Extreme Big Data - Tools and Library

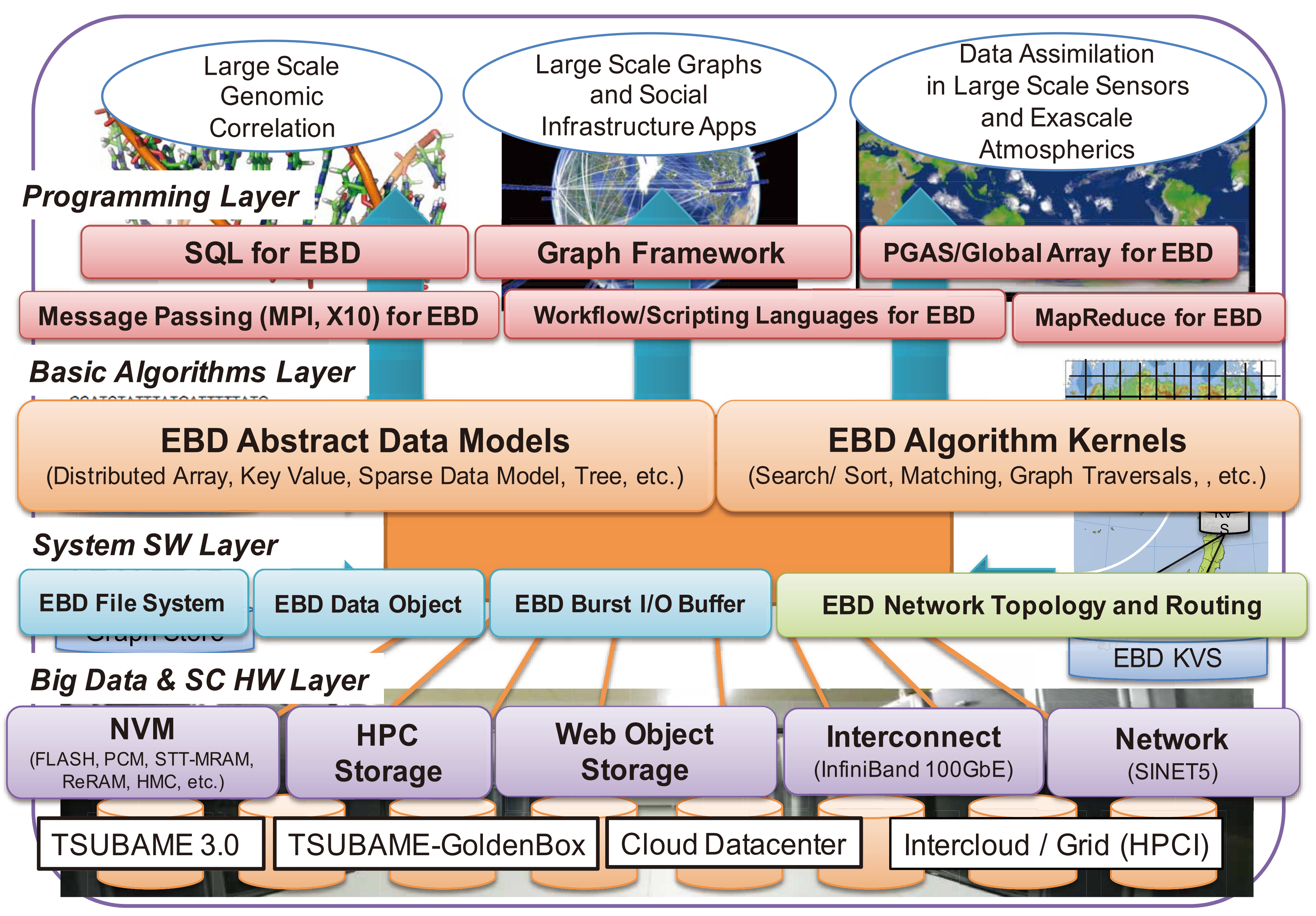
Next Generation Big Data Infrastructure Technologies Towards Yottabyte / Year

Extreme Big Data (EBD) Overview

Our project[1], called EBD, aims to achieve the convergence of extreme supercomputing and big data in order to cope with explosion of data from multiple sources such as massive numbers of sensors whose resolution is increasing exponentially, high resolution simulations generating huge data results, as well as evolution of social infrastructures that allow for "opening up of data silos", i.e., data sources being confined within an institution, much as how scientific data are being handled in the modern era as common asset openly accessible within and across disciplines. Our primary target proxy applications include metagenomics, social simulation, climate simulation with real-time data assimilation, and machine learning. Based on these EBD co-design applications, we define future EBD convergent SW/HW architecture and system. We have several on-going collaboration work with RIKEN AICS, ORNL, LLNL, ETH and JST Graph CREST / Univ. Kyushu.

