Transactions in a Flash

Morgan Price

Scott Nettles

Flash Memory: Pros

- Solid-state memory
- Billions of dollars a year spent
- Persistent
- High-density
- Cheap
- Low latency
- High read bandwidth

Flash Memory: Cons

- Low write bandwidth
- Write once until explicitly erased
- Erased in large blocks
- Erasing is slow
- Limited lifetime
- High cost compared to disks

Key Issues

- Is Flash useful for transaction systems
- Where in the memory hierarchy
  - Disk-like block-oriented device
  - Byte-oriented device
  - Directly accessed as user memory

Our Experiment

- Compare transaction logs using
  - Disk
  - Battery-backed DRAM
  - Flash Memory
- Emulate Flash with DRAM, timers
- Simple, fair comparison
  - No special Flash optimizations

Outline

- **Background**
  - Transaction systems
  - Flash memory
- Design & Implementation
- Experiments
- Performance Results
- Discussion
- Related & Future Work

Transactions

- Reliability
- Resiliency to machine failure
  - Permanent storage
- Lots more to transaction systems besides permanent storage
  - Consistency
  - Concurrent access

Permanent Storage

- Disks
  - file-system
  - raw partitions
- Protection from disk failure
  - mirroring, RAID, tape
- Commit
  - atomic update to persistent data

Fast Commit

- Store updates separately
  - in an append-only log
  - easy to append atomically
- When log fills
  - Truncate & reuse
- Appending disk logs is relatively fast
  - No seeks, just rotational delay
  - Batch transactions to reduce latency

Batched Commits

- High disk latency for writes
- Reduced by batching commits
  - Increases transaction throughput
  - Increases transaction latency
- Requires lots of concurrency
  - Great for database systems
  - Maybe not for other transaction systems
    - persistent heaps
    - filesystem meta-data

Flash Logging

- Write-once
- Erase on truncation
- Automatic wear-leveling
- Low write latency
- High parallelism is feasible
  - hardware failures should be very rare
  - simpler than RAID

Flash Memory

- EEPROM, but with block erases
- 1 transistor per Flash cell
- Flash cells are 30% smaller than DRAM
- Lower cost per bit, eventually
- Intel has samples with >1 bit per cell
  - Flash could be used for main memory
- Cheaper than battery-backed DRAM

Reading from Flash Memory

- Organized similarly to DRAM
- Flash chips with DRAM read interfaces
- Flash read performance matches DRAM

Writing to Flash Memory

- Slow: 6 µs versus 65 ns for read
- Each bit can change from 1 to 0
  - but not back
  - Writing Flash is an AND
- Very low latency compared to disk
- 163 KB/s per chip
  - low bandwidth compared to disk

Erasing Flash Memory

- Erases a whole block: 64 KB
- Conditioning
  - Forces each bit to 0 before erasing
  - Slows down erase
  - Raises lifetimes
- 600 ms latency
- 107 KB/s per chip

Flash Lifetimes

- Write and erase “stress” the chip
- Too-slow blocks have “failed”
  - no data loss
- 100,000 erase cycles guaranteed
- 1,000,000 expected given
  - wear-leveling
  - retirement of (rare) failed blocks

Outline

- Background
- **Design and Implementation**
  - The transaction system
  - Using Flash as a transaction log
  - Flash emulator
  - Device drivers for different access models
- Experiments
- Performance Results
- Discussion

The Transaction System

- Recoverable Virtual Memory (RVM)
- Persistence on regions of virtual memory
- Assumes small persistent working set
  - Must fit in physical memory for good performance

RVM's Transaction Log

- Circular disk-based log
- Truncation forces updates to actual data
  - Big operation with high latency
  - Can be asynchronous

Changes to RVM

- Small
  - Added 500 lines of C out of 20,000 original
- Erase log after truncation
- Erase entire log on recovery
- Minor changes to device configuration
- Occasional in-place updates to meta-data
  - Replaced with a mini-log

Flash Mini-log

- Data block has a log of values
- Commit appends value & mark
- To reclaim space in a block
  - at least two data blocks
  - third block points to valid data block
- Infrequently used
  - performance is not a concern

Sidney

- Persistent heap for Standard ML (SML)
- Based on SML/NJ and RVM
- No changes to Sidney were needed
- No garbage collection
  - Sidney doesn’t use the transaction log
  - Flash & copying garbage collection

The Flash Emulator

- Emulates Flash with DRAM
- Reading Flash is fast
  - RVM forces an unnecessary copy
- Delay writes and erases with timers
- Worst error was 30 ns/byte

Flash Configurations

- Vary the bandwidth
  - as if varying the parallelism
- Emulate battery-backed DRAM
  - just turn off delays
- Vary memory hierarchy

Flash as Kernel Memory

- Simple character device driver
- Emulates Flash with kernel memory
- Erase as ioctl

Flash as a Disk

- Another simple device driver
- Ignore overhead of in-place write semantics
  - Writes are contiguous
- Ignore cost of I/O bus
- Faster writes through page buffers?

