Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories

Micro 2010
NVM-based SSD

• NVMs are replacing spinning-disks
  – Performance of disks has lagged
  – NAND flash showed increased performance
  – Emerging NVM (Phase Change Memory) has more potential

• Legacy of disk-based storage systems inhibit the potential
  – This legacy assumed that storage is slow

• H/W interface are sluggish
  – SATA and SAS are slow for faster NVM

• S/W limits the performance
  – Overhead of I/O stack is large
  – Under Linux, 20,000 instructions for 4KB I/O issue and complete
Architecting a High Performance SSD

This paper talks about architecting high-performance SSD, not NVM optimization.

1. SW layers limiting SSD performance should be optimized.
2. HW/SW interface is critical to good performance.

Careful co-design provides significant benefits.

Moneta prototype allows us to explore design space.

This is only for showing the importance of SW opt. under NVM-based SSD. Actual latency breakdown should show and explain More and more.
Moneta Prototype

- **Four memory controllers**
  - PCMs spread across four
  - Connected by ring network

- **A scheduler**
  - Coordinates data transfers
  - Interface to the host
  - Buffers for in/out data
  - Several state machines
  - Interface to ring network

- **Communicate with host**
  - x8 PCIe 1.1 (2GB/s/dir)
  - Programmed IO (PIO) for I/O request
  - DMA controller for data
Moneta Implementation

- **Four Xilinx Virtex 5 FPGAs**
  - Each implement memory controller
  - Ring networks runs over FPGA-to-FPGA links
  - One also implements Moneta scheduler and PCIe interface
- **DDR2 DRAM emulates PCMs (modeled)**
  - Projected latency: read (48ns) and write (150ns)
  - Same as 64-bit interface
I/O Request through Moneta

1. From host, a request arrives at request queue as PIO write
2. A request = three 64-bit info (sector addr., Data mem. addr., transfer length, tag, op-code, ...)
3. Scheduler allocates buffer for the request’s DMA data transfer
4. (W) Scheduler triggers DMA transfer from host’s mem. to buffer
5. Scheduler checks scoreboard to determine where memory controller has space to receive data
6. Once completed, it raises an interrupt and sets a tag status bit
7. OS processes interrupt and completes requests by reading and clearing the status bit using PIO operations
Baseline Performance

- 4KB I/O request latency (21us = HW 8us + SW 13us)
- Bandwidth of random read under varying access size
- Moneta baseline shows good performance, but room for smaller request (need more optimization, particularly, SW)
Removing I/O Scheduler

- **Linux IO scheduler** tries to sort and merge requests
  - Latency reduction for disks with **non-uniform access latencies**
  - For Moneta, it just adds software overhead
- **Moneta+NoSched** bypasses the scheduler completely
- For 4KB requests, **-2us latency** & **x4 bandwidth**
Atomic Operations

• Baseline Moneta interface requires several PIO write to issue an I/O request
  – Three 64-bit writes for all information

• Moneta+Atomic reduces this to only one
  – A request to fit into 64 bits
  – 8 for tag, 8 for command, 16 for length, 32 for address
  – DMA address field is completely removed

• To specify DMA address, static DMA address is pre-allocated for each tag

• Additionally, concurrency increased due to no need for locks to protect request issue
Atomic Operations

- Moneta+Atomic over Moneta+NoSched
  - Latency reduction (-1 us)
  - Bandwidth increase (460MB/s)
- Big gain in bandwidth from the increased concurrency
Adding Spin-waits

- Interrupt for request completion requires a context switch
  - To wake up the thread that issued the request
  - This adds latency for small requests

- Moneta+Spin allows tread to spin in a loop rather than sleep
  - Spin-waits for small request, but interrupt for large requests
Balancing BW Usage

- **Moneta+Spin** nearly saturates for read-/write-only workloads
- **Mixed of 50% reads and 50% writes**
  - Ideal BW doubles (2GB/s $\rightarrow$ 4GB/s)
  - PCIe link provides **full duplex** (2GB/s for each direction)
  - Moneta+Spin is still far from ideal performance
Balancing BW Usage

- In a single queue, read and write block each other
- Moneta+2Q maintains a Q for read and another Q for write
- Scheduler processes two queues in RR order, so data DMA transfer can be done in both direction simultaneously
Evaluations

- **FusionIO**: FusionIO SSD, 80GB, x4 PCIe
- **SSD**: Intel X-25E SSD, 128GB, SATA2, SW RAID-0
- **Disk**: Disk, 4TB, x4 PCIe, HW RAID-0
- **Moneta**: +Spin, x8 PCIe (2GB/s)
- **Moneta-4x**: +Spin, x4 PCIe (1GB/s)
- **Moneta outperforms due to faster x8 PCIe link for larger req.**
Conclusions

• Faster NVMs are coming
• Moneta prototype is implemented to understand NVM-based storage system
• Both SW and HW optimization opportunities are investigated