Preparing Applications for Persistent Memory

Doug Voigt
Hewlett Packard Enterprise
Persistent Memory (PM) Vision

Persistent Memory Brings Storage

Fast Like Memory

Persistent Like Storage

To Memory Slots

Make data durable without doing I/O
Technology Trends

- The flash revolution marches on
- NVDIMMs are delivering memory speed today
- New PM technologies bring similar speed with much larger capacity
- Fabric memory creates larger shared pools
Advantages of memory speed

- Order of magnitude latency reduction
- Expands in-memory application trend
Mission of the NVM PM TWG

- Accelerate the availability of software that enables NVM (Non-Volatile Memory) hardware.
  - Hardware includes SSD’s and PM
  - Software spans applications and OS’s
- Create the NVM Programming Model
  - Describes application visible behaviors
  - Allows API’s to align with OS’s
  - Has exposed optimization opportunities in networks and processors
Rapid Deployment of PM using RAM Disks

- Avoids application disruption
- Limits potential performance advantage
Memory mapped files

I/O Stack

Kernel

Application

File System

Disk Driver

User

HW

SSD/HDD

Memory "Stack"

Application

Memory Mapped Files

Memory Access

User

HW
Challenges of memory speed

- Applications must change for optimal performance
  - Use PM data structures
  - Different style of error handling
- Processors have volatile write caches
  - Normal write flow does not account for PM
  - Requires the use of flush instructions
- Elimination of Disk IO has ripples
  - Failure atomicity required for data recoverability
  - Leads to more transactional memory access
What Processors Automatically Provide

Data recoverability requires cache flushing
Persistent Memory Data Structures

- Persistent memory data structures must assure recoverability after abrupt power loss
- Single data structures can be made intrinsically recoverable
  - Limit in-place updates
  - Groups of changes are slaved to a single memory access
- Groups of data structures must use transactions
  - Classical approaches apply
  - Implemented using PM data structures
Trivial Example: Append Only Log

Append pseudocode:

<Create new log entry in free space>
Flush(new entry);
filled = filled + size(new entry);  # Atomic update to fundamental data type
Flush(filled);
NVM Library

- [http://pmem.io/nvml](http://pmem.io/nvml)
- PM assist functions
  - Map, Flush, Allocation
- PM Data Structures
  - Log, Block and more
- PM Object
  - Root, Transactions, Type Safety and more
Language extensions for persistent memory

- Features similar to the NVM library can be integrated into standard programming languages
  - More convenient
  - More sophisticated
  - Safer

**Atlas: Leveraging Locks for Non-volatile Memory Consistency**
Failure atomic code sections based on existing critical sections

**NVM support for C Applications**
PM region file management, transactions with locks, heaps
Application Data Access Evolution

Reaping the advantages of persistent memory with step-wise innovation

<table>
<thead>
<tr>
<th>Today</th>
<th>Horizon 1: PM Middleware</th>
<th>Horizon 2: PM Libraries</th>
<th>Horizon 3: Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Compiler</td>
</tr>
<tr>
<td>File System</td>
<td>File System</td>
<td>PM Library</td>
<td>Application</td>
</tr>
<tr>
<td>Disk Driver</td>
<td>File System</td>
<td>PM Library</td>
<td>File System</td>
</tr>
</tbody>
</table>

SSD
Technology Access Matrix

Application View

Memory Access

Block Access

Flush to PM

Normal

Disk

Memory

Underlying Technology
Separate Stacks

Application requires a system with the correct technology.
Technology Access Matrix

Application View

<table>
<thead>
<tr>
<th>Memory Access</th>
<th>Flush to Disk</th>
<th>Flush to PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Access</td>
<td>Normal</td>
<td></td>
</tr>
</tbody>
</table>

Disk  | Memory

Underlying Technology
Dual Stack Scenario with Posix MMap

Emulate memory access with flush to disk
(Existing Posix file system feature called Mmap)
<table>
<thead>
<tr>
<th>Application View</th>
<th>Memory Access</th>
<th>Disk Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Flush to Disk</strong></td>
<td><strong>Normal</strong></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td>RAM Disk</td>
</tr>
<tr>
<td></td>
<td><strong>Flush to PM</strong></td>
<td></td>
</tr>
</tbody>
</table>

Underlying Technology

Disk | Memory
Dual stack Scenario with RAM Disk

Emulate disk with RAM disk driver backed by PM

Unmodified Applications

PM Aware Applications

IO

RAM Disk

Memory Access

SSD/HDD
Dual stack Scenario

Unmodified Applications

SSD/HDD

PM Aware Applications

RAM

Disk

Posix MMap

Memory Access

Memory Access

Memory Access

Memory Access

IO

IO

IO

IO
Do You Want High Availability With That PM?

- There is no hardware flush action over PCI
- AND there is no RDMA completion that indicates persistence
- SO software is involved at the remote node
Remote Access Challenge

- How can remote access PM latency be reduced?
- **SNIA PM Remote Access for High Availability**
  - Remote access taxonomy
  - Data recoverability requirements
  - Model and requirements for remote flush
- Multiple industry parties are responding
  - Open Fabrics Alliance
  - InfiniBand Trade Association
  - Several vendors
Role of the NVM Programming Model

Rally the industry around a view of NVM that is:

- Application centric
- Vendor neutral
- Achievable today
- Reaches beyond storage
  - Applications
  - Memory
  - Networking
  - Processors
Preparing Applications for Persistent Memory

Doug Voigt
Hewlett Packard Enterprise