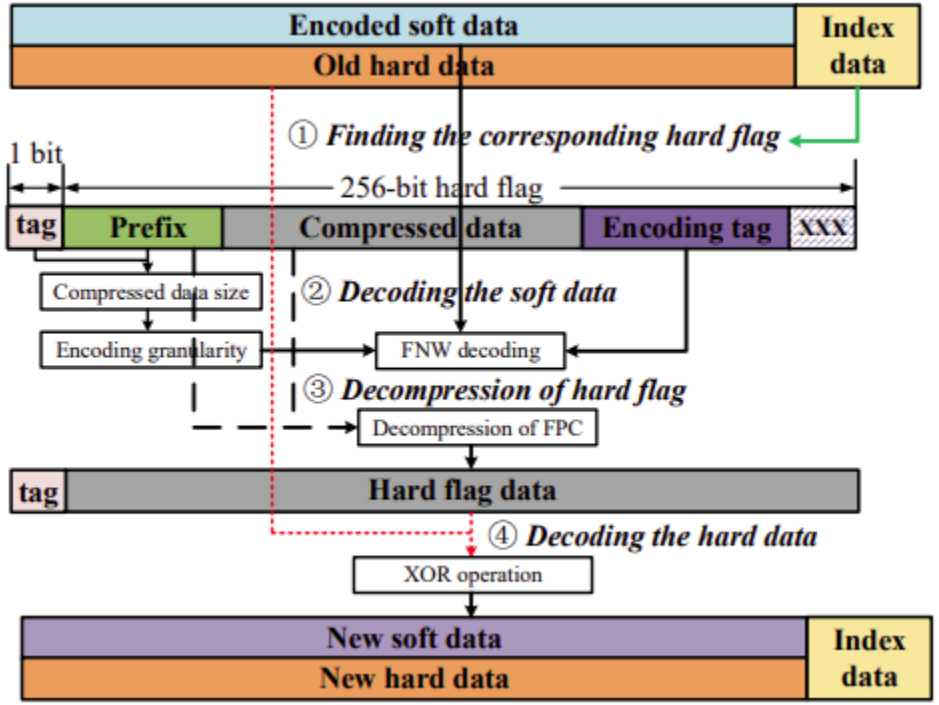
While the empty one is found, the state of this flag is *set to 'Invalid'.*

When a block encoded by HTRE is evicted, the flag state is set to *'Valid'.*

Design

2. Hard Transition Removal Encoding (HTRE) scheme

Decoding procedure:



① Finding the corresponding hard flag via the index data.

② Decoding the data in soft line by FNW decoding.

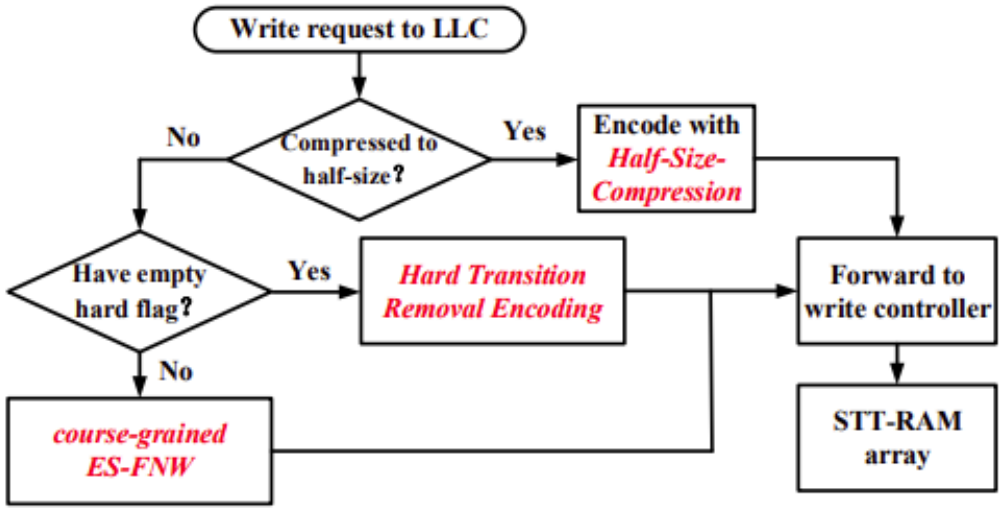
③ Decompressing the hard flag by FPC decompressor.

