

Compiler-Assisted Refresh Minimization for Volatile STT-RAM Cache

Qingan Li, Jianhua Li, Liang Shi,
Chun Jason Xue, Yiran Chen, Yanxiang He

City University of Hong Kong
University of Pittsburg

Outline

- **Background**
- Observation
- Solution
- Experiments

STT-RAM for cache

- STT-RAM compared to traditional SRAM caches:
 - Leakage power consumption
 - Scalability

Memory (65nm)	high density	low leakage power	slow write		high write energy	
	Capacity	Leakage power	read	write	read	write
SRAM (3.66mm ²)	128KB	2.09W	2.25ns	2.26ns	0.90nJ	0.80nJ
STT-RAM (3.60mm ²)	512KB	0.26W	2.32ns	11.02ns	0.86nJ	5.00nJ

SRAM caches vs STT-RAM caches

STT-RAM for cache

- STT-RAM has been proposed as a new candidate for building cache.
- Lots of work has been done to mitigate the costly write operations on STT-RAM
 - smart write buffer, early write termination, hybrid cache, etc.
 - **relaxing the non-volatile property**

Volatile STT-RAM for cache

- Volatile STT-RAM
 - Pros: trade non-volatility for write performance
 - Cons: need refresh schemes to ensure correctness

	Design 1	Design 2	Design 3
Cell size (F2)	23	22	27.3
MTJ sw time (ns)	10	5	1.5
Retention Time	4.27yr	3.24s	26.5 μ s
Write Latency (ns)	10.378	5.37	1.5
Write Dyn. Eng(nJ)	0.958	0.466	0.187


As the retention time decreases, write speed and write dynamic energy are improved.

Refresh schemes for volatile STT-RAM cache

- The reduced retention time (T) of STT-RAM cache needs refresh schemes to avoid data loss.
- Refresh schemes:
 - **Dram-style refresh:**
 - refresh all blocks within every retention time T .
 - **Full-refresh¹:**
 - attach a counter to each block to track its lifespan asynchronously.
 - Only refresh the blocks not written or refreshed within T .
 - **Dirty-refresh²:**
 - Only refresh dirty blocks not written or refreshed within T .

Refresh schemes for volatile STT-RAM cache

- Each refresh operation for STT-RAM cache involves:
 - read data from cache and write them into buffer
 - read data from buffer and write them back into cache



refresh consumes
lots of energy

- Refresh scheme is by-product of volatile STT-RAM design, and consumes lots of energy.

This paper proposes a compilation approach to minimize the required refresh operations to improve the energy efficiency.

Outline

- Background
- **Observation**
- Solution
- Experiments

Observation 1: N -refresh scheme

- Some data blocks are left in idle status for a long time before next use. (An extreme case: dead blocks)
- Previous refresh schemes refresh these blocks many times for little benefits.
- We propose a **N -refresh** scheme:
 - refresh blocks sequentially for at most $(N-1)$ times.
 - can reduce the number of refresh operations
 - only need add $\log N$ bits to the counter attached to each block (extension over **Full-refresh scheme**)

Observation 2: *passive & active* refresh

- To recharge the lifespan of a cache block
 - *passive refresh*: depends on memory trace and cache misses
 - loading data from the main memory to this cache block
 - an instruction writes data to this block (the whole block will be **passively** refreshed)
 - *active refresh*: depends on refresh schemes (full-refresh, dirty-refresh, N -refresh)
- So, within the retention time T , if there is a passive refresh or an active refresh, there is no data loss.

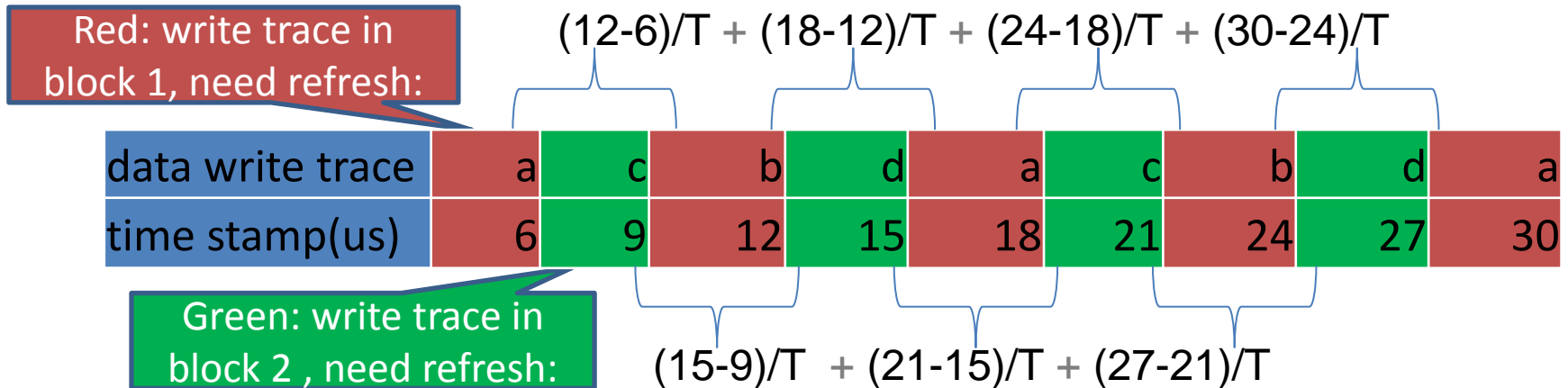
When do we need an *active refresh* ?

