4 THINGS I WISH I KNEW SOONER ABOUT PERSISTENT MEMORY

Rob Dickinson
Datacenter Solutions Group

October 3, 2017
ABOUT ME

NVM Software Architect at Intel
Specializing in persistent memory and databases

- github.com/RobDickinson
- linkedin.com/in/robfromboulder
- @robfromboulder
- robert.a.dickinson@intel.com
I'M NOT ONE OF THOSE INTEL ENGINEERS IN A BUNNY SUIT
FOUR GUIDELINES FOR PERSISTENT MEMORY

- Persistent memory is not exactly like DRAM nor SSD
- Persistent memory will be used in different ways by different applications
- Persistent memory will be used in concert with memory and storage (aka hybrid architectures)
- Persistent memory exposes applications to some classic problems
WHAT IS PERSISTENT MEMORY?

compared to DRAM and SSD
INTEL® PERSISTENT MEMORY

New type of memory:
• Persistent
• 6 TB per two-socket system
• Cheaper than DRAM

Product available:
• SSDs in 2017
• Intel DIMMs for next-gen platforms in 2018
## Compared to Memory and Storage

<table>
<thead>
<tr>
<th>Aspect of Comparison</th>
<th>Memory (DRAM)</th>
<th>Persistent Memory</th>
<th>Storage (SSD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>GB</td>
<td>TB</td>
<td>TB</td>
</tr>
<tr>
<td>Power-Safe Durability</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Access Model (native units)</td>
<td>Byte Addressable (CL)</td>
<td>Byte Addressable (CL/ECC)</td>
<td>Blocks (4K)</td>
</tr>
<tr>
<td>Latency</td>
<td>Nanoseconds</td>
<td>Nanoseconds</td>
<td>Microseconds</td>
</tr>
<tr>
<td>Wear Leveling</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cost</td>
<td>$$$</td>
<td>$$</td>
<td>$</td>
</tr>
</tbody>
</table>
OPTIMIZED SYSTEM INTERCONNECT

- Internal
- Platform Link Xfer & Protocol
- Driver
- File System / Stack

Latency from App (usecs)

NVMe NAND SSD (4kB read)

Intel® Optane™ SSD (4kB read)

Intel® DIMM (64B read)
STANDARD FOR PERSISTENT MEMORY PROGRAMMING

SNIA Technical Working Group
Defined 4 programming modes required by developers

Spec 1.0 developed, approved by SNIA voting members and published

Interfaces for PM-aware file system accessing kernel PM support
Interfaces for application accessing a PM-aware file system
Kernel support for block NVM extensions
Interfaces for legacy applications to access block NVM extensions

MEMORY-MAPPED FILES

- Application
  - Load/Store Access
- Application
  - Page Fault Access
- User Space
- Kernel Space
- DRAM
HYBRID ARCHITECTURES
WITH PERSISTENT MEMORY
1. WAL ACCELERATION

Put WAL on persistent memory
Leave main data storage on SSD
2. PERSISTENT CACHE

Slower than DRAM cache
Never needs to be reloaded from SSD

(multi-level cache variations)
3. HYBRID B+ TREE

Inner nodes are in DRAM
Leaf nodes are in persistent memory
github.com/pmem/pmemkv
CLASSIC PROBLEMS
RE-EXPOSED
Having direct access to persistent memory, bypassing the kernel and filesystem, brings classic problems into userspace (where programmers aren't used to seeing them)
1. TRANSACTIONS & LOCKING

Probably the hardest problems of all...
Preventing torn updates using transactions
Resetting abandoned locks
2. MEMORY ALLOCATION & GC

Regular memory allocators aren't appropriate
A leak in persistent memory remains leaked
3. ERROR DETECTION & HANDLING

Coping with media failures or corruption
Drivers & filesystems hide these problems
GETTING STARTED WITH PERSISTENT MEMORY

- Consider persistent memory as its own unique tier
- Look for break-out opportunities to hybridize your architecture
- Use libraries to assist in safely using persistent memory
  - pmem.io
  - github.com/pmem/nvml/
  - software.intel.com/en-us/persistent-memory
- Start with emulation, but validate & tune on a real platform