This Research is Supported by JST, CREST

100,000 Times EBD "Convergent" System Architecture

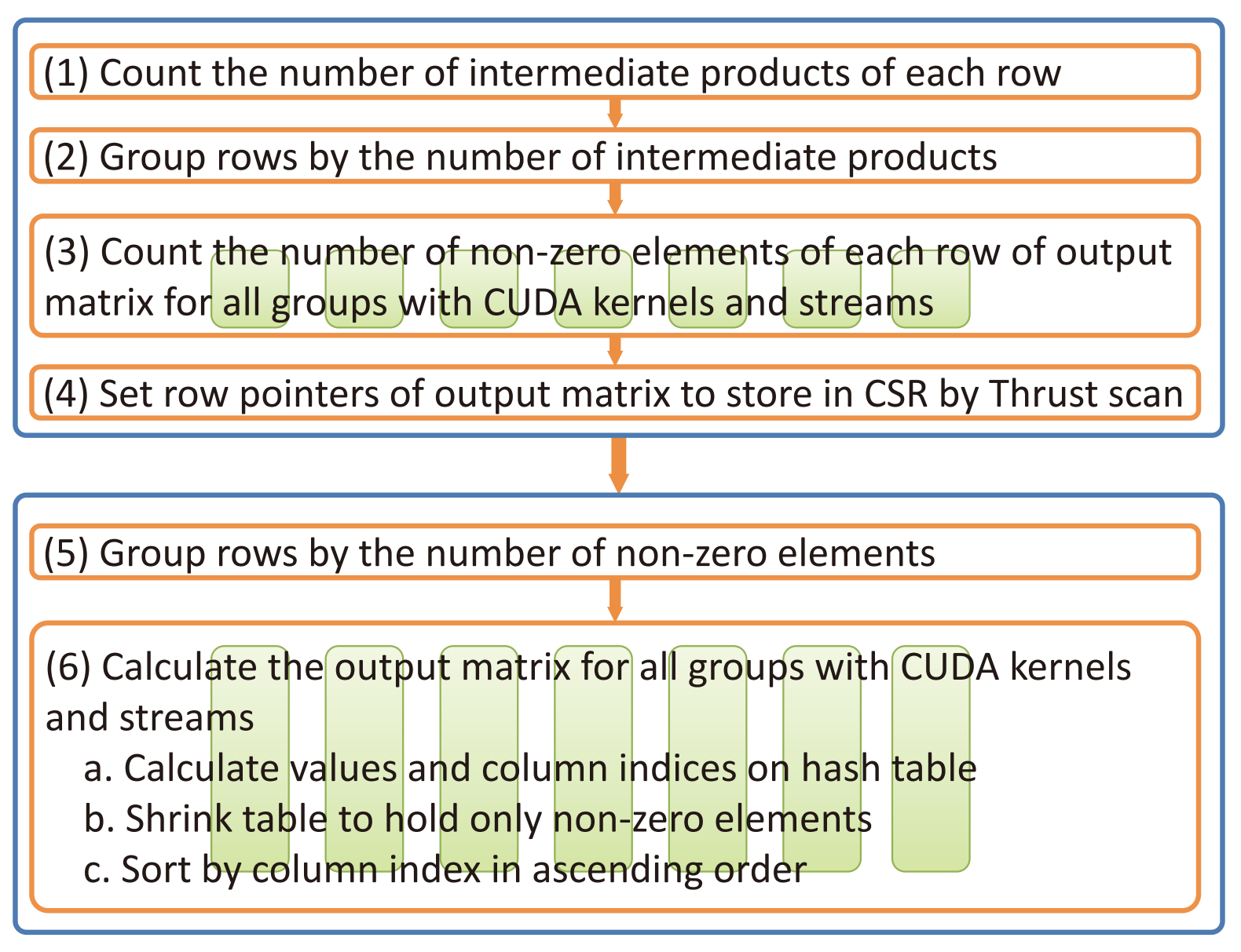

Large Scale Genomic Correlation
Large Scale Graphs and Social Infrastructure Apps
Data Assimilation in Large Scale Sensors and Exascale Atmospherics
Programming Layer
SQL for EBD
Graph Framework
PGAS/Global Array for EBD
Message Passing (MPI, X10) for EBD
Workflow/Scripting Languages for EBD
MapReduce for EBD
Basic Algorithms Layer
EBD Abstract Data Models
(Distributed Array, Key Value, Sparse Data Model, Tree, etc.)
EBD Algorithm Kernels
