Enhancing the Lifetime of Non-Volatile Memory by Hybrid Cache-Replacement-Policy

Presented by : Sampath Kumar S, 206116021

Guide : Ms B.Shameedha Begum

Overview

Non-Volatile caches

- Due to increase in the number of cores in modern computers large on-chip cache are required.
- Scalability of conventional SRAM and DRAM is constrained by
  - Low Cell density
  - High Leakage power
- Non-Volatile Memory is better alternative for SRAM and DRAM due to High Cell density and Low Leakage power.
- Besides above advantages, NVM suffers from low endurance value, which inturn reduces the lifetime of the NVM caches.

Problem Statement

- Due to limited write endurance and high write energy in Non Volatile Memory (NVM), Effective write mechanism is required to utilize the full efficiency of NVM.
- The cache replacement policy has been proposed to **improve the lifetime** of the cache sub systems with the help of Wear-leveling technique.

Existing Solutions

EqualWrites

- Using Wear-Leveling Technique, EqualWrites reduces the Intra-Set variations by redirecting writes of the hot blocks (Maximum writes) to the cold blocks (Minimum writes).

LER Replacement Algorithm

- The idea is to place the incoming block in a line that incurs the minimum error rate in WRITE operation. This is done by comparing the contents of the incoming block with lines in a cache set.

Drawbacks of Existing Solutions

- Existing Solution does not utilize the full capacity and efficiency of the Non-Volatile Memory.
- The major disadvantage of Non-Volatile Memory is their high error rate during write operation, which is not considered.
- EqualWrites is concentrating on distributing the writes across the set, not considering the number of bit changes within victim line.

Proposed Methodology

- During new data write into memory cell, bit flipping (either **0 to 1** or **1 to 0 transition**) can happen.
- The error rate during **0 to 1 transition** is 100 times higher than **1 to 0 transition**, because of voltage fluctuation.
- The high error rate during write operation and unequal writes can be controlled by designing a **Hybrid Cache-Replacement-Policy** which in turn improves the lifetime of the Cache Subsystems.

Algorithm 1 Availability of Redirection block

```
1:  begin
2:  avail = 'N'
3:  target_set = extractSet(new_block)
4:  // Checking whether any redirection block available in target set
5:  for (i = assoc - 1 to 0) do
6:      if (block[i].counter == 0) then
7:          avail = 'Y'
8:          break from for loop
9:      end if
10: end for
```

- **Redirection Block :** Writes on hot Block (higher frequency of writes) will be redirected to cold block (lower frequency of writes).
- **Counter :** Ensure writes are evenly distributed across the set with minimal variation of threshold.

Algorithm 2 Identification of Target block

```
1: find_target = 0
2: target = NULL
3: // Searching among the invalid block in target set
4: for (i = assoc - 1 to 0) do
5:     if ( 0to1trans is minimum and block[i] == Invalid ) then
6:         target = block[i]
7:         inv = 'N'
8:         find_target = 1
9:     else
10:        if ( Inverse 0to1trans is minimum and block[i] == Invalid ) then
11:            target = block[i]
12:            inv = 'Y'
13:            find_target = 1
14:        end if
15:    end if
16: end for
17: // Searching among the valid block in target set
18: if (find_target == 0) then
19:     for (i = assoc - 1 to 0) do
20:         if ( 0to1trans is minimum) then
21:             target = block[i]
22:             inv = 'N'
23:         else
24:             if ( Inverse 0to1trans is minimum) then
25:                 target = block[i]
26:                 inv = 'Y'
27:             end if
28:         end if
29:     end for
30: end if
```

Algorithm 3 Redirecting writes from hot block to Cold block

```
1:  if (target.counter == threshold − 1 and avail == 'N') then
2:      for (i = assoc - 1 to 0) do
3:          if (block[i] <> target ) then
4:              block[i].counter- -
5:          end if
6:      end for
7:  else
8:      if (target.counter == threshold − 1 and avail == 'Y') then
9:          new_target = minimum 0to1trans and target.counter == 0
10:         if (block[new_target == Valid) then
11:             write Data[new_target] to Data[target] and mark clean or dirty
12:             write new data to Data[new_target] and mark dirty
13:         else
14:             write new data to Data[new_target] and mark valid and dirty
15:             mark Data[target] as Invalid
16:         end if
17:         block[target].counter = block[new_target].counter = threshold/2
18:         target = new_target
19:     else
20:         block[target].counter++
21:     end if
22: end if
23: return target
```