Outline

- Design & Implementation
- **Experiments**
  - Benchmarks
  - The Flash memory simulated
  - The benchmarking environment
- Performance Results
- Discussion
- Related & Future Work

C++ Debit-Credit

- Closely based on TPC-B
- Does not scale database size by TPS
  - Fixed at 100,000 accounts
- Each transaction modifies 448 bytes
  - RVM writes 748 byte commit record
- Run 50,000 transactions and truncate
- 16 MB log fills twice

Sidney Debit-Credit

- Written in SML instead of C++
- Sidney implicitly logs writes
  - 80 bytes sent to RVM
  - 548 byte commit record
  - 16 MB log fills once

SML/NJ Compiler

- Compiles 38 of its own files
- Stores data in persistent heap
- Commits after each file
- Lots of actual computation
- 15 KB transactions

Flash Memory Details

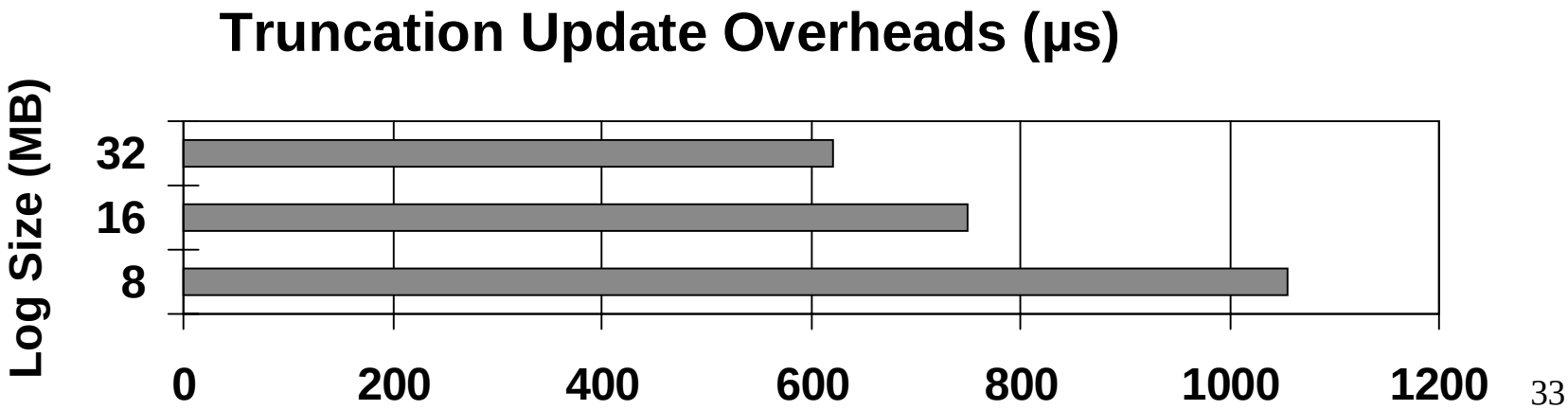
- Parallelism from 4 to 64
  - Our memory controller goes up to 512-way
- Intel's 28F016SV: 2M by 8 bits
  - 65 ns reads, 6 µs writes (163 KB/s)
  - 32 erasable blocks of 64 KB each: 107 KB/s
  - Two 256-byte page buffers
    - bulk writes at 465 KB/s
  - Suspends slow operations for faster

Benchmarking Machine

- SGI Challenge-L
  - 4 R4400 processors at 250 MHz
  - 384 MB of main memory
  - IRIX 5.3
- Disks rotate at 7200 RPM
  - limits RVM's disk log to 120 TPS
- ~6 MB/s of raw disk bandwidth
- Flash emulator delays accurate to 30 ns/byte

