

ME 290-R:  
General Purpose Computation  
(CAD/CAM/CAE) on the GPU  
(a.k.a. Topics in Manufacturing)

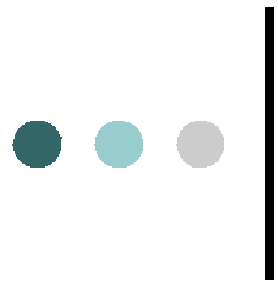
Sara McMains  
Spring 2009  
GPUTeraSort



# GPUSort:

High Performance Graphics Co-processor Sorting  
for Large Database Management

N. Govindaraju, J. Gray, R. Kumar, D. Manocha



# Sorting is Important

- ⌚ Definition - *External memory sorting*
  - | Sorting data sets larger than system RAM
- ⌚ One of the most studied computer science problems
  - | Authors cite [Friend 56] as early work in external sorting
- ⌚ Huge sort tasks arise frequently
  - | Web Indexing
  - | Geographic Information Systems (GIS)
  - | Data Mining
  - | Supercomputing



# GPUs Potentially Help

- ⌚ GPUs have 10x greater memory bandwidth than modern CPUs
- ⌚ GPUs have 10x more operations/second than modern CPUs
  - | This also means we will need a parallel sorting algorithm
    - | Bitonic [Batcher 68]
    - | Periodic Balanced Sorting Network [Dowd 89]
    - | Odd-Even Merge-Sort [Kolb, Latta, Rezk-Salama, 2004]
- ⌚ Growth of GPU power faster than growth of CPU capability



# GPU Sorting Limitations

- ⌚ Previous GPU sorting limited
  - | Could not sort data sets larger than graphics memory
    - | 1GB on the current bleeding-edge GPU
    - | 256MB - 512MB on most consumer-level GPU
  - | Could not handle keys larger than 32 bits
  - | Could not keep up with modern disk I/O systems



# GPUSort Advantages

- ⌚ Effectively distributes work between CPU and GPU
- ⌚ “Up to 10 times faster” than prior in-memory sorting algorithms
  - | Actual results - only 6-7x faster
- ⌚ Little wasted bandwidth
  - | Attains peak disk I/O performance on an inexpensive PC
  - | Near-peak memory bandwidth on GPU
- ⌚ Scalable
  - | Data set sizes, key sizes



# PennySort Benchmark

⌚ Microsoft sort benchmark formulated as follows:

| Assume computer has a lifetime of three years

•  $T = 94,608,000$  sec

|  $T/\text{system cost in pennies} = \text{one penny worth of compute time}$

| Sort maximum amount of data in one penny's worth of time

⌚ Essentially a metric of a sort algorithm's price/performance ratio



# PennySort Benchmark

☞ Benchmark has two categories:

## | Indy

- Formula 1 Race – “Special purpose”
- Input is known to be 100-byte records with 10-byte keys.

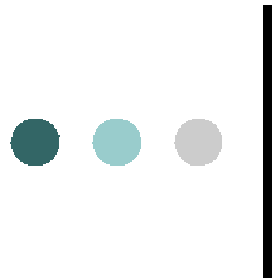
## | Daytona

- Stock Car Race – “All-purpose”
- Sort code must handle arbitrary key widths and record sizes.



# PennySort Benchmark: GPUTeraSort

- ⌚ 4x improvement over 2005 Daytona PennySort record
- ⌚ 1.4x improvement over 2003 Indy PennySort record
  - | Speed plateau since 1995 at about 200k records/sec/CPU
- ⌚ GPUTeraSort: 2006 Indy PennySort winner!



# External Memory Sorting

- ⌚ External memory sorting typically performs 2 phases:
  - ⌚ Phase 1: Produce a set of sorted files
  - ⌚ Phase 2: Produce ordered permutation of input data file
    - ⌚ Uses files from phase 1 as input
- ⌚ Two broad categories of external memory sorting
  - | Distribution-based
  - | Merge-based



# External Memory Sorting: Distribution-Based

- ⌚ Distribution-Based
  - ⌚ Phase 1:
    - ⌚ Partitions input into  $S$  keys
    - ⌚ Generates  $S$  disjoint buckets for keys
      - ⌚ Keys in bucket  $i$  come before keys in bucket  $i+1$
  - ⌚ Phase 2 sorts contents of each bucket independently

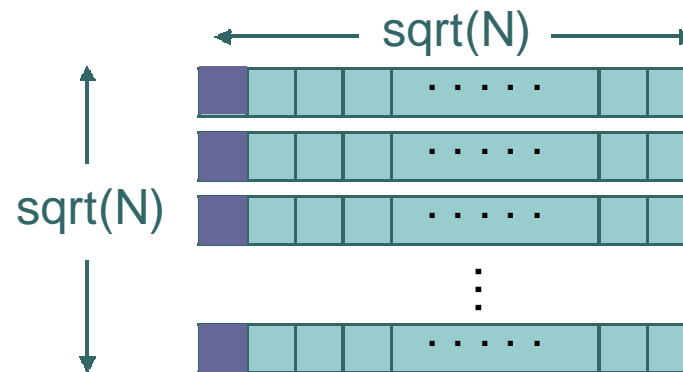


# External Memory Sorting: Merge-Based

- ☪ Merge-Based
  - ☪ Phase 1
    - ☪ Partitions input into roughly equal-size chunks
    - ☪ Sorts these chunks in main memory writing output to disk
      - ☪ The sorted output chunks are often called “runs”
  - ☪ Phase 2
    - ☪ Merge “runs” in memory and write single sorted output to disk
- ☪ GPU memory size is hard constraint
  - | Pre-determined chunk sizes are convenient so we use merge-based

# Merge-Based Sorting: Memory Requirements

- RAM requirements increase as the square root of input size



- Need memory to fit:
  - Each  $\sqrt{N}$  key array into memory to sort for Phase 1
  - One key from each of  $\sqrt{N}$  arrays for merge in Phase 2