- Given a data write trace, *Dtrace*, with timing information

data write trace	a	c	b	d	a	c	b	d	a
time stamp(us)	6	9	12	15	18	21	24	27	30

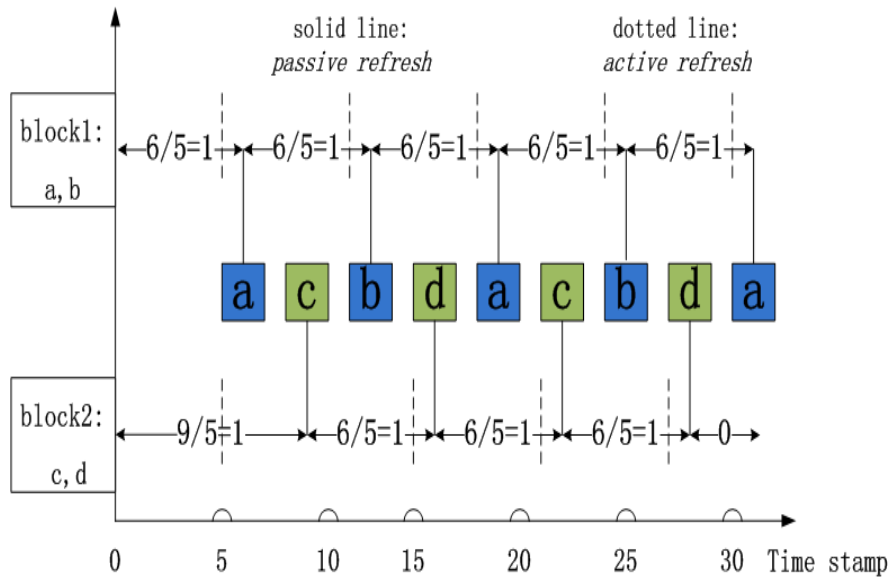
allocation : {a,b}, {c,d}
retention time **T**: 5

- Transform the data write trace into a block write trace: is the interval between two writes less than **T**?

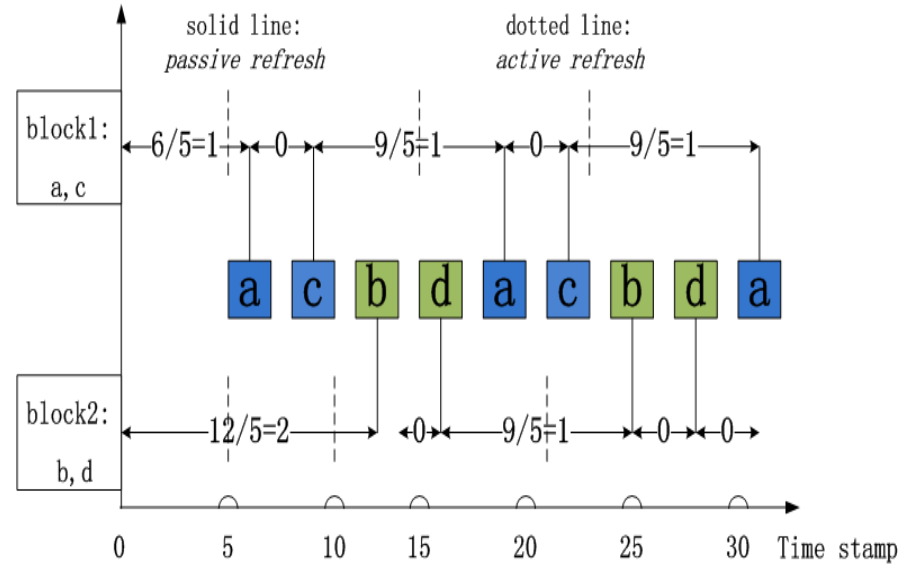


Reduce *active refresh* by data layout

data write trace	a	c	b	d	a	c	b	d	a
time stamp(us)	6	9	12	15	18	21	24	27	30



allocation 1: {a,b}, {c,d}
Totally 9 *active refresh*.



allocation 2: {a,c}, {b,d}
Totally 6 *active refresh*.

We found that, by re-arrange the data layout, we can change the behavior of *passive refresh* such that, the required number of *active refresh* is minimized.