(Search/ Sort, Matching, Graph Traversals, , etc.)
System SW Layer
EBD File System
EBD Data Object
EBD Burst I/O Buffer
EBD Network Topology and Routing
EBD KVS
Big Data & SC HW Layer
NVM
(FLASH, PCM, STT-MRAM, ReRAM, HMC, etc.)
HPC Storage
Web Object Storage
Interconnect
(InfiniBand 100GbE)
Network
(SINET5)
TSUBAME 3.0
TSUBAME-GoldenBox
Cloud Datacenter
Intercloud / Grid (HPCI)

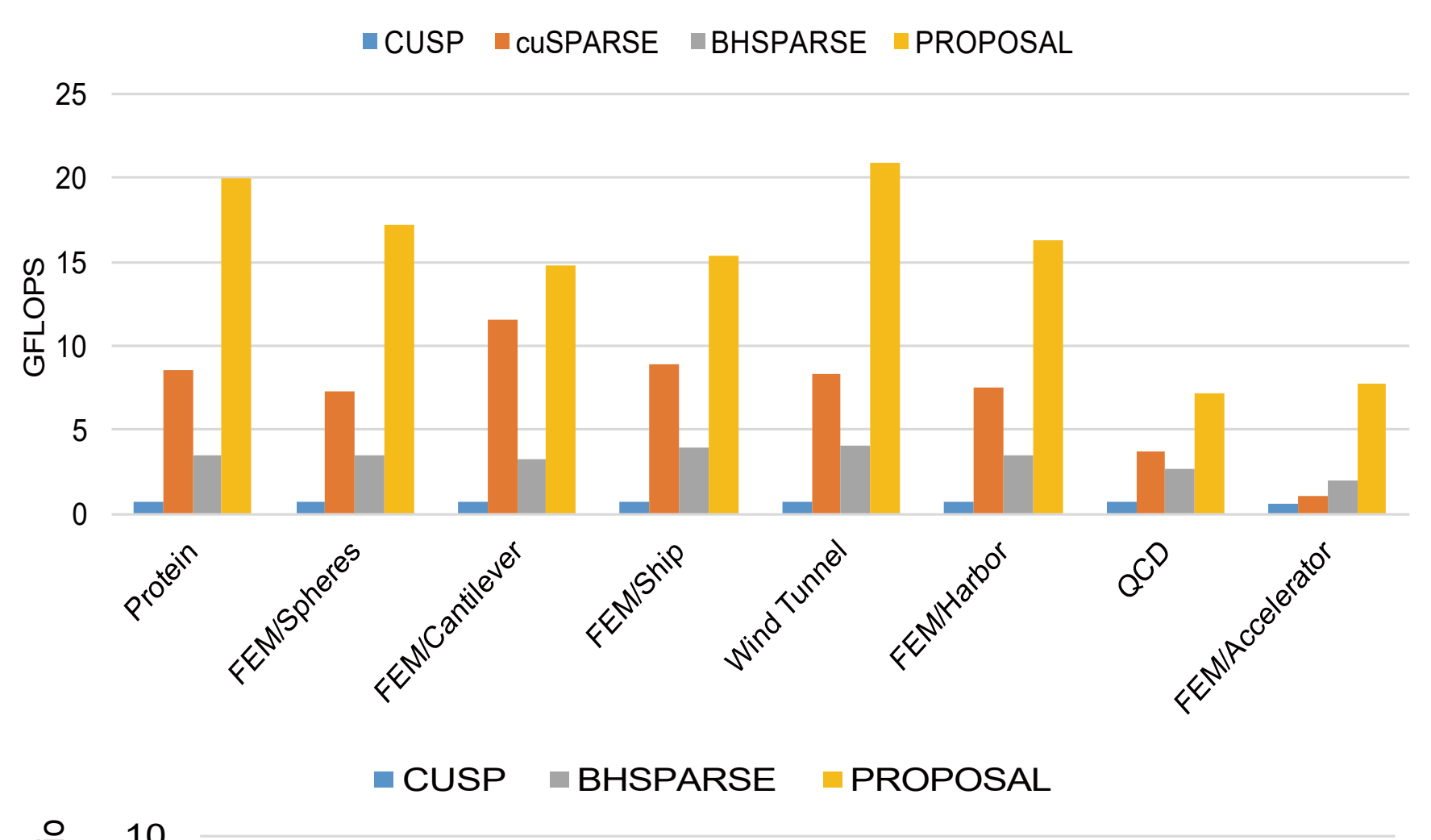
[1] S. Matsuoka, et al. "Extreme Big Data (EBD): Next generation big data infrastructure technologies towards yottabyte/year", Supercomputing frontiers and innovations, 2014.

