Silent Shredder: Zero-Cost Shredding For Secure Non-Volatile Main Memory Controllers

Amro Awad (NC State University)  
Pratyusa Manadhata (Hewlett Packard Labs)  
Yan Solihin (NC State University)  
Stuart Haber (Hewlett Packard Labs)  
William Horne (Hewlett Packard Labs)
Outline

- Background
- Related Work
- Goal
- Design
- Evaluation
- Summary
Outline

- Background
- Related Work
- Goal
- Design
- Evaluation
- Summary
Emerging NVMs

Emerging NVMs are promising replacements for DRAM.
- Fast (comparable to DRAM).
- Dense.
- Non-Volatile: persistent memory, no refresh power.

Examples:
- Phase-Change Memory (PCM).
- Memristor.

Source: http://www.techweekeurope.co.uk/
Emerging NVMs

+ NVMs have their drawbacks:
  + Limited endurance (e.g., PCM has ~$10^8$ writes per cell).
  + Slow writes (e.g., PCM has ~150ns write latency).
  + Data Remanence attacks are easier!

+ Requirements for using NVMs:
  + Encrypt Data.
  + Reduce number of writes, e.g., DCW and Flip-N-Write

Encryption reduces efficiency of DCW and Flip-N-Write
Data Shredding

Data Shredding: The operation of zeroing out memory to avoid data leak.

- It prevents data leak between processes or virtual machines.
- Expensive:
  - Up to 40% of page fault time could be spent in zeroing pages.
  - For tested graph analytics apps, about 41.9% of memory writes could result from shredding.
Example of Data Shredding

1- Request allocation

2- Zero out

3- Request allocation

4- Zero out

VM

NVM
How to implement shredding?

<table>
<thead>
<tr>
<th>Technique</th>
<th>No cache pollution</th>
<th>Low-processor time</th>
<th>No Bus Traffic (indirectly)</th>
<th>No Memory Writes</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular stores</td>
<td>✗</td>
<td>✗</td>
<td>✗ (indirectly)</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Non-Temporal Stores</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>DMA-Support Non-Temporal Bulk Zeroing [Jiang, PACT09]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>RowClone (DRAM specific) [Shehadri, MICRO 2013]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Can we shred without writing?
Threat Model

- Physical access to the memory.
- Snoop memory bus.
Encryption/Decryption Process

- Encryption/Decryption: CTR-mode.

  + The IV must change every time you encrypt new data.
  + Key insight: IV used for encryption = IV used for decryption.
We use Split-Counter Scheme [C. Yan, ISCA 2006]:

4KB Page (64 Cache lines)

- Cache line 0: 512-bits
- Cache line 1: 512-bits
- Cache line 63: 512-bits

- Major (per page)
- Minor
- Cache line address
- Padding

Initialization Vectors
Typical Shredding

Non-temporal Bulk Shredding

Zero Page X → Encryption/Decryption → Write encrypted → NVM

Read & update counters → Counter Cache

Page X
Our Proposal: Silent Shredder

- Key idea: instead of zeroing shredded page, make it unintelligible
  - By changing the key or IV prior to decryption

- Design options:
  - Have a key for every process
    - Impractical: the memory controller needs to know process ID.
    - Shared data requires same key.
  - Increment all minor counters of a page
    - Increases re-encryption frequency: minor counters will overflow faster.
  - Increment the major counter
To achieve software compatibility, would like to have zero cache lines for new/shredded pages.

Shredding: Increment major counter and zero all minor counters.

Zero-filled cache lines are returned for zeroed minor counters.

When minor counter overflows, it starts from 1.
1. Shred $p$

3. Increment $M$
   reset $m1 \ldots m64$

5. Done

2. Invalidate $p$

4. Acknowledge

Design
Design

1. Miss x

2. Read the minor counter of the block x

3a. No: fetch x

3b. Yes

=0?

4. Return the fetched block
   Or a zero-filled block

Counter Cache

Minor counters

Tag

Major Ctr

LLC

MUX

D_k

MC

NVMM
Evaluation Methodology

- To evaluate our design, we use **Gem5** to run a modified kernel.
  - Added shred command to execute inside kernel’s `clear_page` function.

- **Baseline** uses non-temporal stores bulk zeroing.

- We use multi-programmed workloads from SPEC 2006 and PowerGraph suites.

- Warm up 1B then run 500M instructions on each core (~4B overall) from initialization and graph construction phases.

- We assume battery-backed Counter Cache.
## Configurations

<table>
<thead>
<tr>
<th>Processor</th>
<th>CPU</th>
<th>8-Cores, X86-64, 2GHz clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L1 Cache</td>
<td>2 cycles, 64KB size, 8-way, LRU, 64B block size</td>
</tr>
<tr>
<td></td>
<td>L2 Cache</td>
<td>8 cycles, 512KB size, 8-way, LRU, 64B block size</td>
</tr>
<tr>
<td></td>
<td>L3 Cache</td>
<td>Shared, 25 cycles, 8MB size, 8-way, LRU, 64B block size</td>
</tr>
<tr>
<td></td>
<td>L4 Cache</td>
<td>Shared 35 cycles, 64MB size, 8-way, LRU, 64B block size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main Memory (NVM)</th>
<th>Capacity</th>
<th>16GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Channels</td>
<td>2 channels</td>
</tr>
<tr>
<td></td>
<td>Channel bandwidth</td>
<td>12.8 GB/s</td>
</tr>
<tr>
<td></td>
<td>Read/Write latency</td>
<td>75ns/150ns</td>
</tr>
<tr>
<td></td>
<td>IV Cache</td>
<td>10 cycles, 4MB capacity, 8-way associativity, 64B blocks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th>OS</th>
<th>Gentoo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kernel</td>
<td>3.4.91</td>
</tr>
</tbody>
</table>
Characterization

Shredding Rate

# Shreds/Million Instructions

Benchmark
Results

48.6% write reduction
44.6% (very high shredding)

50.3% read traffic reduction
46.5% (Very high shredding)
Results

3.3x reads speed up
2.8x (very high shredding)

6.4% IPC Improvement
19.3% (very high shredding)
Other Use Cases

- Bulk zeroing: Silent Shredder can be used for initializing large areas.
- Large-Scale Data Isolation: Fast data shredding for isolation across VMs or isolated nodes.
- Fast and efficient virtual disk provisioning when using byte-addressable NVM devices.
- Garbage collectors in managed programming languages.
Summary

- We eliminate writes due to data shredding.
- Our scheme is based on manipulating IV values.
- Silent Shredder leads to write reduction and performance improvement.
- Applicable to other cases.
Thanks!

Questions
Encryption Assumption

+ Encryption: CTR-mode.
+ Same IV should never be reused for encryption.
+ OTP generation doesn’t need the data.
Any IV-based encryption scheme needs to guarantee the following:

- **Counter Cache Persistency**
  - Counters must be kept persistent either by battery-backed, using write-through cache or using NVM-based counter cache.

- **IVs’ and Data Integrity**
  - IVs and Data must be protected from tampering/replaying.
  - Authenticated encryption, e.g., Bonsai Merkle Tree, can be used.
Backup slides
Costs of Data Shredding

- Increasing overall number of main memory writes.
  - Our experiments showed that up to 42% of main memory writes can result from shredding.