Using Algorithms Designed for Disks

Michael A. Bender
SUNY Stony Brook, Tokutek

Martin Farach-Colton
Rutgers, Tokutek

Bradley C. Kuszmaul
MIT CSAIL, SUNY Stony Brook, Lincoln Laboratory, Tokutek

The Story So Far...

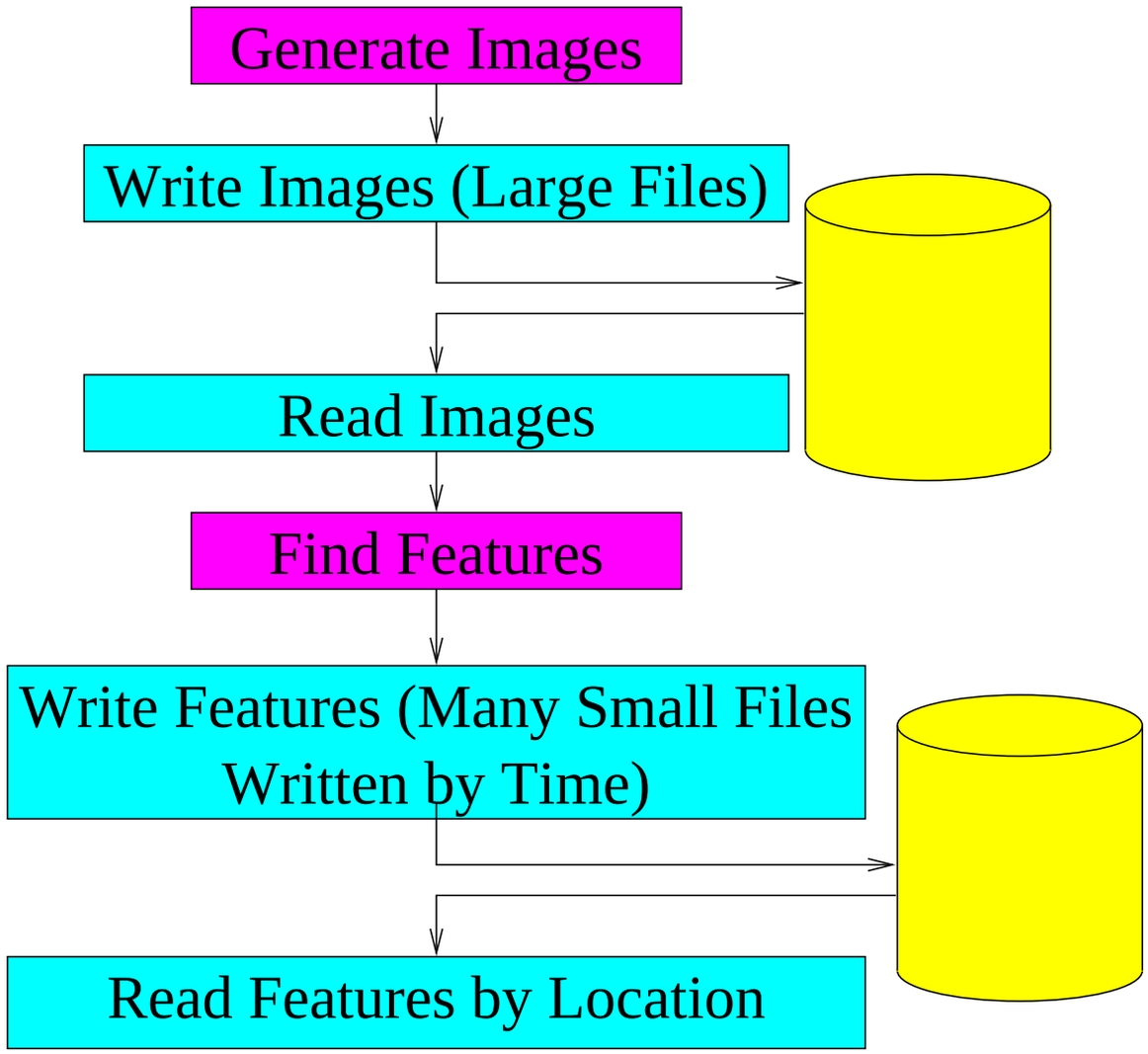
Martin focused on the data structures.
This talk focuses on applications:

- Scalable Synthetic Compact Application #3
- Terabyte Sort (most of this talk)

Scalable Synthentic Compact Application #3 (SSCA#3):

Sensor Processing and Knowledge Formation

http://www.highproductivity.org/SSCABmks.htm



SSCA#3 models a synthetic aperture radar system looking for "features", and consumes Computation and I/O.
The feature-I/O challenges storage systems built using traditional data structures.
We built an I/O-only version in C (available at the HPCS web site).