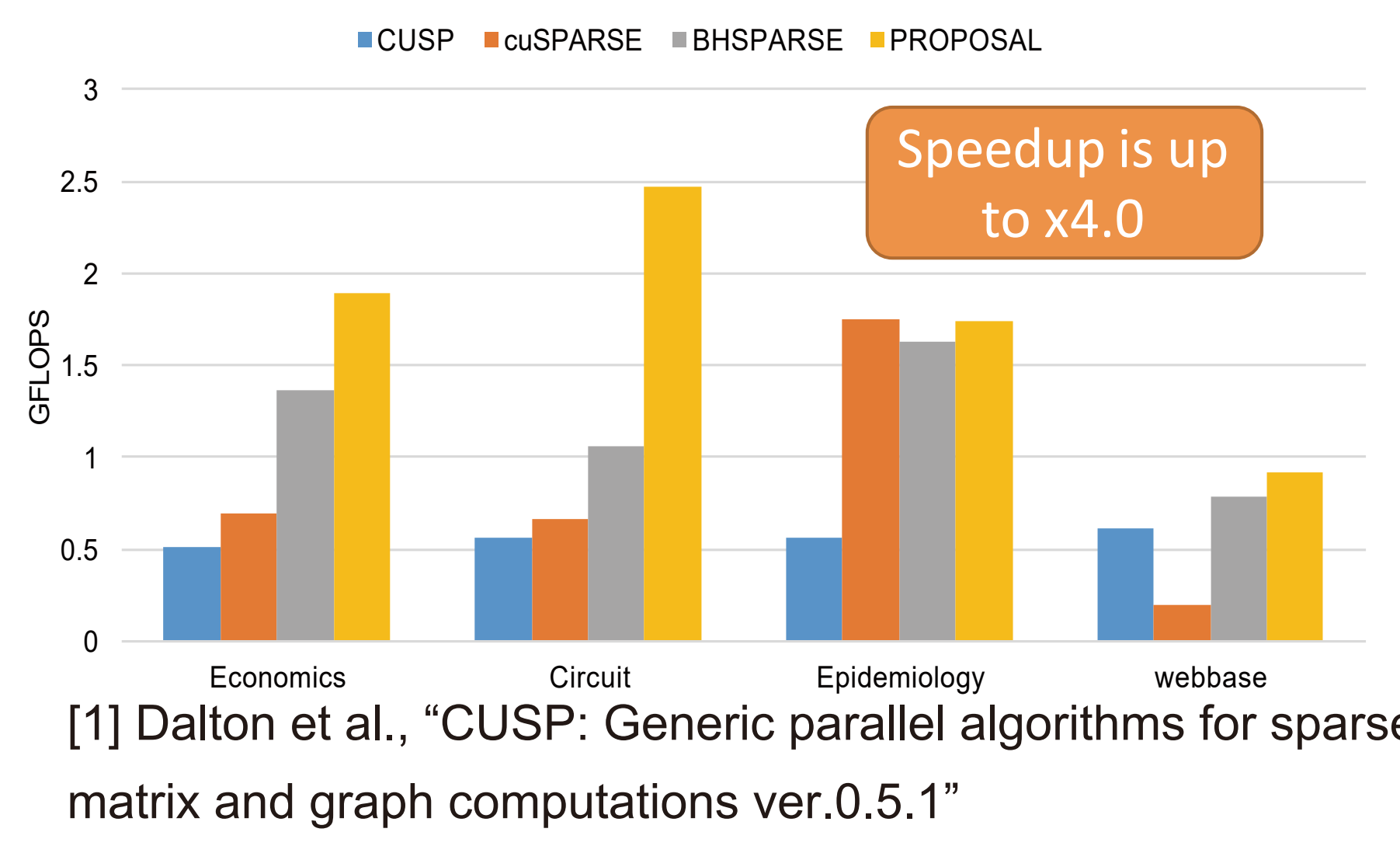
SpGEMM on GPU

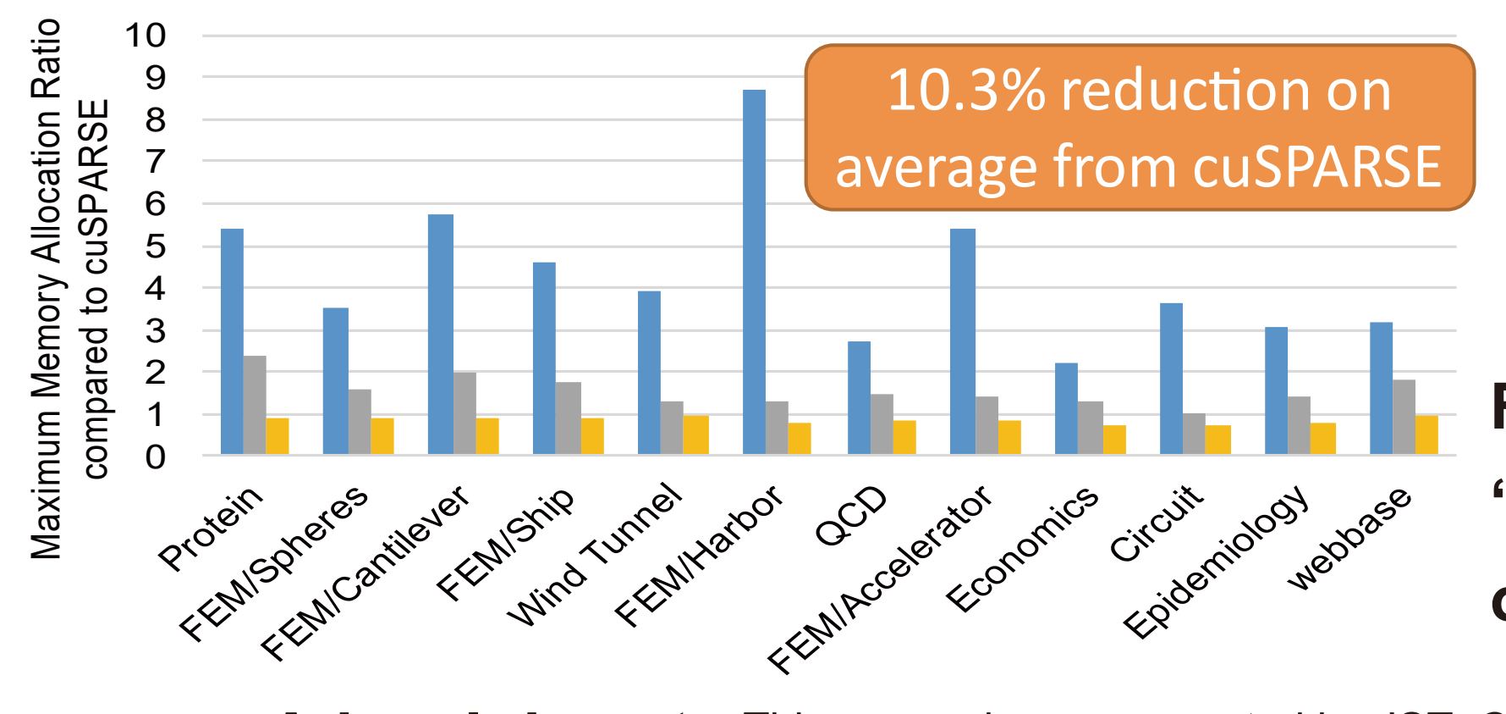
We have devised new Sparse General Matrix-Matrix Multiplication algorithm on GPU, which achieves further speedups and reduces memory usage so that various matrix data can be applied by utilizing GPU' s on-chip shared memory and appropriate assigning of GPU resources.