Experimental Setup

- **Tools :** GEM5 Simulator
- Evaluation using **PARSEC** benchmark suite
- **Operating System :** Ubuntu 14.04
- **Language :** C++, Python

Evaluation Parameters

- **Number of 0 to 1 Transitions :** Total number of 0 to 1 transitions occur during write operation.
- **Distribution of Writes :**

$$Distribution\ Percentage_i = \frac{TotalWrite_i}{MaxWrite_{set}}$$

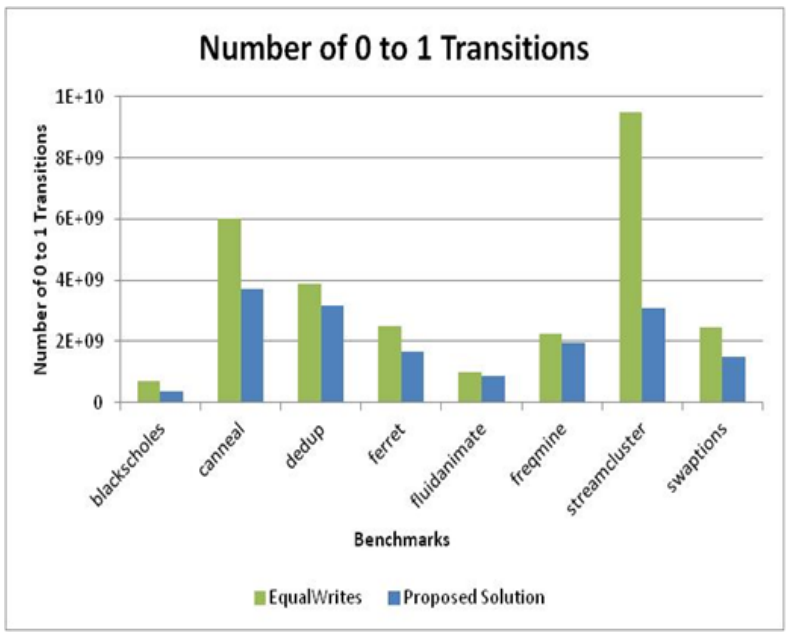
- **Relative Lifetime :** Relative lifetime is computed by considering LRU as baseline

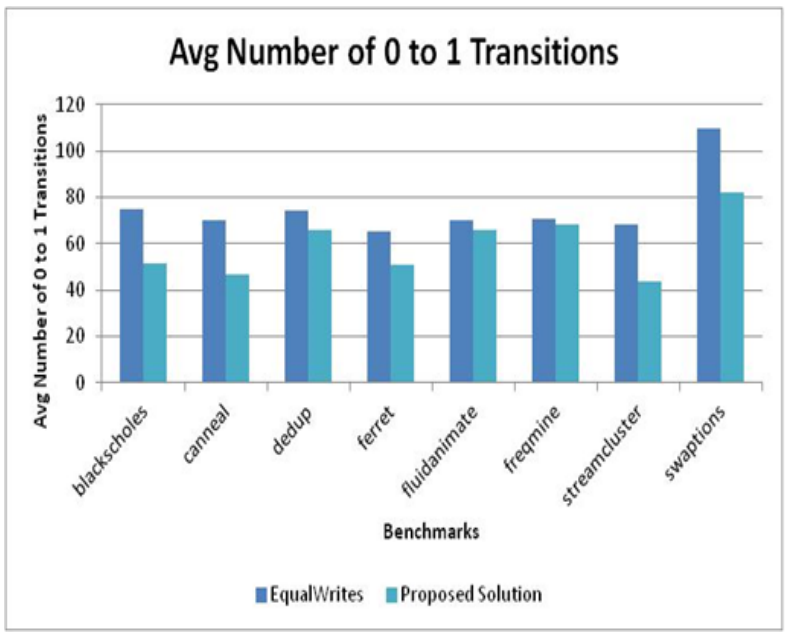
$$Relative\ Lifetime = \frac{Lifetime_{Baseline}}{Lifetime_{Proposed}}$$

- **Relative Performance :** IPC is computed to evaluate the performance of the proposed solution.