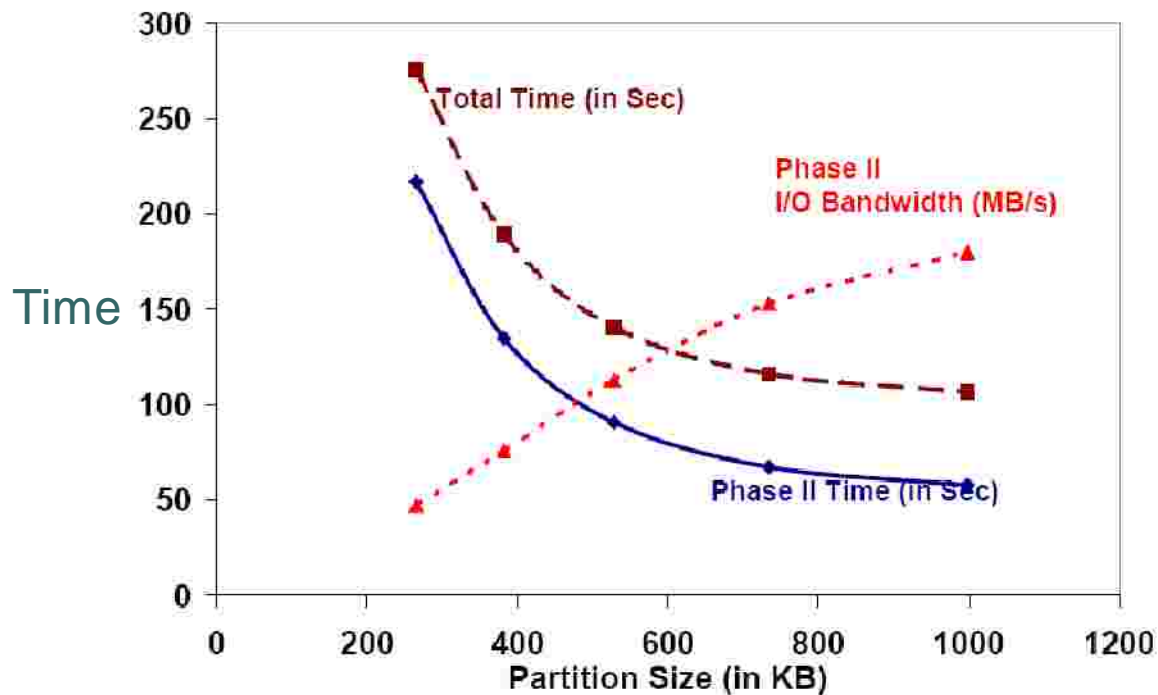
# External Memory Sorting Issues

- ⌚ Three impediments to high performance on commodity CPUs:
  - | Cache misses when partition size  $>$  CPU cache size
    - | Considerable latency when waiting on main memory
  - | I/O performance under-utilized
    - | Sorting comparisons bog down CPU
  - | Memory interfaces shared among processors
    - | Computational power of multi-processors starved



# CPU External Memory Sorting

Performance of an optimized merge-based sort on Dual 3.6 GHz Xeon



Phase 2 speed increases with partition size (I/O bound)  
Phase 1 time is roughly constant since partitions fit in cache



# GPUSort: Gory Details

- ☞ Recall that we have two phases...
- ☞ Phase 1 consists of five stages
  - | Can be sequential, but should be pipelined to execute in parallel
  - | Paper implements stages in pipeline

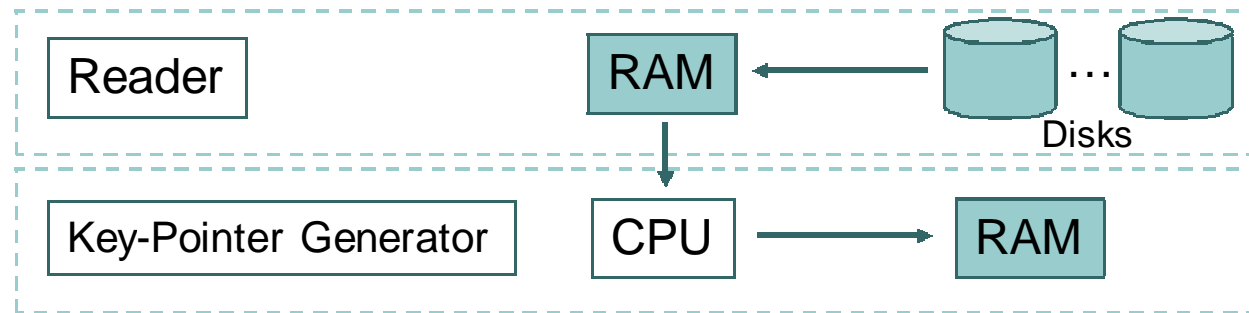
# Phase 1 Data Flow



## I Reader

- Reads phase 1 partition (~100MB) into RAM from striped disks
- < 10% CPU utilization

# Phase 1 Data Flow

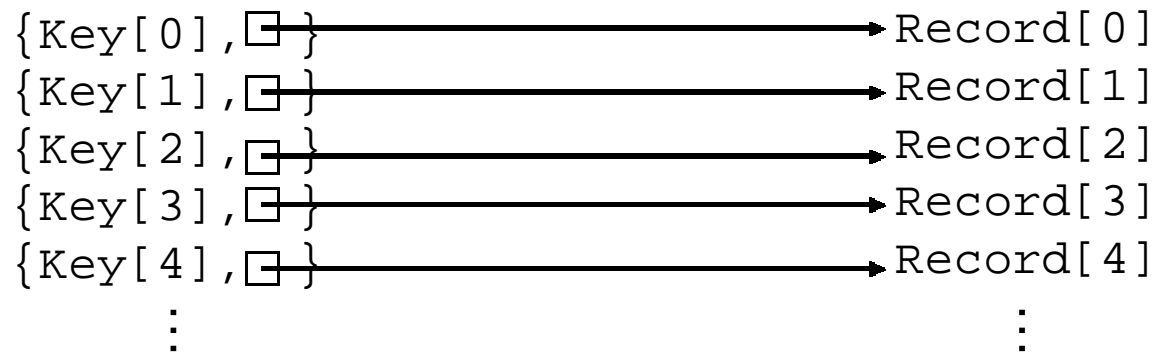


## I Key-Generator

- Computes {key, record-pointer} pairs
- Memory bandwidth-intensive, but not compute-intensive

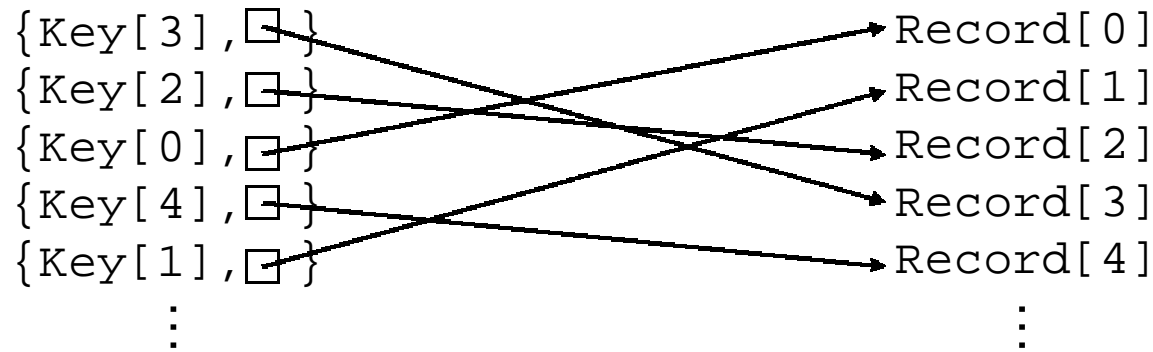


# Key-Generator



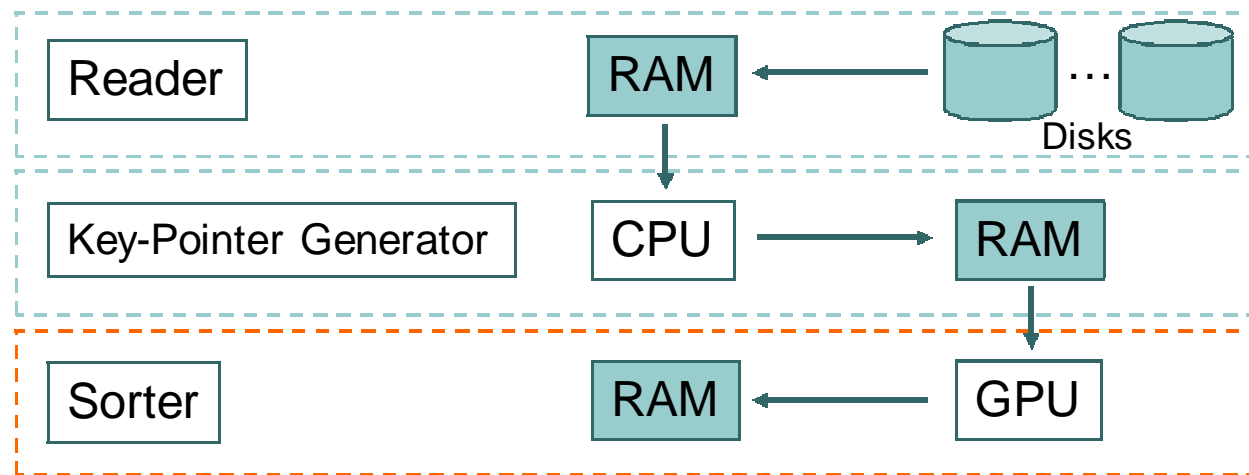
- | For each record
  - Generate some “comparable” key
    - String
    - Integer
    - Floating-point number
  - Store the memory location of record data with key

