# Fast, Scalable Parallel Comparison Sort On Hybrid Multicore Architectures

Dip Sankar Banerjee, Parikshit Sakurikar
and Kishore Kothapalli

IIIT, HYDERABAD

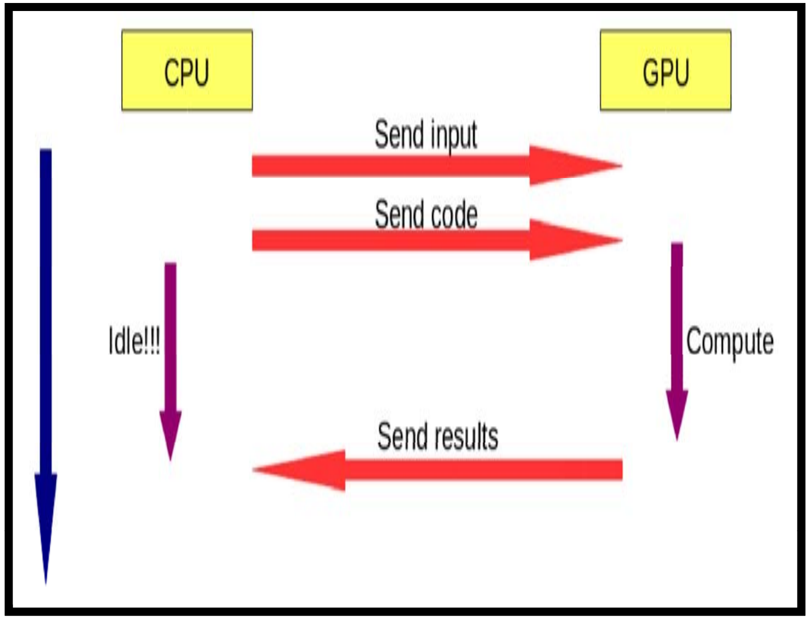
*Center For Security ,Theory, and Algorithmic Research*
*International Institute of Information Technology*
*Hyderabad, INDIA*

# GPGPU

- General purpose computation on GPUs. (GPGPU) is very common and widely practiced.
- Provides the lowest cost to FLOPS ratio.
- A many-core device which consists of:
  - Symmetric multi-processors.
  - Low power cores in each SM.
  - SIMD programmability.
  - Shared and global memory.
- High use in general purpose computations and remarkable results in widely used primitives.

# Accelerators in Computing

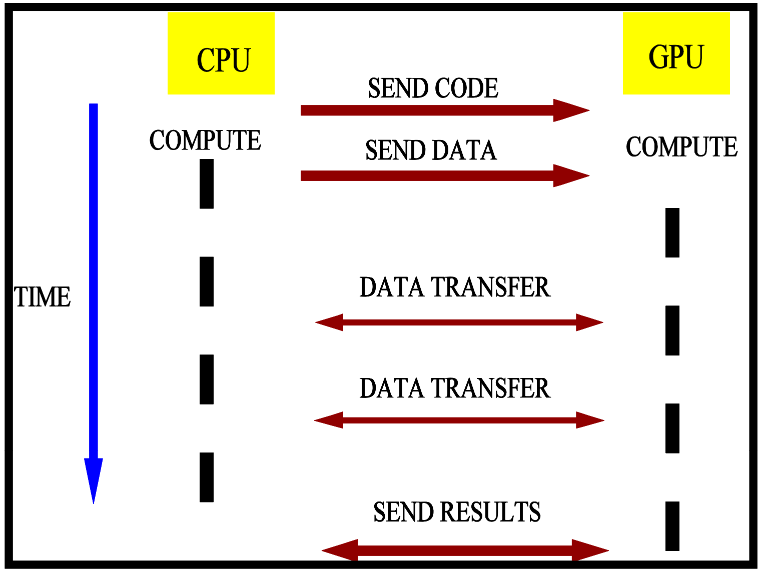
- Typical usage model
  - Transfer input from CPU to the accelerator
  - Transfer program to the accelerator
  - Execute on the accelerator
  - Transfer results back to the CPU.




- Above model necessitated partly because the accelerators do not have I/O capability.
  - Truly auxiliary devices.

# Accelerators in Computing

- The big issue: Utilizing multiple multicore devices for computation.
- CPU Utilization for solving generic problems:
  - CPUs have high compute power cores.
  - Computational power of CPUs is also on the rise.




- Hybrid Multi-core Computing
  - Use all resources available(in a single platform).
  - Provides a higher level of parallelism and efficiency.

