Fast Non-Volatile Memories are Coming and We are Not Ready

Steven Swanson

Rajesh Gupta, Adrian Caulfield, Laura Grupp, Ameen Akel, Joel Coburn, Todor Mollov, Arup De, Michael Wei, Hung-Wei Tseng, Pravin Prabhu

Non-volatile Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
The Need for Fast Storage

• Humanity is generating data at an amazing rate
  – 5.6 Exabytes in 2002
  – Estimated >> 45 Exabytes in 2010

• Storage is not a problem

• Analysis is a problem
  – Unstructured data
  – Graph-based analyses
  – Interactive analysis
Non-volatile Memories

• NAND Flash is here!
  – Dense -> Cheap
  – Faster than disk (25-200us)
  – Reliability and scalability problems

• Storage class memories are coming
  – DRAM-like speed
  – Flash-like density
  – Phase change memory, spin torque transfer memory, the memristor, Race track memory
The Future of Storage

• Random 4KB Reads from user space

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat.</td>
<td>7.1ms</td>
<td>68us</td>
<td>12us</td>
<td>6.5us</td>
</tr>
<tr>
<td>BW: 2.6MB/s</td>
<td>250MB/s</td>
<td>1.3GB/s</td>
<td>6.5GB/s</td>
<td></td>
</tr>
<tr>
<td>1x</td>
<td>104x</td>
<td>591x</td>
<td>1092x</td>
<td>= 2.2x/yr</td>
</tr>
<tr>
<td>1x</td>
<td>96x</td>
<td>500x</td>
<td>2600x</td>
<td>= 2.4x/yr</td>
</tr>
</tbody>
</table>
Hardware, OS, and Applications are unprepared

• Systems assume slow disk
  – IO stack and drivers are inefficient
  – IO interfaces (e.g., SATA) are slow.

• Large, unneeded, or unoptimized overheads
  – Misguided, disk-centric schedulers
  – Useless caching

• Avoid IO at all costs $\rightarrow$ enormous complexity
  – DB buffer managers
  – Sequential IO interfaces
Today’s Talk

Using the Moneta array to quantify the software problem

NV-heaps: Making the persistent objects safe and fast with SCM
Can existing applications leverage the benefits of SCM?
Storage technologies under test

- 128GB SW SSD Raid
- 4TB HW Disk Raid
- SATA
- PCIe
- DDR
- PCIe + SATA
- 8x Nehalem
- SCM
- 80GB FusionIO
- 64GB Moneta
- 64GB NVRamDisk
Moneta: PCIe-attached SCM

[To appear Micro 2010]

PCle 1.1 x8 (2GB/s Full Duplex)

DMA

Request Queue

2x 8KB Buffers

Scheduler

Mem Ctrl

Start Gap

16GB

Mem Ctrl

Start Gap

16GB

Mem Ctrl

Start Gap

16GB

Mem Ctrl

Start Gap

16GB

Ring (4GB/s)

Built on the BEE3 FPGA prototyping platform
Emulating fast NV memories in Moneta

• Adjust DDR timings to match SCM projections
  – RAS-CAS Delay – read from the array
  – Post-write delay -- write to the array
  – Variable from DRAM times to microseconds.

• Wear-leveling – “Start-gap” [micro 2009]

• All the data here are for PCM
  – 67ns read
  – 215ns write
The Moneta software stack

- Optimizations
  - Baseline
  - No scheduler
  - Atomic command issue
  - Spin wait for completion
- Removed 2/3 of SW latency
- Removed all locks
- What remains?
  - Interrupt processing
  - Entering/leaving the kernel
The Moneta software stack

0.9 Million IOPS
Emulating a NVRamDisk

- NVRamDisk is an DDR3-attached array of SCM
- Modified Linux RamDisk Driver
  - Insert delays to model latency impact of SCM
  - By default, the driver is similar to spirit to the Moneta driver.