④ Decoding hard data through simple XOR operation.

Design

3. Implementation of OSwrite

Write operation



If cache line cannot be encoded by HSC and HTRE, then it is encoded by ES-FNW by storing the encoding tags in index data.

Encoding type flag

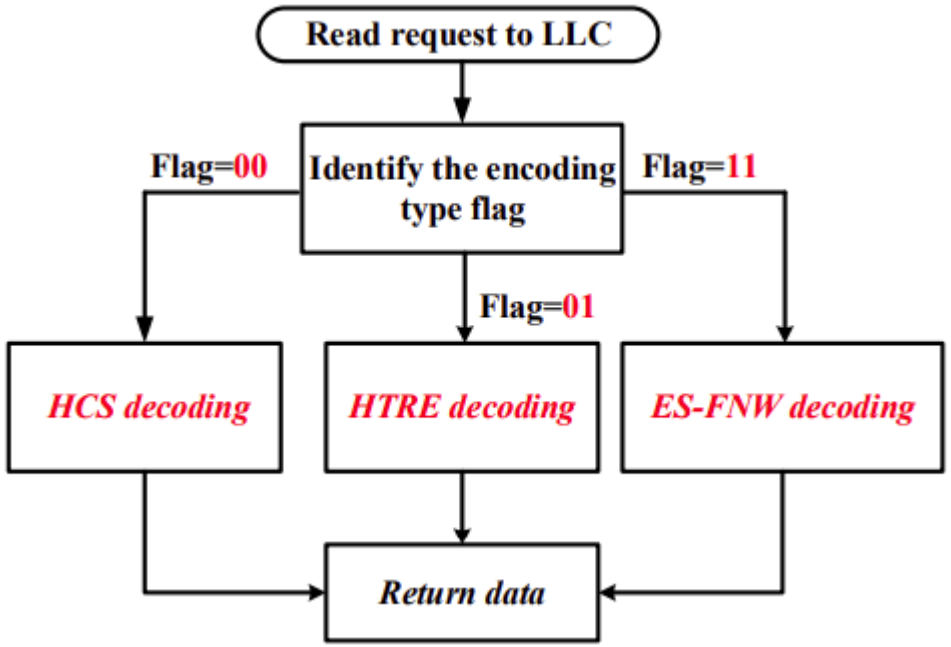
Encoding scheme	Encoding type flag
HSC	00
HTRE	01
ES-FNW	11

Writing the encoding type tag of ‘00’ and ‘01’ is a one-step write operation.

Design

3. Implementation of OSwrite

Read operation



Overhead analysis

Capacity overhead:
index data + hard flag array 12.55%

Hardware overhead:

Implementation	Latency	Energy
FPC	Encoding: 2*ns* Decoding: 1*ns*	Encoding: 2.1*pJ* Decoding: 1.2*pJ*
Flip-N-Write	Encoding: 1*ns* Decoding: 0.1*ns*	negligible
Search logic	0.26*ns*	1.9*pJ*
Multiplexer	1.49*ns*	1.68*pJ*

The decoding procedure is on the critical path, and encoding latency can be hidden by the long write latency.

Evaluation

We use gem5 to implement Oswrite, and the benchmarks are from CPU SPEC 2006.

