Sorting

Arvind Krishnamurthy
Fall 2004

Practical Performance Target (circa 1992)

- Sort one billion large keys in one minute on one thousand processors.

- Good sort on a workstation can do 1 million keys in about 10 seconds
  - just fits in memory
  - 16 bit Radix Sort

- Performance unit: $\mu s$ per key per processor
  - $s \sim 10$ for single Sparc 2
Performance Targets in 2001 (Jim Gray)

- Datamation (1 April, 1985)
  - Sort a million hundred-byte records
  - tests file system, IO system, utility access
- Minute Sort:
  - Sort as many records as you can in a minute
  - Report rate and price ($\text{cost}/1\text{e}6$)
  - *Indy-MinuteSort: a Formula-1 sort where price is no object.*
  - *Daytona-MinuteSort: a stock sort where price is no object.*
- Penny Sort:
  - Sort as much as you can for a penny.
  - Proposed Change to PennySort: Performance/Price Sort
  - Compute GB/$ of a two pass sort: performance per dollar.
  - *Indy-PennySort: a Formula-1 biggest-bang-for-the buck sort.*
  - *Daytona-PennySort: a stock sort giving the biggest-bang-for-the buck*
Datamation Results

- Was one hour, now less ½ second
- Benchmark retired in 2001

Local Sort Performance

(11 bit radix sort of 32 bit numbers)

Entropy in Key Values

Entropy = -\sum p_i \log p_i

p_i = Probability of key i
Bitonic Sort

A bitonic sequence is one that is:
1. Monotonically increasing and then monotonically decreasing
2. Or can be circularly shifted to satisfy 1

A half-cleaner takes a bitonic sequence and produces
1. First half is smaller than smallest element in 2nd
2. Both halves are bitonic

A full sort is a series of half-cleaners starting with depth 1 and increasing to depth of (log n)
Column Sort

1. Sort
2. Transpose - block to cyclic
3. Sort
4. Transpose - cyclic to block w/o scatter
5. Sort
6. Shift
7. Merge
8. Unshift

work efficient

Treat data like n x P array, with n >= P^2, i.e. N >= P^3

Column: Breakdown

Measured

N/P

16384 32768 65536 131072 262144 524288

Sort1
Sort2
Sort3
Merge
Trans
Untrans
Shift
Unshift

P= 64, random
Histo-radix sort

Per pass:
1. compute local histogram
2. compute position of 1st member of each bucket in global array
   - $2^r$ scans with end-around
3. distribute all the keys

Only $r = 8, 11, 16$ make sense for sorting 32 bit numbers

Histo-Radix Sort

Each Pass
- form local histograms
- form global histogram
- globally distribute data
Radix: Breakdown

Radix: Stream Broadcast Problem

- Processor 0 does only sends
- Others receive then send
- Receives prioritized over sends

\[(P-1) (20 + L + (n-1)g) \text{?} \]
Sample Sort

1. compute P-1 values of keys that would split the input into roughly equal pieces.
   - take S~64 samples per processor
   - sort PS keys
   - take key S, 2S, ... (P-1)S
   - broadcast splitters
2. Distribute keys based on splitters
3. Local sort
4. possibly reshift

Sample Breakdown

![Sample Breakdown Chart](chart.png)
Comparison

- Good fit between predicted and measured (10%)
- Different sorts for different sorts
  - scaling by processor, input size, sensitivity
- All are global / local hybrids
  - the local part is hard to implement and model