# Outline

- General Hybrid Computing Platform
- Problem Statement
- Our Solution
- Implementation Details
- Results
- Conclusion

# Hybrid Multicore Platforms

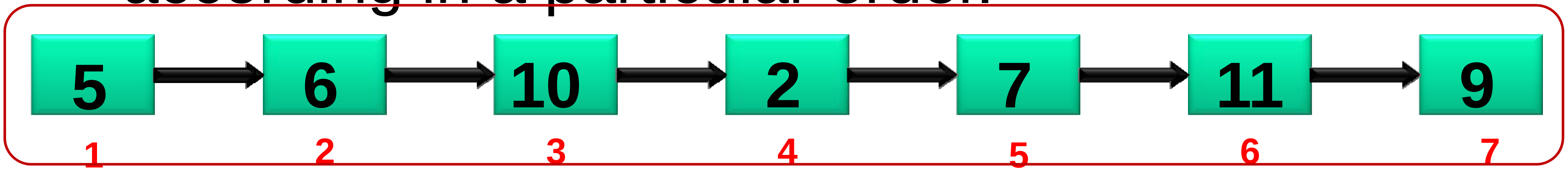
- Target of our research is to validate the implementation of algorithms on both high-end systems as well as commodity low-end system.
- A high end system will have a high throughput GPU connected to a multi-core CPU.
  - An Intel i7 980 coupled with an NVidia GTX 580 GPU
- A low-end system is typically found on commodity systems such as laptops and desktops.
  - An Intel Core 2 Duo E7400 CPU coupled with an NVidia GT520 GPU.

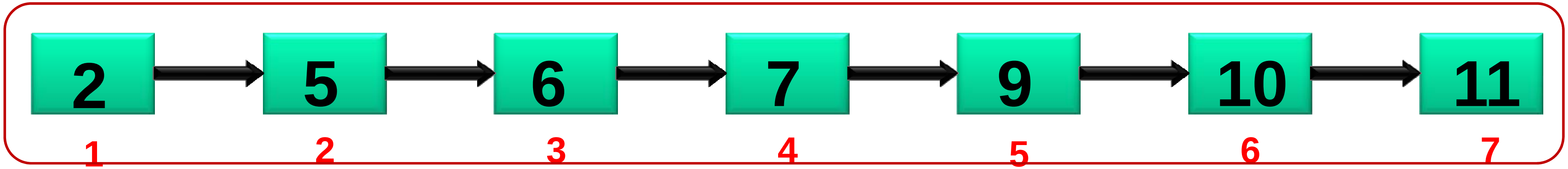
# Our Results

- In this work we implemented comparison sort on a hybrid multicore platform.
- We used hybrid sample sorting on the platform using different data sets.
- Our sorting implementation is 20% better than the current best known parallel comparison sorting, due to Davidson et. al. at InPAR 2012.
- Our results are on an average 40% better than the GPU Sample Sorting algorithm published at IPDPS 2010.

# Problem Definition

- Sorting is a fundamental algorithm which finds massive application in scientific computations, databases, searching, ranking etc.
- The problem is to arrange a certain set of input according in a particular order.




- Sorting is an irregular operation and is not entirely suited for GPU or parallel architectures.