Outline

- Background
- Observation
- **Solution**
- Experiments

Problem

- Given a data write trace, $Dtrace$, with timing information,
 - **allocate** data objects into memory blocks
 - **transform** the data write trace into a block write trace, $Btrace$, by mapping the data to the owner memory blocks
 - **split** $Btrace$ into a set of sub-traces, where each sub-trace recording writes to the same block
 - **compute** the required number of *active refresh* for each sub-trace

$$nRfr_{b^i} = \sum_{j=1}^N \frac{\hat{a} \hat{e} (TS_{bw_{j+1}^i} - TS_{bw_j^i})}{\hat{e} T_{\hat{u}} + \hat{e} (TS_{bw_1^i} - TS_{begin})} + \frac{\hat{e} (TS_{end} - TS_{bw_N^i})}{\hat{e} T_{\hat{u}}}$$

- Solution: find an allocation to minimize: $\sum_i nRfr_{b^i}$

Problem (more details)

- How to **compute** the required number of *active refresh* for each sub-trace

$$nRfr_{b^i} = \sum_{j=1}^N \left\lfloor \frac{(TS_{bw_{j+1}^i} - TS_{bw_j^i})}{T} \right\rfloor + \left\lfloor \frac{(TS_{bw_1^i} - TS_{begin})}{T} \right\rfloor + \left\lfloor \frac{(TS_{end} - TS_{bw_N^i})}{T} \right\rfloor$$

- T : the retention time
 - N : the number of writes to b
 - bw_j^i and bw_{j+1}^i : a pair of consecutive writes to b
 - TS_{bw} : the time stamp of a block write, bw
 - TS_{begin} & TS_{end} : the starting/ending time of the program
- the objective:

$$\min \sum_i nRfr_{b^i}$$

This problem is hard to solve.
Need simplification!

Simplified problem

- Given a data write trace, *Dtrace*, with timing information,
 - extract** a set of sub-traces from *Dtrace*, where each sub-trace recording writes to a pair of data objects

(a,b) sub-trace	a	b	a	b	a
time stamp(us)	6	12	18	24	30

(b,c) sub-trace	c	b	c	b
time stamp(us)	9	12	21	24

(a,c) sub-trace	a	c	a	c	a
time stamp(us)	6	9	18	21	30

(b,d) sub-trace	b	d	b	d
time stamp(us)	12	15	24	27

(a,d) sub-trace	a	d	a	d	a
time stamp(us)	6	15	18	27	30

(c,d) sub-trace	c	d	c	d
time stamp(us)	9	15	21	27

Simplified problem

- For each sub-trace corresponding to objects x and y , if assigning x and y into the same block, the cost is approximated as:

$$\text{cost}_{x,y} = \sum_{k=1}^M \left[\left(TS_{dw_{k+1}} - TS_{dw_k} \right) / T \right]$$

- M : the number of data writes in the sub-trace
- dw_k and dw_{k+1} : a pair of consecutive writes to x or y

otherwise, the cost is zero.

- The goal is to find an allocation (mapping data to memory blocks) to minimize:

$$\sum_{i \in \text{Data}} \sum_{\substack{j \in \text{Data} \\ j \neq i}} \text{cost}_{i,j} \cdot x_{i,j}$$

