



Extreme Big Data (EBD) Overview

The diagram illustrates the architecture of EBD Proxy Applications, showing the flow of data and processing between various components:

- Top Level Applications:**
 - Genomics (DB matching):** Represented by a DNA helix structure.
 - Social Simulation (Graph):** Represented by a globe with network connections.
 - Weather (Stream Sensor):** Represented by a satellite image of Earth.
- Intermediate Processing and Storage:**
 - Programming Framework for EBD Object Store:** A red box on the left.
 - Resource Management for Interactive Queries and Analyses:** A red box on the right.
 - EBD Object Store:** A central orange cylinder representing the core storage layer.
- Data Sources and Formats:**
 - Genomic Data:** A sequence of nucleotides: ATGAACCAAGAACAATTTT, CCATCTATTTATGATTTTATG, GCCAATATTTCTTAAATCGT, TTAATGATTCGCCCTAGTTT.
 - Graph Store:** A blue cylinder connected to a network graph.
 - Cartesian Plane:** A map with a grid overlay.
 - EBD KVS:** A map showing key-value storage locations.
- Infrastructure:**
 - Extreme Computing Infrastructure:** A red box at the bottom, representing the underlying hardware and network.

Arrows indicate the flow of data and processing: from the top applications down to the EBD Object Store, and from the EBD Object Store down to the Extreme Computing Infrastructure. The EBD Object Store also interacts with the EBD Bag, Graph Store, Cartesian Plane, and EBD KVS.

HAMAR is a MapReduce-style programming for next-gen supercomputers with many-core accelerators and non-volatile memory devices. Our framework handles memory overflow from GPUs by dividing data into multiple chunks and overlaps CPU-GPU data transfer and computation on GPUs as much as possible.^[1]

We have developed extremely fast breadth first search (BFS) implementations for large-scale distributed environments and NVM-based hierarchical memory machines^[2]. We have achieved several notable results on the Green500 and the Green Graph500, including becoming world #1 on the Graph500 (June 2014)^[3] on K Computer and #1 on the Green Graph500 (November 2013)^[4] on TSUBAME-KFC, based on our implementations.

Figure 1: Performance of the GRASP system. The chart shows the performance in GTEPs for four datasets. The performance increases from 100 GTEPs for SC'11 to 1,280 GTEPs for ISC'14.

Dataset	Performance (GTEPs)
SC'11	100
ISC'12	317
SC'12	462
ISC'14	1,280

Optimizations	SC11	ISC12	SC12	ISC14
2D decomposition	✓	✓	✓	✓
vertex sorting	✓			
direction optimization				✓
data compression	✓	✓	✓	
sparse vector with pop counting				✓
adaptive data representation	✓			✓
overlapped communication			✓	✓
shared memory		✓		✓

1. Hybrid-BFS (Beamer'11)

Top-down
Bottom-up

Source

2. Proposal

DRAM

Holds highly accessed data

NVM

Holds full size of Graph

Load highly accessed graph data before BFS

3. Experiment

CPU	Intel Xeon E5-2690 × 2
DRAM	256 GB
NVM	EBD-1/O 2TB × 2

mSATA SSD × 3
 RAID Card (RAID 0)

RAID

RAID

Med an GigaTEPS
(@ 1m Traversed Edges Per Second)

Limit of DRAM Only

4 times larger graph with 6.9% of degradation

SCALE

23 24 25 26 27 28 29 30 31

Collaboration work with JST Graph CREST project
and RIKEN AICS

We have developed several GPU-based sorting implementations, including support for variable length keys^[5], large-scale distributed environments^[6], and out-of-core GPU memory management^[1].

Distribution Sorts:

- not cmp based → linear/sublinear time

MSD radix sort
variable length keys
high efficiency on small alphabets
long keys

LSD radix sort (THRUST)
good for short length keys
only fixed-length keys

integer sorts

GPUs are good at counting numbers

computational economics (C,G,T)

apple
apricot
banana
kiwi

mergesort, etc.)