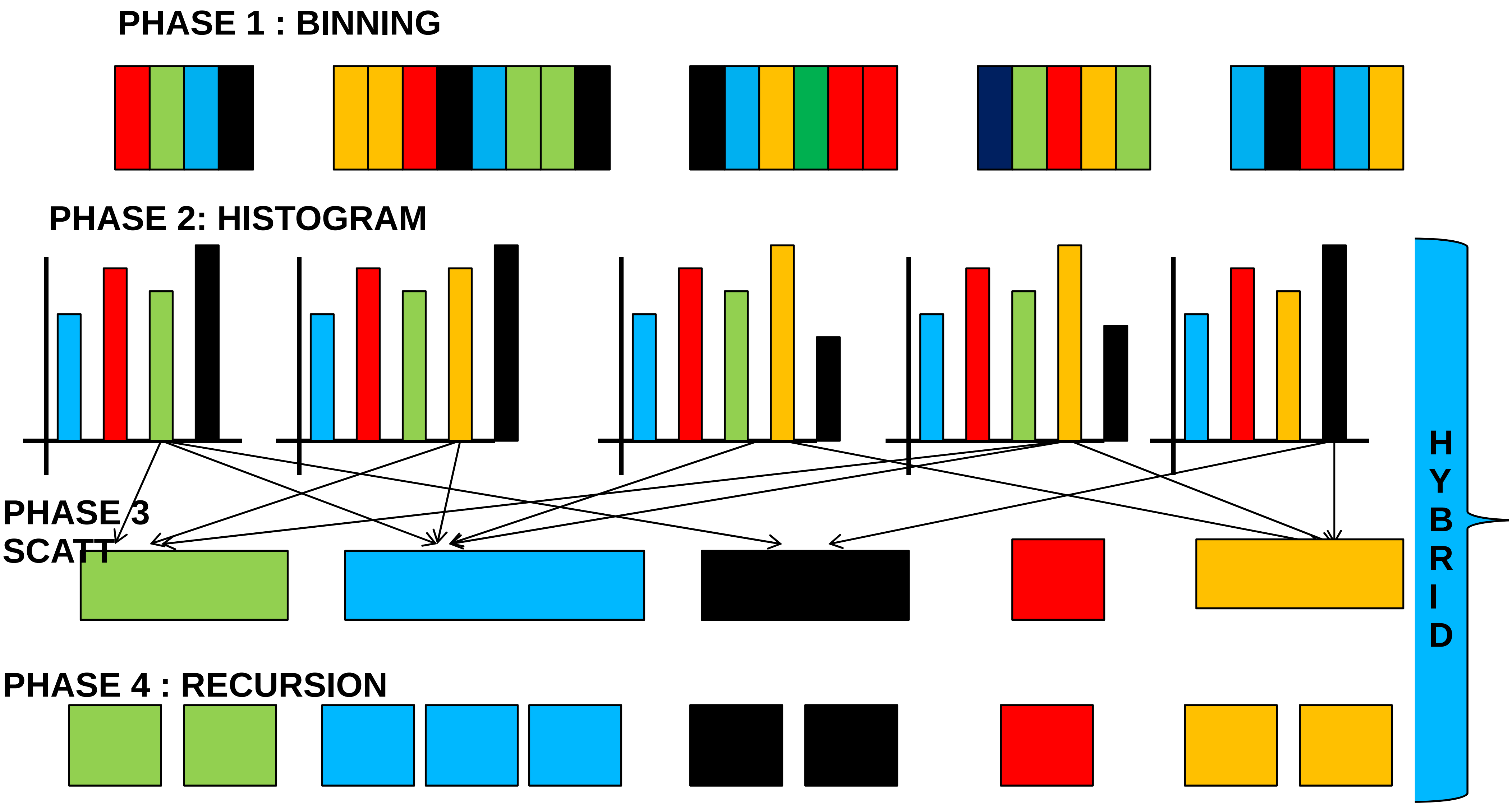
# Parallel Sorting

- Effective use of all available processors by creating independent sub-problems.
- Quick sort is a popular sorting technique where sub-problems are created and solved in a recursive fashion.
- Sample sort is a generalization of the quick-sorting algorithm that chooses many pivots and hence creates higher number of sub-problems.
- Each of these sub-problems can be efficiently allocated to either a CPU or a GPU for sorting.

# Algorithm Overview

- **Phase I**
  - Create sqrt(n) bins where n in no. of input elements.
  - Efficiently bin elements using a BST.
- **Phase II**
  - Compute histograms of the bins allocated to each CPU and GPU.
- **Phase III**
  - Scatter elements across all SMs on GPU and cores in CPU in an synchronous manner.
- **Phase IV**
  - Recurse Phases I-III and until bin sizes are reduced to a certain threshold.
  - Sort the small bins.

# Algorithm Overview



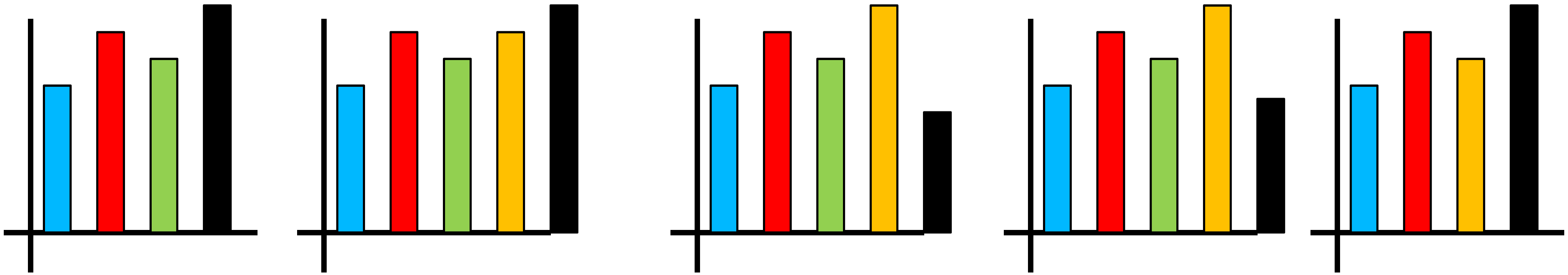
# PHASE I : BINNING

**PHASE 1**

- Select splitters at uniform intervals of sqrt(n)
- Form a Binary Search Tree using the splitters
- Now set a threshold for separation of the labels between the CPU and the GPU.
- Transfer GPU labels to the device
- Use BST on both CPU and GPU to bin the elements.