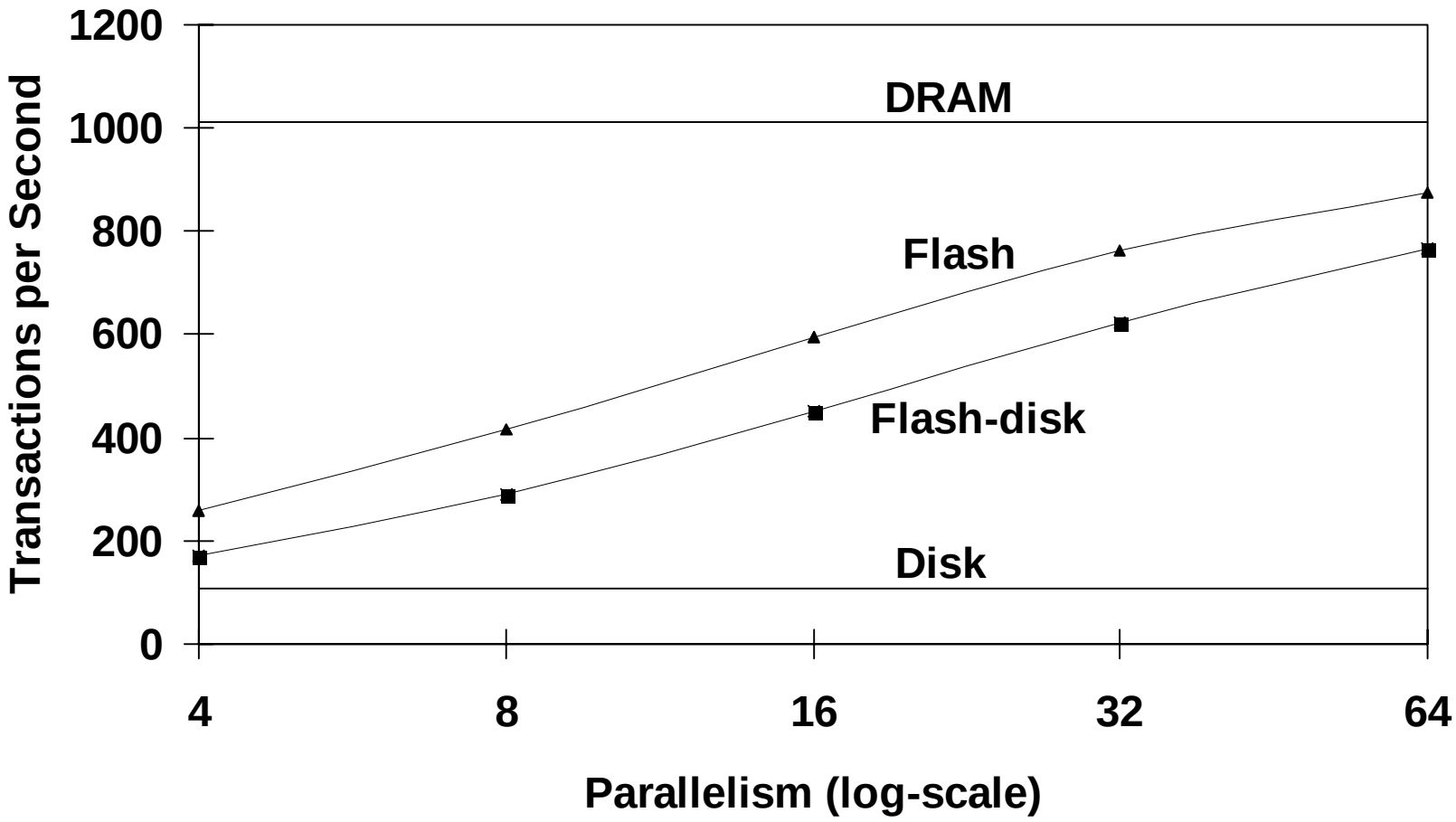
RVM Configuration

- RVM data stored in the EFS filesystem
- Disk-based log stored on a raw partition
- Log size fixed at 16 MB
  - not including the mini-log



Performance Evaluation

- Vary Flash parallelism
  - Flash results in significant speedups
  - Flash is bandwidth limited
  - Disk is latency limited
- Vary the memory hierarchy
  - Flash-disks suffer from fragmentation
  - Kernel overheads are insignificant



