A Comparison of Sorting: Shared-Everything vs. Shared-Nothing Machines

Andrea and Remzi Arpaci-Dusseau

Computer Science Division
University of California, Berkeley

http://now.cs.berkeley.edu

Talk Overview

- Parallel talk on parallel sorting
  - AlphaSort as example of sorting on shared-everything SMPs
  - NOW-Sort as example of sorting on shared-nothing clusters

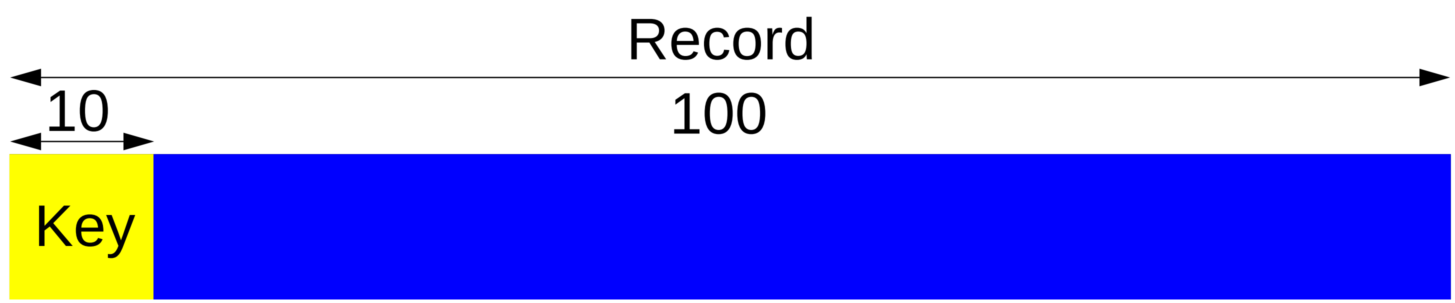
- Outline
  - Background
  - HW/SW differences
  - Single-node sorting
  - Parallel Sorting
  - Conclusions

Why Sorting?

- Why external sorting (disk-to-disk) as a case-study?
  - Memory and I/O intensive
  - Important to database community
  - Parallel algorithms understood
  - Established competition

Benchmark Definition

- 10 byte keys inside 100 byte records



- Datamation (1985): How long to sort 1 million keys?
  Problem: Not I/O benchmark anymore
- MinuteSort (1994): How much sorted inside of 1 minute?
  Problem: More records than memory
  Two-pass sort instead of one-pass

- Price-performance
  - Calculate cost to run over 5 years
  - Multiply by time for sort

Pre-AlphaSort

- Datamation results:

System	Year	CPUs	Disks	Time (secs)	Cost/sort
Tandem	1986	2	2	3600	$4.61
Beck	1988	4	4	6000	1.92
Tandem	1990	3	6	980	1.25
Cray YMP	1986	1+	1	26	1.25
Kitsuregawa	1989	16	1	320	0.41
Baugsto	1990	16	16	180	0.23
Sequent	1990	8	4	83	0.27
Baugsto	1990	100	100	40	0.26
Intel iPSC/2	1992	32	32	58	0.37

- Observations
  - Number of disks usually equals number of CPUs
  - Cost per key drops 10x over 6 years

AlphaSort and Beyond

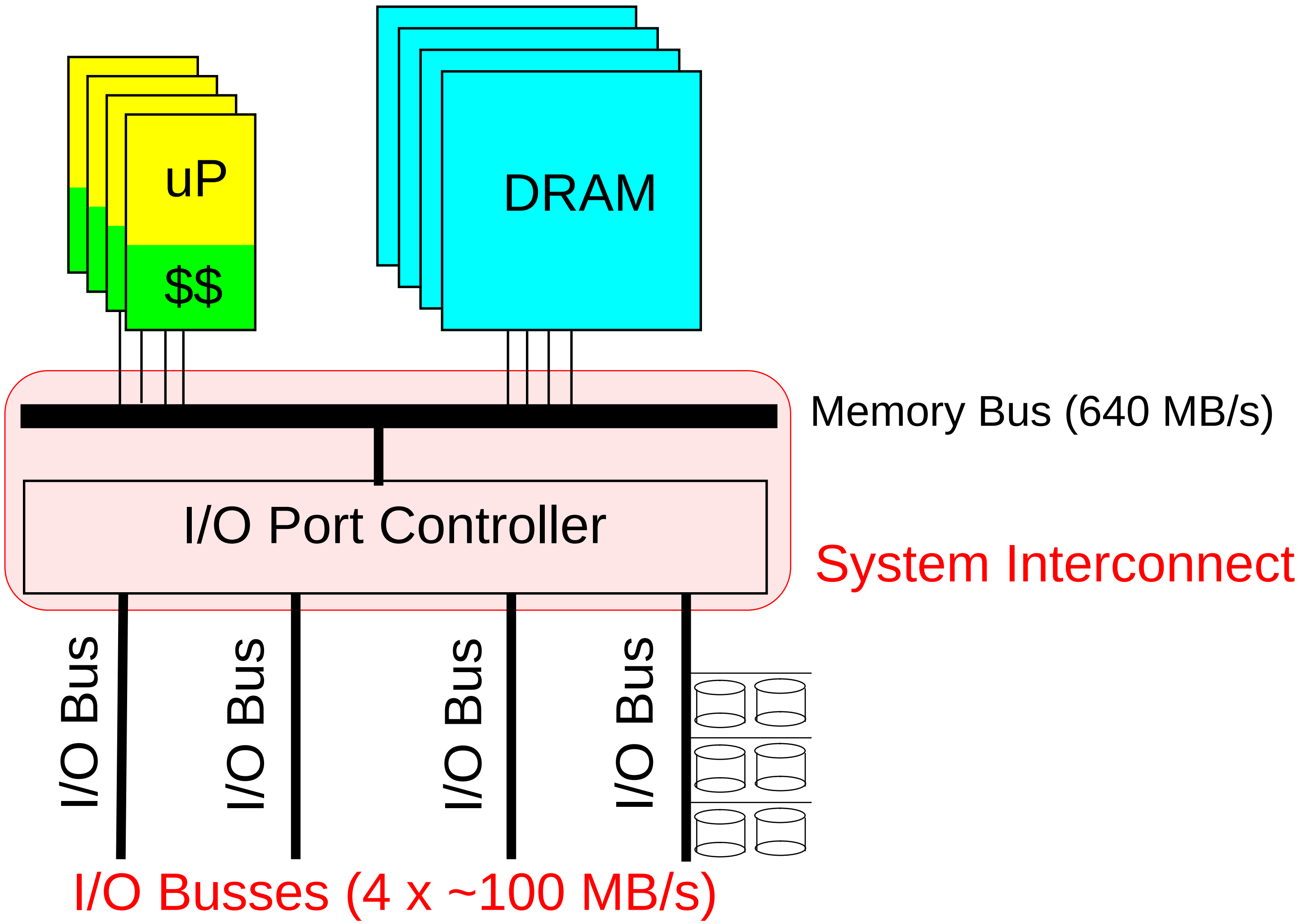
- Have disks, will sort:

System	Year	CPUs	Disks	Datamation	Cost
AlphaSort	1993	3	36	7.0 s	$1 Million
SGI XFS	1995	12	96	3.52 s	~$2 Million
NOW-8	1996	8	32	??	$200 K
NOW-64	1996	64	128	??	$1.2 Million
NOW-95	1996	95	190	??	$1.8 Million

- SMP world
  - AlphaSort shows server-class machines sort well
  - Commodity processors, memory, and disks
- Can we take “commodity” to the NEXT STEP?
  - Can commodity desktop workstations challenge sorting behemoths?
  - NEED: to empirically evaluate workstation clusters

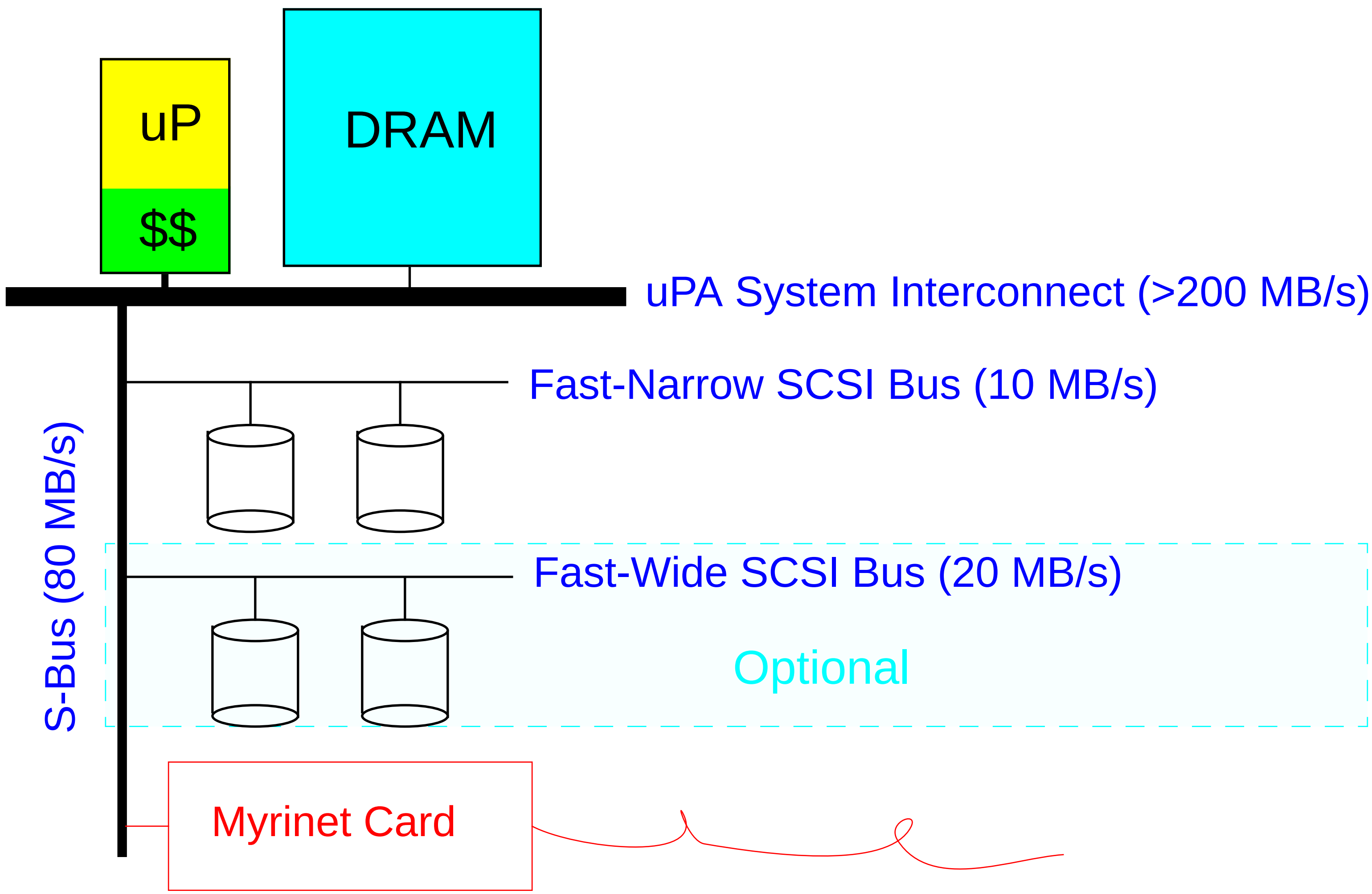
AlphaSort Hardware

- Commodity processors, memory, and disks



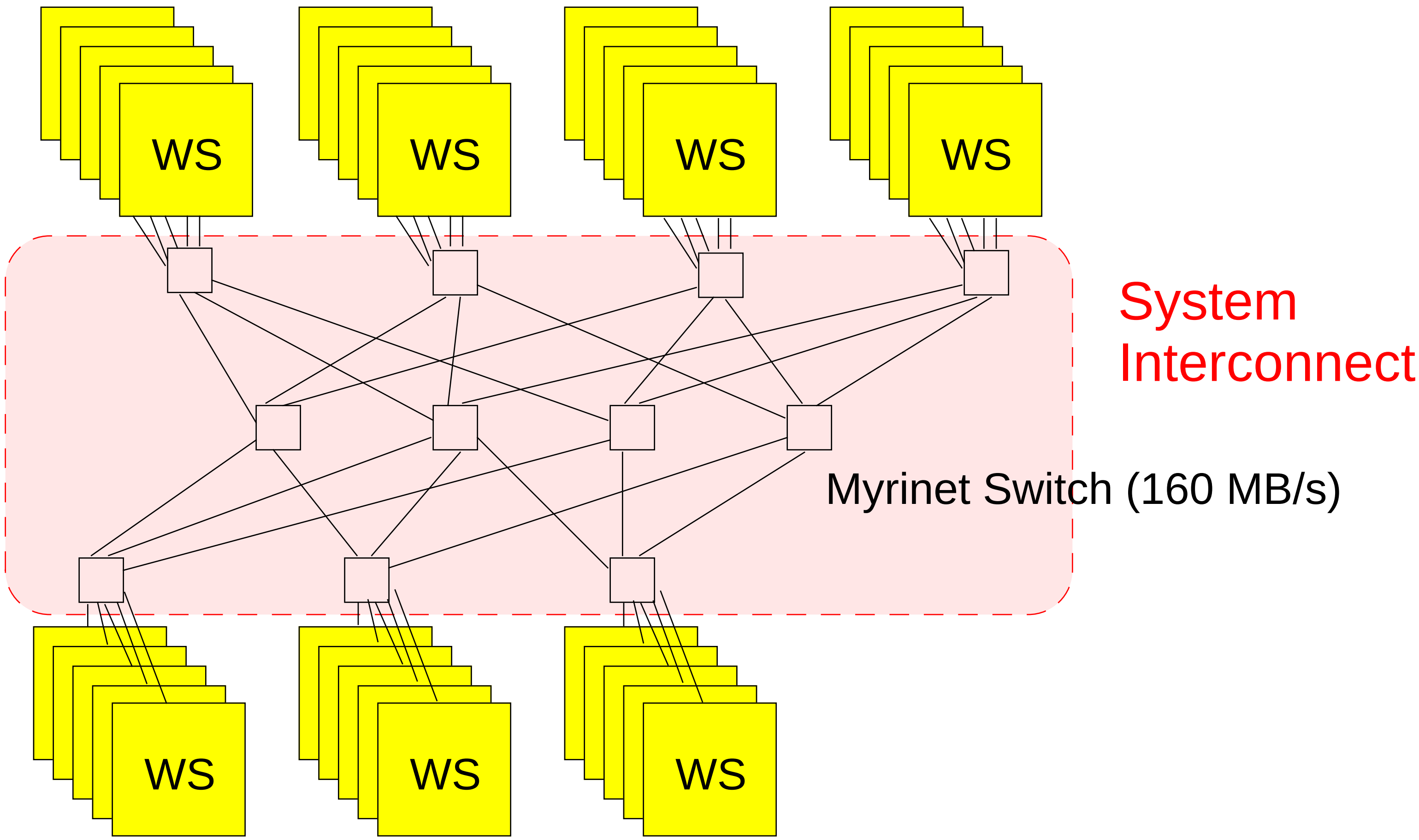
NOW-Sort Hardware: Single Machine

- Entire workstation is commodity!



NOW-Sort Hardware: Cluster

- Cluster: commodity WS + commodity switch



AlphaSort Programming Environment

- Abstraction of parallelism: Threads
- Communication paradigm: Shared Memory
  - Threads use memory buffers to share data
  - Explicit locking used to ensure proper behavior
  - Communication performance:
    - ~1 us latency, ~100 MB/s bandwidth
    (modern machine: 1/2 latency, 2x bandwidth)
- Operating system: DEC OpenVMS
  - Not UNIX!

NOW-Sort Programming Environment

- Abstraction of parallelism: Multi-level
  - One-process per node (coarse-grained)
  - Multiple-threads per process (fine-grained)

- Communication paradigm: Active Messages
  - Moves keys + records between nodes
  - Bandwidth, not latency, is important for sorting
  - Communication performance:
    - 10 us latency, 35 MB/s bandwidth

- Operating system: GLUnix + N copies of Solaris
  - Need parallel environment (job control, process start-up)