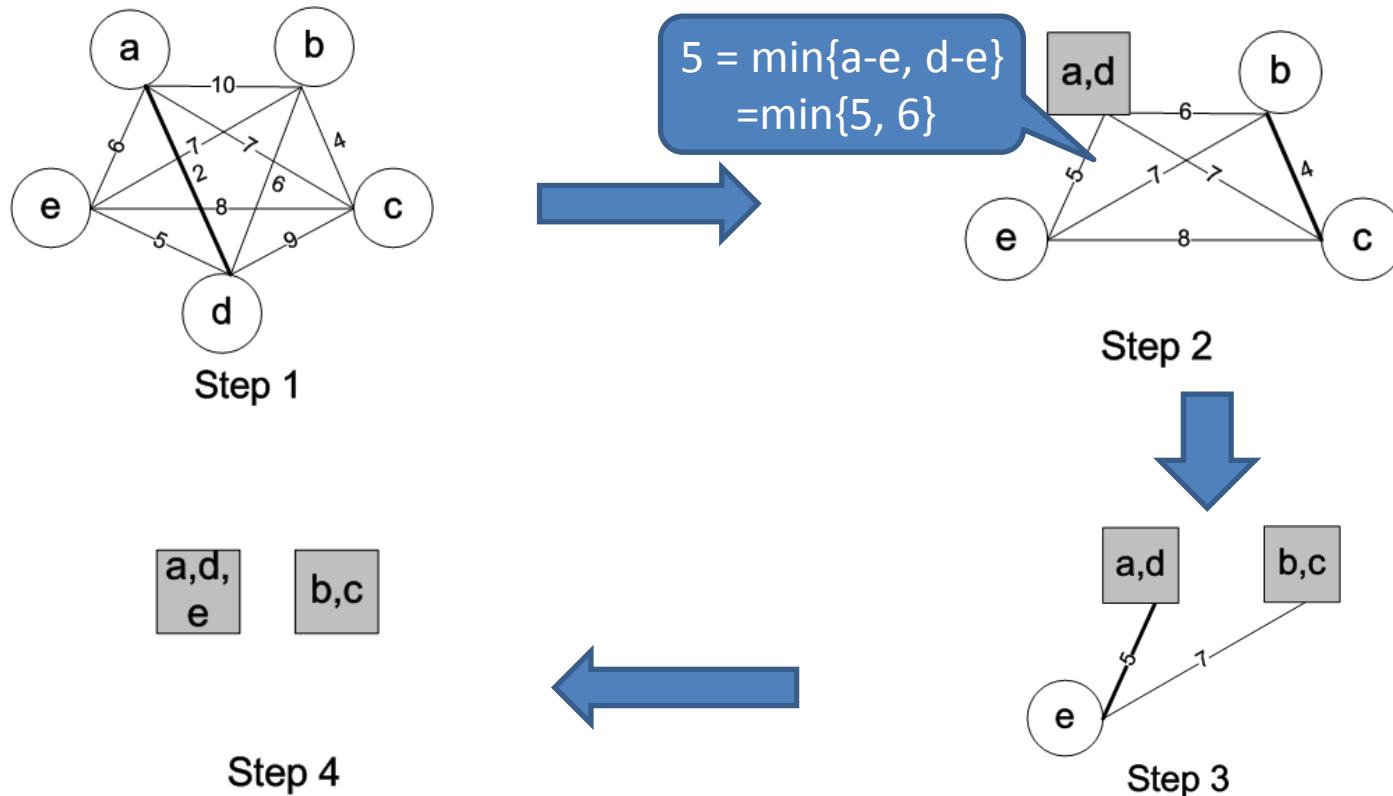
$x_{i,j}$ is a binary variable to indicate whether i and j are allocated into the same block

Heuristic algorithm

- This simplified problem can be easily modeled as a quadratic assignment problem (QAP), which is NP-hard in general.
- A heuristic algorithm to solve this simplified problem
 1. Encode the problem using a complete graph G : encode each data object as the vertex, and $cost_{i,j}$ as the edge weight
 2. Allocate a pair of objects into the same block, if the related edge weight is minimum among all edges of G
 3. Update G by merging this pair of objects
 4. If the remaining graph is not empty, go to 2

Heuristic algorithm

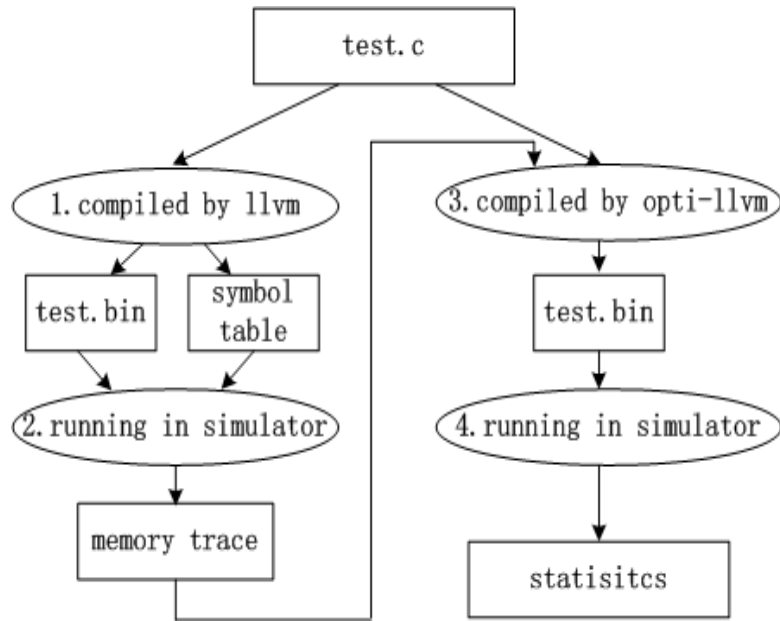
- $cost_{i,j}$ is encoded in a complete graph, as the edge weight
- assume each block can hold only three objects
- merge function: $\min \{edge1, edge2, \dots\}$