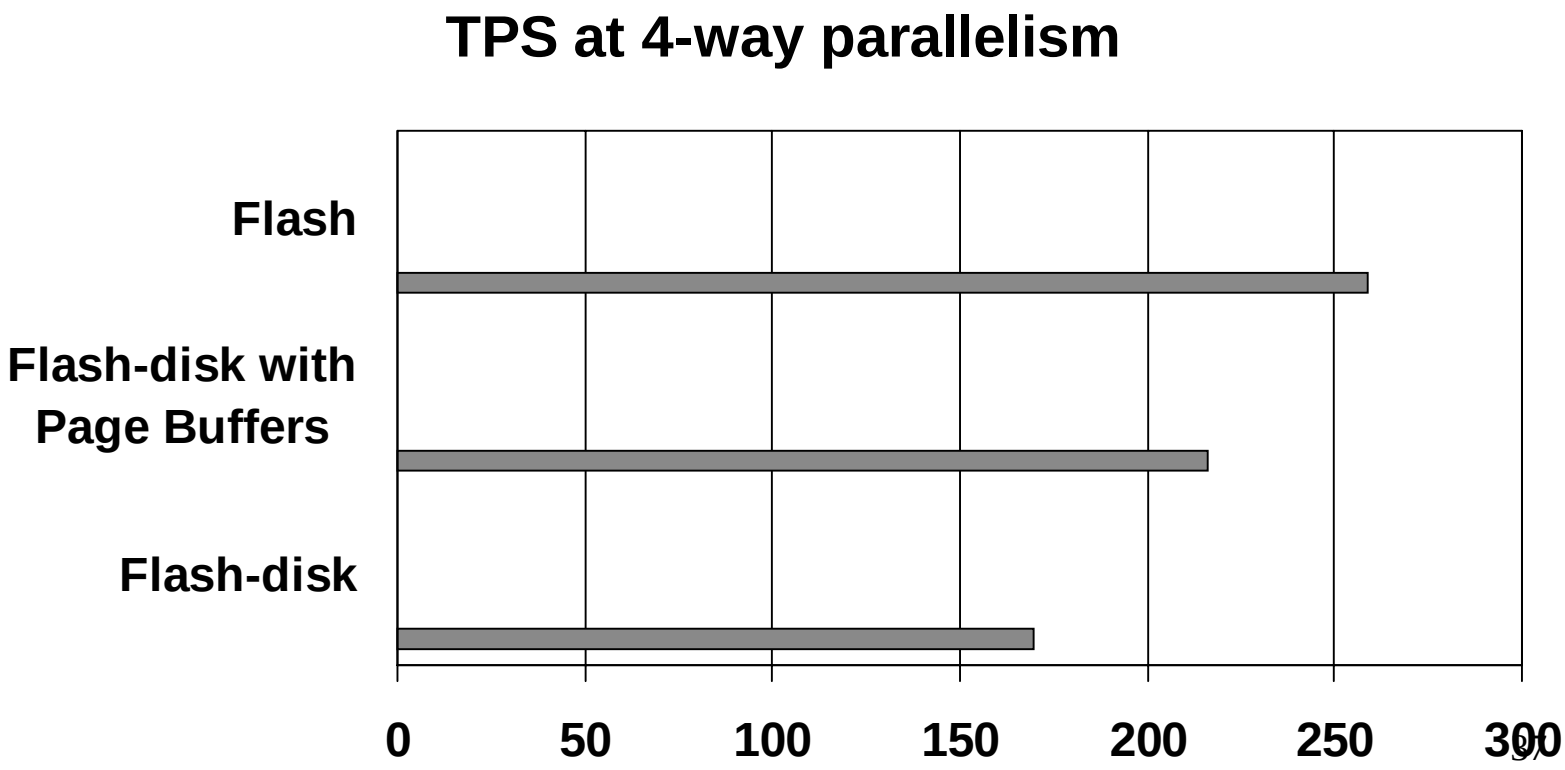
- Flash log is bandwidth-limited
- Flash approaches battery-backed DRAM

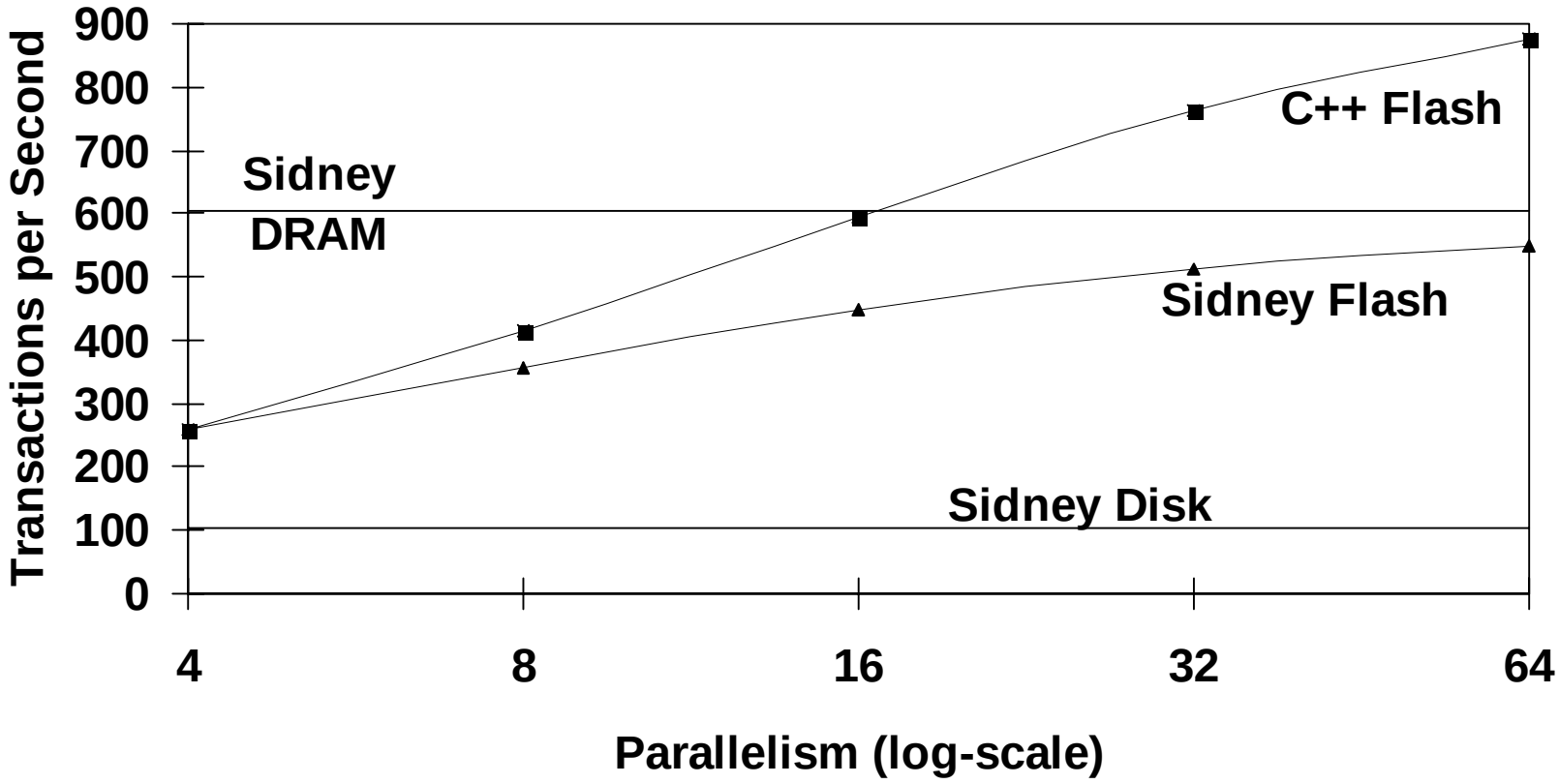
Why is the Flash-Disk Slow?