System configuration

Cores	4-Core, 2.0GHz, out-of-order
L1 I/D cache	private, 32KB per core, 2-way; LRU, 2-cycle latency
L2 Cache	private, 512KB per core, 64B cache line; 8-way, LRU, 10-cycle latency
L3 Cache	MLC STT-RAM based cache, 16MB; SLC STT-RAM based cache, 8MB ; shared, 64B cache line, 16-way, LRU;
Main Memory	4GB, DDR-1600

STT-RAM parameters

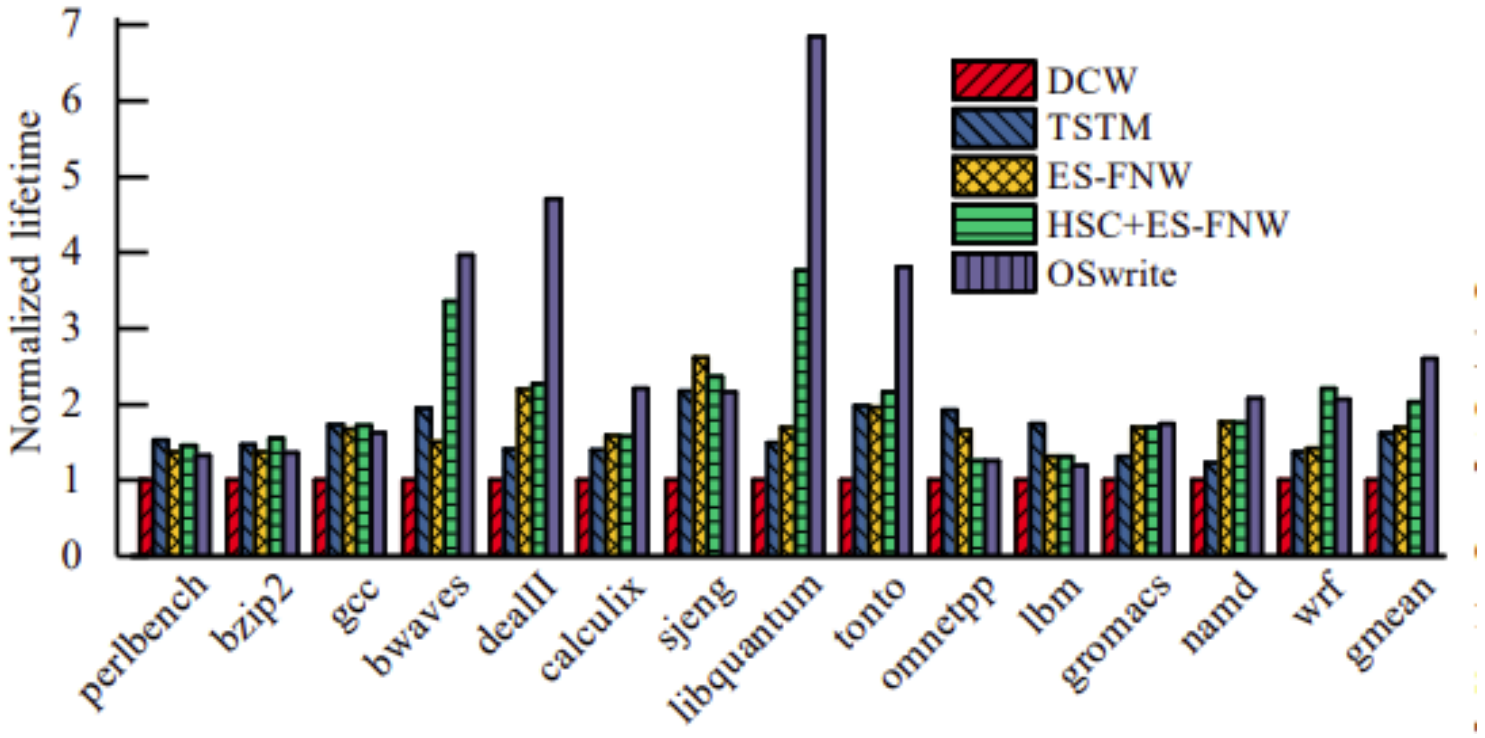
	SLC	MLC
Read Latency (*Cycles*)	5.5	S:4.08 H:5.94
Write Latency (*Cycles*)	15.5	S: 15.34 H:34.24
Read Energy (*nJ*)	0.216	S:0.22 H:0.43
Write Energy (*nJ*)	0.839	S:0.843 H:2.502