# Key-Generator



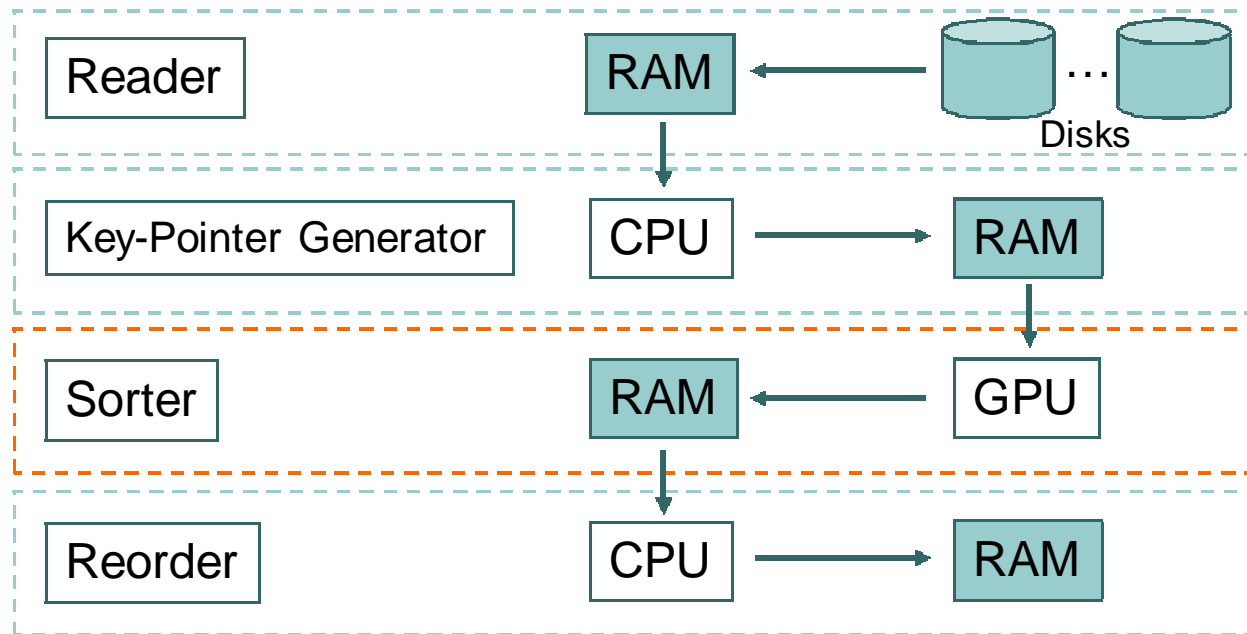
- | Sorting involves moving the keys around memory several times
- | Now we will only have to copy keys and pointers during sort
- | The potentially large records need only be moved (at most) once

# Phase 1 Data Flow



- | Sorter (typically bottleneck)
  - Computationally intensive and memory-intensive...
  - Implemented on GPU

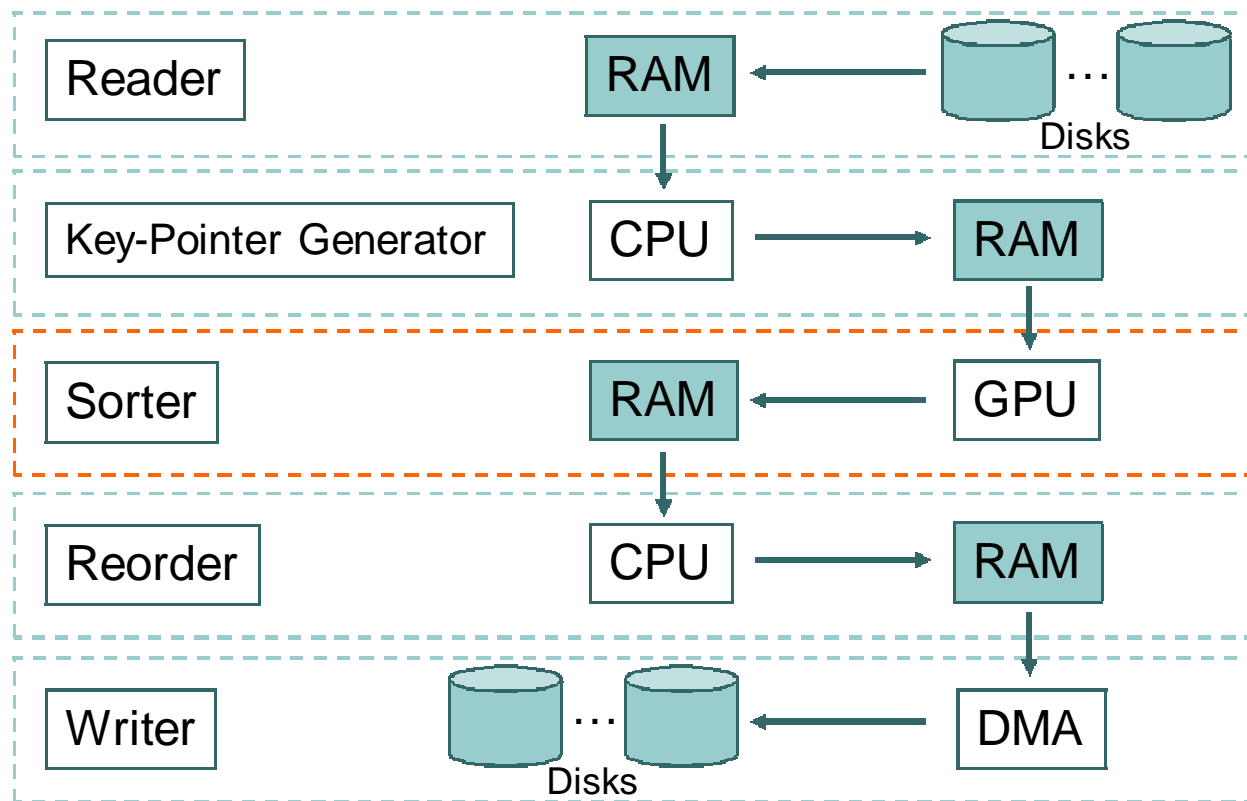
# Phase 1 Data Flow



## Reorder

- Uses sorted key-pointer pairs to generate a sorted output buffer
- Memory bandwidth-intensive

# Phase 1 Data Flow



## Writer

- Writes output runs onto disks (still need to merge in phase 2)
- < 10% CPU utilization