- Not due to kernel overheads
  - Turn off cycle timers
  - Battery-backed DRAM vs. fast Flash-Disk
  - No difference
- Due to fragmentation
  - Writes 67% more bytes per transaction

Using Page-Buffers

- Triples the write bandwidth (optimistically)
  - Sector size < Parallelism * Page Buffer Size





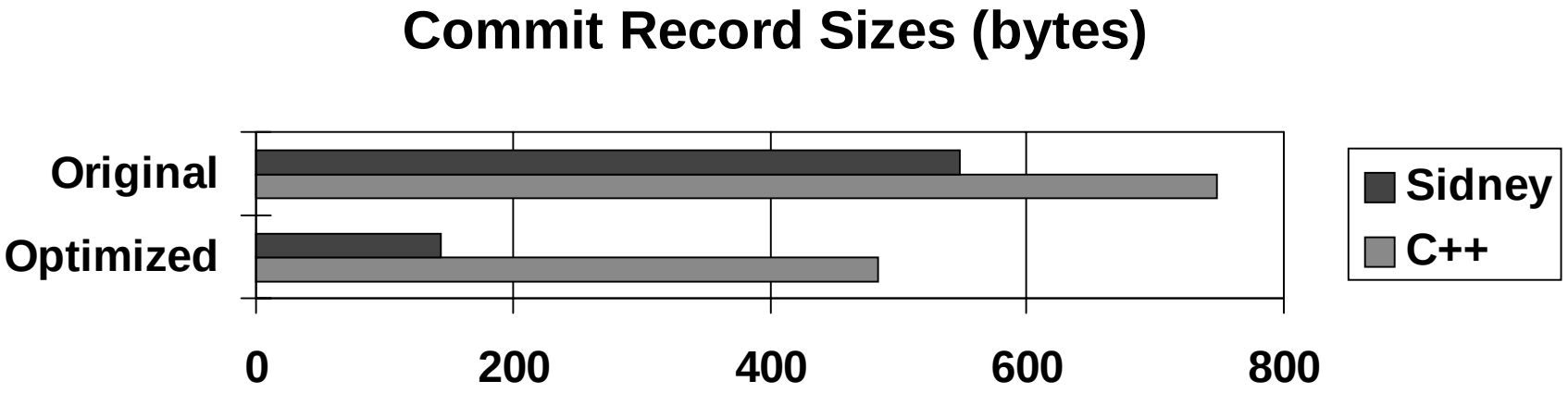
- Low peak throughput
  - CPU overheads rise from 0.3 to 1.0 ms

Why isn’t Low-Parallelism Sidney Slower?

