

# Sorting

Shimin Chen

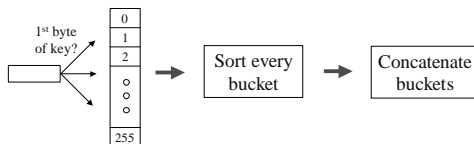
School of Computer Science  
Carnegie Mellon University  
22 March 2001

## Overview

- A brief review of common sort algorithms
- Sort benchmarks
- A base case: AlphaSort
- Improving Sort Performance
- SuperScalar sort and NOW sort
- What do we learn?

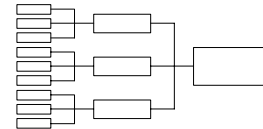
## Warm up: in memory sorting

- Comparison based
  - Quick sort, Merge sort, Heap, Tournament, etc.
- Key structure based
  - Bucket sort, etc.



## External sorting

- Create initial runs
  - Replacement selection
  - Quick sort
- Merge runs



## Overview

- A brief review of common sort algorithms
- **Sort benchmarks**
- A base case: AlphaSort
- Improving Sort Performance
- SuperScalar sort and NOW sort
- What do we learn?

## Purpose

- To test
- Sorting algorithm
  - IO architecture
  - Operating system
- Mimic real sorting problems

## Datamation Benchmark (1985)

- Records
  - 1 million, 100 bytes each
  - 10 byte keys in random order
- External: disk to disk sort
- Elapsed time:
  - From launching to termination
  - Including time to ensure output on disk
- Cost of running the sort
  - Cost of the system / 5 years × Elapsed time

Shimin Chen - 22 March 2001

7

## MinuteSort & PennySort (1994)

- MinuteSort
  - Sort as much as possible in one Minute
  - Otherwise same as Datamation
- PennySort
  - Sort as much as possible for less than a penny
  - Otherwise same as Datamation

Shimin Chen - 22 March 2001

8

## Some results

- Datamation:
  - 1987, Tandem/Tsukerman, 980s
  - 1988, Cray1/Weinberger, 28s
  - 1993, AlphaSort, 9s
  - 1994, AlphaSort, 7s
  - 1996, SuperScalar Sort, 5.1s (raw disk)
  - 1996, NOW Sort, 2.41s
  - .....
  - 2000, Mitsubishi H/w sorter, 0.998s

Shimin Chen - 22 March 2001

9

## Results

- Minute Sort
  - 2000, NOW+HPVMSort, 21.8 GB
- Penny Sort
  - 2000, HMSort, 4.5 GB

Shimin Chen - 22 March 2001

10

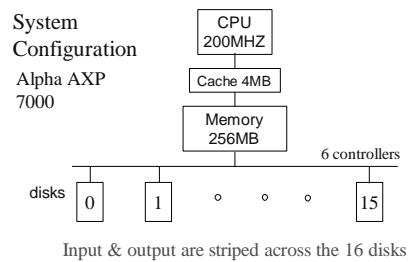
## Overview

- A brief review of common sort algorithms
- Sort benchmarks
- **A base case: AlphaSort**
- Improving Sort Performance
- SuperScalar sort and NOW sort
- What do we learn?

Shimin Chen - 22 March 2001

11

## AlphaSort (9.1s result)



Shimin Chen - 22 March 2001

12

## Strategy

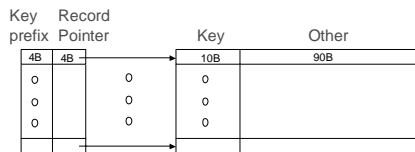
- 100MB input can fit in memory
  - One pass sort
  - Read data, sort data, write data
- Overlap sorting with IO as much as possible
- Triple buffers used for reading and writing to fully utilize disk bandwidth

## AlphaSort Algorithm

1. Launching: till 0.14s
  - Open striped input files, create output files, allocate 110MB memory
2. Read input: till 3.87s
  - Async read stride after stride
  - IO bound: 27MB/s transfer rate
  - Create sorted runs of 10MB each