# Sorter Stage

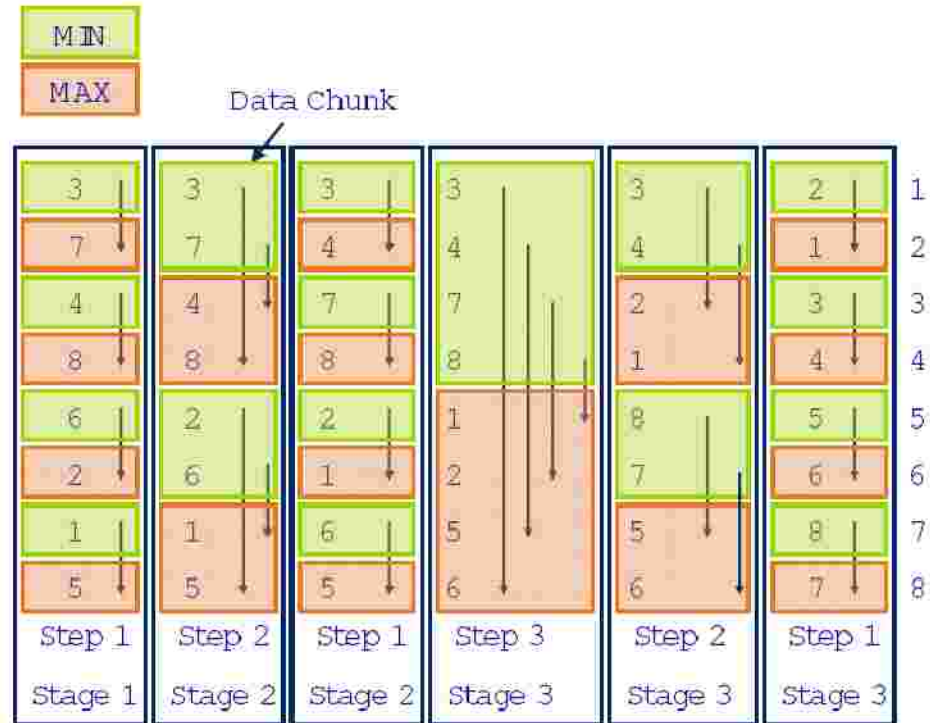
- | Execute sorter stage on GPU
  - Frees CPU cycles to improve I/O performance
  - Reduces memory contention on key-generator and reorder stages



# GPU Sorting Review

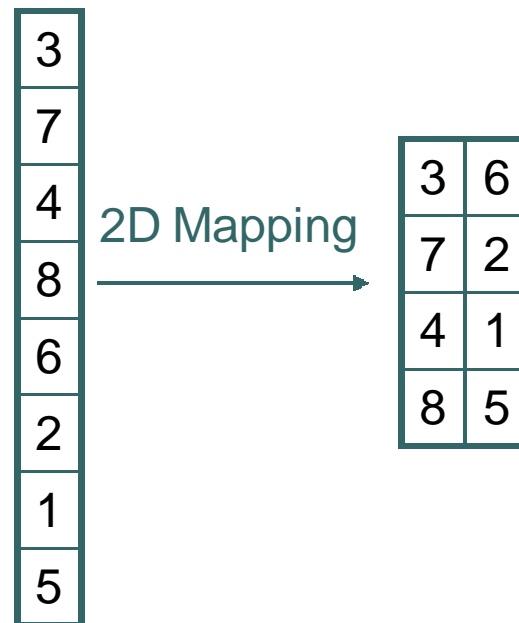
## Bitonic Sorting Network

- Output of stage  $k$  is  $N/2^k$  sorted sets, each set with  $2^k$  values
- Output of each stage is input to next stage