Outline

- Background
- Observation
- Solution
- **Experiments**

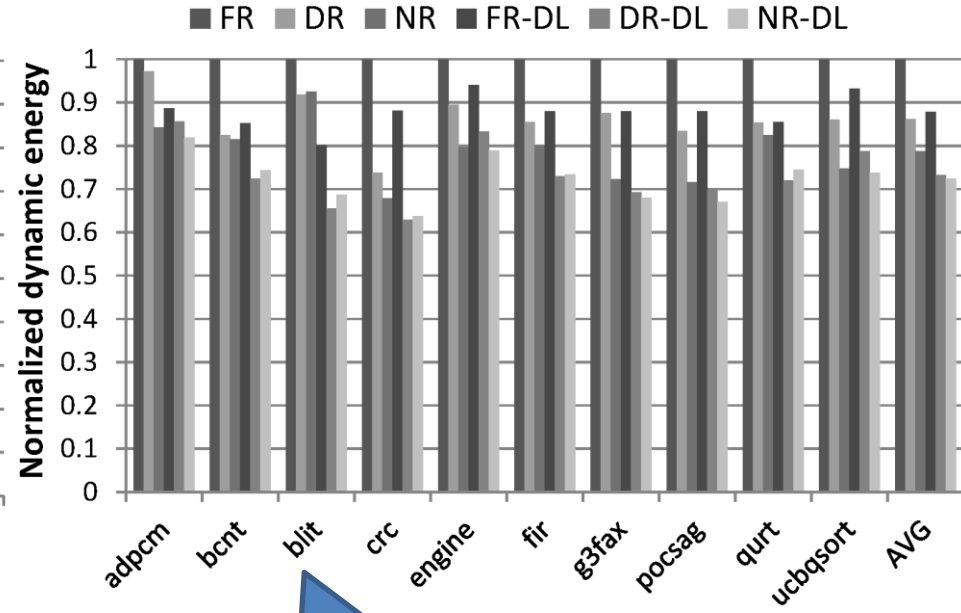
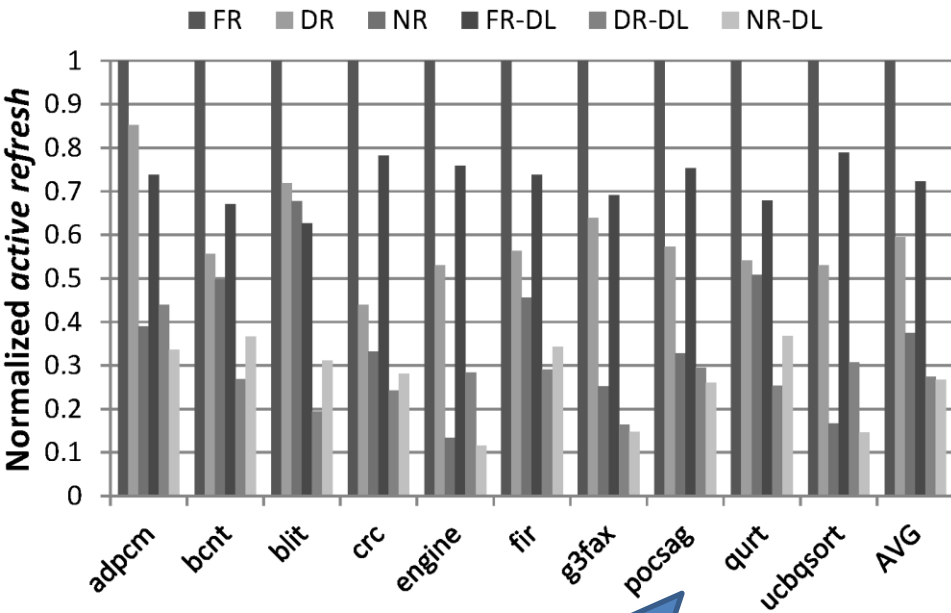
Experimental setup



Parameter	Value
processor	single core, in order execution, 2 GHz
data cache	16KB, 32B line size, LRU, 4-way write allocation, write back
	read/write latency: 2/4 cycles
	retention time: 53000 cycles (26.5 μ s)
	read/write dynamic energy: 0.035/0.187 nJ refresh dynamic energy: 0.356 nJ
main memory	300 cycles latency

	Full-refresh	Dirty-refresh	N-refresh
Default data layout	FR	DR	NR
Proposed data layout	FR-DL	DR-DL	NR-DL