Number of 0 to 1 Transitions

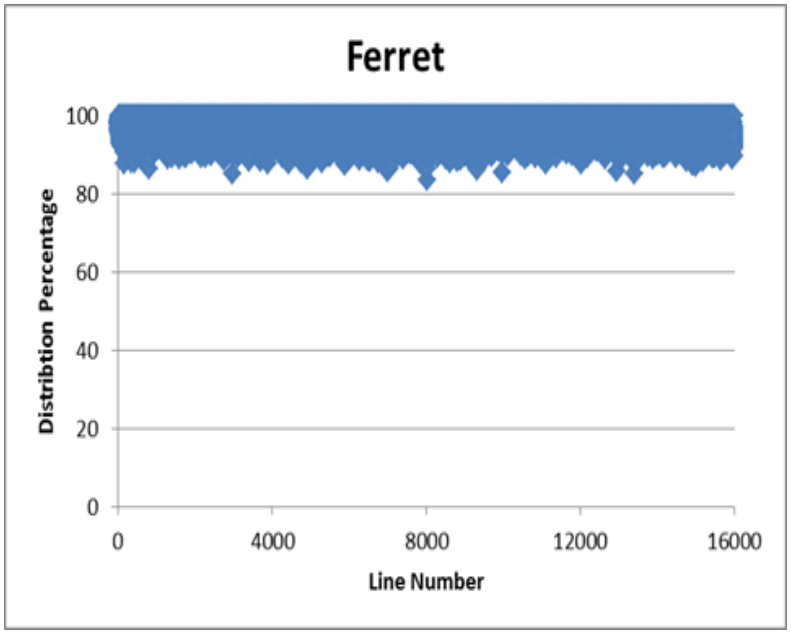
- Proposed solution reduces the number of 0 to 1 transitions by **42%** and average number of 0 to 1 transitions during write operation on cache line by **21%**

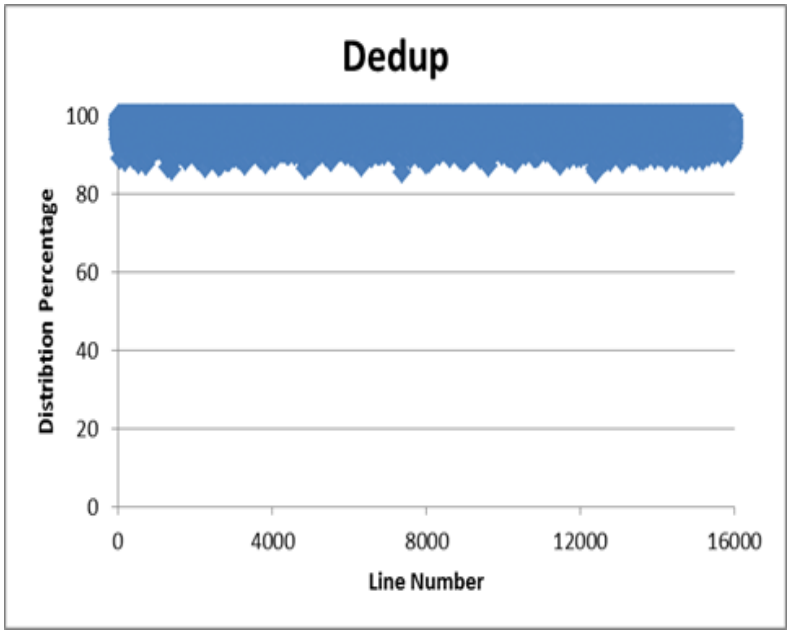




Distribution of Writes

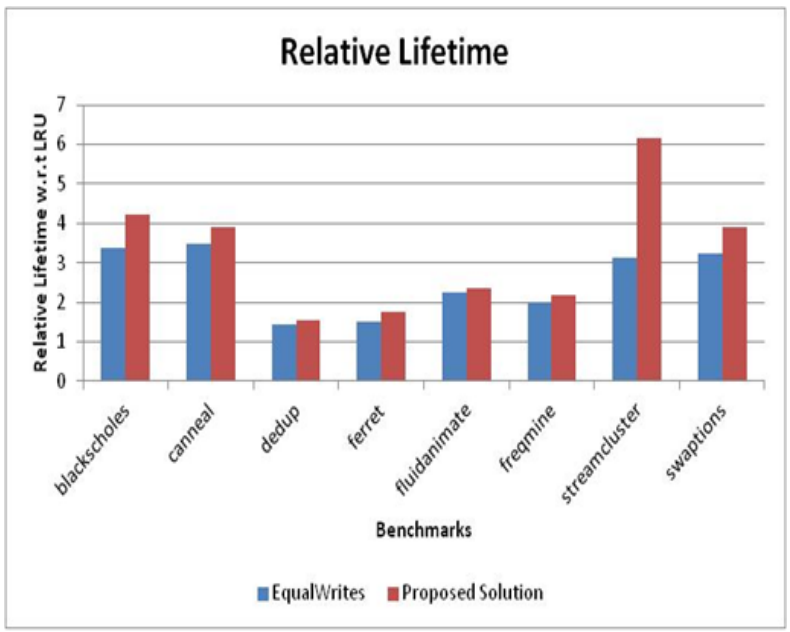
- Proposed solution is distributing the write operation across the block within the set by above 85%.