- N log(N) cmp operations
- slow when cmp is expensive

can't have to examine all characters

Figure 10 is a line graph showing the performance of various sorting algorithms in terms of Keys/second (millions) versus the Number of keys (millions). The x-axis ranges from 0 to 20 million keys, and the y-axis ranges from 0 to 250 million keys/second. The legend identifies five data series:

- CUDA K20 MSD sort + OpenMP Core i7 120threads var_fen_keys** (Blue line): Shows the highest performance, peaking around 210 million keys/second at 10 million keys and then slightly declining.
- Sequential MSDSort, Core i7_2930k var_fen_keys(1000k avg)** (Red line): Shows performance peaking around 80 million keys/second at 10 million keys and then declining.
- CUDA_K20 + CUDA_recursive_sort (var_fen)** (Green line): Shows the lowest performance, peaking around 40 million keys/second at 10 million keys and then declining.
- OpenMP_Xeon_E5_2687v3_320threads_var_fen_keys(1000k avg)** (Purple line): Shows performance peaking around 150 million keys/second at 10 million keys and then slightly declining.
- OpenMP_Xeon_E5_2687v3_320threads** (Cyan line): Shows performance fluctuating around 150 million keys/second across the range of keys.

Top Graph: KeyRate (millions) vs. # of processes (2 processes per K20x)

- HykSort threads** (red line with 'x' markers): Shows the highest performance, reaching approximately 20 million KeyRate at 2048 processes. An annotation '0.25 [TB/s]' is placed near the start of this line.
- HykSort GPU + 6 threads** (blue line with 'x' markers): Reaches approximately 15 million KeyRate at 2048 processes. An annotation 'x1.4' is placed near the end of this line.
- HykSort GPU + 2 threads** (green line with 'x' markers): Reaches approximately 10 million KeyRate at 2048 processes. An annotation 'x3.61' is placed near the end of this line.
- HykSort GPU** (orange line with 'x' markers): Reaches approximately 5 million KeyRate at 2048 processes. An annotation 'x389' is placed near the end of this line.

Bottom Graph: KeyRate (millions) vs. # of processes (2 processes per K20x)

- HykSort threads** (red line with 'x' markers): Reaches approximately 60 million KeyRate at 1024 processes.
- HykSort GPU + 6 threads** (blue line with 'x' markers): Reaches approximately 50 million KeyRate at 1024 processes.
- HykSort GPU + 2 threads** (green line with 'x' markers): Reaches approximately 40 million KeyRate at 1024 processes.
- HykSort GPU** (orange line with 'x' markers): Reaches approximately 30 million KeyRate at 1024 processes.
- PCuM-100** (purple line with 'x' markers): Reaches approximately 20 million KeyRate at 1024 processes.
- PCuM-200** (brown line with 'x' markers): Reaches approximately 15 million KeyRate at 1024 processes.
- PCuM-50** (pink line with 'x' markers): Reaches approximately 10 million KeyRate at 1024 processes.

Annotations and Text:

- 0.25 [TB/s]**: Annotation for HykSort threads in the top graph.
- x1.4**, **x3.61**, **x389**: Performance ratio annotations in the top graph.
- 2x faster than K20x**: Annotation for HykSort threads in the bottom graph.
- x2.2 speedup compared to CPU-based implementation when the # of PCI bandwidth increase to 50GB/s**: Annotation for HykSort threads in the bottom graph.
- 8.8% reduction of overall runtime when the accelerators work 4 time faster than K20x**: Annotation for HykSort threads in the bottom graph.

[5] A. Drozd, M. Pericas, S. Matsuoka, “Efficient String Sorting on Multi- and Many-Core Architectures”, BigData Congress 2014.

[6] H. Shamoto, K. Shirahata, A. Drozd, H. Sato, S. Matsuoka, “Large-scale Distributed Sorting for GPU-based Heterogeneous Supercomputers”. IEEE BigData 2014.