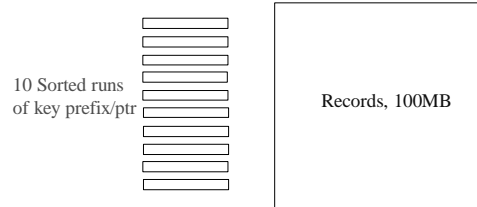
## Key prefix extraction

- Whenever 1MB is read, extract (4B key prefix, 4B record ptr) into an array



## QuickSort to create runs

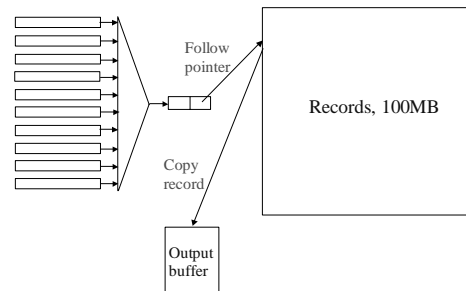
- Whenever 10MB is read, sort the array containing 100,000 key prefix/ptr with Quicksort



## Algorithm Cont'd

3. Sort the last 10MB run: till 4s
  - During this 0.12s period, no IO
4. Merge runs and output: till 8.9s
  - Merge the 10 sorted runs and copy records to output buffer
  - When a full stripe (64KB) is filled, write the buffer
  - IO bound

## Merge runs



## Algorithm Cont'd

5. Terminating: till 9.1s

- 0 ~ 0.14s launching
- ~ 3.87s read input and create 9 runs (IO bound)
- ~ 4s create the 10<sup>th</sup> run
- ~ 8.9s merge runs and output data (IO bound)
- ~ 9.1s terminating

## AlphaSort summary

- Mainly IO bound: 8.6s read, write
- CPU memory to memory sort: 6s (overlapped)

## Overview

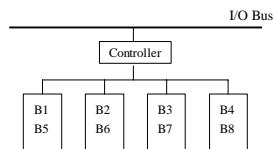
- A brief review of common sort algorithms
- Sort benchmarks
- A base case: AlphaSort
- **Improving Sort Performance**
- SuperScalar sort and NOW sort
- What do we learn?

## IO Bottleneck

- Sorting is IO bound!
  - What about memory to memory sorting part?  
Can be overlapped with async IO
- Improve IO bandwidth

## IO Bandwidth: AlphaSort

- Striping
  - Access multiple blocks in parallel



## Striping

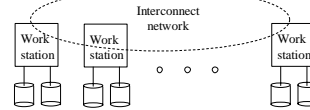
- Hardware striping: RAID0
- Software striping: user library
  - Less expensive
  - Allow multiple controllers: more bandwidth
  - Different stride size for different disk speed
- AlphaSort uses user library way

## Requires balanced I/O system

- If system is not balanced, may not get maximal benefit
  - UltraSPARC with Narrow SCSI Bus (NOW)
    - 5400RPM Hawk disk
    - 1 disk ~ 5.5MB/s
    - 2 disks ~ 8.3MB/s  $\neq 5.5 \times 2$
    - Saturated!
- Disk B/W ~ Controller B/W ~ I/O Bus B/W

## Another way to get IO bandwidth

- NOW (Network Of Workstations)
  - Ideally #workstations  $\times$  B/W per workstation
- But sorting needs data exchange among machines
  - Need fast network connection



## IO Bandwidth: summary

- Striping
- Balanced IO architecture
- Network of Workstations

## Memory to memory sort

- Why is it important at all?
- Naïve implementation could use 23s to sort 1 million records (shown in AlphaSort)
  - Then the bottleneck becomes in memory sorting!
- In memory sorting still needs careful consideration

## What are the major concerns?

- Cache performance!
- Why is it important?
  - A cache miss = 10-100 CPU cycles
  - CPU speed increases 70% per year
  - Memory speed can not catch up
  - Gap becomes larger and larger
- Improving cache performance is more important than reducing instructions

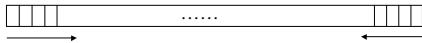
## AlphaSort

- QuickSort the runs
  - 10MB records have 800KB prefix/ptr, so fit in the 4MB cache
- Tournament tree to merge 10 runs
  - So should fit in the first level on-chip cache

## What if not fit in cache?

### QuickSort

- Sequential access: use many items in one cache line, reduces cold cache misses

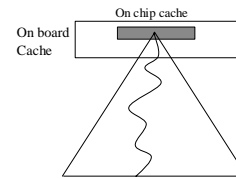


- Divide and Conquer: when the sub-array fit in cache, we can sort the sub-array and there will be no cache miss

## What if not fit in cache?

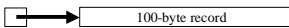
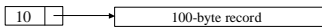
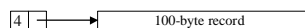
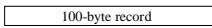
### Tournament is terrible

