Non-Volatile Memories are Promising

NVMs (STT-RAM, PCM, ReRAM, etc)

- Low density
- High leakage
- Volatile

- High density
- Low leakage
- Non-volatile
A 3.3ns-Access-Time 71.2μW/MHz 1Mb Embedded STT-MRAM Using Physically Eliminated Read-Disturb Scheme and Normally-Off Memory Architecture

Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Naoharu Shimomura, Jiro Hiroyuki Hara,

Test Chip for STT-MRAM Cache

This work was partly supported by the Ministry of Education, Culture, Sports, Science and Technology.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>65-nm CMOS 47-nm MTJ</td>
</tr>
<tr>
<td>Macro size</td>
<td>0.8196mm²</td>
</tr>
<tr>
<td>Capacity</td>
<td>1Mb</td>
</tr>
<tr>
<td>I/O width</td>
<td>256bits</td>
</tr>
<tr>
<td>Read speed</td>
<td>3.3ns</td>
</tr>
<tr>
<td>Write speed</td>
<td>3.0ns</td>
</tr>
<tr>
<td>Read energy</td>
<td>71.2μJ/MHz</td>
</tr>
<tr>
<td>Write energy</td>
<td>166.2μJ/MHz</td>
</tr>
</tbody>
</table>
Asymmetric R-W Energy Introduces New Challenges

Writes consume higher energy than reads
Asymmetric R-W Energy Introduces New Challenges

Impact of write energy cannot be ignored

Innovative cache management policies are required
Mitigating Write Energy in NVMs

- Bypass dead write [HPCA'14 etc]
- Bit-level [MICRO'09 etc]
- Hybrid cache [ISCA'09 etc]
- Data migration [HPCA'14 etc]

Inclusion property (non-inclusion)

Energy Efficient?
Existing Cache Inclusion Properties

Writes to LLC:
- **LLC miss + Dirty Victim**

Non-inclusion (noni)

- L2
- LLC
- Memory
- Clean victim
- Dirty victim
- LLC hit
- Back-invalidate

Writes to LLC:
- **Clean + Dirty Victim**

Exclusion (ex)

- L2
- LLC
- Memory
- Clean victim
- Dirty victim
- LLC hit

Larger effective capacity

Variant inclusion properties incur variant # writes

Write traffic need to be considered for NVMs
Why a New Inclusion Property could Benefit NVM?

- Exclusion is more energy efficient in SRAM LLC
- Since leakage power dominates the energy consumption
Why a New Inclusion Property could Benefit NVM?

- Exclusion is more energy efficient in SRAM LLC
  - Since leakage power dominates the energy consumption
- No dominant inclusion property in NVM LLC
  - Since impact of high write energy cannot be ignored

![Chart showing normalized STT-RAM LLC EPI for different benchmarks with exclusion and non-inclusion properties.]

- Favor exclusion
  - Zeusmp
  - GemsFDTD
  - Omnetpp
- Favor non-inclusion
  - Xalancbmk
Why a New Inclusion Property could Benefit NVM?

• Exclusion is more energy efficient in SRAM LLC
  • Since leakage power dominates the energy consumption
• No dominant inclusion property in NVM LLC
  • Since Impact of high write energy cannot be ignored

Need a design that captures workload behavior
Naive Solution: Dswitch

- **Dswitch**: Dynamic inclusion switching
  - Change inclusion property based on the workload behavior

![Diagram showing relative write traffic vs. relative LLC misses](image)

- **Exclusion saves more write energy**
- **Exclusion saves more leakage energy**
- **Exclusion provides better performance**
Is There a Better Approach?

- Dswitch can only select from either non-inclusion or exclusion for the entire cache
- We can exploit the strength of existing properties and enable cache-line-level management
Is There a Better Approach?

Non-inclusion (noni):
- Clean victim
- Dirty victim
- LLC hit

Exclusion (ex):
- Clean victim
- Dirty victim
- LLC hit

Memory

L2

LLC
Is There a Better Approach?

Non-inclusion (noni):
- Redundant data-fill
- Duplicate all data
- No clean insertion

Exclusion (ex):
- No clean insertion
Is There a Better Approach?

