The Impact of the NVM Programming Model

Andy Rudoff
Intel Corporation
The Impact of the NVM Programming Model

... and Persistent Memory!

Andy Rudoff
Intel Corporation
Questions

- Why now?
  - Basic programming model is decades old!

- What changes?
  - Incremental changes vs major disruptions

- What does this mean to developers?
  - Emphasize the D in SDC…
Questions

- Why now?
  - Basic programming model is decades old!

- What changes?
  - Incremental changes vs major disruptions

- What does this mean to developers?
  - Emphasize the D in SDC…
Two Movements Afoot

Why now?
- Basic programming model is decades old!

Block Mode Innovations
- Atomics
- Access hints
- NVM-oriented operations

Emerging NVM Technologies
- Performance
- Performance
- Perf… okay, Cost
SNIA* NVM Programming TWG

• Members
  □ EMC, Fujitsu, Fusion-io, HP, HGST, Inphi, Intel, Intuitive Cognition Consulting, LSI, Microsoft, NetApp, PMC-Sierra, Qlogic, Red Hat, Samsung, Seagate, Sony, Symantec, Viking, Virident, VMware
  □ Calypso Systems, Cisco, Contour Asset Management, Dell, FalconStor, Hitachi, Huawei, IBM, IDT, Marvell, Micron, NEC, OCZ, Oracle, SanDisk, Tata Consultancy Services, Toshiba

* Storage Networking Industry Association: www.snia.org
SNIA NVM Programming TWG

- Charter: Develop specifications for new software “programming models” as NVM becomes a standard feature of platforms
  - Scope:
    - In-kernel NVM programming models
    - Kernel-to-application programming models
  - Programming models specify the exact technical behavior, up to (but not including) the OS specific API semantics

- APIs
  - Each OSV codes the programming models to specific to OS
Programming Model vs. API

- OSVs own their kernel APIs
  - Cannot define these in a committee and push on OSVs
  - Cannot define one API for multiple OS platforms
    - Serious differences on how things work in the kernel
    - Goes against independent innovation
  - Next best thing is to agree on overall model
    - With OSV collaboration
    - Then engage OSV to define and implement API

- Similar situation in user-space
  - A common API doesn’t always make sense
    - Violates the “when in Rome” design principle
    - Example: the UNIX* versus the Windows* event models
  - Ultimately: want OSV to ship and maintain the API
Questions

- Why now?
  - Basic programming model is decades old!

- What changes?
  - Incremental changes vs major disruptions

- What does this mean to developers?
  - Emphasize the D in SDC…
Today’s Topic

- Why now?
  - Basic programming model is decades old!

- What changes?
  - Incremental changes vs major disruptions

- What does this mean to developers?
  - Emphasize the D in SDC…
Next Generation Scalable NVM

Resistive RAM NVM Options

<table>
<thead>
<tr>
<th>Family</th>
<th>Defining Switching Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase Change Memory</td>
<td>Energy (heat) converts material between crystalline (conductive) and amorphous (resistive) phases</td>
</tr>
<tr>
<td>Magnetic Tunnel Junction (MTJ)</td>
<td>Switching of magnetic resistive layer by spin-polarized electrons</td>
</tr>
<tr>
<td>Electrochemical Cells (ECM)</td>
<td>Formation / dissolution of “nano-bridge” by electrochemistry</td>
</tr>
<tr>
<td>Binary Oxide Filament Cells</td>
<td>Reversible filament formation by Oxidation-Reduction</td>
</tr>
<tr>
<td>Interfacial Switching</td>
<td>Oxygen vacancy drift, diffusion induced barrier modulation</td>
</tr>
</tbody>
</table>

Many candidate next generation NVM technologies. Offer ~ 1000x speed-up over NAND, closer to DRAM
Exploiting Next Generation NVM

With Next Generation NVM, the NVM is no longer the bottleneck
- Need optimized platform storage interconnect
- Need optimized software storage access methods
Exploiting Next Generation NVM

- With Next Generation NVM, the NVM is no longer the bottleneck
  - Need optimized platform storage interconnect
  - Need optimized software storage access methods

App to SSD IO Read Latency (QD=1, 4KB)
Persistent Memory
Persistent Memory
Persistent Memory Definition

- Byte-addressable
  - As far as the programmer is concerned
- Load/Store access
  - Not demand-paged
- Memory-like performance
  - Would reasonably stall a CPU load waiting for PM
- Probably DMA-able
  - Including RDMA

- For modeling, think: Battery-backed DRAM
PMem in This Presentation…

- Is **not** tablet-like memory for entire system
  - “Transparent Persistent Memory” is another topic

- Is **not** NAND Flash
  - At least not directly
    - perhaps with aggressive caching

- Is **not** block-oriented
Persistent Memory Attributes

- PM does not
  - Surprise the program with unexpected latencies
    - No major page faults
  - Kick other things out of memory
  - Use the page cache unexpectedly

- PM stores aren’t durable until data is flushed
  - Is this a new, inconvenient attribute of PM?
  - Or is this something that’s been around for decades?