  - Scheduling not an issue (dedicated environment)

Outline

- Outline
  - *Background*
  - *HW/SW differences*
  - Single-node sorting
  - Parallel Sorting
  - Conclusions

AlphaSort: Single-Node Overview

- Uses multiple threads
- One thread reads records from disk into buffer
  - Another thread sorts buffer
- Merge multiple sorted-runs into one run
- Gather records into contiguous buffer and write to disk

NOW-Sort: Single-Node Overview

- Single thread
- Read records from disk
  - Simultaneously partially sort by placing into buckets
- Sort each bucket
- Gather records into contiguous buffer and write to disk

AlphaSort: Reading from Disk

- Single 1993 SCSI disk: Read 4.5 MB/s, Write 3.5 MB/s
  - Problem: 25s to read 100 MB and 30s to write
- Solution: Software striping
  - File striping on top of OpenVMS file system

    Stripe definition file: *N* pairs of (File name, Block size)

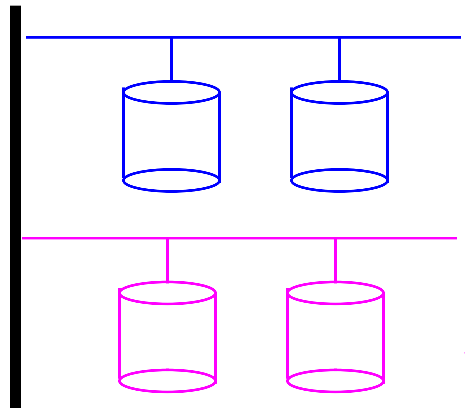
    `Open` with new interface: Opens *N* base files
- Compare many-slow vs. few-fast disks (same capacity: 36 GB)

	Many Slow Disks	**Few Fast Disks**
Drives	36 RZ26	12 RS28 + 6 Velocitor
Disk Speed	1.8 MB/s	SCSI: 4 MB/s IPI: 7 MB/s
Stripe Rate:	64 MB/s	52 MB/s (not 90 MB/s!!)
List Price	$85, 000	$122,000

- Many-slow has better performance and price-performance

NOW-Sort: Reading from Disk

- SCSI disks faster (and cheaper)



- Variable stripe sizes for *heterogenous* disks

Disks	SCSI Bus	Read Bandwidth
2 Hawks	Narrow	8.3 MB/s (bus saturated)
2 Barracudas	Wide	13.0 MB/s
All 4	Both	16.0 MB/s (naive)
All 4	Both	20.5 MB/s (2:3 ratio)

