Redrawing the Boundary Between Software and Storage for Fast Non-Volatile Memories

Steven Swanson
Director, Non-Volatile System Laboratory
Computer Science and Engineering
University of California, San Diego
Humanity processed 9 Zettabytes in 2008.

Welcome to the Data Age!

*http://hmi.ucsd.edu
Solid State Memories

• NAND flash
  – Ubiquitous, cheap
  – Sort of slow, idiosyncratic
• Phase change, Spin torque MRAMs, etc.
  – Not (yet) ready for prime time
  – DRAM-like speed
  – DRAM or flash-like density
• DRAM + Battery + flash
  – Fast, available today
  – Reliability?
Integrating Them Into Systems

PCle-attached NVMs
- Low Latency
- High bandwidth
- Flexible interface
- Flexible topology
- Scalable

DDR-attached NVMs
- Lowest Latency
- Highest bandwidth
- Rigid interface
- Restricted topology
- Limited scalability
Software Latency Will Dominate

<table>
<thead>
<tr>
<th>Technology</th>
<th>Software's Contribution to Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Drive</td>
<td>0.3%</td>
</tr>
<tr>
<td>SSD (2012)</td>
<td>19.3%</td>
</tr>
<tr>
<td>PCIe-Flash (2007)</td>
<td>21.9%</td>
</tr>
<tr>
<td>PCIe-PCM (2010)</td>
<td>10.4%</td>
</tr>
<tr>
<td>PCIe-Flash (2012)</td>
<td>21.9%</td>
</tr>
<tr>
<td>PCIe-PCM (2012)</td>
<td>70.0%</td>
</tr>
<tr>
<td>BB DRAM (2012)</td>
<td>88.4%</td>
</tr>
<tr>
<td>DDR NVM (2014)</td>
<td>94.09%</td>
</tr>
</tbody>
</table>
Software Energy Will Dominate

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Year</th>
<th>Software's Contribution to Energy/IOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Drive (2006)</td>
<td></td>
<td>0.4%</td>
</tr>
<tr>
<td>SSD (2012)</td>
<td></td>
<td>97.0%</td>
</tr>
<tr>
<td>PCIe-Flash (2012)</td>
<td></td>
<td>75.8%</td>
</tr>
<tr>
<td>PCIe-PCM (2013)</td>
<td></td>
<td>78.6%</td>
</tr>
</tbody>
</table>
Software’s New Role in Storage

For Disks: Adding Software Could Make the System Faster

For NVMs: Adding Software Makes It Slower

• Reduce software interactions
• Remove disk-centric features
• Redesign IO software (OS, FS, languages, and apps)
Two Case Studies

Moneta: SSDs for Fast NVMs

- Database
- File System
- IO Stack
- Moneta Driver

NV-Heaps: NVMS over DDR

- Programming Model
  - File System
  - IO Stack
  - Block Driver

- CPU
- PCIe
- DDR
- DRAM
- NVM

- NVM
- NVM
- NVM

Two Case Studies
Moneta Internals: Performing a Write

- Host via PIO
- Request Queue
- Scoreboard
- Tag Status Registers
- DMA Control
- Transfer Buffers
- Ring Control
- Ring (4 GB/s)
- 16GB NVM
- 16GB NVM
- 16GB NVM
- 16GB NVM
The Moneta Prototype

- FPGA-based implementation
- DDR2 DRAM emulates PCM
  - Configurable memory latency
  - 48 ns reads, 150 ns writes
  - 64GB across 8 controllers
- PCIe: 2 GB/s, full duplex
Optimizing Software for Moneta

- Optimizations
  - Remove IO scheduler
  - Redesigned HW interface
  - Remove locks

- SW latency drops from 13.4 us to 5us
  - 62% reduction in latency
  - Increased concurrency

- Bandwidth increases by up to 10x
The App Gap

- We must close the App Gap for NVM success
Where is the App Gap (Part I)?

- Databases
- Sci Apps
- Web Apps
- File Servers

File system

IO Stack

Driver

Moneta 1.0

4KB Accesses

Sustained MB/s

 Writes

Read/Write

Reads

- Moneta 1.0 + fs
- Moneta 1.0
Eliminating FS and OS overheads

- Separate protection mechanisms from policy
  - Move protection checks to hardware
  - Allow applications to access Moneta directly

- Protection overheads are almost eliminated
Some Progress on the App Gap

![Bar Chart]

- **Speedup vs. Kernel Space**
- **Kernel Space**
- **User Space**

- BDB-Btree
- BDB-Hash
- MySQL-Select
- MySQL-Update
- MySQL-Complex
- PSQL-Select
- PSQL-Update
- PSQL-Complex
- Mean

**Non-volatile Systems Laboratory**
Where is the App Gap (Part II)?

- **Write-ahead logging**
  - e.g. ARIES
- **Extra IO operations**
  - Wasted software latency/power
  - Wasted IO performance
- **Multi-part atomic writes**
  - Apps can create and commit write groups
- **The application can read and change the log contents**
Bandwidth Comparison

2 to 3.8X improvement
ARIES – Write Ahead Logging for Databases on Disks

• ARIES provides useful database features
  – Atomicity
  – Durability
  – High Concurrency
  – Efficient recovery