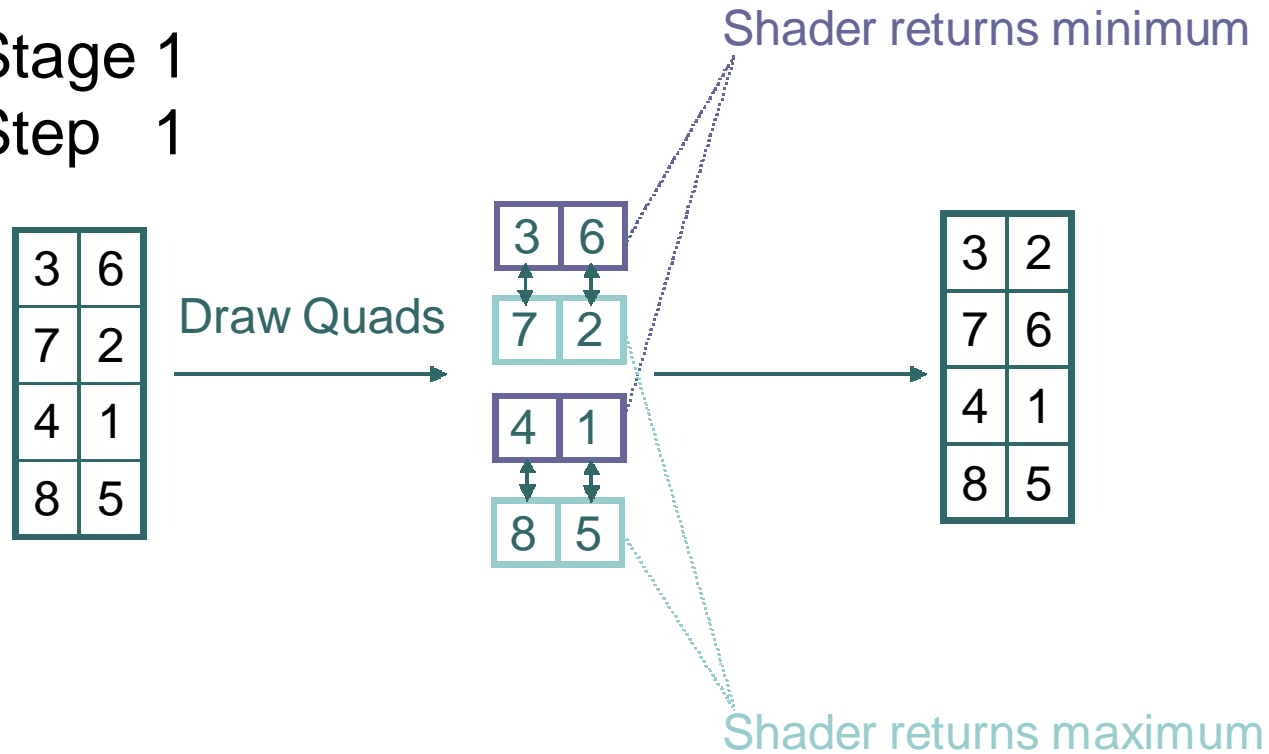
# GPU Sorting Review





# GPU Sorting Review

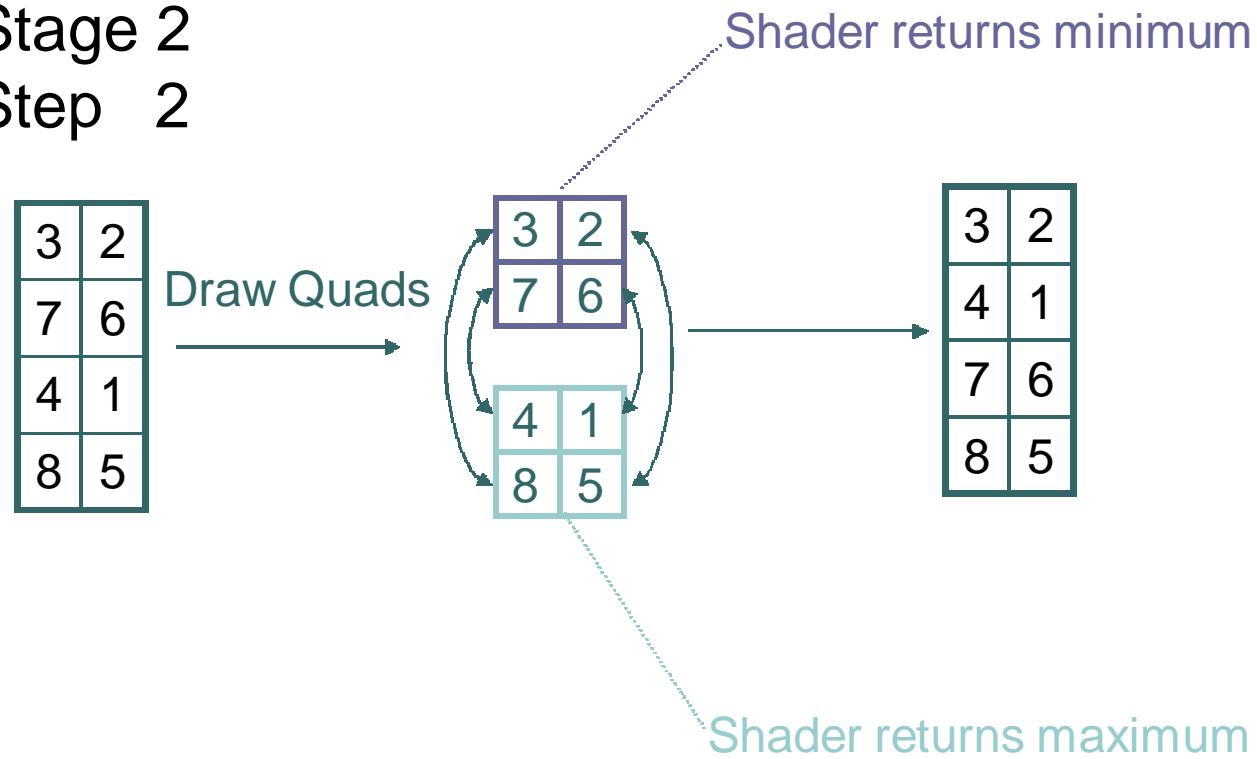
Stage 1  
Step 1





# GPU Sorting Review

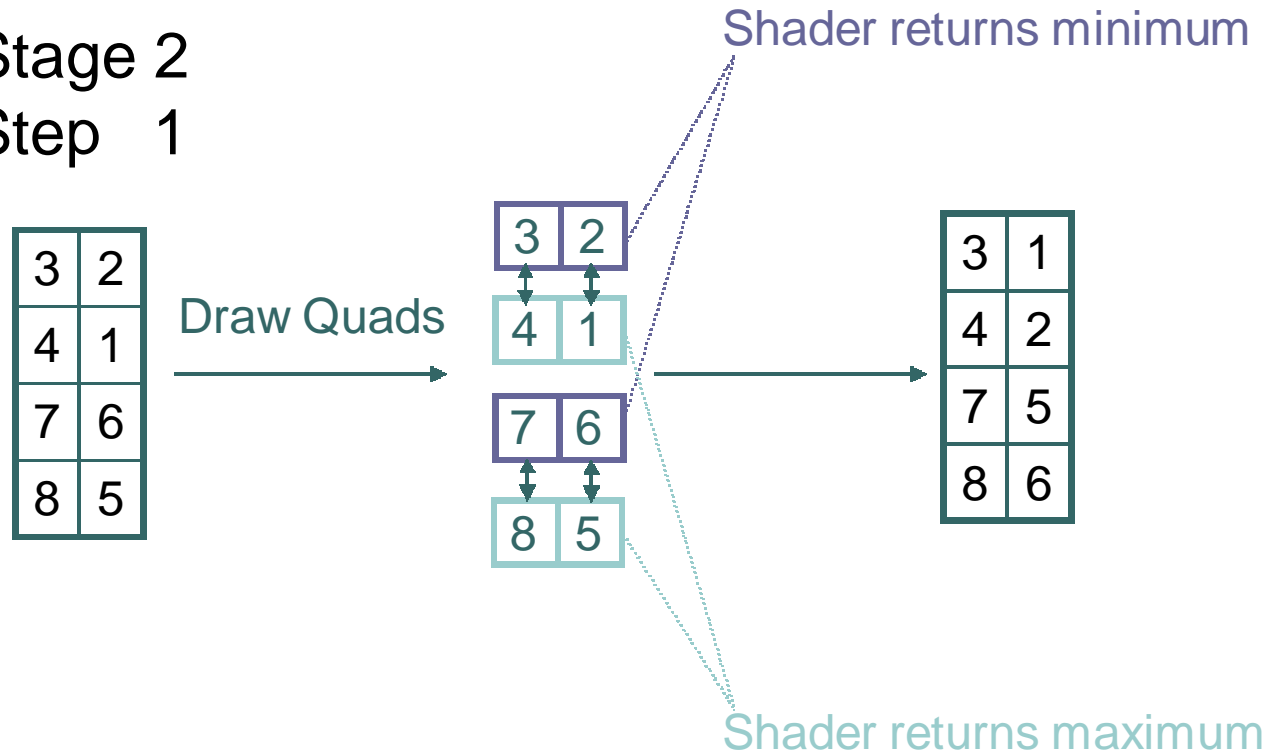
Stage 2  
Step 2





# GPU Sorting Review

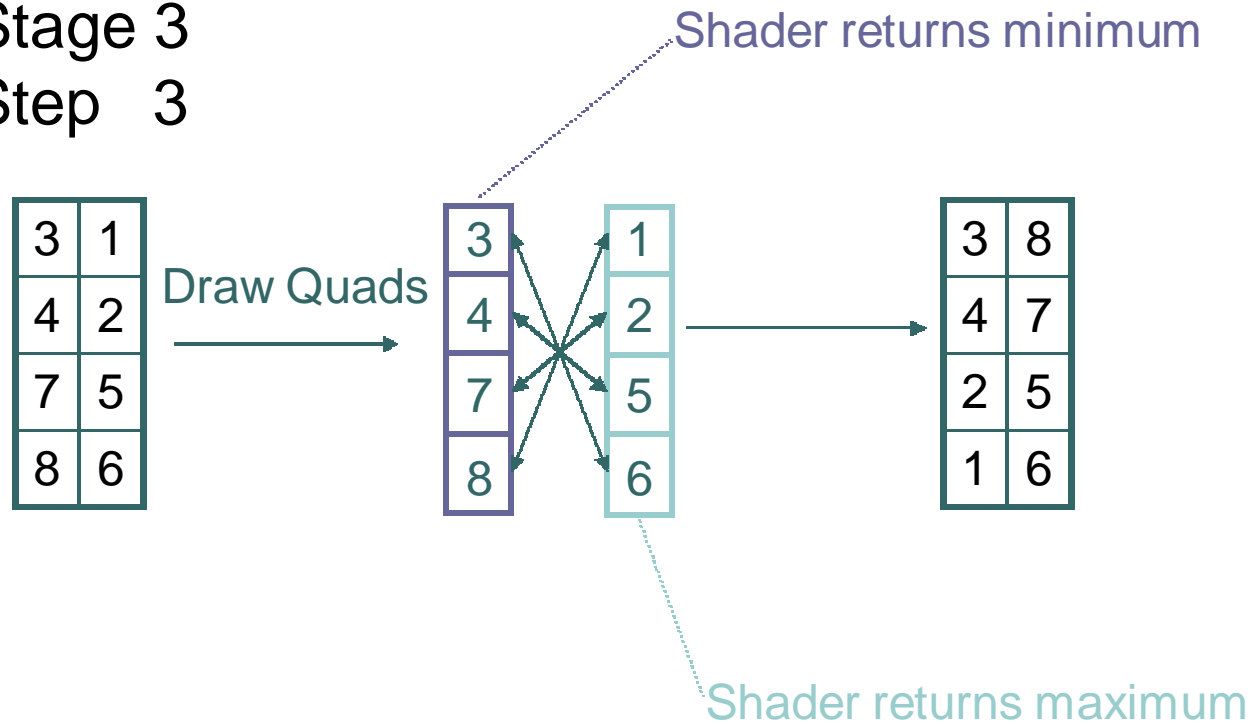
Stage 2  
Step 1





# GPU Sorting Review

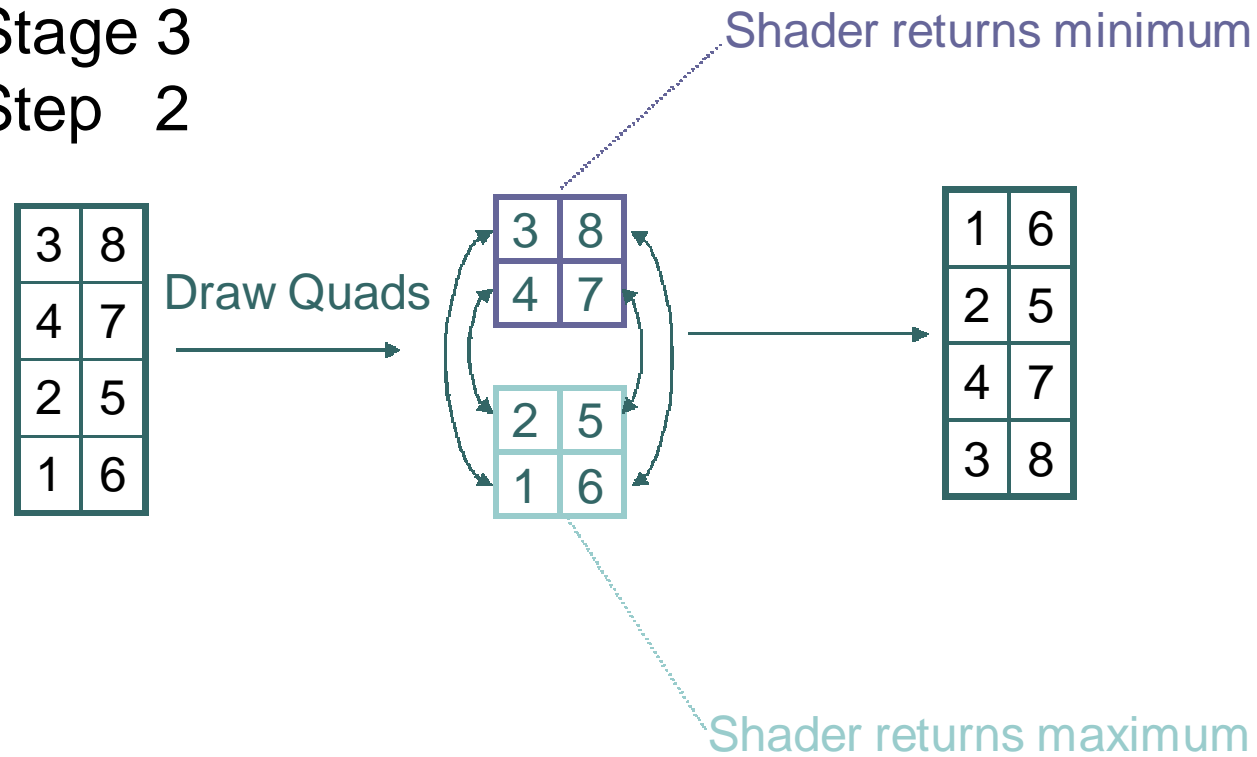
Stage 3  
Step 3





# GPU Sorting Review

Stage 3  
Step 2





# GPU Sorting Review

Stage 3  
Step 1

1	6
2	5
4	7
3	8

Draw Quads

1	6
2	5
4	7
3	8

Shader returns minimum

Shader returns maximum

1	5
2	6
3	7
4	8

Sorted!



# GPU Data Representation

- | Two possible data representation methods
  - Single-array:
    - Single CPU array of size  $4W \cdot H$
    - Stored as a texture of size  $W \times H$
  - Four-array:
    - 4 individual CPU arrays of size  $W \cdot H$
