All About Persistent Memory Flushing

Andy Rudoff, Intel
Steve Byan, Oracle
By the End of This Talk, You Will Know…

- Why flushing matters for persistent memory
  - And when you don’t have to worry about it
- The difference between:
  - Visibility and persistence
  - msync/fsync, FlushFileBuffers
  - Optimized Flush
  - x86 instructions: CLFLUSHOPT/CLWB
  - Deep Flush
  - Flushing to remote persistence
Why Flushing Matters
Why Store Barriers Matter
Why Store Barriers Matter

On crash recovery:
completion of transaction B
implies
completion of transaction A
How the Hardware Works

- MOV

Core

L1 L1
L2
L3

CPU CACHES

CLWB + fence
-or-
CLFLUSHOPT + fence
-or-
CLFLUSH
-or-
NT stores + fence
-or-
WBINVD (kernel only)

WPQ

ADR
-or-
WPQ Flush (kernel only)

Custom
Power fail protected domain
indicated by ACPI property:
CPU Cache Hierarchy

Minimum Required
Power fail protected domain:
Memory subsystem

Power fail protected domain:
Memory subsystem
How the Software Stack Works

- Management UI
  - Management Library
  - Standard Raw Device Access

- Application
  - Standard File API

- Application
  - Load/Store

- Application
  - Load/Store

- File System
  - Persistent Memory Aware File System
  - MMU Mappings

- NVDIMM Driver

- NVDIMM

“DAX”

User Space

Kernel Space
Flushing for Application Programmers

- Is the flushing requirement new for pmem?
  - Memory-mapped files have always worked this way:
    - Stores are not guaranteed persistent until flush API is called
    - Stores are visible before they are persistent

- Do standard flushing APIs work with pmem?
  - Yes, standard APIs work as expected
    - msync() on Linux
    - FlushFileBuffers() on Windows
    - The kernel will use instructions like CLWB as necessary

- Can Applications just flush with CLWB from user space
  - Only when supported by the kernel/file system
  - Libraries like NVML determine when it is safe
Definitions

- Standard Flush
  - msync/fsync/FlushFileBuffers
  - 30+ year-old interfaces, same semantics

- Optimized Flush
  - Optionally-supported, faster flushing
  - Can avoid locks, kernel calls, rounding up
More Definitions

- Deep Flush
  - Flush to smallest failure domain available to SW
  - Gives up performance for higher RAS
- Remote Flush
  - A store barrier for RDMA to pmem
<table>
<thead>
<tr>
<th>OS Detection of NVDIMMs</th>
<th>ACPI 6.0+</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Exposes pmem to apps</td>
<td>DAX provides SNIA Programming Model</td>
</tr>
<tr>
<td></td>
<td>Fully supported:</td>
</tr>
<tr>
<td></td>
<td>• Linux (ext4, XFS)</td>
</tr>
<tr>
<td></td>
<td>• Windows (NTFS)</td>
</tr>
<tr>
<td>OS Supports Optimized Flush</td>
<td>Specified, but evolving (ask when safe)</td>
</tr>
<tr>
<td></td>
<td>• Linux: <strong>unsafe</strong> except Device DAX</td>
</tr>
<tr>
<td></td>
<td>• (and new file systems like <strong>NOVA</strong>)</td>
</tr>
<tr>
<td></td>
<td>• Windows: <strong>safe</strong></td>
</tr>
<tr>
<td>Remote Flush</td>
<td>Proposals under discussion</td>
</tr>
<tr>
<td></td>
<td>(works today with extra round trip)</td>
</tr>
<tr>
<td>Deep Flush</td>
<td>Upcoming Specification</td>
</tr>
<tr>
<td>Transactions, Allocators</td>
<td>Built on above via libraries and languages:</td>
</tr>
<tr>
<td></td>
<td>• <a href="http://pmem.io">http://pmem.io</a></td>
</tr>
<tr>
<td></td>
<td><strong>Much more language support to do</strong></td>
</tr>
<tr>
<td>Virtualization</td>
<td>All VMMs planning to support PM in guest</td>
</tr>
</tbody>
</table>
|              | (KVM changes upstream, Xen coming, others too…)**
Can Visibility == Persistence?