Jim Gray's Sorting Benchmark Contest

http://research.microsoft.com/barc/SortBenchmark/

Sort disk-resident data, & write it to disk.

There are six prizes each year: 3 categories $\times$ 2 classes.

Three Categories

Terabyte: How fast can you sort a terabyte?

Minute: How much can you sort in one minute?

Penny: How much can you sort for a penny. (Amortize machine cost over three years. A $1000 machine gives about 15m of runtime.)

2 Classes

Indy: Special-purpose sort. (Take advantage of keys being uniformly random and records being all the same size.)

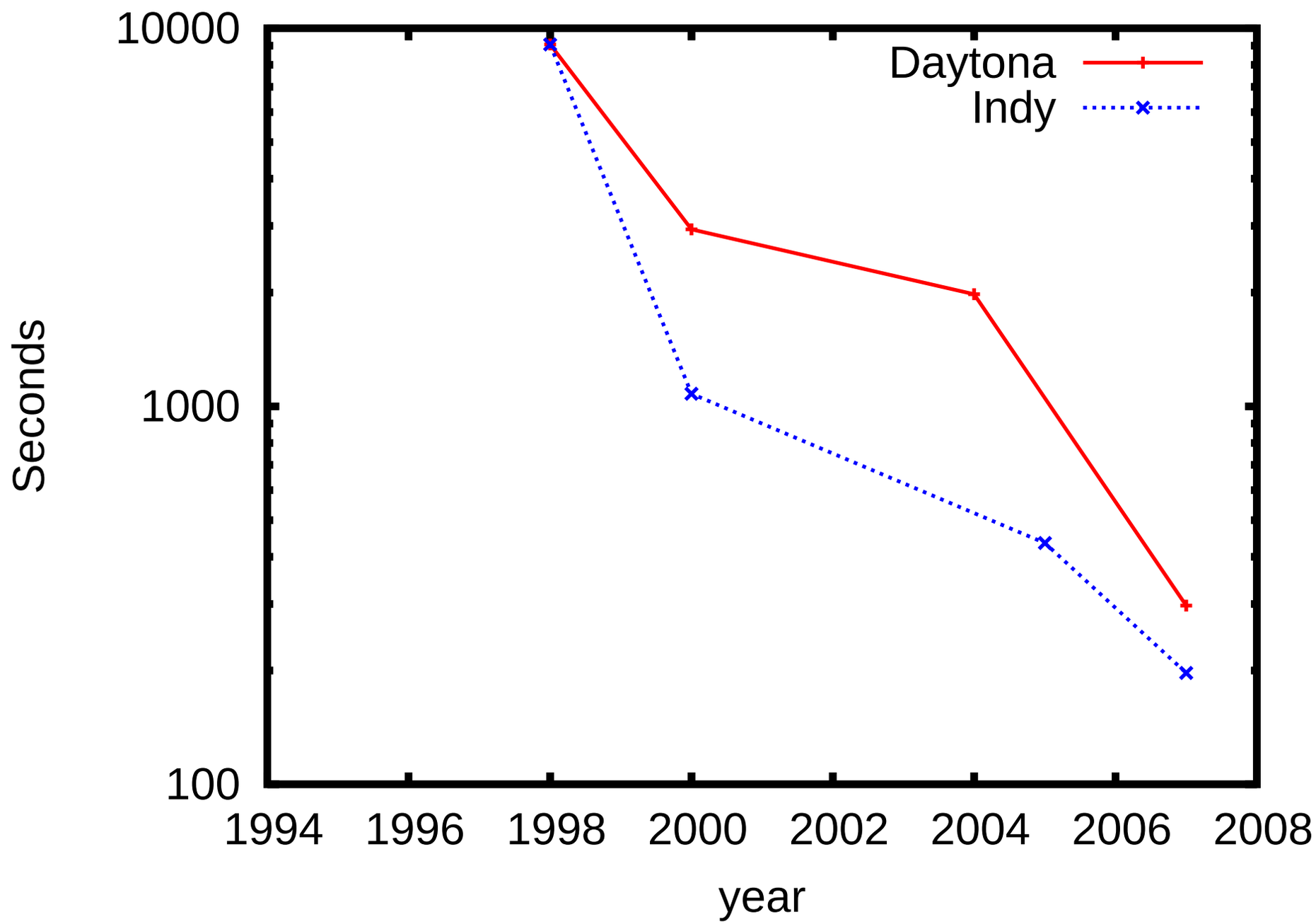
Daytona: General-purpose sort.