- Random access: a lot of cache misses



## AlphaSort uses Key-prefix/Ptr

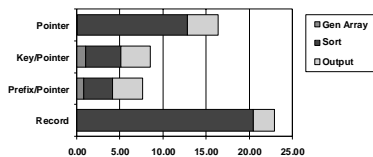
Altogether four methods:

- Pointer sort: 
- Key-pointer: 
- Key prefix-ptr: 
- Record sort: 

## Compare the four methods

- Building the array:
  - Key or key prefix extraction cost
- Key comparison:
  - Ptr sort need to retrieve the record
- Swap
  - Record sort need to swap the full record
- Final result
  - All ptr based need to copy results

## Comparison

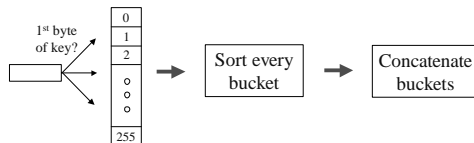


## How to select?

- Large record, not skewed key distribution: key prefix extraction
- Large record: key extraction
- Short record: record sort

## Another way: Bucket sort

- Use the first  $m$  bits of the key to divide the input into  $2^m$  buckets
  - Avoid  $2^m$  comparisons / key – no need to merge
  - Bucket size can be made smaller than cache size



Shimin Chen - 22 March 2001

37

## Bucket sort cont'd

- $K=2^m$  buckets, how to choose?
  - Average bucket size is small, so sorting in the bucket can fit in cache
  - $K \leq \# \text{cache lines}$ 
    - Avoid conflict miss
  - $K \leq \# \text{TLB entries}$ 
    - TLB: cache for virtual memory page table entry
    - Avoid TLB thrashing
- Can put (key prefix, pointer) into the bucket

Shimin Chen - 22 March 2001

38

## Moving data: 64bit vs 32 bit

- Use 64bit load and store to move data
- Some systems have 64bit integer load/store instructions
- Most systems have 64bit floating point number load/store instructions
- 64bit instead of 32bit move reduces instructions for free!

Shimin Chen - 22 March 2001

39

## Memory sort: summary

- Cache performance
- QuickSort has good cache performance
- Tournament is terrible if not fit
- Bucket sort avoids comparison – even better
- For long record + the uniform distribution of keys  
Use (key prefix, pointer)
- 64bit data move

Shimin Chen - 22 March 2001

40

## Operating system issues

- Memory allocation
  - OS usually zero the newly allocated area upon first touch
  - Many cache misses: 12s/GB (AlphaSort)
  - Alphasort: threads touch the allocated area first when with multi-threads

Shimin Chen - 22 March 2001

41

## Read vs. Mmap

- read() system call
  - File system may perform its own buffering
  - Lower effective memory size, Copy cost
  - Use non-buffered read if possible
- Mmap() (NOW)
  - Map file directly into VM
  - But uses LRU replacement
- Mmap() with madvise()
  - Can further improve by giving hints

Shimin Chen - 22 March 2001

42

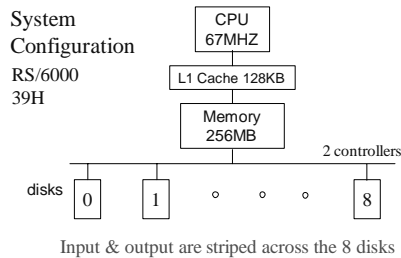
## Process or thread structure

- When use multi-threading
- When use NOW
- Efficient synchronization of all the processes or threads is needed

## Overview

- A brief review of common sort algorithms
- Sort benchmarks
- A base case: AlphaSort
- Improving Sort Performance
- **SuperScalar sort and NOW sort**
- What do we learn?

## SuperScalar sort (5.1s)



## IO Bandwidth

- Every disk  $7\text{MB/s} \times 8 = 56\text{MB/s}$  maximum
- Controller and IO channel supports  $80\text{MB/s}$  peak
- Use software disk striping on raw disks
  - 100MB reads 2.1s:  $47\text{MB/s}$
  - 100MB writes 2.55s:  $39\text{MB/s}$

## Algorithm

1. Launching: till 0.1s
  - Open files, allocate shared memory segment
2. Read phase: till 2.2s
  - Double buffers for every disk
  - Two async reads per disk all the time
  - When a block is read, use bucket sort to distribute key prefix/pointer into buckets

## Bucket sort details

- Use the first 7 bits of keys to put into 128 buckets
- Key prefix/pointer pair
  - Key prefix is the next 31 bits (so total 38 bits used)
  - 8 bytes per record
- Why 128 buckets?
  - $1\text{ million} / 128 = 7812$  (key prefix, pointer)  $< 64\text{KB}$
  - Sort in the bucket below requires twice space
    - $64\text{KB} \times 2 = 128\text{KB} = \text{L1 cache size}$