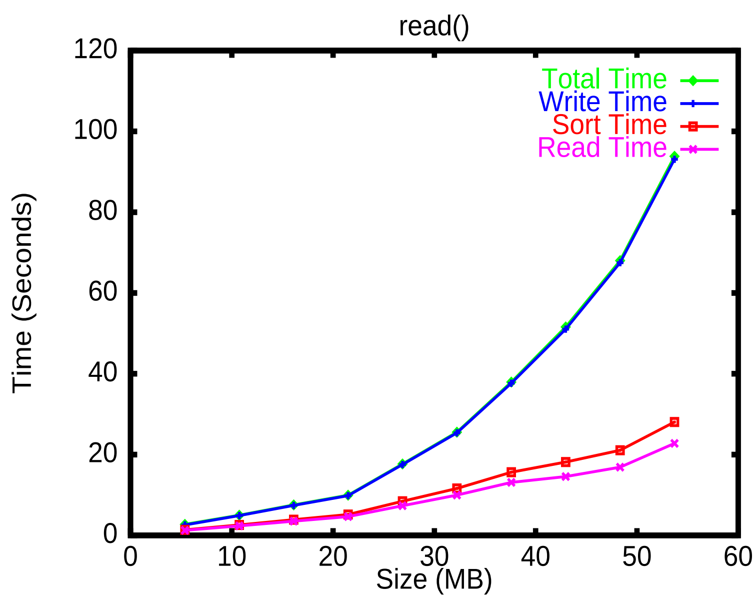
- Fast-Narrow SCSI bus is bottleneck

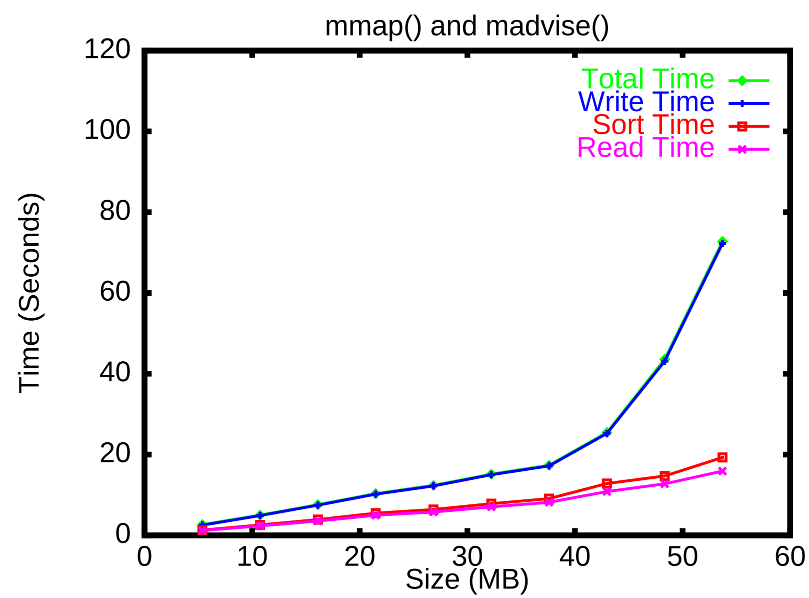
AlphaSort: Managing Memory

- Desired interface for file I/O
  - Sequentially read from input file, access, throw away
  - OpenVMS has unbuffered I/O
- Manage prefetching explicitly with threads (triple-buffering)
  - One thread reads 1 MB buffer
  - One thread sorts 1 MB buffer
- Throw lots of memory at problem
  - Use 256 MB and 384 MB of main memory to sort 100 MB!

NOW-Sort: Managing Memory

- Compare `read` and `mmap` with 64 MB real memory





- Results
  - `read`: double buffering painful
  - `mmap:` (not shown) poor replacement policy for input file
  - `mmap+madvise:` good performance/semantics, 10% memory tax

AlphaSort: Internal Sorting

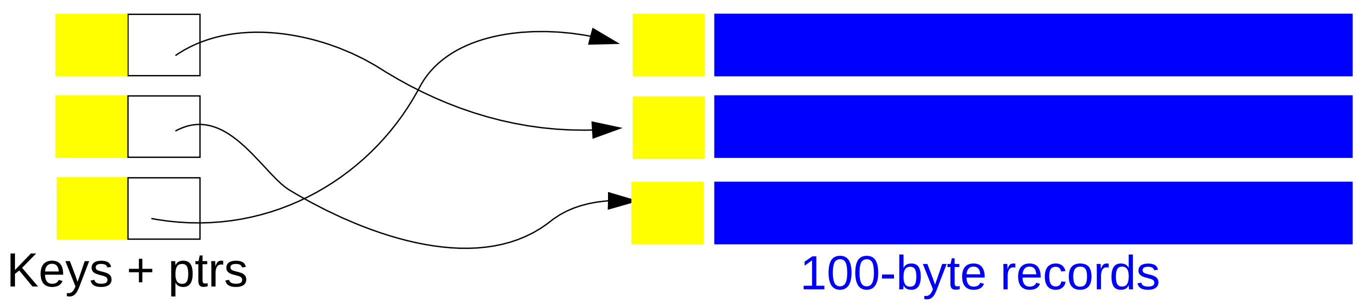
- Previous: Disk-to-disk sorts used *Replacement-Selection*
  - Build tournament-tree in memory
  - Output minimum element; replace with next key
  - If next key smaller, start next run; Otherwise, continue
- Run length is expected to be: *2 * Buffer Space*
  - Great when little available memory and for two-pass sorts
  - Poor cache behavior: Random accesses to tree
  - Sorting 1 MB buffers at a time

    ~ 100 KB of keys

    8 KB of on-chip cache
- AlphaSort: *QuickSort* instead
  - Good cache behavior: Mostly sequential accesses

AlphaSort: Internal Sorting

- Improve memory system behavior further
  - Remember: 90 extra bytes for every 10-byte key
- Three options:
  - *Record sort:* Swap full record within QuickSort; Copy 100 bytes
  - *Pointer sort:* Keep pointer to record; Save copy by derefencing
  - *Key sort:* Keep `(key, pointer)`; Save copy with more memory



- Performance regimes
  - Short records (*R* <= *16* bytes): Use record sort
  - Otherwise: use key sort

AlphaSort: Internal Sorting

- Optimization:
  Fit more `(key,pointer)` pairs in cache and align by cache-line
  - Use partial key for comparisons (4 bytes vs. 10 bytes)
  - Dereference pointer and access full key only on ties
  - 25% improvement on QuickSort time

AlphaSort: Merging Sorted Runs

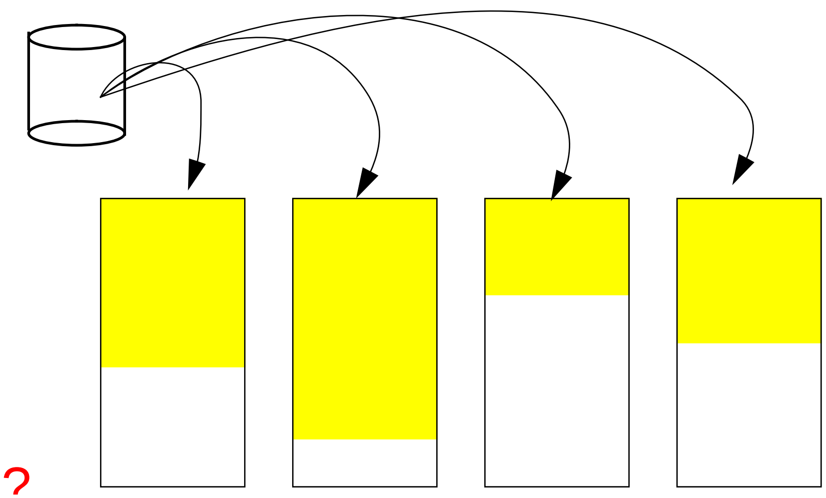
- Merge multiple 1 MB sorted runs into one final run

- Use previously criticized replacement-selection tree
  - Much smaller tree (10 runs in Datamation benchmark)
  - Good cache behavior
  - Usually examine only first word kept in `(partial key,pointer)` pair

NOW-Sort: Internal Sorting

- Learn from AlphaSort
  - Use key sort: (partial key, pointer) and NO replacement-selection!
- Quicksort
  - Sorts all key+ptr pairs
- Partial bucket sort + quicksort

