Overview of the NVM Programming Model

JANUARY 28, 2014, SAN JOSE, CA
Motivation for NVM Programming Models

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

Emerging NVM Technologies
- Performance
- Performance
- Perf… okay Cost!
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
The NVM Software Stack

![App to SSD IO Read Latency (QD=1, 4KB) Diagram]
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

- Think “battery-backed RAM”
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
PM Volume

- **NVM Volumes**
  - PM Capable
    - A list of physical ranges of NVM
- **Operations**
  - GET_RANGESET
    - ...
Uses for 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
PM File

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

- Standards
  - NVDIMM Driver
  - Application
  - File System

User Space
- Standard File API
- Load/Store

Kernel Space
- PM-Aware File System
- MMU Mappings

Persistent Memory
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
Re-thinking the Stack

Persistent Memory

User Space

Kernel Space

Interposing “filter” Drivers

Application

Standard File API

Load/Store

Application

Standard File API

File System

Block Driver

PM-Aware File System

MMU Mappings

Persistent Memory
Building on NVM.PM.FILE

- 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
Is PM Transparent?

- Totally transparent
  - HW-supported full-system persistence
  - Client first, server much later

- Transparent above a certain level in stack
  - Exactly why we have NVM.PM.VOLUME
  - Also SSDs

- Transparent via library or language run-time
  - JVM seems like an obvious target
Exploring PM Semantics

- Portable, testable code using memory-mapped files
  - Should “just work”
  - Except for performance, of course

- Atomics provided by CPU
  - Typically what can be done in a single store
  - Beyond that, additional HW or SW needed
  - PM atomic requirements for each app is an area of research

- Interaction with CPU caches
  - Analogous to memory-mapped files and page cache
Summary

❖ The NVM Programming Models are aligning the industry
  ◆ Gaining common terminology
  ◆ Not forcing specific APIs

❖ Now that we’ve got it, what do we do with it?
  ◆ PM models expose it
  ◆ New PM models build on existing ones
  ◆ We have not yet found our models to be limiting us
    ◄ “yet”…

❖ Emerging technologies are going to drive work in this area, increasing as the cost comes down
For More…

- SNIA NVM Programming TWG
  - http://snia.org/forums/sssi/nvmp

- Linux Pmem Examples:
  - https://github.com/pmem/linux-examples

- Linux PMFS:
  - https://github.com/linux-pmfs

- andy.rudoff@intel.com