    - Stored as RGBA components of a texture of size  $W \times H$
- | In both schemes, keys and pointers are stored in separate textures



# Improved GPU Sorting

- | Single-array has the following two advantages over four-array:
  - Mapping from CPU->GPU is direct
    - Data transfer operation need not worry about interleaving 4 arrays
  - Fewer memory accesses in early steps of each bitonic sort stage
    - When sorting of 2- and 4-element arrays, can fetch a single texel



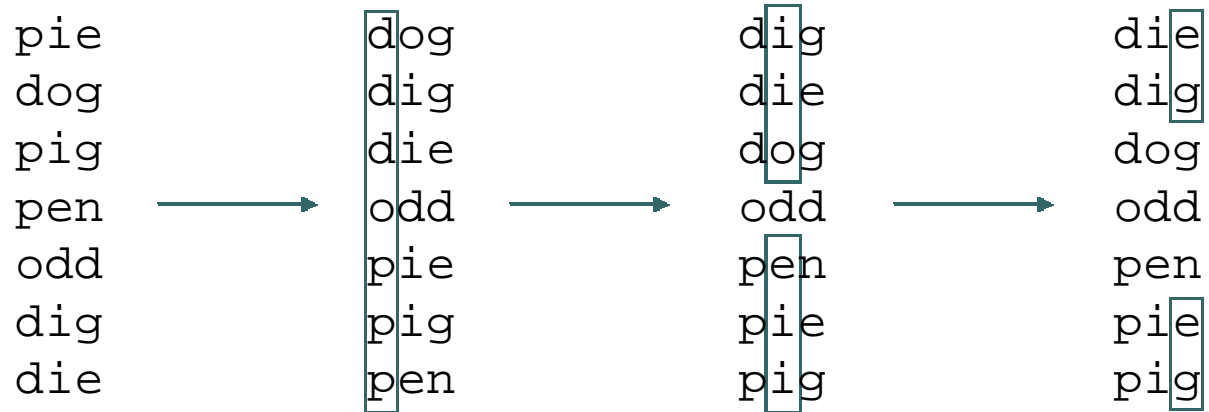
# Sorting Wide Keys

- | What if our keys are larger than our a single number?
  - | For example, greater than 32 bits on a GPU
- | Use hybrid radix-bitonic sort to handle “wide” key



# Radix Sort

- | Sort keys in pieces - from “most significant” to “least significant”
- | Humans do this to alphabetize strings:



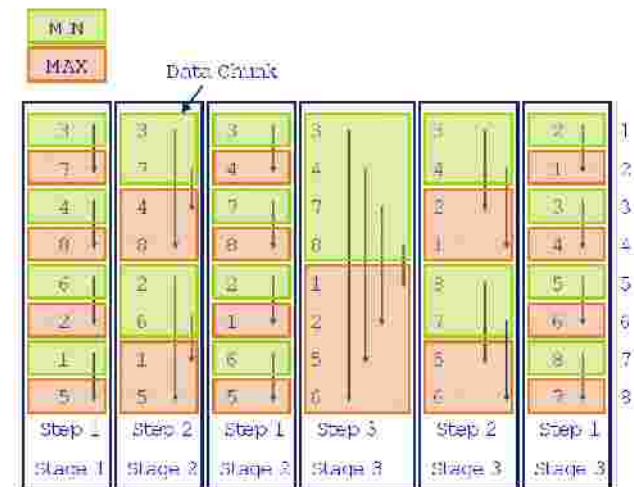


# Sorting Wide Keys

- | In best case, we still do only 1 bitonic sort
- | In worst case, we do  $B/4$  bitonic sorts, where keys are  $B$  bytes
- | Proceed as follows:
  - | Do bitonic sort on most-significant 4 bytes of key
  - | While there are lists of size  $> 1$  with equal keys
    - | For each contiguous list with equal keys, do bitonic sort on next most-significant 4 bytes
- | Only works if we avoid special IEEE values (NaN, 8)
  - | Mask first two bits to '10'

# Performance Analysis

- | Computational complexity:
  - | We do  $(N/2)$  comparisons each step
  - | There are  $(\log N)$  bitonic sort stages
    - | The longest stage has  $(\log N)$  steps
  - | Comparison complexity of  $O(N \log^2 N)$



- | In practice, using an NVIDIA 7900 GTX, GPUteraSort gets
  - | 14 GOPs/sec
  - | 49 GB/s memory bandwidth (56 GB/s is peak!)



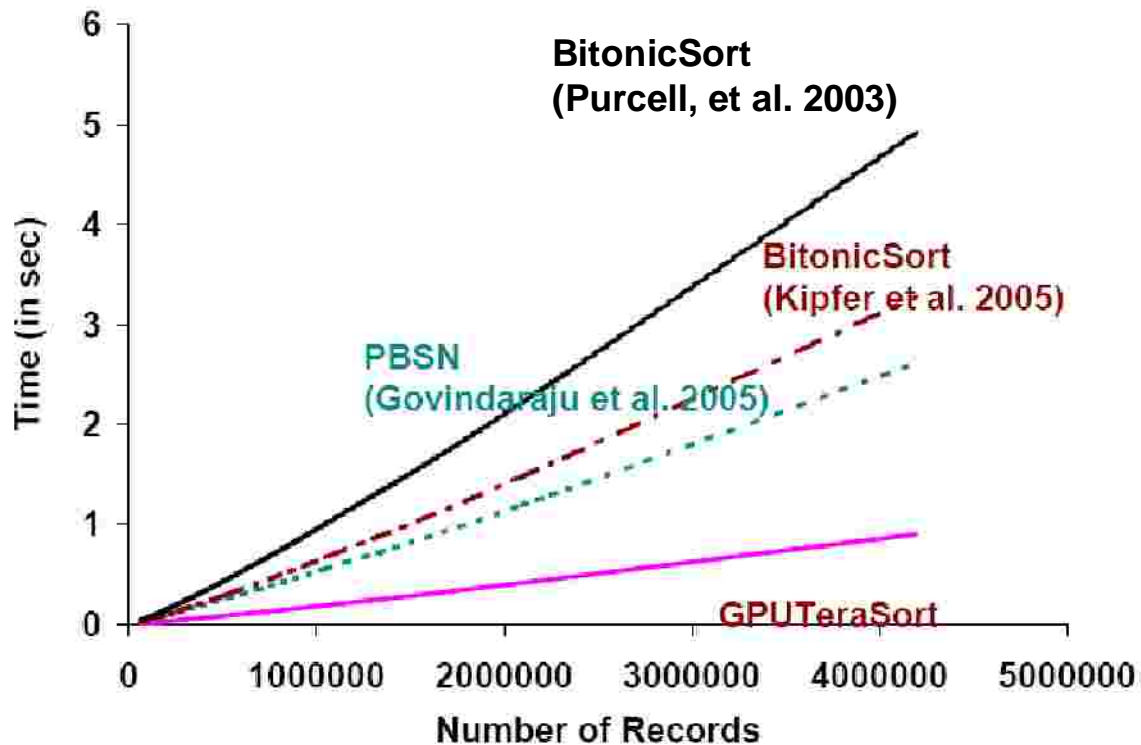
# Performance Analysis: Bandwidth

- | Data is transferred to and from GPU only once
  - | Not really... For wide keys, lists of equal keys computed on CPU
  - | But we can do this at the same time we sort another list on GPU
- | Transfers between disks and main memory are  $O(N)$
- | Claim is that in practice, less than 10% of sort time is data transfer
  - | Aren't we pipelined?
  - | 10% of the time is spent waiting for reader/writer to finish first/last chunk



# Performance vs. GPUs

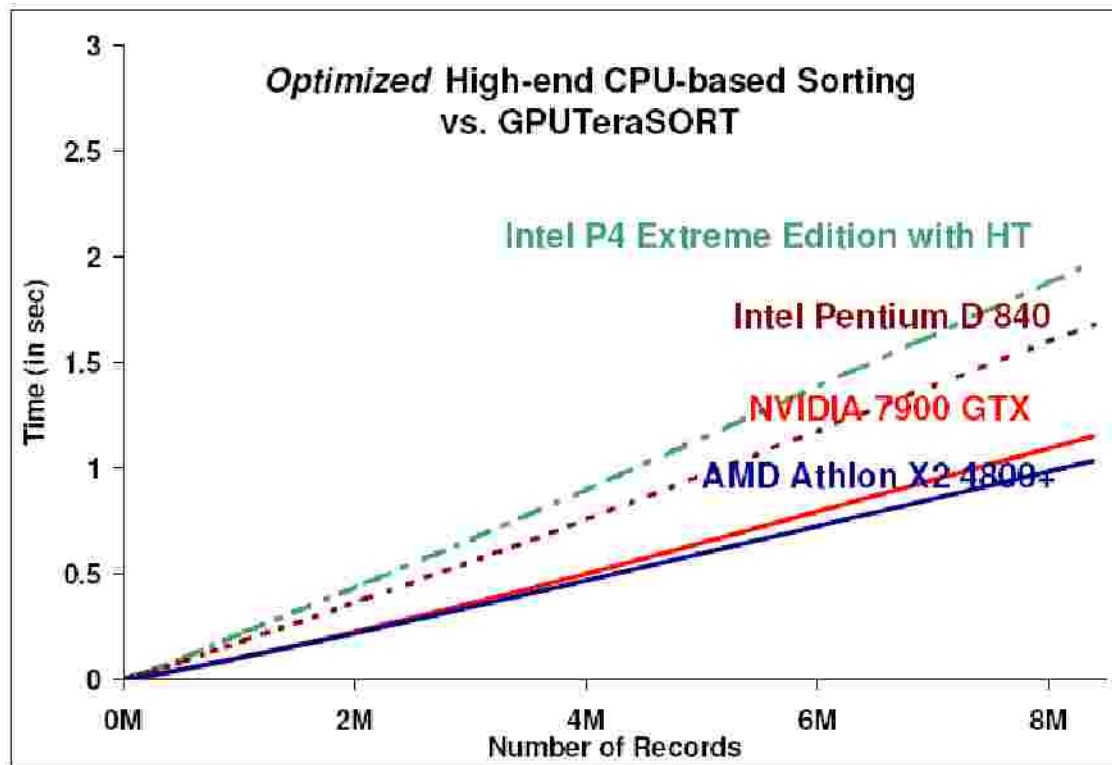
- ⌚ 3-7x performance advantage over previous GPU sorts
- ⌚ GPUs benchmarked with 32-bit keys on a 7800 GTX