Normalized to FR: Energy

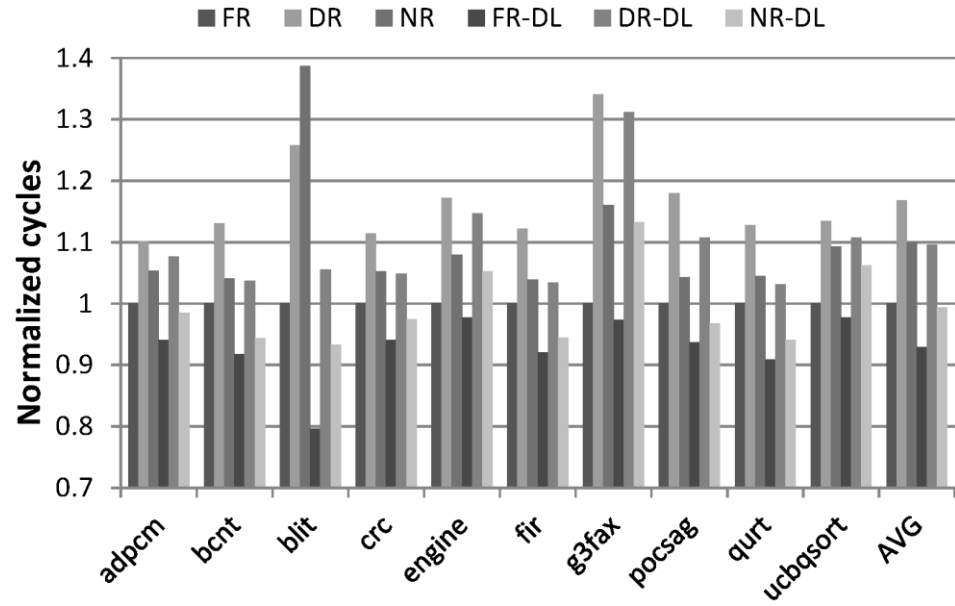
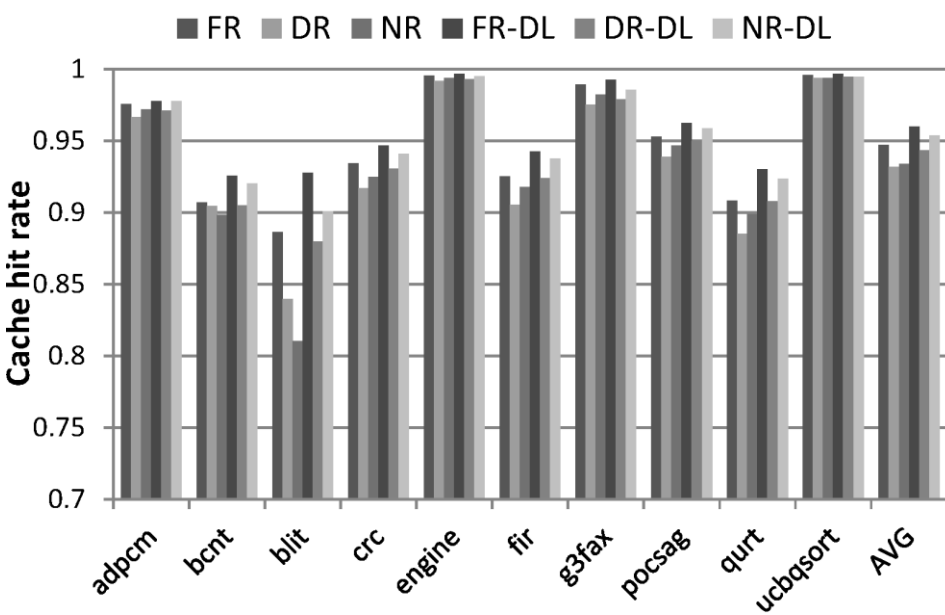


Reduces active refresh:
 FR-DL: 27.8%
 DR-DL: 72.7%
 NR-DL: 73.3%

Reduces dynamic energy:
 FR-DL: 12.1%
 DR-DL: 26.8%
 NR-DL: 27.6%

Proposed data layout	FR-DL	DR-DL	NR-DL
Refresh	FR-DL	DR	NR
Dirty-refresh	FR-DL	DR-DL	NR-DL

Normalized to FR: Performance



Increase cache hit rates:

- FR-DL: 1.3%
- DR-DL: -0.6%
- NR-DL: 0.7%

Reduce cycles:

- FR-DL: 8.1%
- DR-DL: -9.6%
- NR-DL: 0.6%

data layout	FR-DL	DR	DR-DL	Dirty-refre

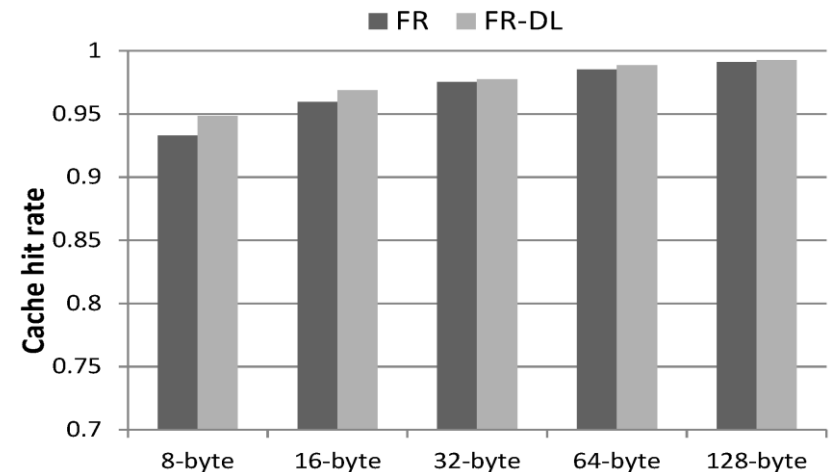
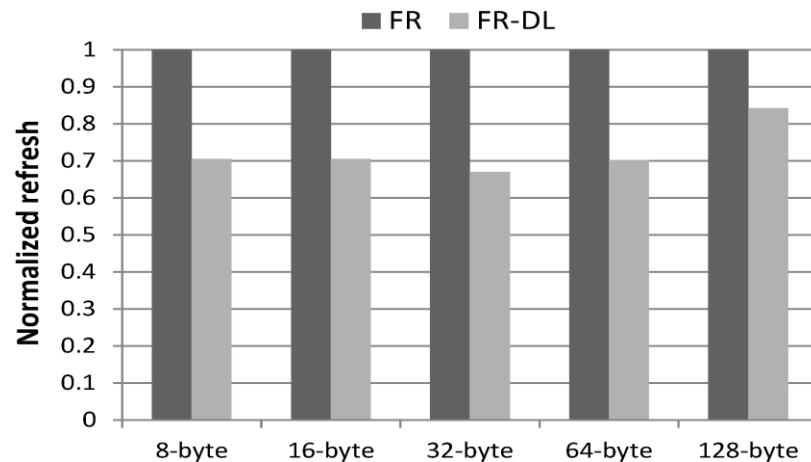
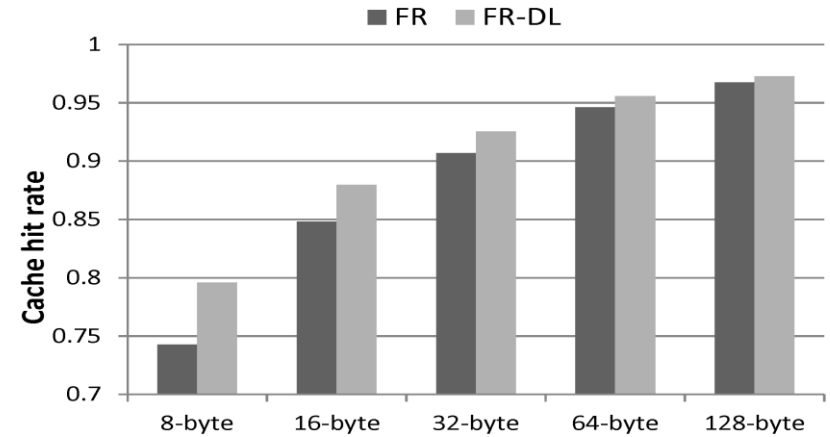
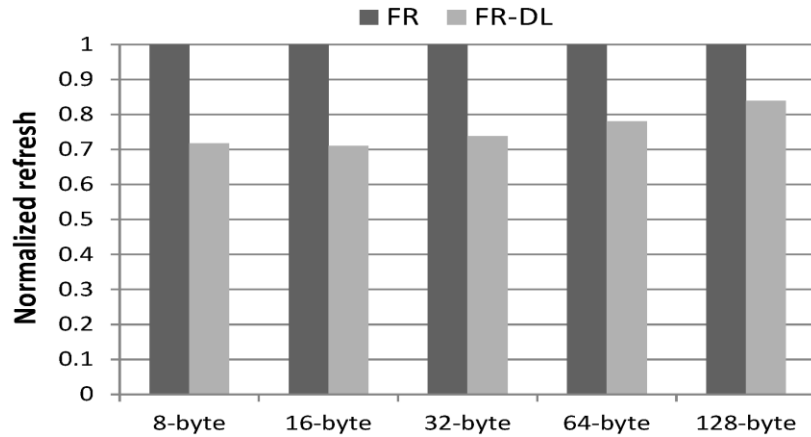
Conclusion

- Volatile STT-RAM cache has been proposed for better performance. But it requires refresh schemes to avoid data loss. And, refresh schemes bring about additional energy overhead.
- This paper proposes a compilation approach to minimize the refresh overhead by re-arranging data layout.
- Experiments show that, this approach can reduce refresh operations (73.3%), energy consumption (27.6%), and cycles (0.6%).

Thank you!

