Consistent, Durable, and Safe Memory Management for Byte-Addressable Non-Volatile Main Memory

Iulian Moraru*, David Andersen*, Michael Kaminsky°, Niraj Tolia‡, Nathan Binkert†, Partha Ranganathan

*Carnegie Mellon University, °Intel Labs, ‡Maginatics, †Amiato, ◇Google
New memory technologies
New memory technologies

NVRAM: phase change, spin torque, memristor

• Memory that is both fast and non-volatile
• Will help build fast, robust systems
New memory technologies

NVRAM: phase change, spin torque, memristor

• Memory that is both fast and non-volatile
• Will help build fast, robust systems

NVRAM much faster than HDD and Flash
New memory technologies

NVRAM: phase change, spin torque, memristor

- Memory that is both fast and non-volatile
- Will help build fast, robust systems

NVRAM much faster than HDD and Flash

![Diagram showing access time comparison between DRAM, Flash, and access time in ns (logscale)]
New memory technologies

NVRAM: phase change, spin torque, memristor
- Memory that is both fast and non-volatile
- Will help build fast, robust systems

NVRAM much faster than HDD and Flash
New memory technologies

NVRAM: phase change, spin torque, memristor

- Memory that is both fast and non-volatile
- Will help build fast, robust systems

NVRAM much faster than HDD and Flash

- Placed on the memory bus (alongside DRAM)
New memory technologies

NVRAM: phase change, spin torque, memristor

• Memory that is both fast and non-volatile
• Will help build fast, robust systems

NVRAM much faster than HDD and Flash

- Placed on the memory bus (alongside DRAM)
- Needs new ways of handling persistent state
NV main memory - assumptions

- Mapped into process address space
- Accessed through CPU loads/stores
- Persistent namespace
NV main memory - assumptions

- Mapped into process address space
- Accessed through CPU loads/stores
- Persistent namespace
NVDRAM - challenge

High performance and Robustness (no data loss)

→ Give developers the right tools to develop data structures for NVRAM
Sources of data loss
Sources of data loss

Wear-out
Sources of data loss

Wear-out

Erroneous writes
Sources of data loss

Wear-out

Erroneous writes

CPU caches
Sources of data loss

Wear-out

Erroneous writes

CPU caches

...
Sources of data loss

Wear-out → Memory allocator

Erroneous writes

CPU caches

...
Sources of data loss

Wear-out

Erroneous writes

CPU caches

Memory allocator

Virtual memory protection
Sources of data loss

- Wear-out
- Erroneous writes
- CPU caches
- ...
Malloc for NVRAM - goals
Malloc for NVRAM - goals

• Avoid frequently re-allocating the same block

• HW wear leveling may slow down applications that write often to one location
Malloc for NVRAM - goals

- Avoid frequently re-allocating the same block
- HW wear leveling may slow down applications that write often to one location
- Minimize # of metadata updates in NVRAM
- Wear-out & speed
Malloc for NVRAM - goals

- Avoid frequently re-allocating the same block
- HW wear leveling may slow down applications that write often to one location

- Minimize # of metadata updates in NVRAM
  - Wear-out & speed

- Make metadata robust to accidental corruption
  - Avoids extensive loss of (persistent) data
Problems with allocators for DRAM

• Reuse recently freed blocks
• More writes than necessary
  • Metadata is embedded in allocated / free blocks
  • Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location

3

7
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location

```
size
3
```
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location

![Diagram](image)
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location

size

\[ \begin{array}{c}
1 \\
2 \\
3 \\
\end{array} \]
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
- Metadata corruption can cause data loss
Problems with allocators for DRAM

• Reuse recently freed blocks
• More writes than necessary
  • Metadata is embedded in allocated / free blocks
  • Frequent writes to one location

• Metadata corruption can cause data loss
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated/free blocks
  - Frequent writes to one location
- Metadata corruption can cause data loss
Problems with allocators for DRAM

• Reuse recently freed blocks
• More writes than necessary
  • Metadata is embedded in allocated / free blocks
  • Frequent writes to one location

• Metadata corruption can cause data loss
Problems with allocators for DRAM

- Reuse recently freed blocks
- More writes than necessary
  - Metadata is embedded in allocated / free blocks
  - Frequent writes to one location
- Metadata corruption can cause data loss
NVMalloc

DRAM

NVRAM
NVmalloc

free / in-use bitmap

000000...

DRAM

NVVRAM
NVMalloc

free / in-use bitmap

010001...

DRAM

NVRAM
NVmalloc

**free / in-use bitmap**

```
010001...
```

**free lists**

- **size 1:**
  - [ ] [ ] ...

- **size 2:**
  - [ ] ...

- **size 3:**
  - [ ] ...

---

**DRAM**

---

**NVRAM**
NVmalloc

free / in-use bitmap

010001...

free lists

size 1:

size 2:

size 3:

DRAM

NVRAM

Sunday, November 3, 13
NVMalloc

free / in-use bitmap

010001...

free lists

size 1:

size 2:

size 3:

DRAM

NVRAM
NVMalloc

free / in-use bitmap

```
010001...
```

free lists

size 1:

size 2:

size 3:

DRAM

NVRAM

header:
state, size, checksum
NVMalloc

free / in-use bitmap

010001...

free lists

size 1:

size 2:

size 3:

DRAM

NVRAM

header:

state, size, checksum
NVMalloc

free / in-use bitmap

[010001...]

free lists

size 1:

size 2:

size 3:

header:
state, size, checksum
NVmalloc

free / in-use bitmap

010001...