  Examine key, distribute
  to bucket based on value
  of top *B* bits

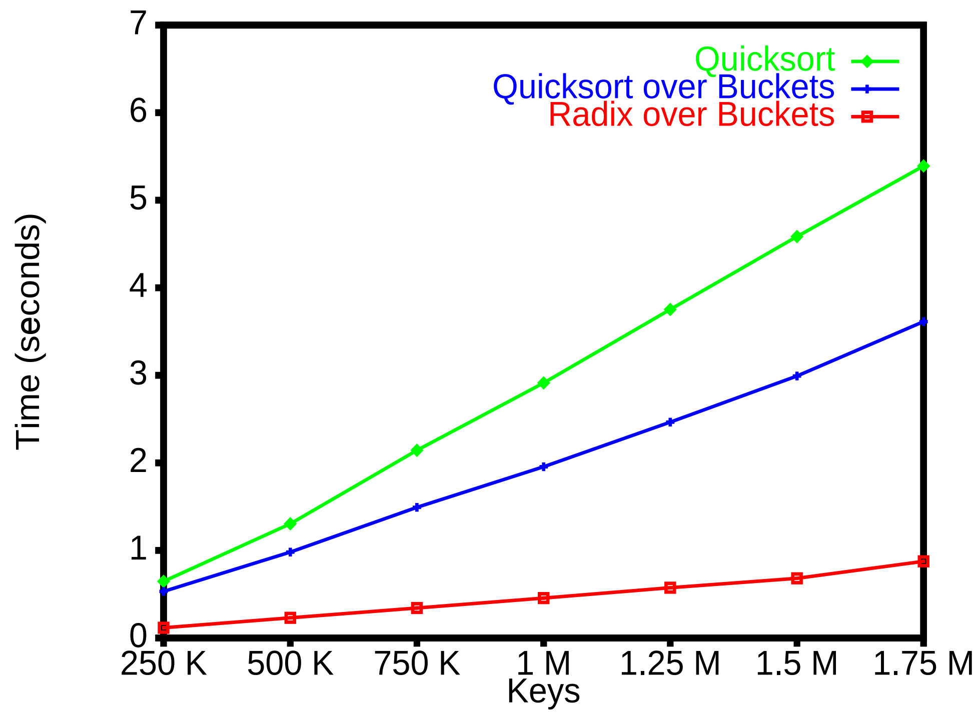
  (Records still in separate array)

  Sort each bucket individually

  How big should each bucket be?
- Partial bucket sort + radix sort
  - Buckets only get half as many keys

NOW-Sort: Internal Sorting Comparison

- Quantitative comparison:



- Key: optimizes for L2 cache
- Result: internal sort time negligible

AlphaSort: Gather and Write

- Records are gathered into output buffer
  - Follow `(partial key, pointer)` pairs
- Write buffer to disk

NOW-Sort: Gather and Write

- Records are gathered into output buffer
  - Follow `(partial key, pointer)` pairs
- Write buffer to disk

AlphaSort: Single-Node Results

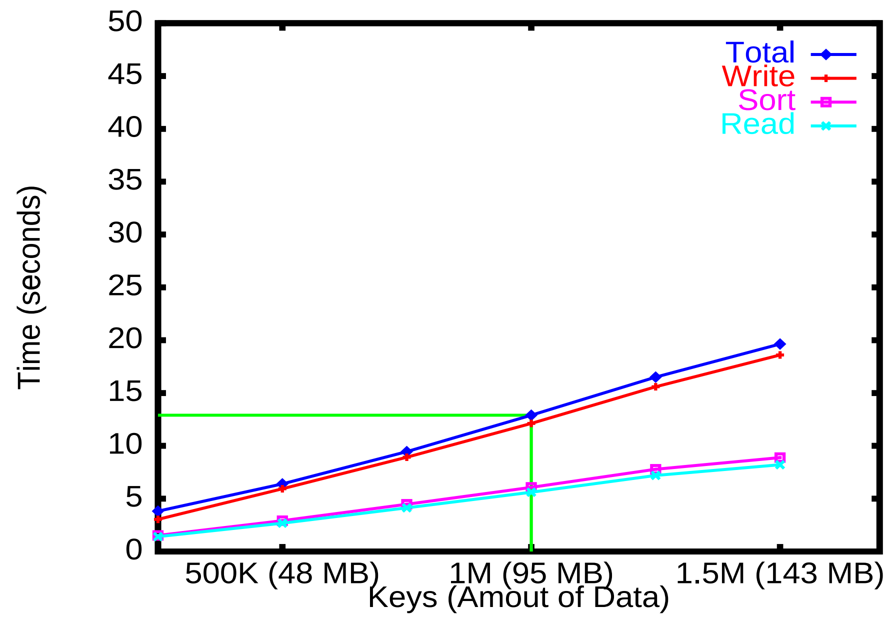
- Time on Datamation benchmark (October 1993):

System	Controllers	Drives	MB	Time (s)	Price	$/Sort
Cray YMP	?	1	?	26	$7.5M	1.25
Intel iPSC/2	?	32 (32 P)	?	58	$1M	0.37
DEC-3000	5 SCSI	10 RZ26	256	13.7	$97K	0.009
DEC-4000	4 Fast SCSI	12 RZ26	384	11.3	$166K	0.014
DEC-7000	6 Fast SCSI	16 RZ74	256	9.1	$247K	0.014

- Better performance and price-performance than previous best
  - Best price-performance on low-end model (DEC-3000)
- Still very expensive systems

NOW-Sort: Single-Node Results

- Cumulative performance on 4-disk systems



System	Controllers	Drives	MB	Time (s)	Price	$/Sort
UltraSPARC 1	2 Fast-SCSI	4	256	13 s	$23k	0.002

- Sorting “on the cheaps”
- Absolute performance lower (no records, yet...)

AlphaSort: Single-Node Conclusions

- Commodity processor & disks beat Cray & 32-node Intel iPSC/2
- Software striping to utilize multiple disks
  - Many slow disks better than few fast disks
- Threads useful for overlapping reading and sorting
  - Unbuffered I/O
- Pay attention to memory hierarchy
  - Use cache-sensitive internal sort: QuickSort
  - Reduce memory copying: Set up `(partial keys, pointers)`

NOW-Sort: Single-Node Conclusions

- Commodity workstations sort pretty well
  - Good performance, better price/performance
  - Internal Fast-Narrow SCSI not quite sufficient
- Software striping to utilize multiple disks
  - Different stripe-sizes for heterogeneous disks
- Mmap with mmadvise necessary without unbuffered I/O
  - No threads necessary
- Pay attention to memory hierarchy
  - Sort `(partial keys, pointers)` in buckets that fit in L2 cache

Outline

- Outline
  - *Background*
  - *HW/SW differences*
  - *Single-node sorting*
  - Parallel Sorting
  - Conclusions