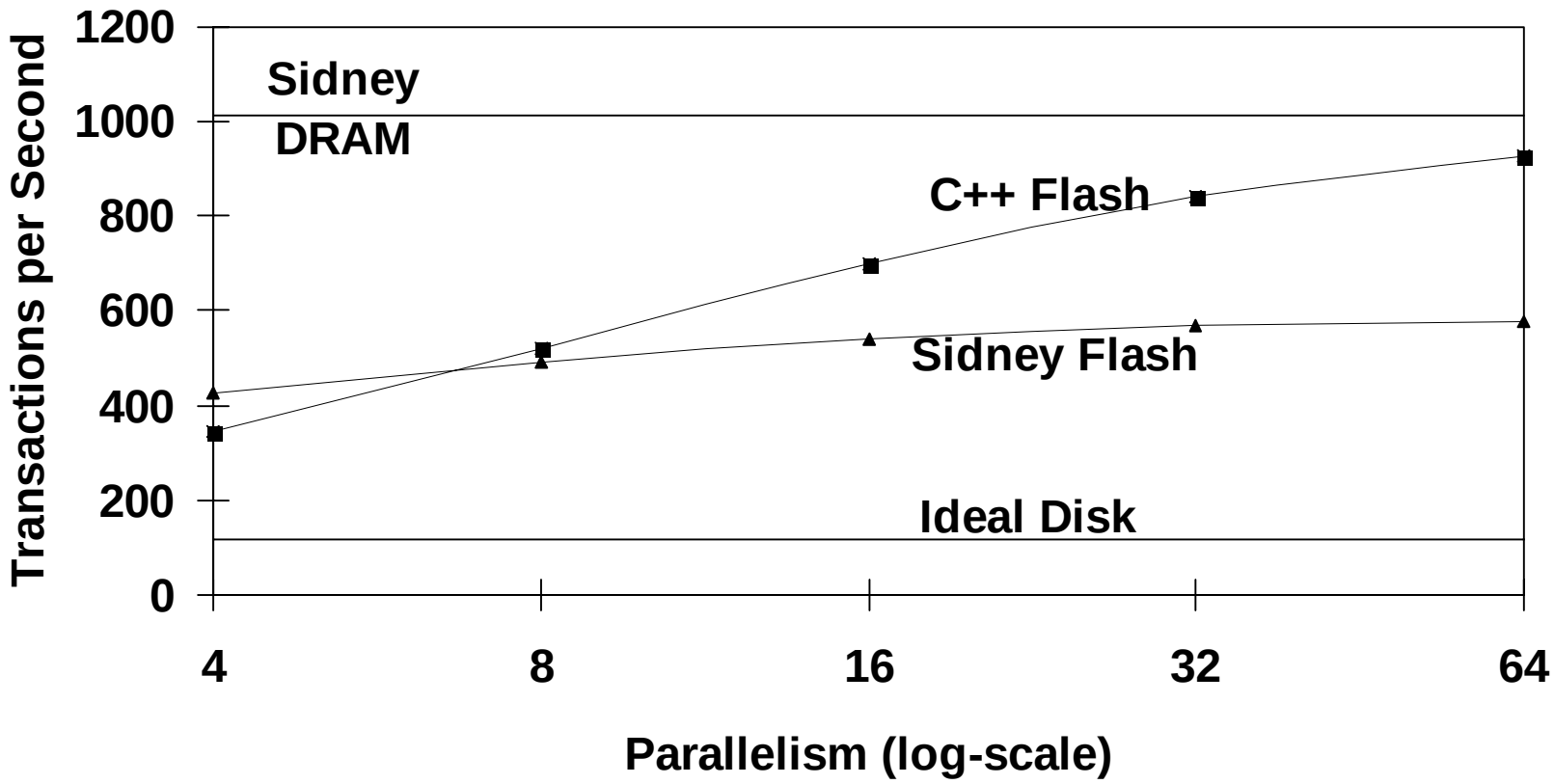
- Big log bandwidth savings!
  - Writes 548 vs. 748 bytes
- Flash requires compact logs

A Better Log Representation

- RVM has high header overheads
  - 76 bytes per transaction
  - 56 bytes per range modified
- Reasonable headers are 8 byte each!