![SCM](image)

64GB NVRamDisk
## Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Footprint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic IO</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raw IO</td>
<td>64 GB</td>
<td>Read, Write, read + write</td>
</tr>
<tr>
<td>File IO</td>
<td>64 GB</td>
<td>Read, Write, read + write</td>
</tr>
<tr>
<td><strong>Database Applications</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berkeley-DB Btree</td>
<td>16 GB</td>
<td>Transactional updates to btree key/value store</td>
</tr>
<tr>
<td>Berkeley-DB HashTable</td>
<td>16 GB</td>
<td>Transactional updates to hash table key/value store</td>
</tr>
<tr>
<td>BiologicalNetworks</td>
<td>35 GB</td>
<td>Biological database queried for properties of genes and biological-networks</td>
</tr>
<tr>
<td>PTF</td>
<td>50 GB</td>
<td>Palomar Transient Factory sky survey queries</td>
</tr>
<tr>
<td><strong>Memory-hungry Applications</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGEMM</td>
<td>21 GB</td>
<td>Matrix multiply with 30,000 x 30,000 matrices</td>
</tr>
<tr>
<td>NAS Parallel Benchmarks</td>
<td>8-35 GB</td>
<td>7 apps from NPB suite modeling scientific workloads</td>
</tr>
</tbody>
</table>
Raw Bandwidth

![Bar Chart]

- **Bandwidth (GB/s)**
- **4MB Accesses**
  - Read
  - RW
  - Write
- **4KB Accesses**
  - Read
  - RW
  - Write

- **Technologies**:
  - DDR-DRAM
  - DDR-PCM
  - Moneta-DRAM
  - Moneta-PCM
  - FusionIO
  - SSD-RAID
  - DISK-RAID
Latency Hiding

- Latency impacts DDR more
  - 64 byte requests
  - 128 for an 8KB transfer
  - Fine-grain bus contention
- Moneta hides latency well
  - 8KB requests
  - 1 for an 8KB transfer
  - Bus contention occurs once.
File System Performance

![Graph showing file system performance with various technologies like DDR-DRAM, DDR-PCM, Moneta-DRAM, Moneta-PCM, FusionIO, SSD-RAID, and DISK-RAID. The graph plots XFS Bandwidth (GB/s) against Raw Bandwidth (GB/s).]
Database Performance

Log Speedup vs DISK-RAID

- DDR-PCM
- HASTE-PCM
- FusionIO
- SSD-RAID
- DISK-RAID

- XDD 4KB RW
- Btree
- HashTable
- Bio
- PTF
SCMs will make fast IO a software problem

• Optimizing the driver helps a great deal, but costs remain
  – Entering the kernel
  – Interrupt processing
• The file systems destroys SCM performance
• End-to-end application benefits are small
• Research problems abound!!!
Today’s Talk

The Moneta array and quantifying the software problem

NV-heaps: Making the persistent objects safe and fast with SCM
SCM is a *memory*. Why should we treat it like disk?

Expose SCM memory like DRAM and give programmers the tools to use it like DRAM.
Exposing NVM Raw Storage

- Byte-addressable NV memories allow for direct access via load/store instructions.
- Access it with loads and stores.
Two Classes of Persistent Object Systems

• Assume slow storage but provide strong safety guarantees
  – Database frontends – Java Persistence, C# LINQ
  – Object-oriented databases (Thor, Objectstore, Texas, Quickstore)
  – Single-level stores (as400, Opal, etc.)
  – Orthogonally persistent Java

• Assume fast storage and leave everything up to the programmer
  – Rio Vista (battery backed DRAM)
  – Recoverable Virtual memory (RVM)
The Dangers of NV Memory

• All existing programming errors are still possible
  – Memory leaks
  – Multiple-frees
  – Locking errors

• Rebooting/restarting won’t help.

• Average programmers cannot get this stuff right

• System support can make it easier
  – Garbage collection
  – Transactions
New Kinds of Bugs

• Multiple heaps means 5 types of pointers
  – V-to-V – Normal pointers
  – V-to-NV – Useful
  – NV-to-V – inherently unsafe
  – Inter-heap NV-to-NV – Inherently unsafe
  – Intra-heap NV-to-NV – Necessary
Existing Primitives are Error Prone

Insert(Objec t * a, List<Objec t> * l)
{
    ... 
}

• Is a volatile? Is L? Are they in the same heap?
• One wrong call causes permanent corruption
Memory Management, Locking, and Persistent Objects

• Non-GC Memory management and locking disciplines are well-known sources of errors
• Both rely on a program-wide invariant that is...
  – Not specified in the source
  – Not enforced by the system.
• The NV pointer safety relies on a similar invariant.
• Programmers will get it wrong
Getting it Right: BPFS

• BPFS [Condit et. al. 2009]
  – Transactional file system for byte-addressable NV memory

A useful persistent object system for SCMs must be both fast and safe.