Winners

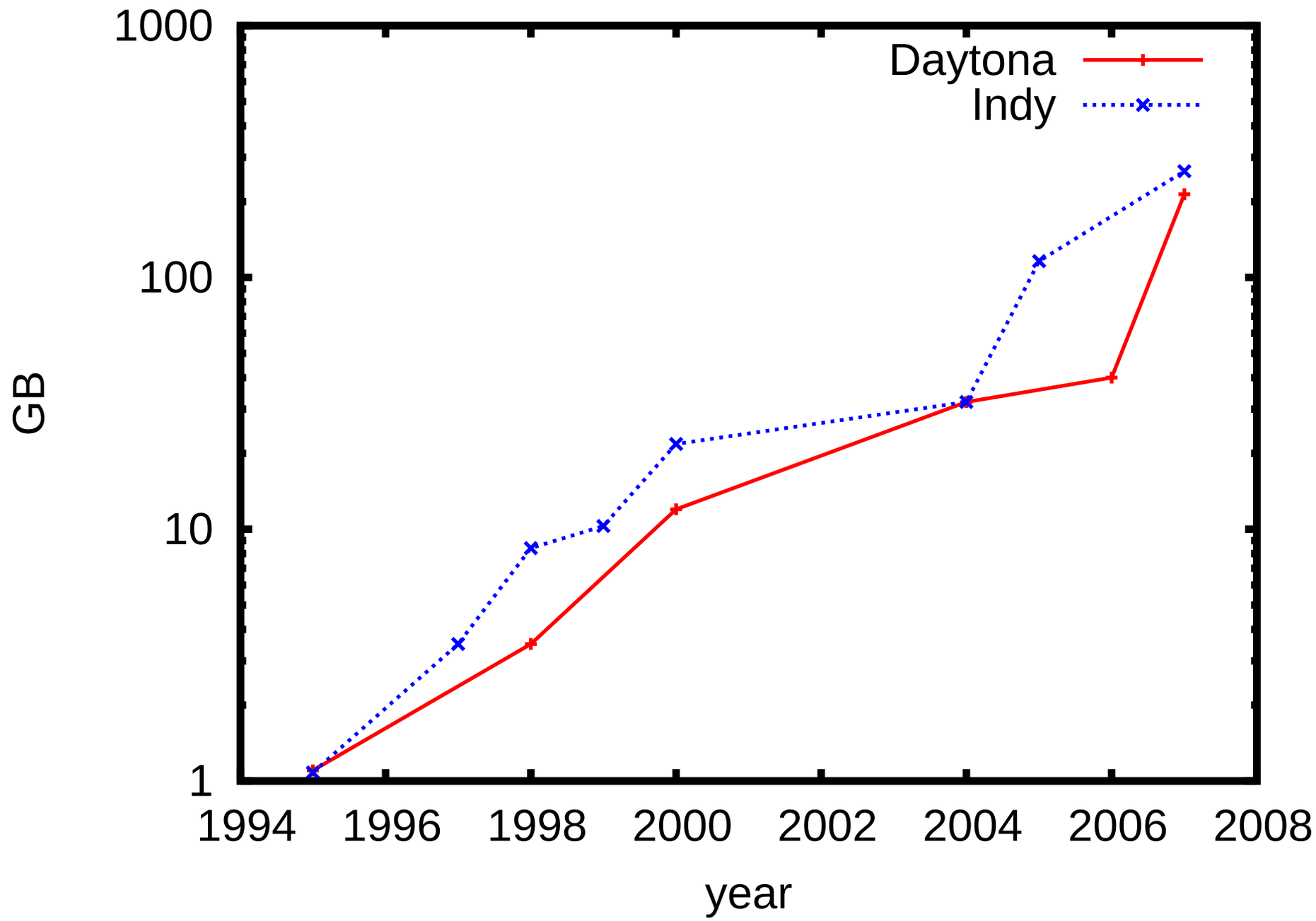
	Indy	Daytona	Year	Machine
Terabyte	197s	297s	2007	LL TX2500; 400 nodes; 3,600 disks
	435s		2005	80 Itanium2; 2,520 disks
		1,980s	2004	32 Itanium2; 2,320 disks
	1,080s	2,940s	2000	1952 SP; 2,168 disks
	9,060s	9,060s	1998	32-proc Origin 2000
Minute	264GB	214GB	2007	LL TX2500
		40GB	2006	32 Itanium 2; 128 disks
	116GB		2005	80 Itanium 2; 2,520 disks
	32GB	32GB	2004	32 Itanium 2; 2,320 disks
	21.8GB	12GB	2000	
	10.3GB		1999	
	8.41GB	3.5GB	1998	
	3.5GB		1997	
	1.08GB	1.1GB	1995	