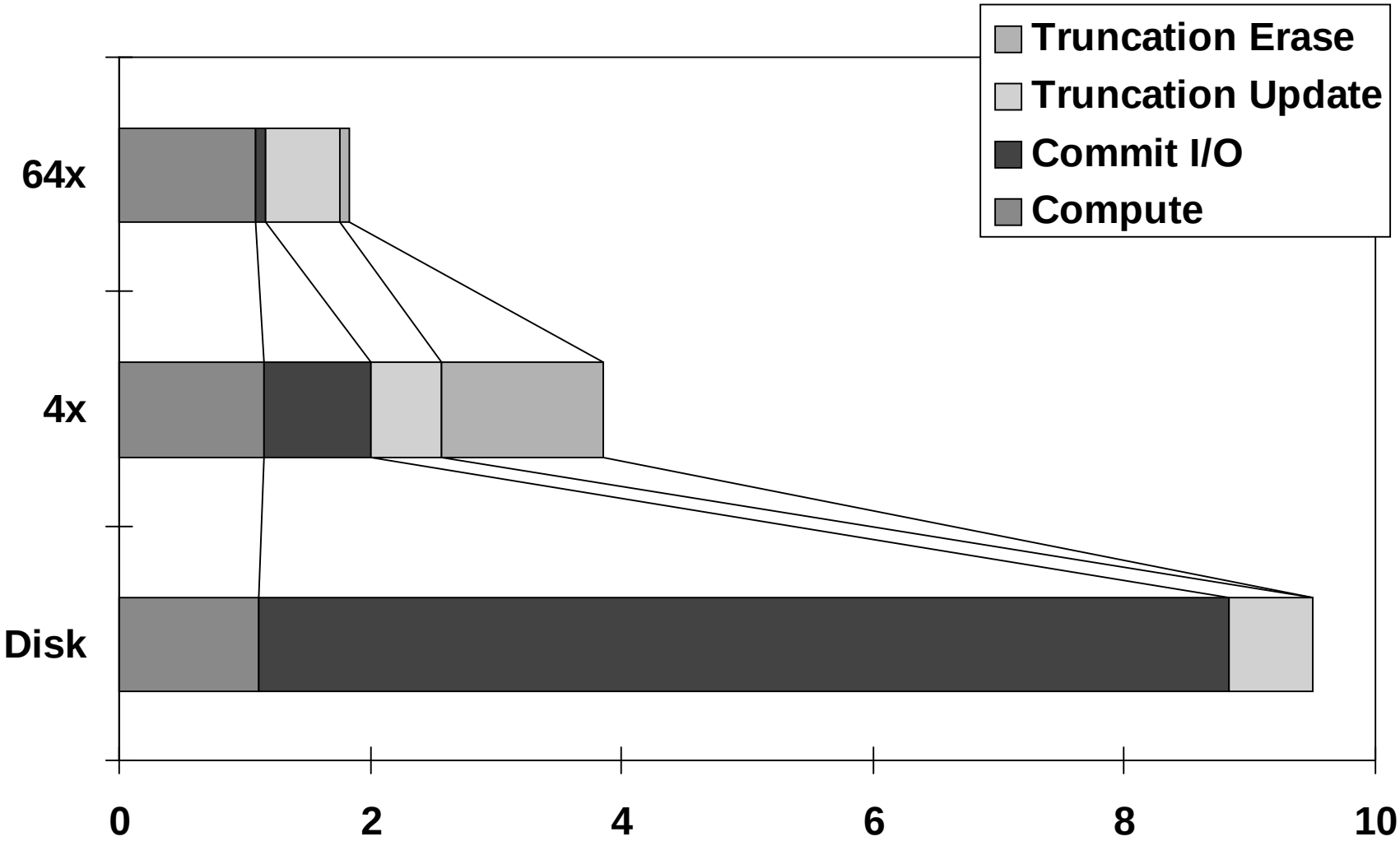


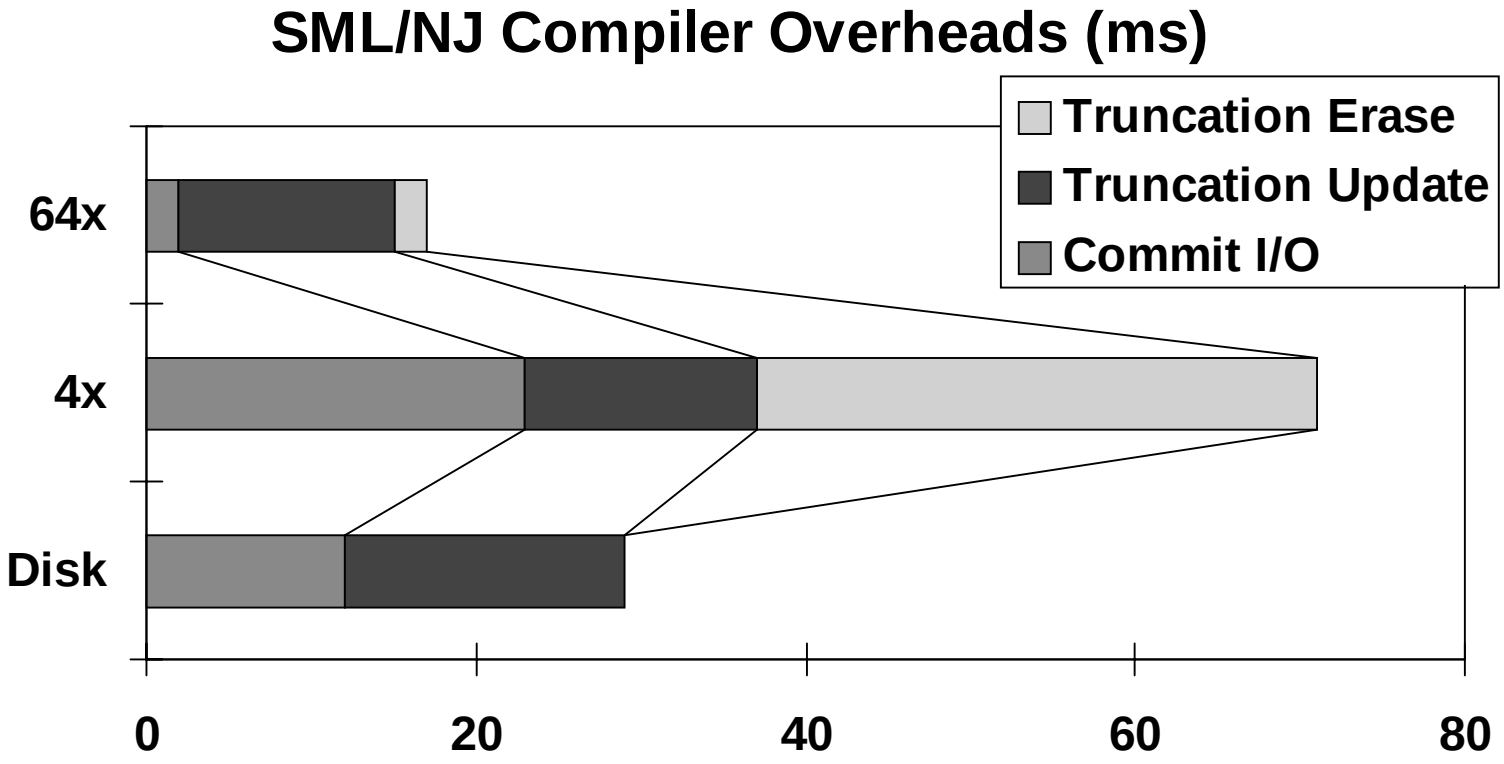
Debit-Credit with Optimized Headers



- Big win for Sidney version
- C++ programmer could do the same
- RVM could compare the modified ranges

Sidney Debit-Credit Overheads (ms)





- 670 ms of Compute time not shown
  - Allows background truncation

Outline

- Performance Results
- **Discussion**
  - Flash Life-time
  - How to Extend it
  - Bigger Transactions
- Related Work
- Future Work
- Conclusions

Flash Life-time

- 1,000,000 erase cycles
  - Conservative given block retirement
- Block retirement by
  - virtual to physical remapping
  - sector remapping
- At least 13 MB of data at 8x
- (13 MB * 1,000,000 erases) /
  (748 bytes * 1000 TPS) = 200 days

Extending Life-time

- More Flash extends life
  - at expense of price-performance
- Header optimizations
  - Extend lifetime to at least 2 years
- Better hardware
- Bursty work-loads
- Actually reading the Flash

Log Compression

- Compiler log compresses by 2x
  - Real application
  - Would double flash lifetime
- Compress/Decompress runs at 1 MB/s
- Improves performance if
  - Write&Erase Bandwidth < 1/2 MB/s
  - Breaks even at 8x parallelism

Hybrid Logging

- For large transactions, want both
  - low-latency (Flash)
  - high-bandwidth (disk)
- Write-ahead logging
  - standard optimization
  - “speculatively” writes to disk
- Use Flash for the final commit write

Related Work: eNVy

- Persistent memory controller
  - 256-bytes wide
  - 2 GB of flash
- Allow in-place update via 64 MB of SRAM