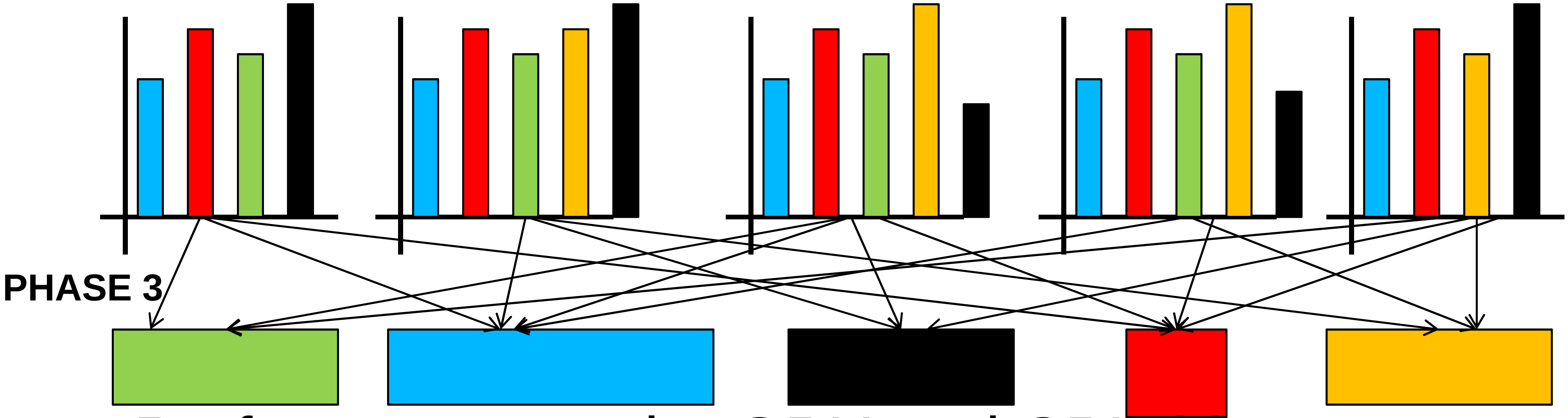
# PHASE II : Histograms

PHASE 2



- Compute histograms in an overlapped fashion on both the CPU and the GPU.
- Store histogram $H_c$ of CPU for LEN/BLOCK size of elements.
- Store $H_g$ of GPU on LEN/BLOCK size of elements.