- Sure, but you need either
  - Write-through caches (performance impact)
  - Flush-on-fail caches (cost)
Flush on Fail Caches

- Battery-protected
- MOV

**CPU CACHES**

- **L1**
- **L2**
- **L3**

**CLWB + fence**
- or-
**CLFLUSHOPT + fence**
- or-
**CLFLUSH**
- or-
**NT stores + fence**
- or-
**WBINVD** (kernel only)

**Custom Power fail protected domain indicated by ACPI property:**
- CPU Cache Hierarchy

**Minimum Required**
- **Power fail protected domain:**
  - Memory subsystem

**Battery-protected**

2017 Storage Developer Conference. © Intel and Oracle. All Rights Reserved.
Can Visibility == Persistence?

- Do volatile algorithms now “just work?”
  - CMPXCHG
    - Mostly yes
  - TSX
    - Mostly no
- The real issue:
  - Many algorithms have volatile assumptions!
Can Visibility == Persistence?

- Bottom line:
  - Some custom platforms will have flush-on-fail
  - Maybe some day all platforms
  - Flushing is not going away soon though
  - Apps should honor the ACPI property
When to use Deep Flush

- To limit metadata corruption from an ADR failure
  - Example: file system journal writes
    - ext4 or NTFS journal
- To give up performance for RAS
  - Example: today, some apps call `fsync()`
- Note that sometimes standard flush (`msync()` or `FlushViewOfFile()`) is fast enough, so just use it and get deep flush for free!
When to Not to use Deep Flush

- Performance critical
  - Remember: deep flush is meant to be used rarely
  - Optimized flush works fine unless HW failure
    - In the HW failure case, deep flush might fail too!

- Advice: writing apps to micromanage flush versus deep flush is a premature optimization
Flushing via Libraries

NVM Libraries

- Open Source
  - [http://pmem.io](http://pmem.io)
  - libpmem
  - libpmemobj
  - libpmemblk
  - libpmemlog
  - libvmem

More libraries being added to the suite over time
If You Feel You Must Flush Yourself

- Use libpmem to determine appropriate flushes
  - Either call it, or steal from it

- But consider using a library that handles flushing for you!
void push(pool_base &pop, uint64_t value) {
    transaction::exec_tx(pop, [&] {
        auto n = make_persistent<pmem_entry>();

        n->value = value;
        n->next = nullptr;
        if (head == nullptr) {
            head = tail = n;
        } else {
            tail->next = n;
            tail = n;
        }
    });
}

---

Transactional (including allocations & frees)
Where Are The Flushes?

PersistentSortedMap employees = new PersistentSortedMap();
...
employees.put(id, data);

No flush calls.
Transactional.
Java library handles it all.

See https://github.com/pmem/pcj
Libpmemobj Replication: Application Transparent (except for performance overhead)

“PMoF”
Why is Remote Flushing Any Different?

- Locally, the flush is fast
  - No time to context switch away
  - Potentially pipelined by hardware visibility rules
- Remote flushes go across the wire
  - Waiting for them prevents pipelining
  - Adding the store barriers to the data stream brings pipelining back into it
What is Async Flush?
Async Flush SNIA Work in Progress

- Specify the semantics of Async Flush and Persist Barrier
  - Goal: compatible semantics for both local and remote access
- What stores must be flushed?
  - Compiler optimizer instruction reordering
  - Superscalar CPU instruction reordering
  - Multithread store ordering
- What flushes must complete for the Persist Barrier?
  - Again, compiler and CPU instruction reordering
  - Thread-local barrier, not across threads
Summary

- Several type of flushes are available
  - HW provides some, SW provides some
- Flushing won’t go away any time soon
- Mostly use standard/optimized flushes
- Mostly apps should depend on libraries
- SNIA TWG still working on some details, but the 99% case is well-defined and implemented on both Windows and Linux