Non-inclusion (noni)

- Redundant data-fill
- Duplicate all data
- No clean insertion

Exclusion (ex)

- No data-fill
- No duplicate data
- Redundant clean insertion
Is There a Better Approach?

Increase effective capacity
Reduce redundant data fill

Memory
Non-inclusion (noni)

L2
LLC hit
Clean victim
Dirty victim

L2
LLC hit
Clean victim
Dirty victim

L2
LLC hit
Clean victim
Dirty victim

Memory

Non data-fill
Duplicate only useful data

No data-fill
No duplicate data

Redundant clean insertion

Redundant data-fill
Duplicate all data

No clean insertion
Is There a Better Approach?

Increase effective capacity
Reduce redundant data fill

Reduce redundant clean insertion

Non-inclusion (noni)

Our target

Exclusion (ex)

- Redundant data-fill
- Duplicate all data
- No clean insertion

- No data-fill
- Duplicate only useful data
- Insert only non-duplicate data
- Redundant clean insertion
Is There a Better Approach?

Increase effective capacity
Reduce redundant data fill

Reduce redundant clean insertion

Which clean insertion is redundant?

Which clean data should be kept in the LLC?

Non-inclusion (noni)

Redundant data-fill
Duplicate all data
No clean insertion

No data-fill
Duplicate only useful data
Insert only non-duplicate data

Our target

No data-fill
No duplicate data

Exclusion (ex)

Redundant clean insertion
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches.
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches.

**Non-inclusion:**

**Exclusion:**
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches

### Non-inclusion:

```
+----------------+  +----------------+  +----------------+
|      L2       |  |      L2       |  |      L2       |
|  Clean        |  |  Clean        |  |  Clean        |
|     ↓         |  |     ↓         |  |     ↓         |
| LLC          |  | LLC          |  | LLC          |
|   A B        |  |   A B        |  |   A B        |
```

### Exclusion:

```
+----------------+  +----------------+  +----------------+
|      L2       |  |      L2       |  |      L2       |
|  Clean        |  |  Clean        |  |  Clean        |
|     ↓         |  |     ↓         |  |     ↓         |
| LLC          |  | LLC          |  | LLC          |
|   A B        |  |   A B        |  |   A B        |
```

- **B** is written at L2
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches

**Non-inclusion:**

- L2 Clean to LLC Clean
- L2 Dirty to LLC Dirty

**Exclusion:**

- L2 Clean to LLC Clean (no loop-block)
- L2 Dirty to LLC Dirty (loop-block: redundant write)

B is written at L2

L2 Hit to LLC Clean

L2 Hit to LLC Empty
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches.

**Non-inclusion**:  
- Data moves from LLC to L2: **Clean** → **Dirty**

**Exclusion**:  
- Data moves from L2 to LLC: **Clean** → **Dirty**

**loop-block**: redundant write  
**non-loop-block**: waste capacity
Loop-Blocks & Non-Loop-Blocks

- **Loop-blocks**: clean data that are not modified during its trip between LLC and upper-level caches.

**Non-inclusion:**

**Exclusion:**

Loop-block distribution: average 20%, up to 80%

Keeping loop-blocks in LLC can reduce redundant writes

Evicting non-loop-blocks earlier can increase LLC capacity

non-loop-block: waste capacity

loop-block: redundant write

B is written at L2
LAP: Loop-Block-Aware Inclusion Property

• **Loop-block prediction**
  • A loop-block is highly possible to continue its loop-block behavior
  • Add **one loop-bit** per block in L2 and LLC for prediction
LAP: Loop-Block-Aware Inclusion Property

• **Loop-block prediction**
  - A loop-block is highly possible to continue its loop-block behavior
  - Add **one loop-bit** per block in L2 and LLC for prediction
LAP: Loop-Block-Aware Inclusion Property

- Loop-block prediction
  - A loop-block is highly possible to continue its loop-block behavior
  - Add **one loop-bit** per block in L2 and LLC for prediction

![Diagram showing loop-block prediction]