- Differences of
  - Data area
  - Scale: eNVy supports I/O rates of 30,000 TPS
  - Custom hardware, SRAM
- Wu & Zwaenepoel, in ASPLOS ‘94

Related Work: Filesystems

- Flash file-systems for mobile computers
  - Low-power
  - High durability
  - Douglas et al, in OSDI ‘94
  - Kawaguchi et al, in Usenix ‘95

Outline

- Related Work
- **Future Work**
  - Real Hardware
  - Improved Logger
  - Redesigning Sidney to use Flash directly
- Conclusions

Real Hardware

- A simple memory controller
  - The Intel 28F016XD has a DRAM interface
- New performance measurements
  - Effect of background truncation on throughput
- Allow page-buffer use on small writes
- System cache structure
  - Forcing persistent writes to Flash
  - Flushing stale data upon erase

Improved Logger

- Header space optimizations
- Replace RVM logger
  - Designed for disks
  - CPU overhead for error checking
- Retirement of slowed blocks
  - page-remapping
- Batched commits

Flash-Based Persistent Heaps

- Use Flash as bulk of main memory
  - Eliminate the disk update overheads
- Most Sidney data is immutable
- Copying garbage collection
  - append-only
  - frees up large chunks to be erased
- Keep mutable data, young data in DRAM

Conclusions

- Flash memory is well-suited for transaction logging.
- Flash logging is
  - easy to implement.
  - fast for small transactions
  - can rival battery-backed DRAM for speed

Thanks to

- Satya and the CODA group for RVM
- Puneet Kumar for C++ Debit-Credit
- Mark Foster for help with memory systems
- Michael Wu for information about eNVy