Two Phases Algorithm : 1st phase counts the number of non-zero elements of output matrix, and 2nd phase calculates the output matrix
→ Reduce memory usage
Grouping rows (2, 4)
→ Better utilization of GPU resources
Two ways threads assignments
→ Improve the load-balance
Hash table on fast shared memory
→ Accelerate (3)counting part and (6)calculation part


(1) Count the number of intermediate products of each row
(2) Group rows by the number of intermediate products
(3) Count the number of non-zero elements of each row of output matrix for all groups with CUDA kernels and streams
(4) Set row pointers of output matrix to store in CSR by Thrust scan
(5) Group rows by the number of non-zero elements
(6) Calculate the output matrix for all groups with CUDA kernels and streams
a. Calculate values and column indices on hash table
b. Shrink table to hold only non-zero elements
c. Sort by column index in ascending order







[1] Dalton et al., "CUSP: Generic parallel algorithms for sparse matrix and graph computations ver.0.5.1"
[2] NVIDIA, "Nvidia cuda sparse matrix library (cuSPARSE)"
[3] Liu et al., "An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data" , IPDPS2014

For more details, please visit SC Regular Poster "Fast Sparse General Matrix-Matrix Multiplication on GPU with Low Memory Usage"

Acknowledgments. This research was supported by JST, CREST (Research Area: Advanced Core Technologies for Big Data Integration).

Increasing GPU Occupancy

Multi-GPU batch-queue systems usually experience large number of idle GPUs due to the scattered idle-GPU problem (Fig.1). We addressed this problem by allowing jobs to utilize remote GPUs and migrating execution on a remote GPU back to a local GPU as soon as one becomes available. This method enables the systems to serve more GPU jobs concurrently while minimizing execution time penalty caused by remote GPU communication.