- PM may not always stay at the same address
  - Physically
  - Virtually
Types of Persistent Memory

- Battery-backed DRAM
  - Practical or not, a good model for software design
- DRAM saved on power failure
  - DRAM speeds at run-time
  - Expense = DRAM + additional logic
- NVM with significant caching
  - Reliance on cache means assumptions on workloads
- Next generation NVM
  - Bit and pieces of information available
  - Still quite a bit unknown/emerging here
Existing Memory use cases

**Volatile**

**Typical Use Case**

- DRAM DIMMs
- High Throughput, High Cost

**SSD Caching**
- DB in HDD/SAN cached by SSD/DRAM

**Persistent**

**Emerging Use Case**

- NVDIMM
  - Raid Cache
  - Persistent RamDisk
  - Checkpointing

- Copy to Flash
  - Sections of DRAM used for Persistent Write Back Cache (WBC)
  - Transfer WBC to Flash (power loss)

- NVM used as Memory
  - Extended App Virtual Memory
  - Auto Commit Mem (persistent)
The Value of Persistent Memory

- Data sets addressable with no DRAM footprint
  - At least, up to application if data copied to DRAM
- Typically DMA (and RDMA) to PM works as expected
  - RDMA directly to persistence – no buffer copy required!
- The “Warm Cache” effect
  - No time spend loading up memory
- Byte addressable
- Direct user-mode access
  - No kernel code in data path
- Intel collaborating with the industry to ensure open standards and choice
The Challenges of PMem

- New Programming Models, APIs
  - But built on decades old memory-mapped file model

- Some Programming Challenges
  - File systems engineers, database engineers always did this
  - Now other application engineers need to learn this
Two Persistent Memory Programming Models
(two so far…)
NVM.PM.VOLUME

- NVM Volumes
  - PM Capable
  - A list of physical ranges of NVM
- Operations
  - GET_RANGESET
  - ...
Who Uses NVM.PM.VOLUME?

- Kernel modules
- File systems
  - Maybe to expose PM
  - Maybe just to use it internally
- Memory management
  - Example: Multi-tiered page cache
- Other storage stack components
  - RAID
  - Caches
  - Clustered I/O
- Future NVM Programming models we haven’t thought of yet
NVM.PM.FILE

- NVM Files
  - PM Capable
  - Native file APIs and management
- Operations
  - Native open/close read/write
  - NVM.PM.FILE.MAP
  - ...

NVM PM FILE Mode

Application

Native File API

Load/Store

User space

Kernel space

PM-Aware File System

MMU Mappings

NVM PM Capable Driver

PM Device

PM Device

PM Device
Application Memory Allocation

- Well-worn interface, around for decades
- Memory is gone when application exits
  - Or machine goes down
Application NVM Allocation

- Simple, familiar interface, *but then what?*
  - Persistent, so apps want to “attach” to regions
  - Need to manage permissions for regions
  - Need to resize, remove, …, **backup** the data
Who Uses NVM.PM.FILE?

- Applications
  - Persistent data sets, requiring addressability without impacting DRAM footprint
  - Persistent caches
- Usages that must reconnect with blobs of persistence
  - Naming
  - Permissions
- Potentially kernel modules requiring some of the above features
Questions

- Why now?
  - Basic programming model is decades old!