I also won the Daytona Penny sort this year on a $300 desktop with 2 disks.
Won 5 out of 6 categories.

Terabyte Sort vs. Time



Minute Sort vs. Time



The Machine: Lincoln Lab TX2500

A cluster focused on I/O.

- 400 dual-processor servers
- 8GB/server (3.2TB total)
- 6 disk RAID in each node
- Infiniband

The TX2500 is probably around #75 on the Top-500 list.

Terabyte is an In-Core Sort on TX2500

1. Read 1TB from disk into the 3.2TB main memory.
2. Determine which node should receive each record so that node i's records are all $<$ node $i+1$'s records.
3. Send each record to the proper node.
4. Sort in a node.
5. Write to disk.

The Algorithm: TokuSort

Data starts on disks:

Disk

$P0$	83 86 77 15 93 35
$P1$	86 92 49 21 62 27
$P2$	90 59 63 26 40 26
$P3$	72 36 11 68 67 29
$P4$	82 30 62 23 67 35
$P5$	29 2 22 58 69 67
$P6$	93 56 11 42 29 73
$P7$	21 19 84 37 98 24

Memory

$P0$	
$P1$	
$P2$	
$P3$	
$P4$	
$P5$	
$P6$	
$P7$	

Step 1: Read Data From Disks to Memory

Disks

$P0$	83 86 77 15 93 35
$P1$	86 92 49 21 62 27
$P2$	90 59 63 26 40 26
$P3$	72 36 11 68 67 29
$P4$	82 30 62 23 67 35
$P5$	29 2 22 58 69 67
$P6$	93 56 11 42 29 73
$P7$	21 19 84 37 98 24

$\Rightarrow$

Memory

$P0$	83 86 77 15 93 35
$P1$	86 92 49 21 62 27
$P2$	90 59 63 26 40 26
$P3$	72 36 11 68 67 29
$P4$	82 30 62 23 67 35
$P5$	29 2 22 58 69 67
$P6$	93 56 11 42 29 73
$P7$	21 19 84 37 98 24

Step 2: Oversample The Data

Memory

$P0$	83	86	77	15	93	35
$P1$	86	92	49	21	62	27
$P2$	90	59	63	26	40	26
$P3$	72	36	11	68	67	29
$P4$	82	30	62	23	67	35
$P5$	29	2	22	58	69	67
$P6$	93	56	11	42	29	73
$P7$	21	19	84	37	98	24

Step 3: Sort the Sample

Memory:

$P0$	83 86 77 15 93 35	77 35	11 21
$P1$	86 92 49 21 62 27	86 21	23 29
$P2$	90 59 63 26 40 26	63 40	35 35
$P3$	72 36 11 68 67 29	11 67	40 63
$P4$	82 30 62 23 67 35	23 35	67 69
$P5$	29 2 22 58 69 67	29 69	73 77
$P6$	93 56 11 42 29 73	93 73	84 86
$P7$	21 19 84 37 98 24	84 98	93 98

$\Rightarrow$

Step 4: Calculate Pivots

How many data items are $\leq$ each pivot?