Compared with several schemes:

- 8MB SLC STT-RAM cache.
- DCW [B.-D. Yang et al'ISCS 07]: Data Comparison Write.
- TSTM [H.Luo et al'DAC 16]: Using 3MLCs to indicate the value of 2 MLCs.
- ES-FNW [J. Xu et al'ICCD 17]: Encoding soft and hard with FNW seperately.
- HSC+ES-FNW: Encoding half-sized cache lines with HSC, and others are encoded by ES-FNW.
- Oswrite: This is our scheme with all optimizations.

Evaluation

1. Lifetime

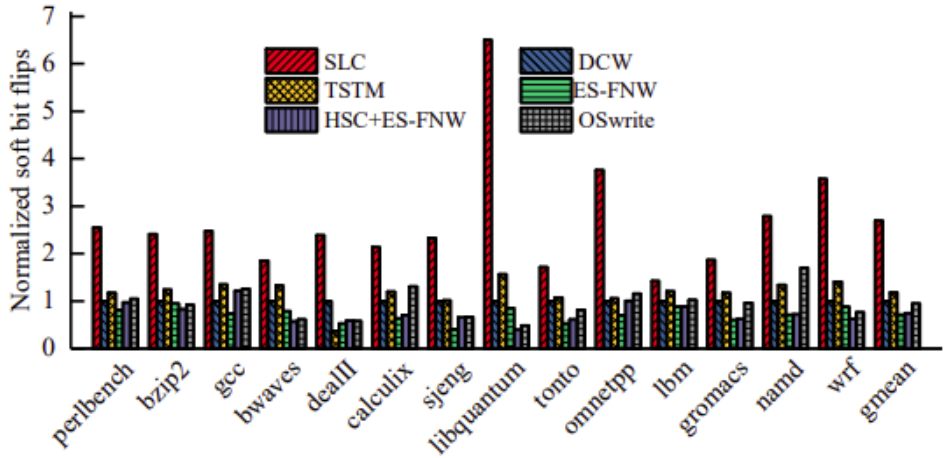


OSwrite can improve lifetime to 2.6× compared with baseline.