- is written at L2
- L2
- 0 0 1
- Hit
- LLC

- is written at L2
LAP: Loop-Block-Aware Inclusion Property

• **Loop-block prediction**
  • A loop-block is highly possible to continue its loop-block behavior
  • Add *one loop-bit* per block in L2 and LLC for prediction
LAP: Loop-Block-Aware Inclusion Property

- Loop-block prediction
  - A loop-block is highly possible to continue its loop-block behavior
  - Add **one loop-bit** per block in L2 and LLC for prediction

![Diagram showing L2 and LLC with loop-block prediction logic]

```
Replacement policy:
1. invalid
2. non-loop-block
3. loop-block
```

- No LLC data-fill
- No invalidation
Lhybrid: LAP for Hybrid LLCs

• Loop-blocks can guide the data placement policy

Store loop-blocks in NVM can reduce writes to NVM

Store non-loop-blocks in SRAM as they are frequently written
Lhybrid: LAP for Hybrid LLCs

- Loop-blocks can guide the data placement policy

**Store loop-blocks in NVM can reduce writes to NVM**

**Store non-loop-blocks in SRAM as they are frequently written**

Loop-block exists in SRAM
Lhybrid: LAP for Hybrid LLCs

- Loop-blocks can guide the data placement policy

Store loop-blocks in NVM can reduce writes to NVM

Store non-loop-blocks in SRAM as they are frequently written

- Loop-block exists in SRAM

[Diagram showing L2 cache, SRAM, and NVM with loop-blocks and non-loop-blocks]
Lhybrid: LAP for Hybrid LLCs

- Loop-blocks can guide the data placement policy

**Store loop-blocks in NVM can reduce writes to NVM**

**Store non-loop-blocks in SRAM as they are frequently written**

- Loop-block exists in SRAM
- No loop-blocks in SRAM
Experimental Setup

- **Simulator**: gem5, NVSim
- **Benchmarks**:
  - Multi-programmed: SPEC2006
  - Multi-threaded: PARSEC
- **Policies**
  - non-inclusion & exclusion
  - FLEXclusion[ISCA’12] & Dswitch

**Hierarchy Representation**:

- L1: 32KB/core
- L2: 512KB/core
- L3: 8MB STT-RAM

**Workload Characteristics**:

- **WH workloads**: writes: $ex > noni$
- **WL workloads**: writes: $ex <= noni$
Result: Energy Efficiency

- LAP is more energy efficient than existing policies
  - 20% and 12% lower EPI than non-inclusion and exclusion
Result: Energy Efficiency

- **LAP is more energy efficient than existing policies**
  - 20% and 12% lower EPI than non-inclusion and exclusion
  - Consume less energy than SRAM-based selective inclusion approach and Dswitch

![Bar chart showing energy efficiency comparison](image)

- **Normalized LLC EPI**
  - Non-inclusion
  - Exclusion
  - FLEXclusion
  - Dswitch
  - LAP

- **Normalized LLC EPI** for different metrics:
  - AvgWL
  - AvgWH
  - AvgAll

- **Comparison Metrics**:
  - AvgWL: 17%
  - AvgWH: 10%
Result: Energy Efficiency

- LAP also saves energy in hybrid LLCs
- Lhybrid can further reduce the energy of hybrid LLCs
Summary

- No existing inclusion property is energy-efficient in NVM LLCs with asymmetric read-write energy

- Propose a new loop-block aware inclusion property to
  - Reduce redundant insertion of clean data in exclusion
  - Reduce redundant data-fill and provide larger effective capacity than non-inclusion

- The loop-block characteristic can also help to guide better data placement in hybrid LLCs

Average results in all workloads:
EPI savings: 20% over non-inclusion; 12% over exclusion
Performance improvement: 2% over exclusion
Scalability to W/R Energy

- LAP continues to provide energy benefits at variant W/R energy ratios
  - W/R energy ratio is the key factor that impacts the amount of energy savings provided by LAP

LAP can be applied broadly across other asymmetric memory technologies
LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches

Hsiang-Yun Cheng, Jishen Zhao, Jack Sampson
Mary Jane Irwin, Aamer Jaleel, Yu Lu, Yuan Xie