free lists

size 1:

size 2:

size 3:

header:

state, size, checksum

don’t allocate list:
Our solution: NVMalloc

• Freed blocks ➔ “don’t allocate list”
  • At least T seconds (e.g. T = 0.2 s)
  • Block will not be allocated more often than 1/T

• Keep most metadata in DRAM
  • In-DRAM bitmap tracks blocks state
  • In-DRAM free lists accelerate allocations

• Checksum in each block header
  • Checksum over: state, size, location
Building blocks

Memory allocator

Virtual memory protection

Cache line counters
Building blocks

- Memory allocator
- Virtual memory protection
- Cache line counters
Problem: erroneous writes

Address space

- stack
- data
- code

Persistent data

File system interface

Narrow interface
Problem: erroneous writes

- stack
- Persistent data
- data
- code
Problem: erroneous writes

- stack
- Persistent data
- data
- code

erroneous overwrite
Problem: erroneous writes

Solution: virtual memory protection
Problem: erroneous writes

Solution: virtual memory protection
(if we can make it fast enough)
Virtual memory protection

• Today’s solution: mprotect
  • Used for protecting in-memory databases (since 1990’s)
  • Used today: in-memory databases, JVM, garbage collecting
Virtual memory protection

• Today’s solution: mprotect
  • Used for protecting in-memory databases (since 1990’s)
  • Used today: in-memory databases, JVM, garbage collecting

High performance penalties:
• synchronous
• syscall overhead
• one request at a time
Asynchronous mprotect

- No waiting
- No system call overhead
- Batching, sorting
  - Mprotect cheaper on ranges
Un-protect: possible scenarios

un-protect request

protect request

---

14
Un-protect: possible scenarios

- Un-protect request
- Protect request

Coalesce

Time
Un-protect: possible scenarios
Un-protect: possible scenarios

un-protect request

x

protect request

x

Coalesce

un-protect request

don un-protect

done

write

Async, if sufficiently in advance of write (e.g. GC)
Un-protect: possible scenarios

Async, if sufficiently in advance of write (e.g. GC)
Un-protect: possible scenarios

- Un-protect request
- Protect request
- Coalesce

Async, if sufficiently in advance of write (e.g. GC)

Handle page fault in kernel, transparently
Building blocks

Memory allocator

Virtual memory protection

Cache line counters
Building blocks

- Memory allocator
- Virtual memory protection
- Cache line counters
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
- Even more so for NVRAM
- No guaranteed order for cache line write-back
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
- Even more so for NVRAM
- No guaranteed order for cache line write-back
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
- Even more so for NVRAM
- No guaranteed order for cache line write-back
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
- Even more so for NVRAM
- No guaranteed order for cache line write-back
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
- Even more so for NVRAM
- No guaranteed order for cache line write-back

Diagram:

- Pointer may be flushed first

Sunday, November 3, 13
Write Caching for NVRAM

- CPU caching: important optimization for DRAM
  - Even more so for NVRAM
  - No guaranteed order for cache line write-back

- Goals:
  - Avoid forcing cache line flushes (costly)
  - Enforce ordering
Cache line counters

• Make applications aware of cache state

  How many cache lines updated by a transaction are still in cache?

• Enforce ordering in software
Evaluation

- Testing setup:
  - Core i7 860 (2.8 GHz), 8 GB DRAM
  - DRAM as proxy for NVRAM
Results: wear leveling

100K operations (10B-4KB, 50% deallocations)

Writes / 64 B block

Block number

malloc
NVMalloc

Writes / 64 B block

Sunday, November 3, 13
Results: wear leveling

100K operations (10B-4KB, 50% deallocations)

malloc concentrates writes & 30%-50% more writes overall than NVMAlloc
NVMalloc in B+ tree

1M insert operations

% of all 64 B blocks

Number of writes per 64 byte block (log scale)
Async. mprotect in B+ tree

1M inserts, 256B values

<table>
<thead>
<tr>
<th></th>
<th>Slowdown</th>
<th>Latency (median)</th>
<th>Latency (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>9.0x (ideal)</td>
<td>200ms</td>
<td>350ms</td>
</tr>
<tr>
<td>Async 200ms</td>
<td>1.7x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Async 0.1ms</td>
<td>7.9x</td>
<td>0.1ms</td>
<td>1.2ms</td>
</tr>
</tbody>
</table>
Async. mprotect in B+ tree

1M inserts, 256B values

- Slowdown (vs no mprotect)
  - Sync
  - Async 200ms: 1.7x slowdown
  - Async 0.1ms: 7.9x slowdown

Latency (median)
- Sync: 200ms
- Async 200ms: 0.1ms
- Async 0.1ms: 1.2ms

Latency (max)
- Sync: 350ms
- Async 200ms: 1.2ms
- Async 0.1ms: 1.2ms

different batching quanta

Sunday, November 3, 13
DRAM as NVRAM proxy

*How would our results change on NVRAM?*

- Same malloc / NVMalloc access patterns
  - But NVMalloc writes less frequently to NVRAM
- Same asynchronous mprotect overhead
  - But lower relative overhead b/c NVRAM is slower
Summary

• Challenges of NVRAM on the memory bus
  • Wear-out
  • Erroneous writes
  • CPU caches

• Building blocks for NVRAM data stores

<table>
<thead>
<tr>
<th>Persistent data stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVMalloc</td>
</tr>
<tr>
<td>Async mprotect</td>
</tr>
<tr>
<td>CLC</td>
</tr>
</tbody>
</table>
Code

github.com/efficient/nvram
Related Work