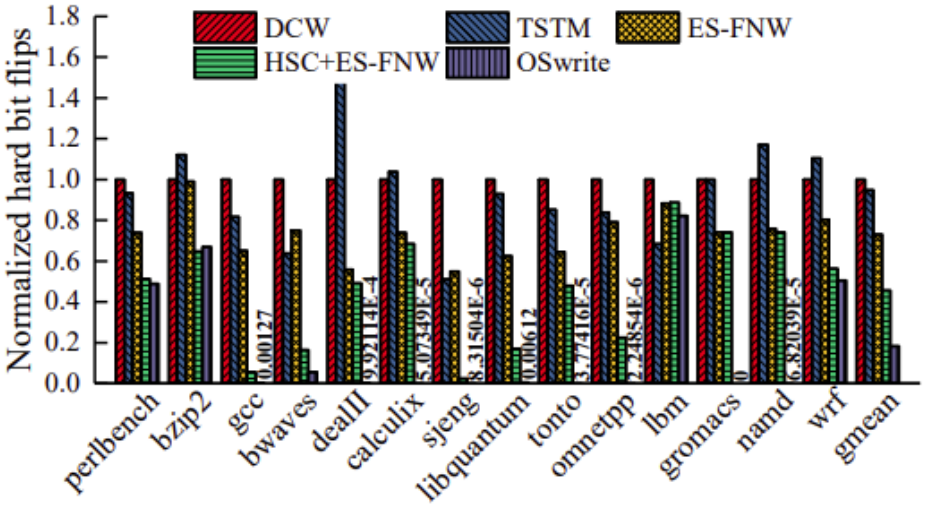
Evaluation

2. Bit flips

Soft bit flips



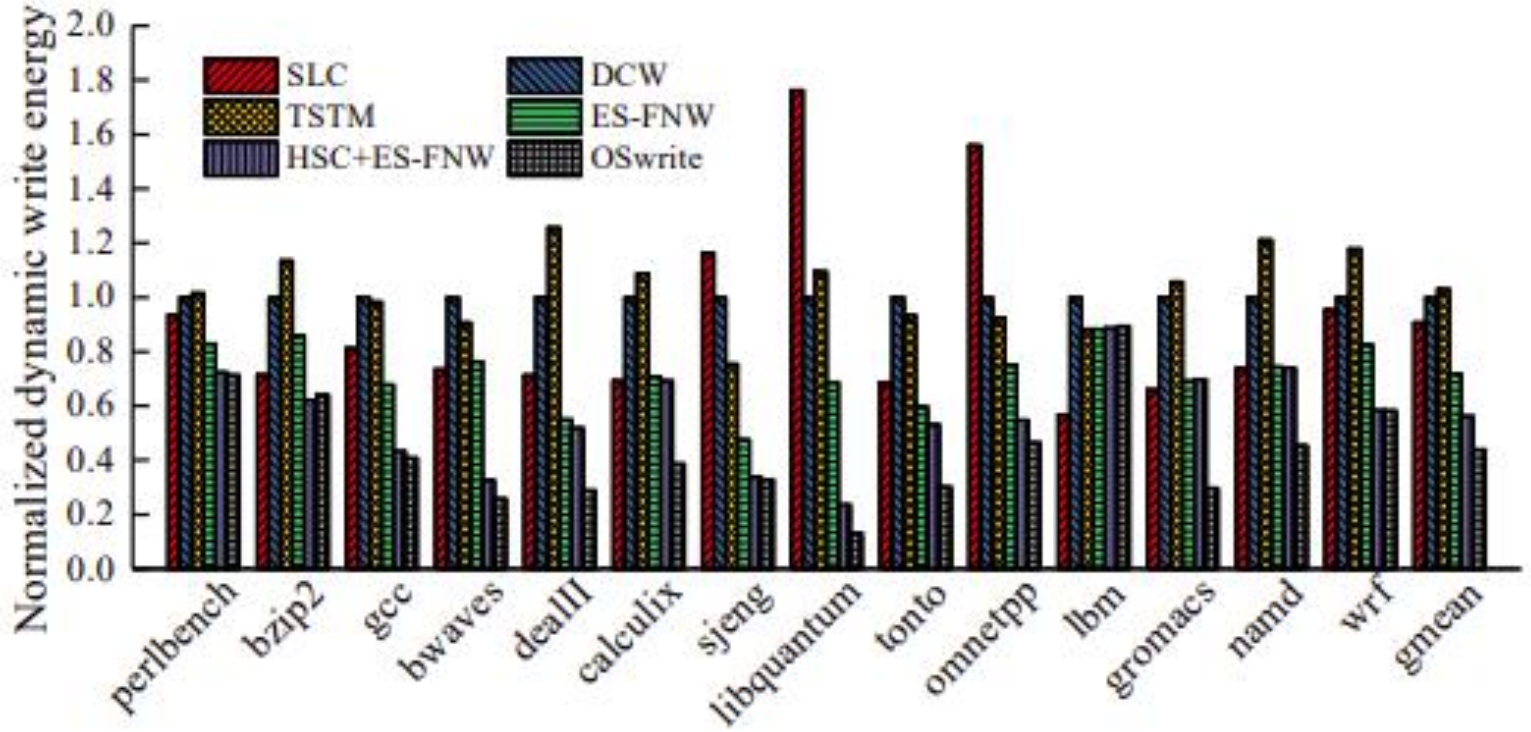
Hard bit flips



Compared with baseline, OSwrite can reduce soft and hard bit flips by 5.3% and 82.8%, respectively.