AlphaSort: Parallel Overview

- Multiple threads: Few changes from single-node algorithm
  - Each thread requests affinity to a processor
  - Root thread responsible for all file operations
  - Worker threads do sorting and memory-intensive operations
- Root reads records into buffers
  - Workers QuickSort each data run
- Root merges multiple sorted-runs
- Workers gather records into contiguous output buffers
  - Root writes sorted buffers to disk

NOW-Sort: Parallel Overview

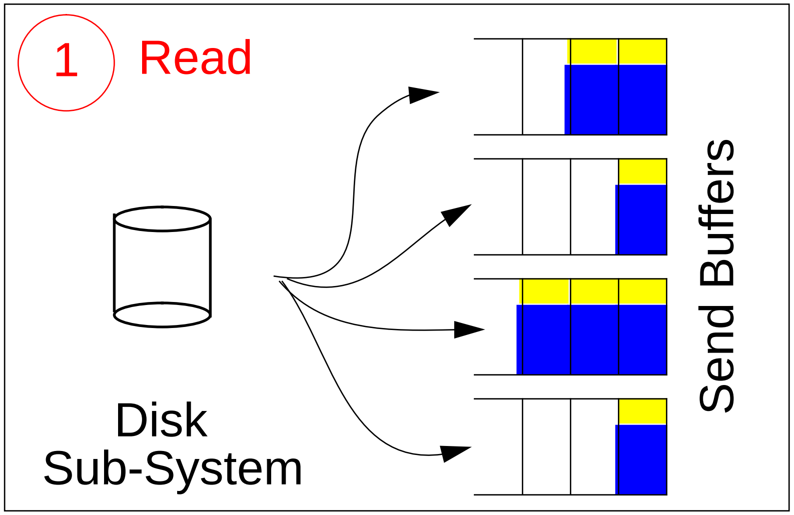
- One-pass parallel sorting
  - Read: get records from disk
  - Communicate: send keys to final destination
    - Shared-nothing machines require explicit partitioning
  - Sort: sort keys
  - Write: gather and write records to disk

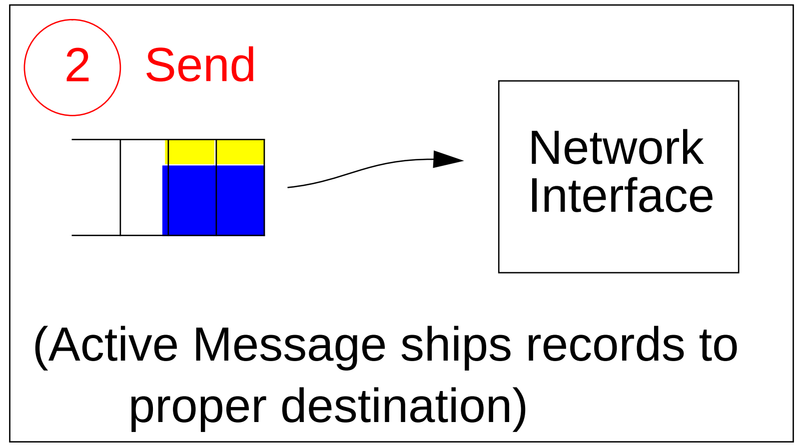
  Same as single-node

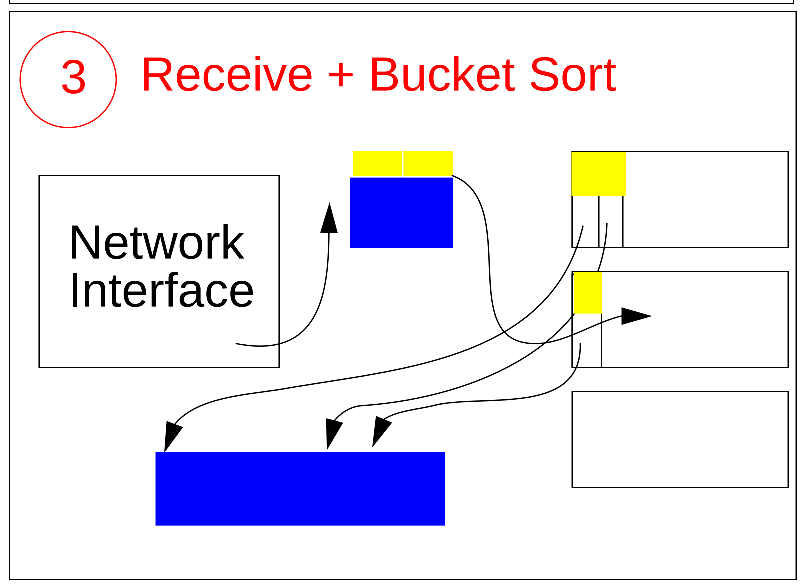
- Exploration of...
  - Programming issues
    - Overlap
  - Performance issues
    - How good is commodity workstation as building block?

NOW-Sort: Partitioning the Data

- "The Three Stages" (of reading and communicating)







1) Read from disk into send buffers

2) When buffer fills, send it!

3) Upon receipt, scatter keys into buckets, records into single buffer (keep pointer from key to record)

AlphaSort: Parallel Programming Issues

- Trivial programming effort
  - No significant changes necessary to add threads
- Correct way to structure computation?
  - Balanced division of work?

    Time to read 1 MB = Time to sort 1 MB / # workers ?

    Time to merge = Time to gather records / # workers ?

- Performance issues not addressed in paper
  - One-to-one correspondence of threads and processors?
  - Processor utilization in each phase?
  - Scalability with more processors?
    DEC AXP systems may have 6 processors, but only measure 3
- Root/workers model adequate only for small-scale parallelism

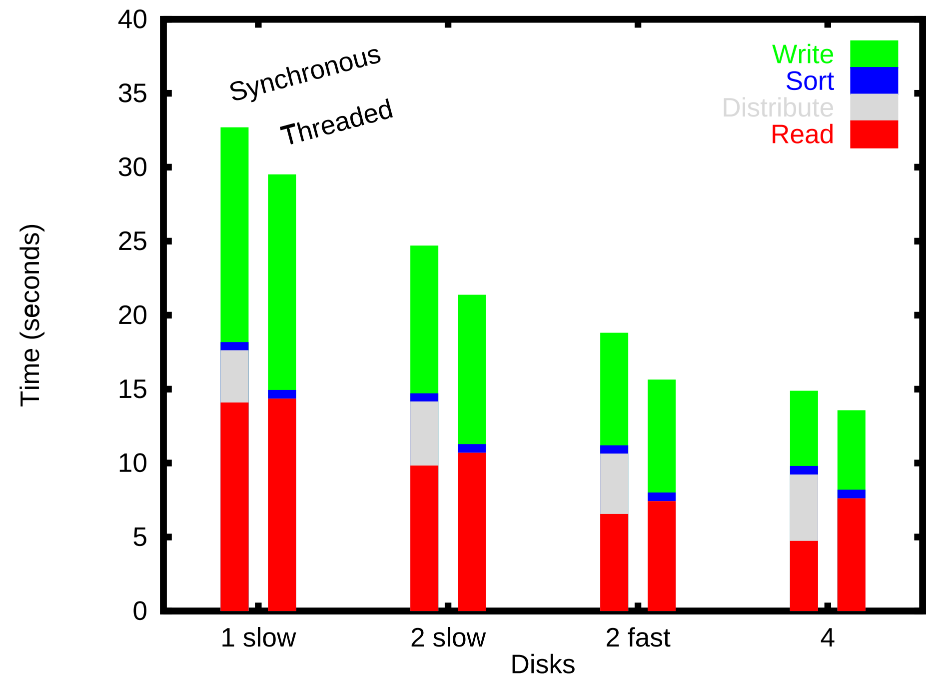
NOW-Sort: Programming Issue

- Open question: can cost of communication be hidden?
  - Overlap only useful if resources are under-utilized