# PHASE III : Histograms



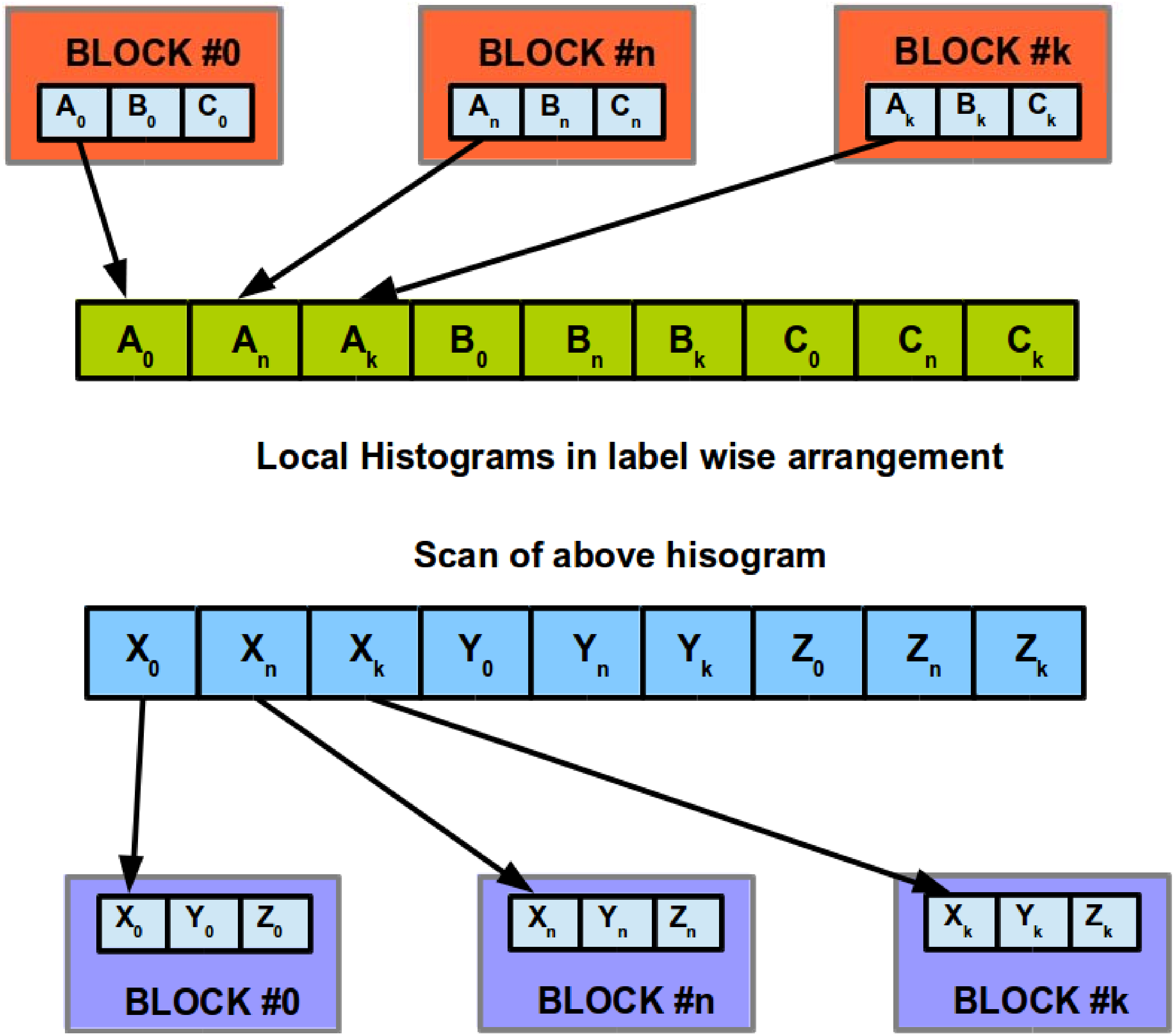

- Perform scan on the GPU and CPU histograms to compute the block-wise offsets.
- Scatter elements in an hybrid fashion to all bins
  - GPU: Perform local scattering in each BLOCK
  - CPU: Perform global scattering across the single BLOCK.

# PHASE IV : Recurse and Sort

- Recurse from phases I to III until the size for each block comes down to a size where we can do a normal quick sort on each thread.

- Separate the bins among the CPU and GPU and apply the sorting on each of the bins until a final sorted sequence is obtained.