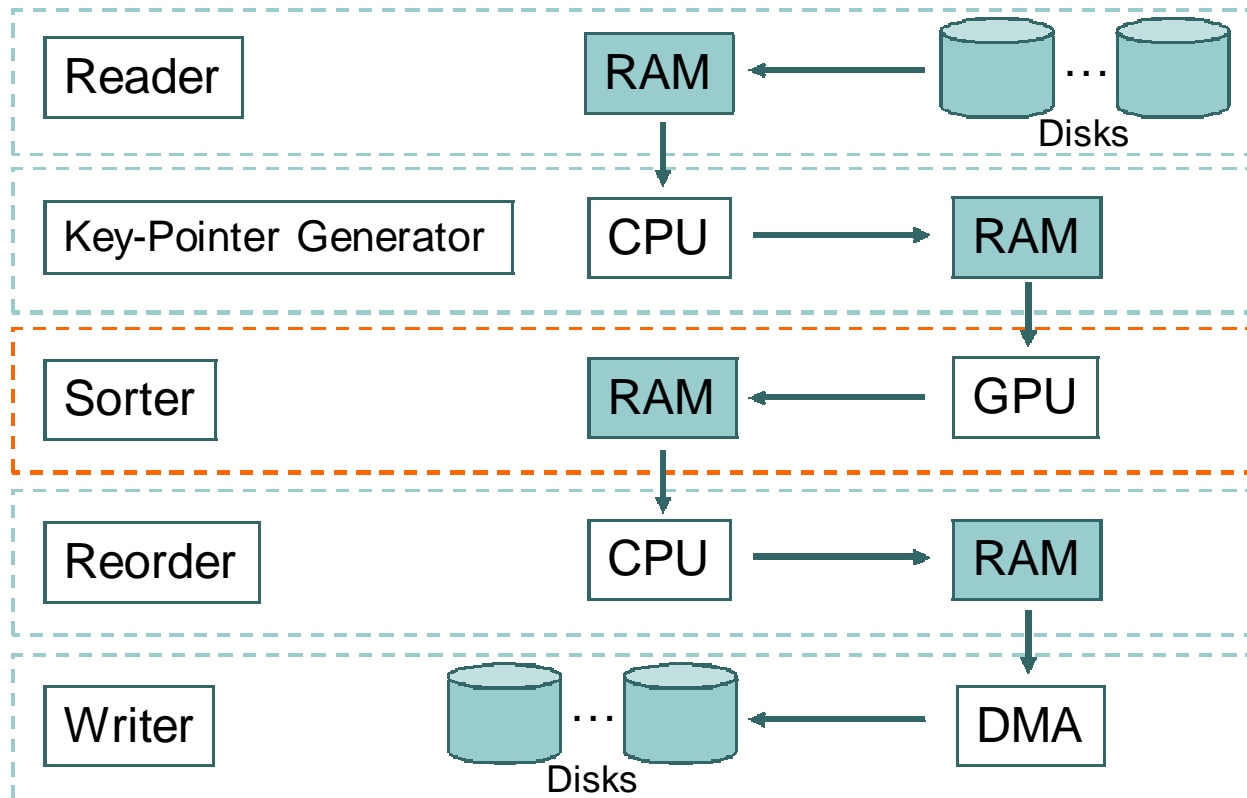


# Performance vs. CPUs

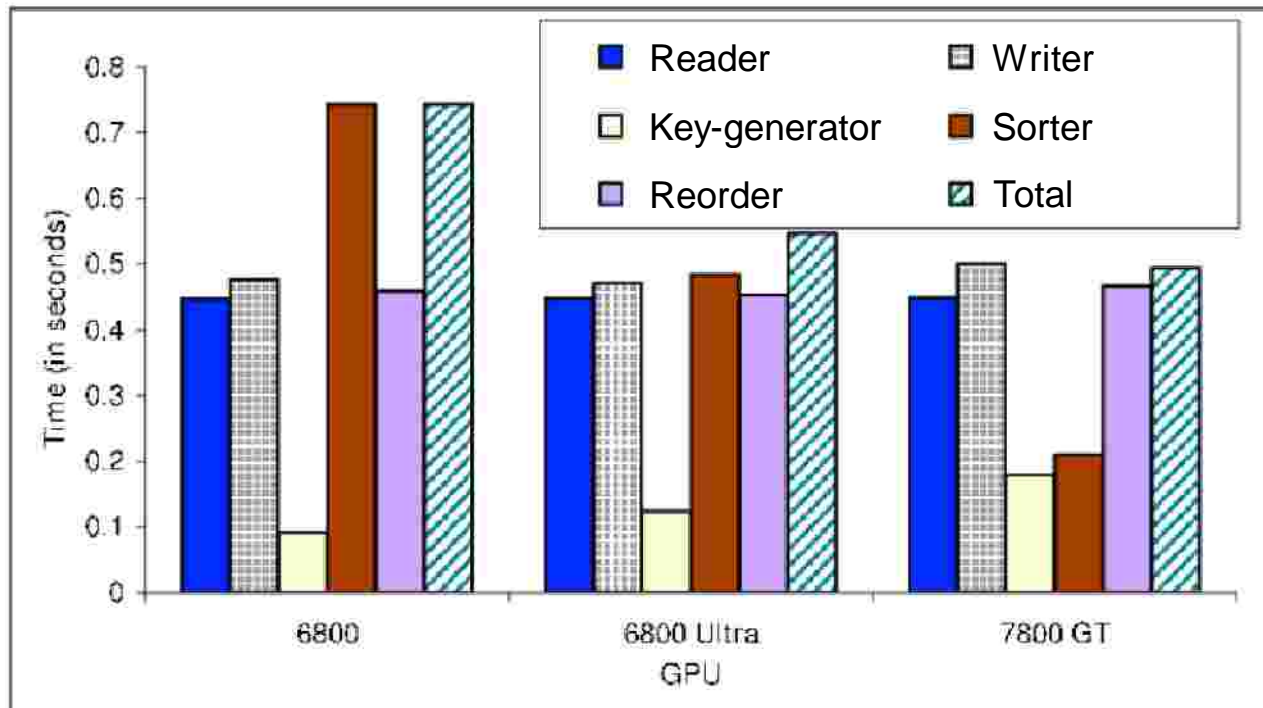
- ☞ Comparable to performance of high-end, commodity CPU sorting



# Recall: Phase 1 Stages



# Bottleneck Analysis



- ⌚ With fastest GPU, disk I/O system is bottleneck
- ⌚ PennySort record set with 9-disk system
  - | 4 read disks, 5 write disks



# Algorithm Limitations

- | Variable-sized keys allowed, but works best with keys of fixed size
  - | Hence, PennySort record on fixed-size keys
- | Not adaptive to input
  - | For example, no speed increase if input is sorted already
- | Requires programmable GPU
  - | Some low-end GPUs (PDAs/mobile phones) not programmable
- | Memory Size Limitation
  - | Each of phase 1's five stages share system RAM
  - | Thus, phase 1 run size is bounded by the smaller of
    - 1/5 the total RAM
    - GPU memory size



# Future Work

- | Clusters of PCs with multiple GPUs
  - | What about a single PC with multiple-GPUs?
- | Accelerating Phase 2 merge with GPUs
- | Integrating GPUSort with SQL systems



# Acknowledgements

- ⌚ Figures from the paper, “GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management.” Naga Govindaraju, Jim Gray, Ritesh Kumar, Dinesh Manocha.
- ⌚ PennySort Information from Jim Gray’s former page (now defunct): <http://research.microsoft.com/barc/SortBenchmark/>
- ⌚ Jason Sanders put together original version of these slides