Evaluation

3. Write energy

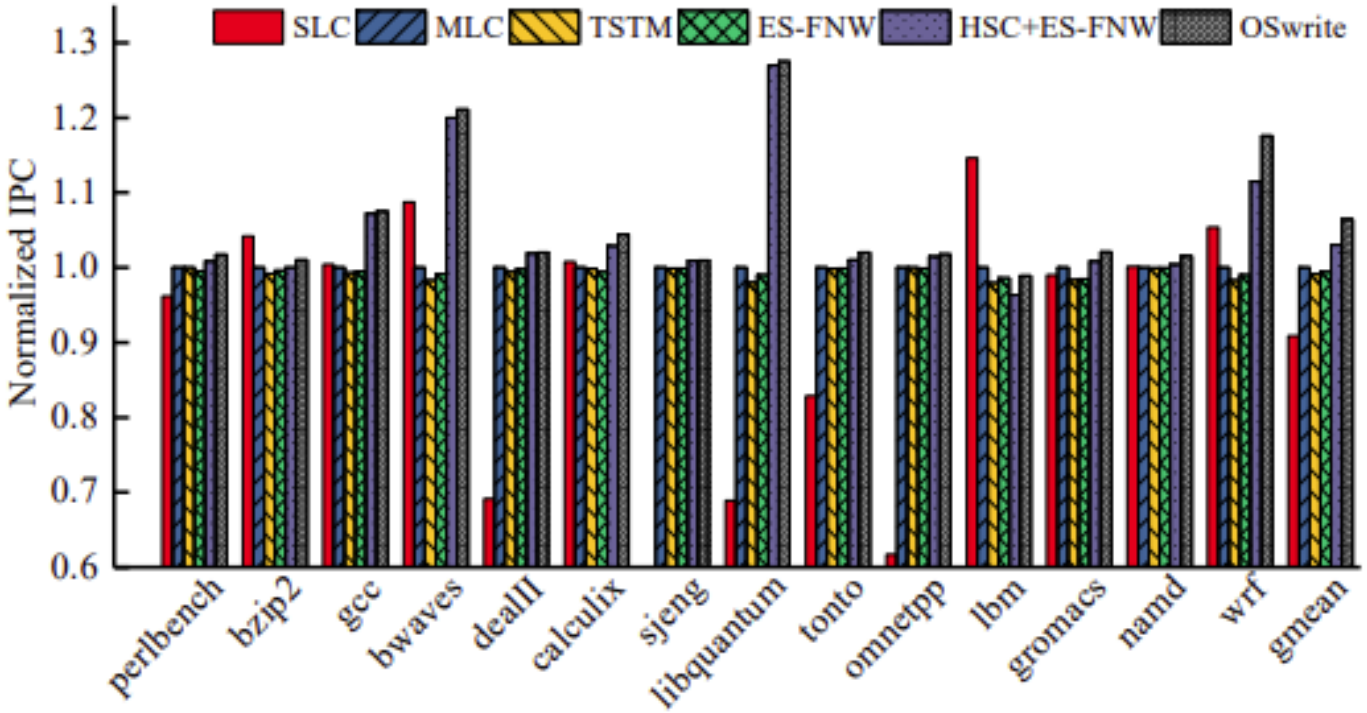


OSwrite can reduce write energy by 56.2% compared with baseline.

Evaluation

4. Performance

$$IPC_{speedup} = \frac{IPC}{IPC_{baseline}}$$



OSwrite can improve performance by 6.4% compared with baseline.

Conclusion

Challenges:

- Two-step write of MLC STT-RAM.
- Limited lifetime.

OSwrite:

- We propose Half-Sized Compression (HSC).
- We propose Hard Transition Removal Encoding scheme (HTRE).
- The implementation of OSwrite.
- OSwrite can improve the lifetime of MLC STT-RAM to *2.6*$\times$, and reduce write energy and improve system performance by *56.2%* and *6.4%*, respectively.

Thanks for your listening!

Q&A