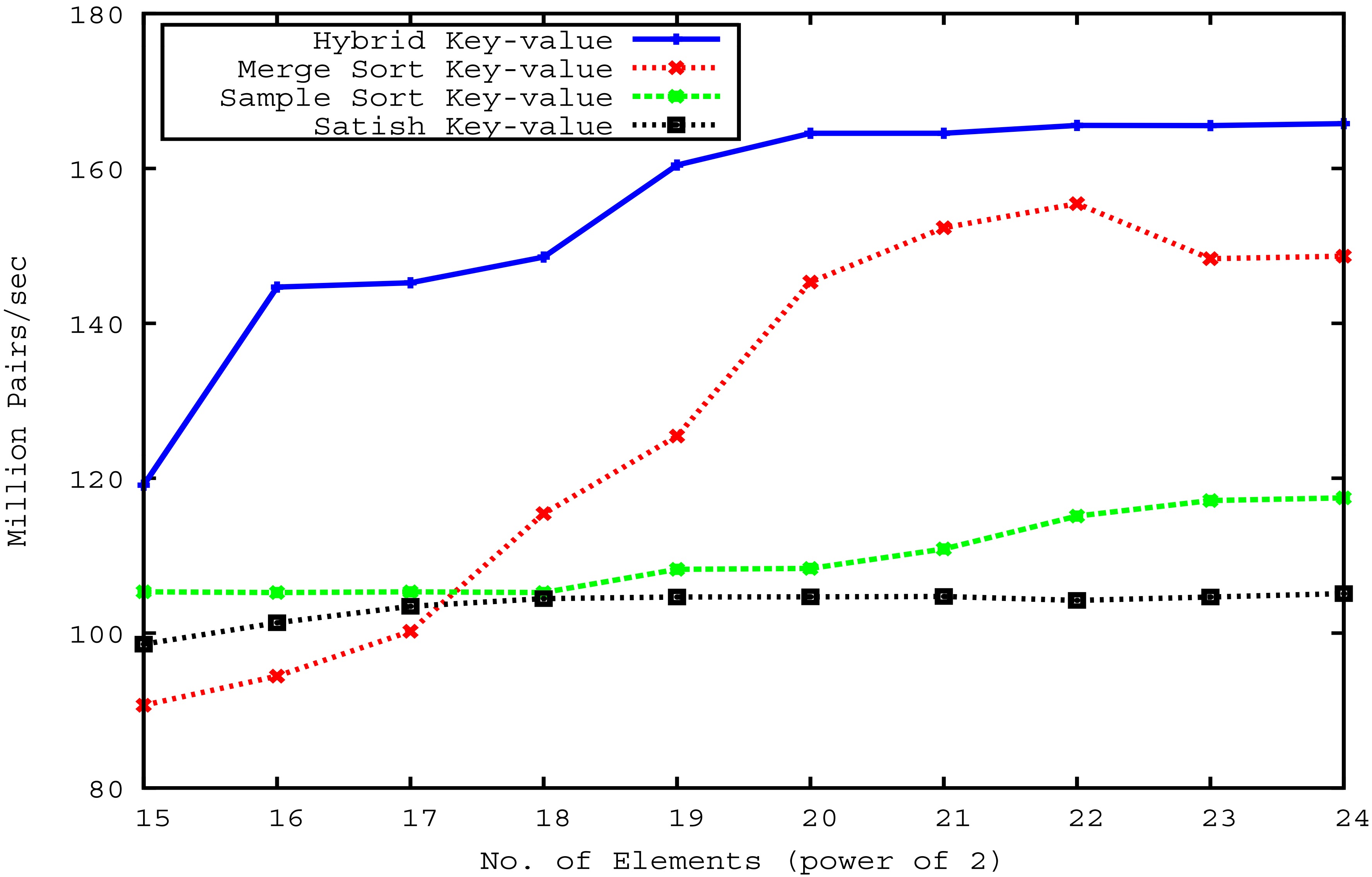
# Memory Access Optimization



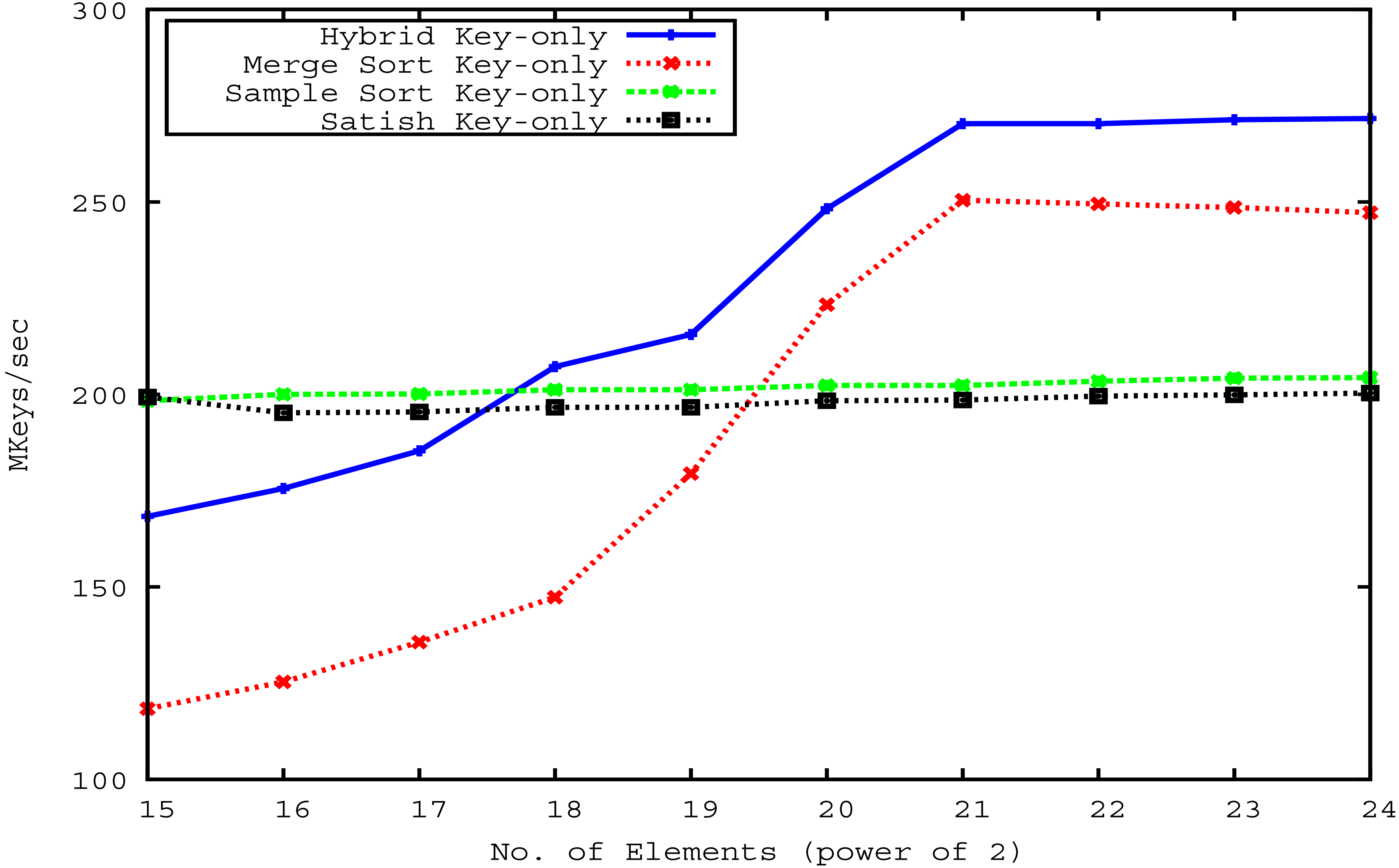
# Memory Access Optimization

- Available memory is a vital resource.
- Reuse of data-structures is vital for synchronization and consolidation.
- We reuse our histogram store in the scattering step where we do not write all the entries for all the labels together.
- Instead writing all the entries in one space, we write it in the order in which it will be read back.
- This facilitates a higher coalescing of reads as well as the re-use of a data-structure.