- What changes?
  - Incremental changes vs major disruptions

- What does this mean to developers?
  - Emphasize the D in SDC…
Memory-Mapped Data Structures

- `mmap()` on UNIX, `MapViewOfFile()` on Windows

- Facility has been around for decades
  - In pretty much all modern operating systems

- Typically demand-paged
  - UNIX also used it for hardware access (e.g., graphics)

- Usually two types:
  - Shared
  - Copy-on-write (“Private”)

2013 Storage Developer Conference. © Intel Corporation. All Rights Reserved.
NVM.PM.FILE programming model “surfaces” PM to application

Still somewhat raw at that point

Build on it with additional libraries

Eventually turn to language extensions
Creating Resilient Data Structures

- Once PMem is mapped into application, a malloc-like interface is required
- Traditional:
  - ptr = pmemalloc(pmem_pool, len)
- New and interesting issues:
  - What if program crashes before anything links to ptr?
    - Memory leak – **Persistent** memory leak
  - What if program is half-way through linking it in?
    - Transactions required? How complex?
  - How to test issues that didn’t happen w/volatile memory?
    - Fault-injection testing
- Not just for malloc – true for all PM data structures
Full Example

- [https://github.com/pmem/linux-examples](https://github.com/pmem/linux-examples)

- libpmem
  - Helper functions for PM API

- libpmemalloc
  - Example of a PM-safe malloc library

- **Fault injection framework**
  - Proof-by-exhaustion of algorithm resilience
  - At least for simple examples…
What Might malloc Look Like?

- Start with a linked-list data structure:
  ```c
  struct node {
      struct node *next_;
      int value;
  };
  ```

- Traditional “malloc” is now done in steps:
  - Reserved the memory
  - Fill it in (prepare for use)
  - Link it in
malloc Example...

- Start with a linked-list data structure:

  ```
  struct node {
    struct node *next_;  
    int value;
  };
  ```

- Traditional “malloc” is now done in steps:

  - Reserved the memory
  - Fill it in (prepare for use)
  - Link it in

  undo↑  interruption  finish↓
Summary

- NVM Programming Models are Evolving
  - Incremental block device features
  - Disruptive: Persistent Memory!

- The Industry is Aligning Around New Models
  - SNIA NVM Programming TWG
  - Emerging Research, Products

- Apps Can Leverage Persistent Memory
  - Using File System/DB Techniques
  - Using Libraries & Language Extensions
For More...

- SNIA NVM Programming TWG
  - [http://snia.org/forums/sssi/nvmp](http://snia.org/forums/sssi/nvmp)

- Linux Pmem Examples:
  - [https://github.com/pmem/linux-examples](https://github.com/pmem/linux-examples)

- Linux PMFS:
  - [https://github.com/linux-pmfs](https://github.com/linux-pmfs)

- andy.rudoff@intel.com
Some Additional Details
np = pmemalloc_reserve(sizeof(*np));

/* link it in at the beginning of the list */
np->next = rootnp;
np->value = value;
pmemalloc_onactive(np, &rootnp, np);

pmemalloc_activate(np);
Breaking `malloc` Into Steps

np = pmemalloc_reserve(sizeof(*np));

/* link it in at the beginning of the list */
np->next = rootnp;
np->value = value;
pmemalloc_onactive(np, &rootnp, np);

pmemalloc_activate(np);
np = pmemalloc_reserve(sizeof(*np));

/* link it in at the beginning of the list */
np->next = rootnp;
np->value = value;
 pmemalloc_onactive(np, &rootnp, np);

pmemalloc_activate(np);
np = pmemalloc_reserve(sizeof(*np));

/* link it in at the beginning of the list */
np->next = rootnp;
np->value = value;

pmemalloc_onactive(np, &rootnp, np);

pmemalloc_activate(np);

undo

interruption

time

finish
if ((np_ = pmemalloc_reserve(pmp, sizeof(*np_))) == NULL) 
    FATALSYS("pmemalloc_reserve");

/* link it in at the beginning of the list */
PMEM(pmp, np_)->next_ = sp->rootnp_;  
PMEM(pmp, np_)->value = value;  
pmemalloc_onactive(pmp, np_, (void **)&sp->rootnp_, np_);

pmemalloc_activate(pmp, np_);
Position Dependence

- Can pointers work across sessions?
- Will a PM file be mapped at the same address every time?
The “C-Clamp”

Interposing "filter" Driver

Persistant Memory

Block Driver

File System

PM-Aware File System

Load/Store

User Space

Kernel Space

Standard File API

Standard File API

MMU Mappings

Application

Application
PM Interposition Points

Application

Load/Store

User Space

Standard File API

PM-Aware File System

MMU Mappings

Kernel Space

libraries (high confidence)

page modification tracking (research topic)

Persistent Memory

PM Interposition Points

Application

Load/Store

User Space

Standard File API

PM-Aware File System

MMU Mappings

Kernel Space

libraries (high confidence)

page modification tracking (research topic)

Persistent Memory