Pivot:	11	21	23	29	35	35	40	63	67	69	73	77	84	86	93	9
Count:	3	7	9	16	19	19	22	30	33	35	37	38	41	43	47	4

Want 6 per processor, so choose these pivots

Choose:	21	23	35	40	63	69	84

The resulting partition sizes are:

7	2	10	3	8	5	6

Now every number ≤ 21 goes to processor 0, numbers from from 22–23 go to processor 1, and so forth.

Step 5: Permute according to the pivots

$P0$	83 86 77 15 93 35
$P1$	86 92 49 21 62 27
$P2$	90 59 63 26 40 26
$P3$	72 36 11 68 67 29
$P4$	82 30 62 23 67 35
$P5$	29 2 22 58 69 67
$P6$	93 56 11 42 29 73
$P7$	21 19 84 37 98 24

$\longrightarrow$

$P0$	15 21 11 2 11 21 19
$P1$	23 22
$P2$	35 27 26 26 29 30 35 29 29 24
$P3$	40 36 37
$P4$	49 62 59 63 62 58 56 42
$P5$	68 67 67 69 67
$P6$	83 77 72 82 73 84
$P7$	86 93 86 92 90 93 98

Step 6: Sort Locally

$P0$	2 11 11 15 19 21 21
$P1$	22 23
$P2$	24 26 26 27 29 29 29 30 35 35
$P3$	36 37 40
$P4$	42 49 56 58 59 62 62 63
$P5$	67 67 67 68 69
$P6$	72 73 77 82 83 84
$P7$	86 86 90 92 93 93 98

Step 7: Write to Disks

Disks

$P0$	2 11 11 15 19 21 21
$P1$	22 23
$P2$	24 26 26 27 29 29 29 30 35 35
$P3$	36 37 40
$P4$	42 49 56 58 59 62 62 63
$P5$	67 67 67 68 69
$P6$	72 73 77 82 83 84
$P7$	86 86 90 92 93 93 98

$\Leftarrow$

Memory

$P0$	2 11 11 15 19 21 21
$P1$	22 23
$P2$	24 26 26 27 29 29 29 30
$P3$	36 37 40
$P4$	42 49 56 58 59 62 62 63
$P5$	67 67 67 68 69
$P6$	72 73 77 82 83 84
$P7$	86 86 90 92 93 93 98

Terabyte Indy Sort Time breakdown

		Time
Step 0:	Startup	9.7s
Step 1:	Read Data	37.1s
Step 2:	Oversample	0s
Step 3:	Sort sample	44.1s
Step 4:	Calculate Pivots	14.0s
Step 5:	Permute	46.4s
Step 6:	LocalSort	23.9s
Step 7:	Write (& fsync)	11.6s

Sorting is a Great Benchmark

- Stresses CPU, Memory, I/O, and Interconnect.
- Not *too* hard. It took me about 1 month full time to get it all working.

Acks:

- Andy Funk (MITLL) helped build a fuse filesystem module.
- Albert Reuther (MITLL) helped me get going on the tx2500.
- Pete Michaleas (MITLL) kept the tx2500 hardware & software running.

Come Talk To Us

We're looking for more applications for data structures and algorithms for efficient disk I/O.
We are:

- Michael A. Bender (`bender@cs.sunysb.edu`)
- Martin Farach-Colton (`bender@cs.rutgers.edu`)
- Bradley C. Kuszmaul (`bradley@mit.edu`)

We have formed a company, *Tokutek*, to commercialize our research on cache-oblivious streaming B-trees.

That’s All Folks

Improvements for Next Time

- MPI: MPI starts too slow, taking 100s to invoke `start_mpi`.

  I wrote my own implementation of the communications primitives using TCP/IP. This cost me performance on the (perhaps 20s in the permute, and 10s in the SortSample).
- Sort can be parallelized (saving about 10s).
- Sort: I should have sorted locally before sampling, saving the Bucketize step.

Perhaps as much as 25% left on the table.

TokuSort Algorithm

1. Each node reads data from its local disk.
2. Sample 64 randomly chosen data items on each node ($64 \times 400 = 25,600$ samples total.
3. Sort the sample.
4. Select pivots from the sample. Use the best 400 of the $25,600$ samples to partition the data evenly into 400.
5. Permute the data according to the pivots.
6. Each node sorts locally.
7. Each node writes to its local disk.