- Answer via quantitative comparison
  - Synchronous algorithm: no overlap
  - Threaded algorithm: overlaps reading and communicating

NOW-Sort: To Thread, or not to...

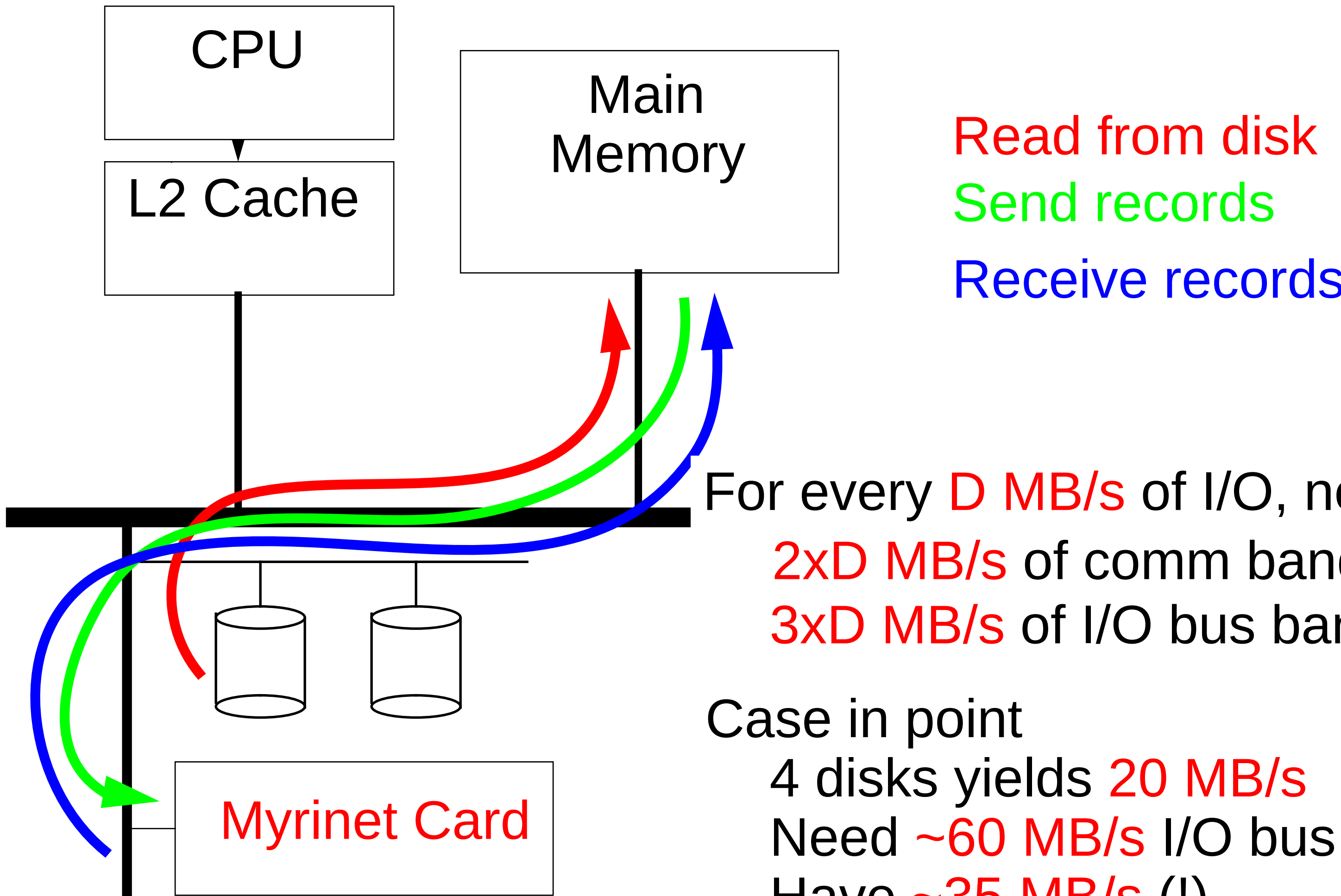
- Threaded vs. Synchronous algorithms



- Threads help in clusters too!
  - Overlap useful in cases w/ 1 or 2 disks
- BUT, no benefit with 4 disks per node; why?

NOW-Sort: I/O Analysis

- Data Movement Analysis



For every D MB/s of I/O, need:
- 2xD MB/s of comm bandwidth
- 3xD MB/s of I/O bus bandwidth

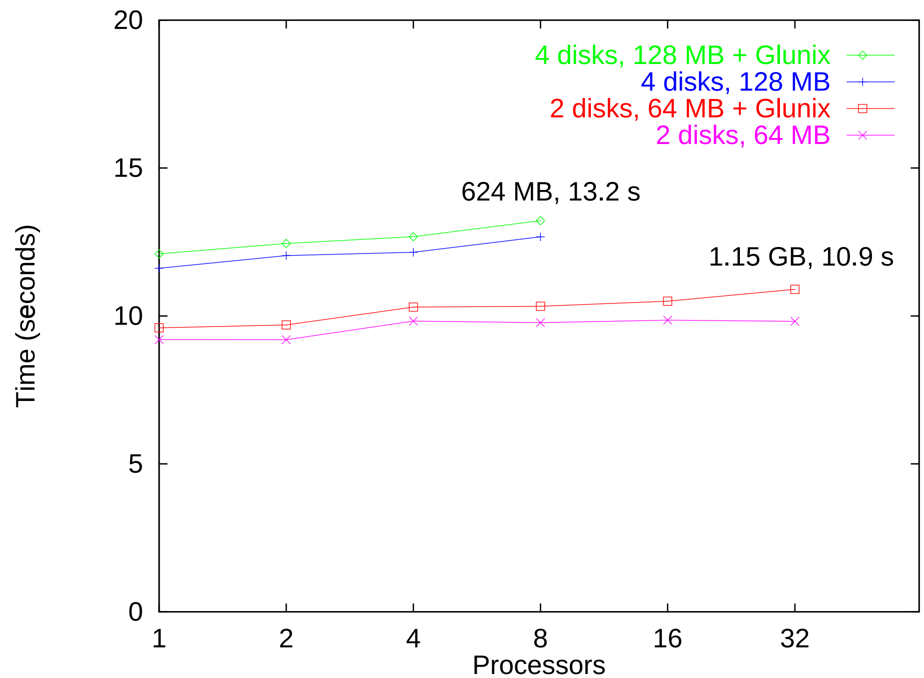
Case in point
- 4 disks yields 20 MB/s
- Need ~60 MB/s I/O bus
- Have ~35 MB/s (!)