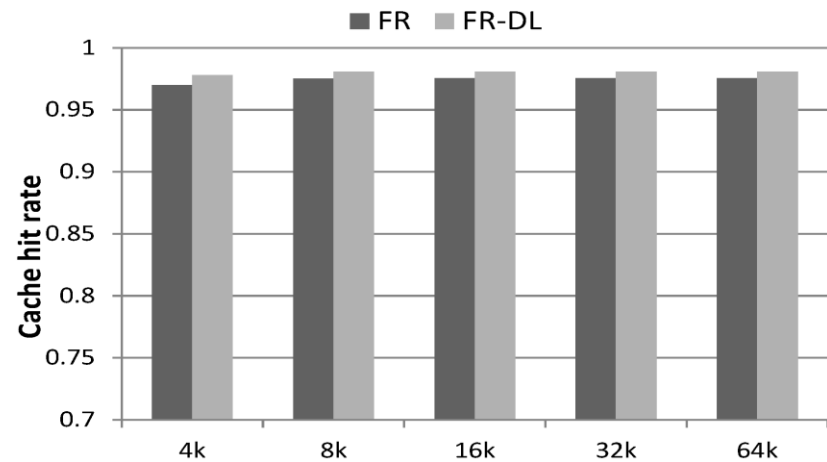
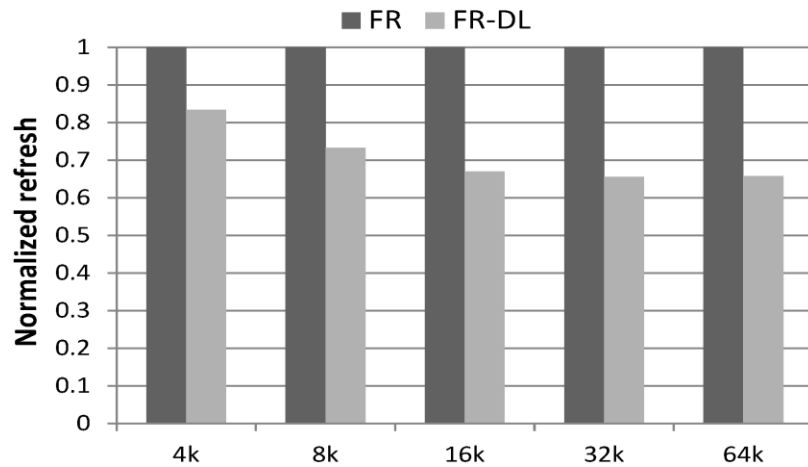
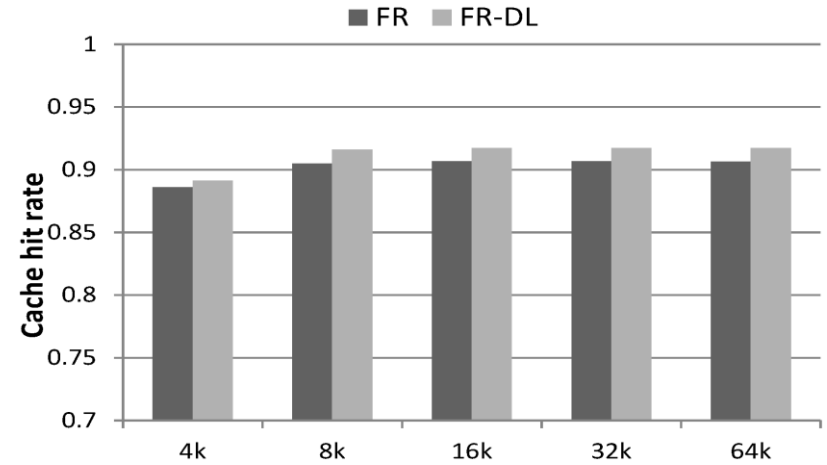
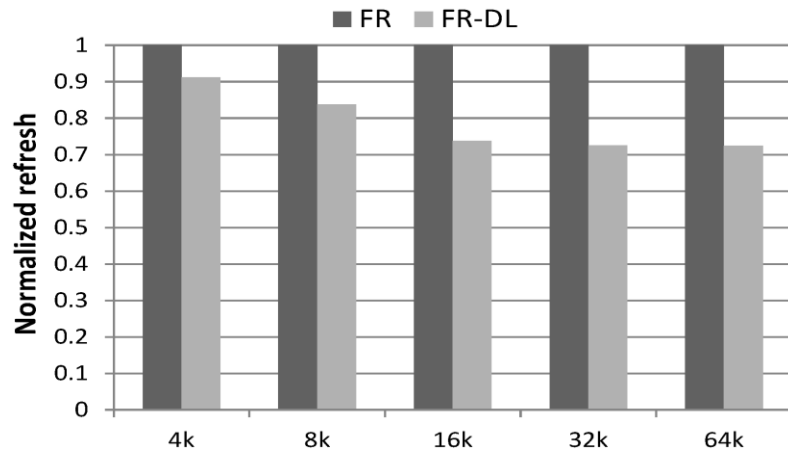
Questions?

Sensitiveness to different cache block size



For *adpcm* and *bcnt*, the proposed method can consistently reduce the *active refresh*, and increase cache hit rates, for varied cache block size.

Sensitiveness to different cache size



For *adpcm* and *bcnt*, the proposed method can consistently reduce the *active refresh*, and increase cache hit rates, for varied cache size.