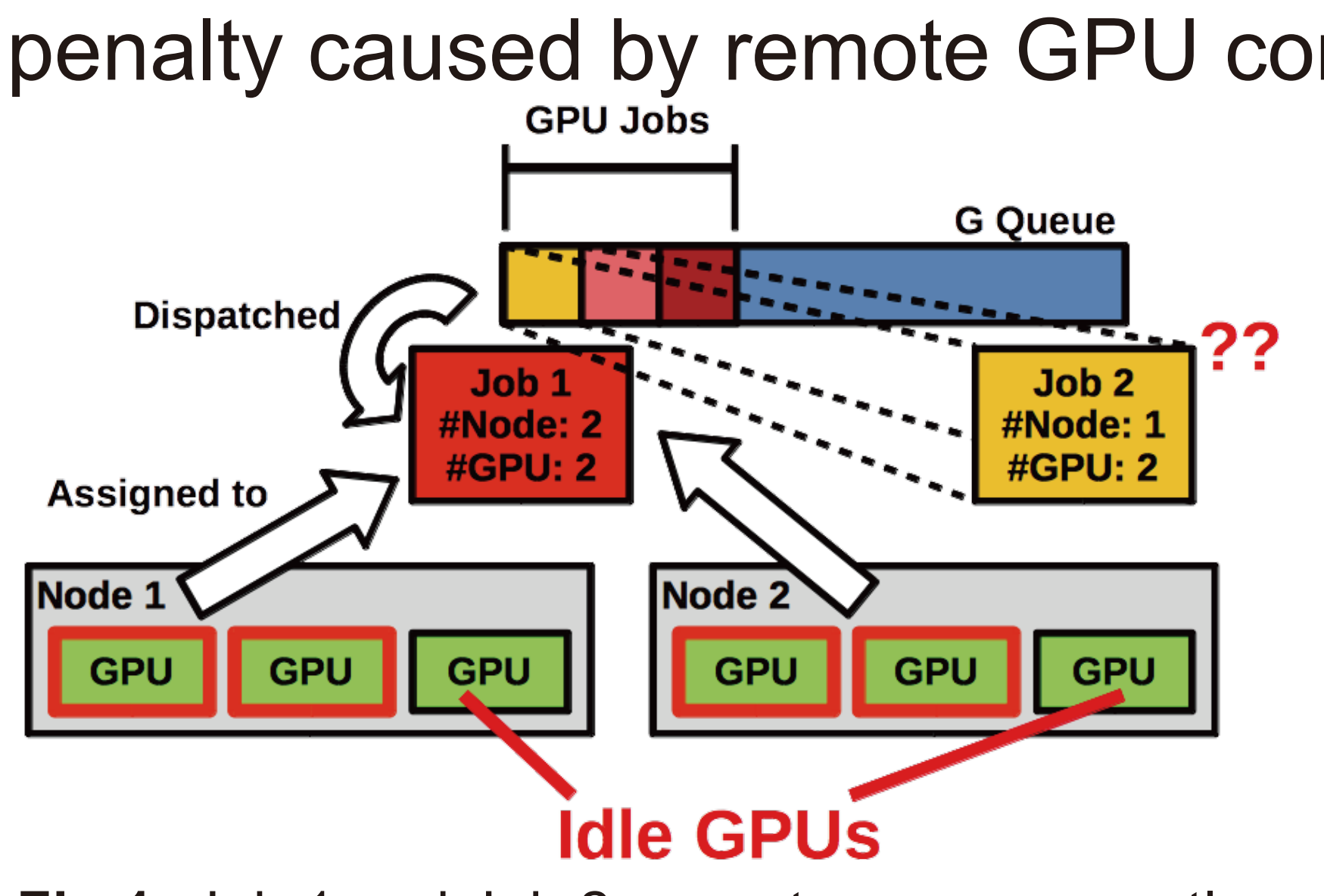


Fig.1: Job 1 and Job 2 cannot run concurrently as Job 2 wants two unoccupied GPUs on the same node.

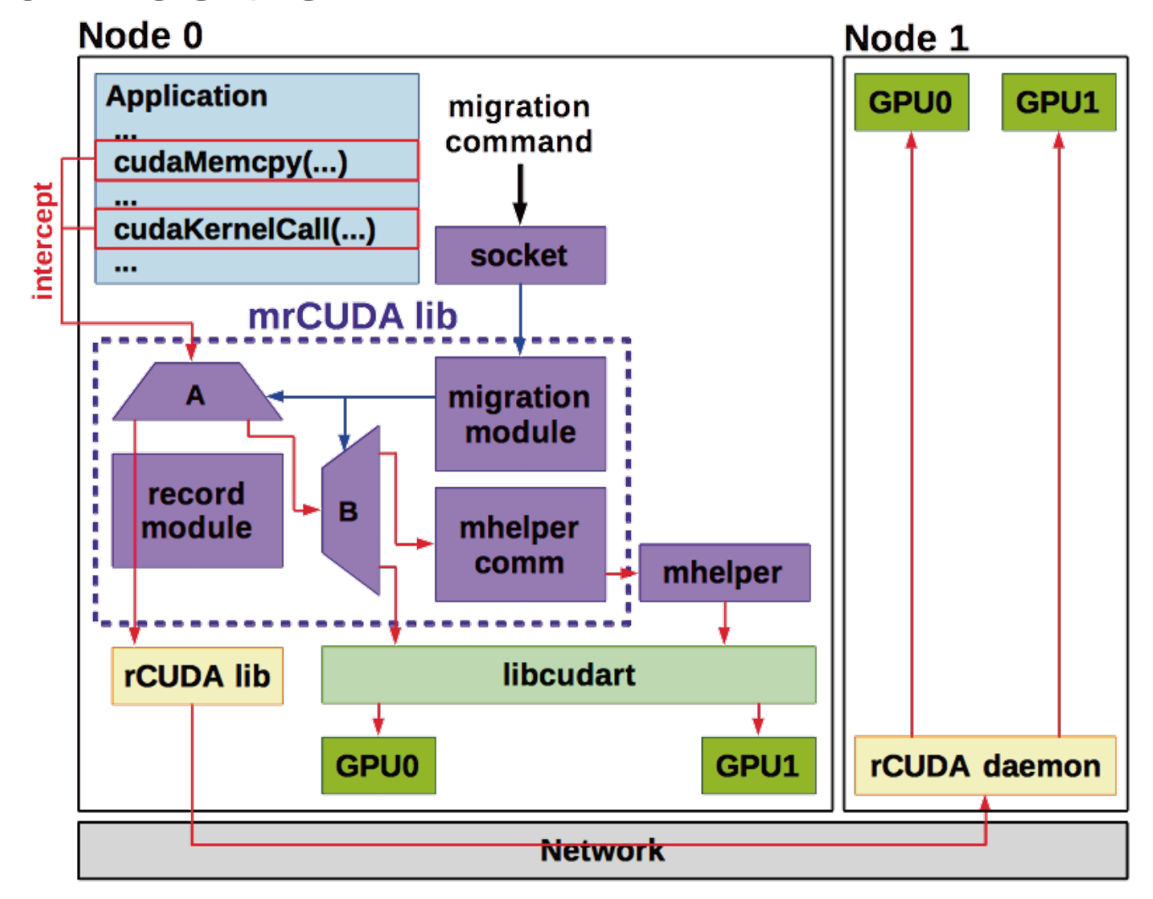


Fig.2: The architecture of mrCUDA, our middleware for handling remote GPU migration on top of rCUDA.