• But it makes disk-centric design decisions
  – Undo *and* redo logs to avoid random synchronous writes and provide IO scheduling flexibility
  – Page-based storage management to match disk semantics

We want to keep these

Rethink these
## ARIES’ Disk-Centric Design Decisions

<table>
<thead>
<tr>
<th>Design Decision</th>
<th>Advantages</th>
<th>Downsides</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-force</td>
<td>Eliminates synchronous random writes.</td>
<td>Requires synchronous writes to the redo log.</td>
</tr>
<tr>
<td>Steal</td>
<td>Longer sequential writes.</td>
<td>Requires an undo log.</td>
</tr>
<tr>
<td>Pages</td>
<td>Matches disk and OS page-based IO model.</td>
<td>Page granularity is not always ideal.</td>
</tr>
</tbody>
</table>
MARS: Rethinking ARIES for Moneta

No-force

Force updates on commit, but offload it to Moneta’s memory controllers.

Steal

Get rid of undo logging. Update redo log contents instead.

Pages

Software can choose object/page granularity.
ARIES vs MARS

![Graph showing comparison between ARIES and MARS across different threads.](image)

- Y-axis: 4KB Swaps TX/sec
- X-axis: Threads (1, 2, 4, 8, 16)

The graph illustrates the performance comparison between ARIES and MARS in terms of 4KB swaps per second across varying thread counts.
Software’s Role in Moneta

• The basic Moneta hardware is (relatively) simple
• Software requirements drove the HW enhancements
• Reduce, Remove, Redesign
  – Simplify the kernel IO stack
  – Eliminate the OS and FS overheads for most accesses
  – Rethink write-ahead logging for new memories
Two Case Studies

Moneta: SSDs for Fast NVMs

NV-Heaps: NVMS over DDR

Database
File System
IO Stack
Moneta Driver

Programming Model
File System
IO Stack
Block Driver

NVM
NVM
NVM

PCIe
CPU
DDR

NVM
DRAM
NVM
Disks and Files Are Passé

DDR NVM is a new kind of memory
- Use conventional file operations to access data
- Use pointers/references
- Leverage strong types
- But make it persistent
The Dangers of Non-Volatile Programming

Dlist Insert(Foo x, Dlist head)
{
    Dlist r;
    r = new Dlist;
    r.data = x;
    r.next = head;
    r.prev = null;
    if (head) { head.prev = r; }
    return r;
}
NV-Heaps: Safe, Fast, Persistent Objects

• Container for NV data structures
  – Generic
  – Self-contained

• Access via loads and stores
  – Memory-like performance
  – Familiar semantics

• Strong safety guarantees
  – ACID transactions
  – Pointer safety
A Type System for NV-Heaps

1. Refine each type with a heap-index
2. Type rules ensure that the heaps match

```c
Dlist<h> Insert(Foo<h> x, Dlist<h> head)
{
    atomic<h> {
        Dlist<h> r;
        r = new Dlist in heapof(r);
        r.data = x;
        r.next = head;
        r.prev = null in heapof(r);
        if (head) { head.prev = r; }
    }
    return r;
}
```
Application: A Persistent Key-value Store

NV-heaps are much faster than the block-based systems. NV-heaps make the cost of persistence very low.
NV-Heaps Wrap Up

• The hardware is here already.
• NV-heaps can change how programmers think about persistent state.
  – Careful system design can avoid the pitfalls
  – And the performance gains are huge
Other Efforts

• PCIe SSDs
  – FusionIO et. al.
  – Intel NVMeExpress prototypes [FAST’12]
  – Texas Memory Systems SSDs

• Atomic Writes
  – Transactional Flash [OSDI’08]
  – FusionIO AtomicWrite [HPCA’11]

• NV-Heaps
  – Slow but safe
    • Stasis [OSDI 06]
    • Orthogonally persistent Java [SIGMOD 96]
    • Many others...
  – Fast but unsafe
    • Recoverable Virtual Memory [SOSP 93]
    • Rio Vista [SOSP 97]
  – Fast and safe
    • Mnemosyne [ASPLOS 11]
Open Challenges
How Can We Elegantly Enrich Storage Semantics?

• Atomic Writes help but,...
• Each applications has its own needs
  • Legacy system architectures
  • Novel data structures
• Can we build flexible semantics for storage?
  • Like shaders in graphics cards?
How do NVMs affect the data center?

- How should we organize a petabyte of NVMs?
  - Centralized? Distributed? Hybrid?
- NVM performance with large-scale system overheads
  - Replication
  - Network latency
  - Thick software layers (e.g. OpenStack, HadoopFS)
Software Costs Will Remain High

PCle and DDR performance will continue to improve.

PCle and DDR power efficiency will continue to improve.

Thick software stacks will push software costs higher.

Applications will demand richer, more sophisticated interfaces.
The App Gap Remains

Log Speedup vs DISK-RAID

Moneta
FusionIO
SSD-RAID
DISK-RAID

XDD 4KB RW
Btree
HashTable
Bio
PTF
Solid State Storage In the Data Age

- Harnessing the data we collect requires vastly faster storage systems
- Fast, non-volatile memories can provide it, but...
- NVMs will force software’s role to change
  - Software will become a drag on IO performance
  - Reduce, Remove, Redesign
- Leveraging NVMs quickly requires a coordinated research that spans system layers
- No aspect of the system is safe!
Thanks!
Questions?