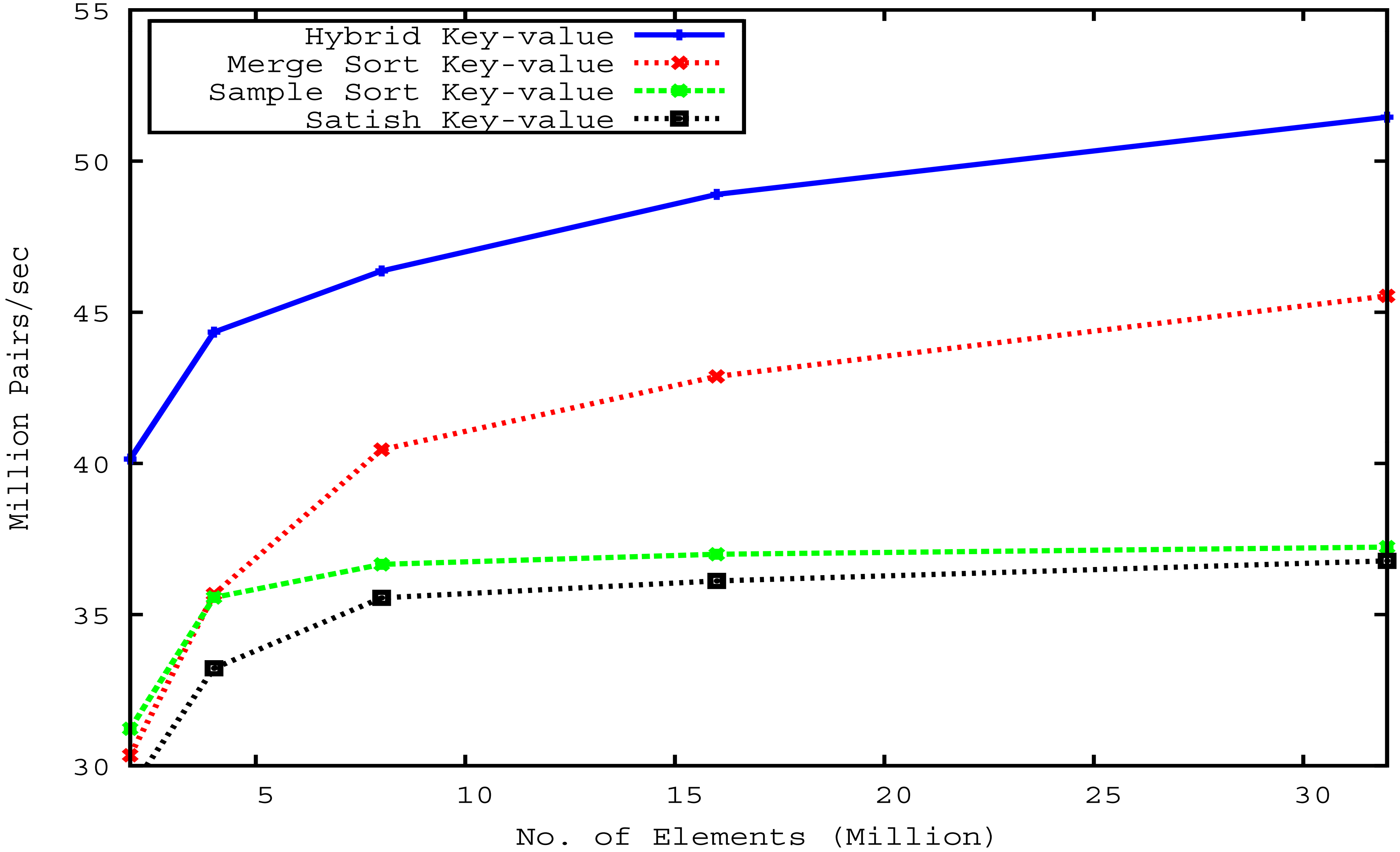
# Results on Key-Value Pairs



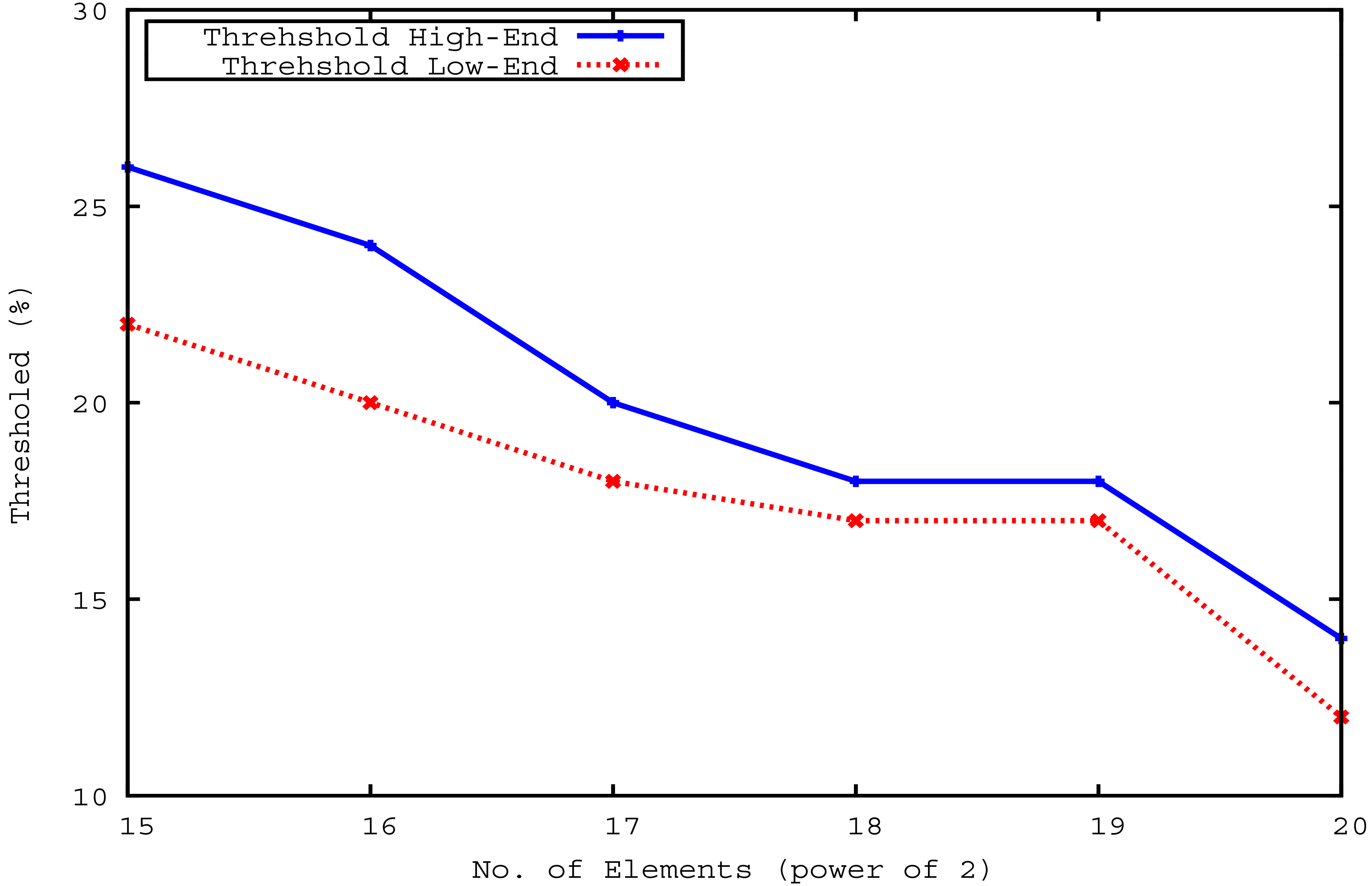
# Results on 32 bit integers



# Results of Key-Value Pairs on Low-End Platform



# Variation of Threshold



# Results

- Our Key-Value pair sorting is on an average **20% better** than the current best known result.
- Our 32 bit sorting results are on an average **23% better** than the current best known result.
- The performance benefit can be attributed to:
  - Hybrid Histogram computation which reduces atomic and irregularity overheads.
  - Overlapped scattering which reduces the memory access latencies.

# Conclusions

- Our implementation clear shows the benefits of a heterogeneous platform.
- Hybrid algorithms show promising results on both high-end as well as commodity level processors.
- Our algorithm can be very easily incremented to sort variable length keys such as strings.
- It will also be of interest to experiment and optimize on other data sets such as Deterministic Duplicates, Staggered Keys, Bucket Sorted Keys.

THANK YOU

Questions ?