## Algorithm cont'd

---

3. Write phase: till 4.85s
  - 2 Output buffers 4MB each
  - Each buffer consists of 512KB per disk (8 disks)
  - Sort the buckets and move records into output buffer
  - When an output buffer is filled, issue async writes
4. Shutdown phase: till 5.1s

## Sorting a bucket

---

- Radix Counting sort
  - Recall: 64KB bucket, 8B prefix/ptr
  - Only sort the 22bits in the prefix
  - Count #entries for every pattern in the first 11-bit, and count #entries for pattern in the second 11-bit
  - Get two  $2^{11}$  array *Count1*, *Count2*
  - Compute ranks of patterns and copy keys to the pattern position

## Checking other bits

---

- Then the records are checked to see if there are any ties
- Break tie by using the other bits
- Prob(first 29 bits are the same) = 0.002

## SuperScalar Sort: summary

---

- Read phase is IO bound, but write phase is CPU bound!
- IO: 2.1s to read, 2.55s to write
- CPU
  - IO related: 2.2s (initiate, service, check)
  - Initialize and release memory: 1.05s
  - Sorting: 0.6s
  - Moving 100MB to output buffer: 1.0s
  - Other: 0.25s

## Cont'd

---

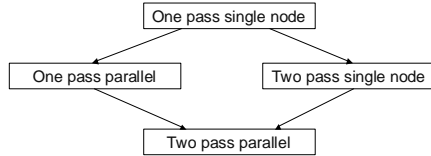
- Software striping
- Bucket sort + radix sort
- IO servicing time is a big issue

## NOW sort

---

- Cluster 1
  - Node: UltraSparc 64MB, two 5400RPM disks, IO bandwidth 8.3 MB/s
  - Up to 64 nodes interconnected by Myrinet
- Cluster 2
  - Node: UltraSparc 128MB, two 5400RPM disks, two 7200 RPM disks, IO B/w 21 MB/s
  - Up to 8 nodes interconnected by Myrinet
- Myrinet: 160MB/s bi-directional

## A set of sort algorithms

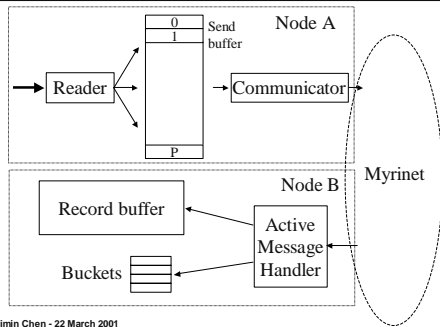


- Single node one pass sort mainly follows SuperScalar sort
  - Software striping
  - Bucket+Partial-radix sort

## One pass parallel (P processors)

- Read: each processor reads records from its local disk into memory
- Communicate: records are sent to one of  $P \times B$  local or remote buckets based on key values
- Sort and write

## How to synchronize?



## Two pass sorts

- Create run using one pass sorts
- Merge runs locally

## NOW sort: summary

- A set of sort algorithms
- Use striping and NOW to get IO bandwidth
- Memory sort algorithm follows SuperScalar sort
- Interesting synchronization structure

## Overview

- A brief review of common sort algorithms
- Sort benchmarks
- A base case: AlphaSort
- Improving Sort Performance
- SuperScalar sort and NOW sort
- **What do we learn?**

## What do we learn?

---

- Sorting is not only memory sorting algorithm
  - IO bandwidth important!
  - Striping, NOW to improve IO bandwidth
- Cache performance is important for memory sorting algorithm

## Discussion

---

- Bucket sorting applicable to non-uniform distribution?
- One-pass sorting vs Two-pass sorting?

## References

---

1. Most sorting related terms can be found here:  
<http://hissa.nist.gov/dads/>
2. Anon et al. *A Measure of Transaction Processing Power*. 1985. (redbook)
3. C. Nyberg et al. *AlphaSort: A Cache-Sensitive Parallel External Sort*. 1994 (redbook)
4. Sort benchmark results:  
<http://research.microsoft.com/barc/SortBenchmark/>
5. R.C. Agarwal. *Super Scalar Sort Algorithm for RISC Processors*. SIGMOD'96
6. A.C. Arpaci-Dusseau et al. *High-Performance Sorting on Networks of Workstations*. SIGMOD'97