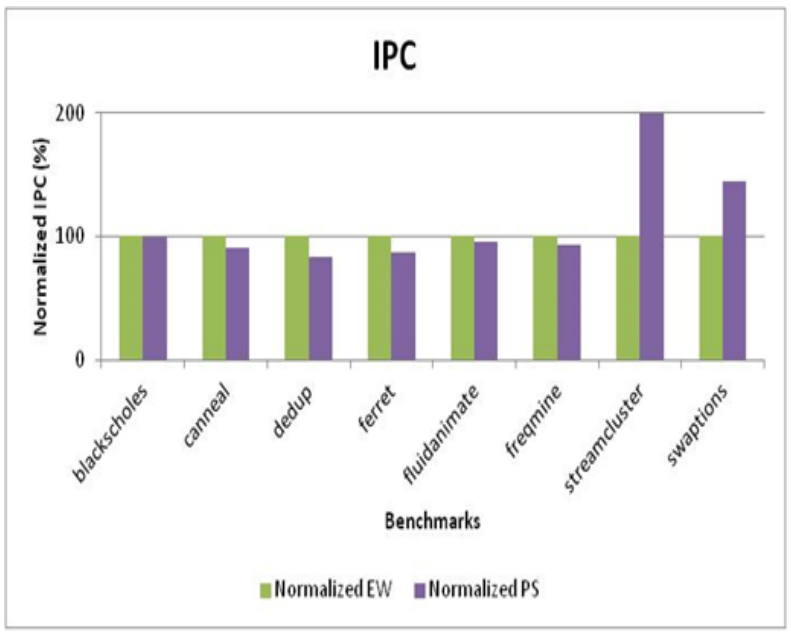




Relative Lifetime and Performance

- Relative Lifetime is calculated by considering **LRU** as baseline. Proposed solution improves the lifetime of the NVM by **27%** with **1% performance overhead**.





Summary

- Proposed Hybrid Cache-Replacement-Policy effectively utilizing the minimal endurance value available and reducing the probability of error rate.
- Proposed solution reduces write variations among the block, probability of write error rates and number of writes into the cell.
- Improves the lifetime of NVM caches by 27% with about 1% performance overhead.

References

[1] Cho, Sangyeun, and Hyunjin Lee., *"Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance."*, Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on. IEEE, 2009.

[2] Mittal, Sparsh, and Jeffrey S. Vetter., *"AYUSH: A technique for extending lifetime of SRAM-NVM hybrid caches"*, IEEE Computer Architecture Letters 14.2 (2015): 115-118.

[3] RJue Wang, Xiangyu Dong, Yuan Xie, Norman P. Jouppi, *" i^2 WAP: Improving Non-Volatile Cache Lifetime by Reducing Inter- and Intra-Set Write Variations"*, Mathematical Methods and Techniques in Engineering and Environmental Science.

[4] Monazzah, Amir Mahdi Hosseini, Hamed Farbeh, and Seyed Ghassem Miremadi., *"LER: Least-error-rate replacement algorithm for emerging STT-RAM caches."*, IEEE Transactions on Device and Materials Reliability 16.2 (2016): 220-226.

[5] Sparsh Mittal, Jeffrey S. Vetter, *EqualWrites: Reducing Intra-set Write Variations for Enhancing Lifetime of Non-Volatile Caches*, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 24, NO. 1, JANUARY 2016

[6] Yang, Byung-Do, et al. *"A low power phase-change random access memory using a data-comparison write scheme."*, Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on. IEEE, 2007.

[7] Bienia, Christian, et al. *"The PARSEC benchmark suite: Characterization and architectural implications."*, Proceedings of the 17th international conference on Parallel architectures and compilation techniques. ACM, 2008.

[8] Binkert, Nathan, et al. *"The gem5 simulator."*, ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.

Thank You