• Well worth the effort – write once, use many times
• 3 PhDs/persistent data structure does not scale.
NV-heaps: Safe Persistent Objects

• Safety First
  – Garbage collection
  – Transactions
  – Pointer safety
• Expose raw SCM performance.
• Scalability
  – Volatile storage requirement is constant and small.
  – Operations are $O(\text{touched data})$ not $O(\text{storage size})$
• Easy to use
  – All of the above
  – Leverage existing file systems and tools.
  – Intuitive separation between V and NV data.
class NVList : public NVOBJECT {
    DECLARE_POINTER_TYPES(NVList);
public:
    DECLARE_MEMBER(int, value);
    DECLARE_PTR_MEMBER(NVList::NVPtr, next);
};

void remove(int k)
{
    NVHeap * nv = NVHOpen("foo.nvheap");
    NVList::VPtr a =
        nv->GetRoot<NVList::NVPtr>();
    AtomicBegin {
        while(a->get_next() != NULL) {
            if (a->get_next()->get_value() == k) {
                a->set_next(a->get_next()->get_next());
            }
            a = a->get_next();
        }
    } AtomicEnd;
}
Implementing NV-heaps

- ACID Transaction Management
- ACID GC & pointer manipulation
- NVM allocation

- Locking, logging, and recovery
- Reference counting
- Pointer assignments
- Pointer type enforcement
- Reclamation
- Memory mapping
- Allocation and Deallocation
- Relocatability
The NV-heap Allocator

• Raw allocation and de-allocation
  — Per-thread free lists
  — Fixed-sized, write-ahead logging for durability.
  — Assignment is part of allocation
  — Epoch barriers for consistency [Condit et. al., 2009]

• Mapping
  — “Execute in place” support in Linux
  — Relative pointers for relocation
Garbage Collection and Pointer Safety

• Reference-counting GC
  – Per-object locks protect reference counts
  – Weak references for cycles

• Dynamic type system prevents dangerous NV pointers.
  – Wide pointers allow run-time checks on assignments
  – A static type systems is also possible.
Scalable locking for NV-heaps

- NV-heaps require per-object locks
- Volatile locks don’t scale
  - V storage rises with NV-heap size.
- Simple non-volatile locks don’t scale
  - On recovery, all locks need to be released
  - Recovery time scales with NV-heap size
Generational NV Locks

- Every NV-heap has a generation number
- A generational lock is an integer
  - If the integer > generation #, the lock is held
  - Atomic incr/decr acquires/releases
- Opening an NV-heap increments the generation number, freeing all the locks.
General, ACID Transactions

• Software transactional memory system
  – Object-based undo logging
  – Eager conflict detection with locks and version numbers
  – Flattens nested transactions.

• Logging
  – Per-thread NV write logs and V read logs
  – Using GC objects and pointers.
Evaluation Methodology

• Similar to NVRamDisk
  – But there’s no driver interaction in the common case
  – Emulate SCM with profiling and simulation

• Four workloads
  – Swaps per second
  – Btree
  – Hash Table
  – Six Degrees of separation

• Three other systems
  – Stasis [Sears, et. al., 2006]
  – Berkeley DB – Native implementations
  – Rio Vista-like interface: “Unsafe”
Speedup vs Disk

- 170-30,000x faster than disk
- Raw performance gap: 10,000x
Direct access and the cost of safety

- 8x faster than BDB
  - The benefit of direct access!
- 7x slower than “Unsafe”
  - The cost of safety.

Log Speedup vs BerkeleyDB

- Btree
- SPS
- Hash
- 6-Degrees
- Average

BerkelyDB SCM, NV-heaps, Unsafe
Conclusion

• Software is not ready for SCM
  – Old “optimizations” are wasted effort
  – Moneta shows that redesigning existing software can fix some of this

• NV-heaps can do better by rethinking the abstractions
  – Provide safe, easy to use, persistent objects
  – 7x overhead vs raw SCM
  – Very large application-level improvements
Thanks!

Rajesh Gupta
Questions?