NOW-Sort: Scaling

- Performance issue: Scaling

  8-node cluster
  128 MB
  4 disks

  32-node cluster
  64 MB
  2 disks



- Results:
  - NOW-Sort can scale with the best of them
  - 2x processors and 2x data -> same time to sort

AlphaSort: Datamation Results

- Multiprocessor performance for Datamation benchmark

System	CPUs	Controllers	Drives	Time (s)	Price	$/Sort
DEC-7000	3	7 Fast -SCSI	28 RZ26	7.0	$312k	0.014
DEC-4000	2	4 SCSI, 3 IPI	12 + 6	8.2	$312k	0.016
DEC-7000	1	6 Fast-SCSI	16 RZ74	9.1	$247	0.014

- Startup: Initializing address space expensive
  - Worker threads touch pages while root reads input files
  - VMS allocates and zeroes physical page

    12 seconds for 1 GB address space

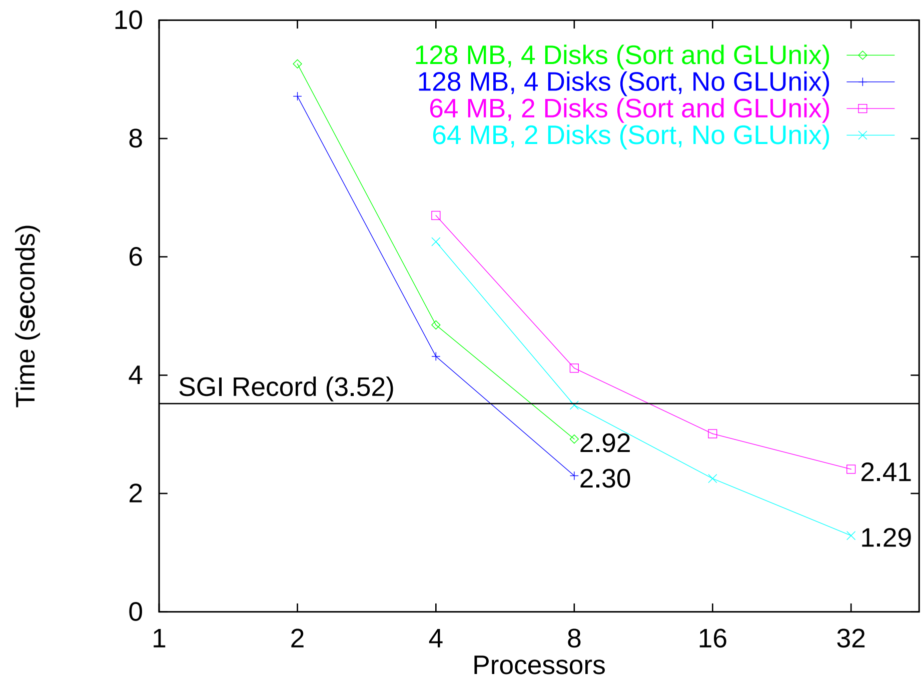
    Nearly 2 seconds for 100 MB

NOW-Sort: Datamation Results

- Bottom line: How long to sort 1-million records?

8-node cluster
128 MB
4 disks

32-node cluster
64 MB
2 disks



- NOW-Sort does well
  - Both clusters break records
  - BUT, start-up time is high

AlphaSort: Parallel Conclusions

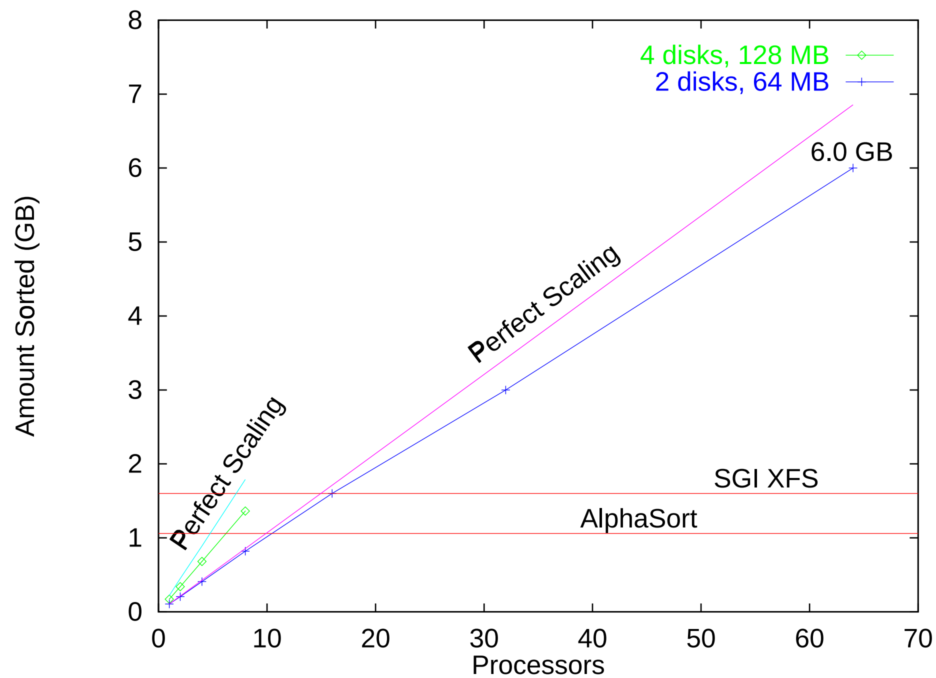
- Little additional programming complexity on multiprocessor
  - Thread programming model
  - Pooling of resources: No explicit partitioning of data
- Model not scalable past small number of procesors (3!)
- Performance of system difficult to decode

NOW-Sort: Parallel Conclusions

- Commodity clusters can sort!
  - Workstations + switches effective data movers
- Managing parallelism
  - Local: fine-grained, producer/consumer
  - Global: coarse-grained, bulk-synchronous
  - Interestingly, more difficulty with local synchronization
- Scaling
  - Has scaled well to 64 nodes
- Performance isolation
  - Understand behavior of single-node, tune for high-performance

Coda: MinuteSort

- MinuteSort: much better benchmark
  - How much can one sort in a minute?



- PennySort: how much can you sort for a penny?
  - No one has entered yet (will YOU?)