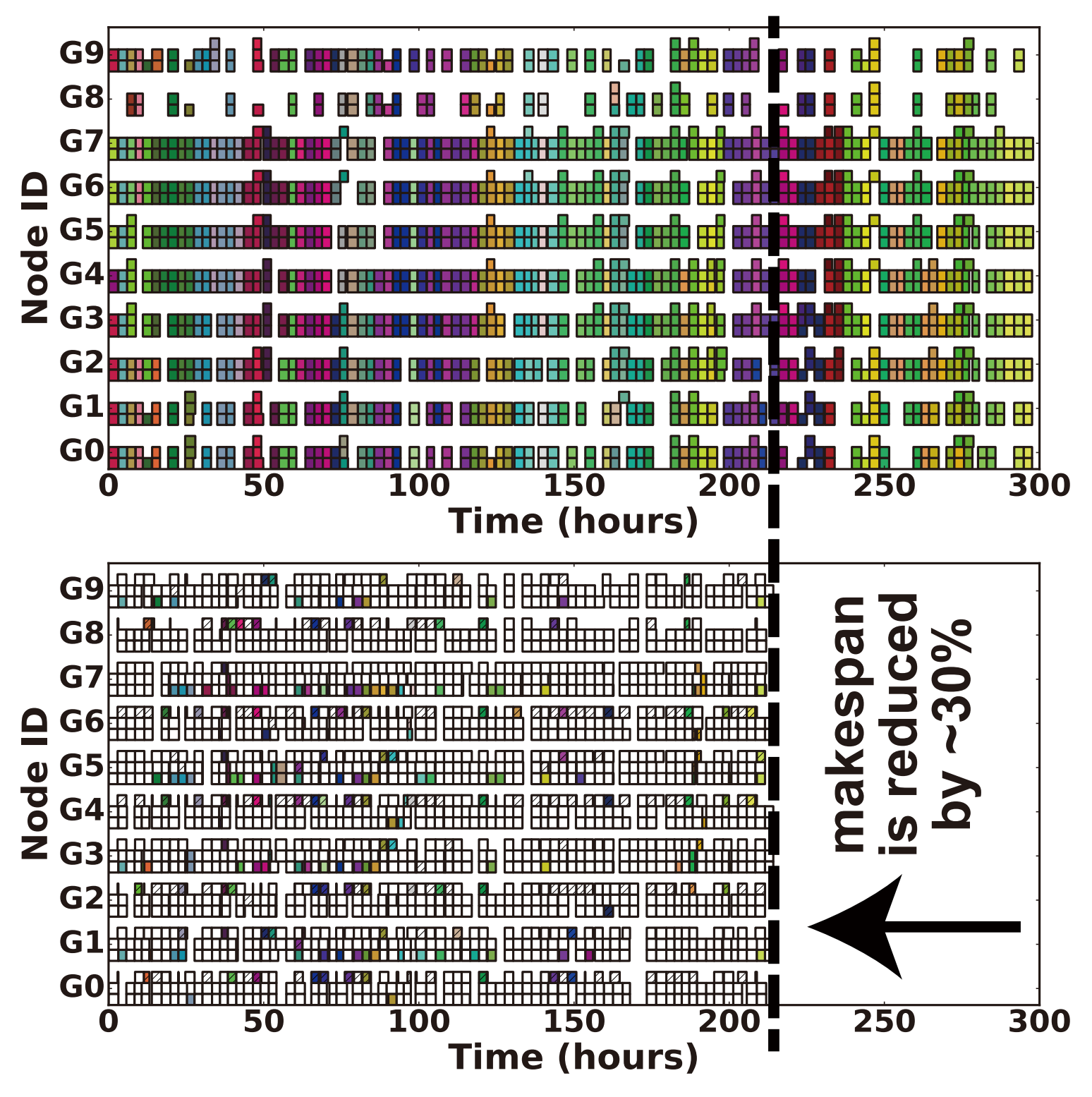


Fig.3: GPU occupancy patternswhen using FCFS (top) and our method (bottom).

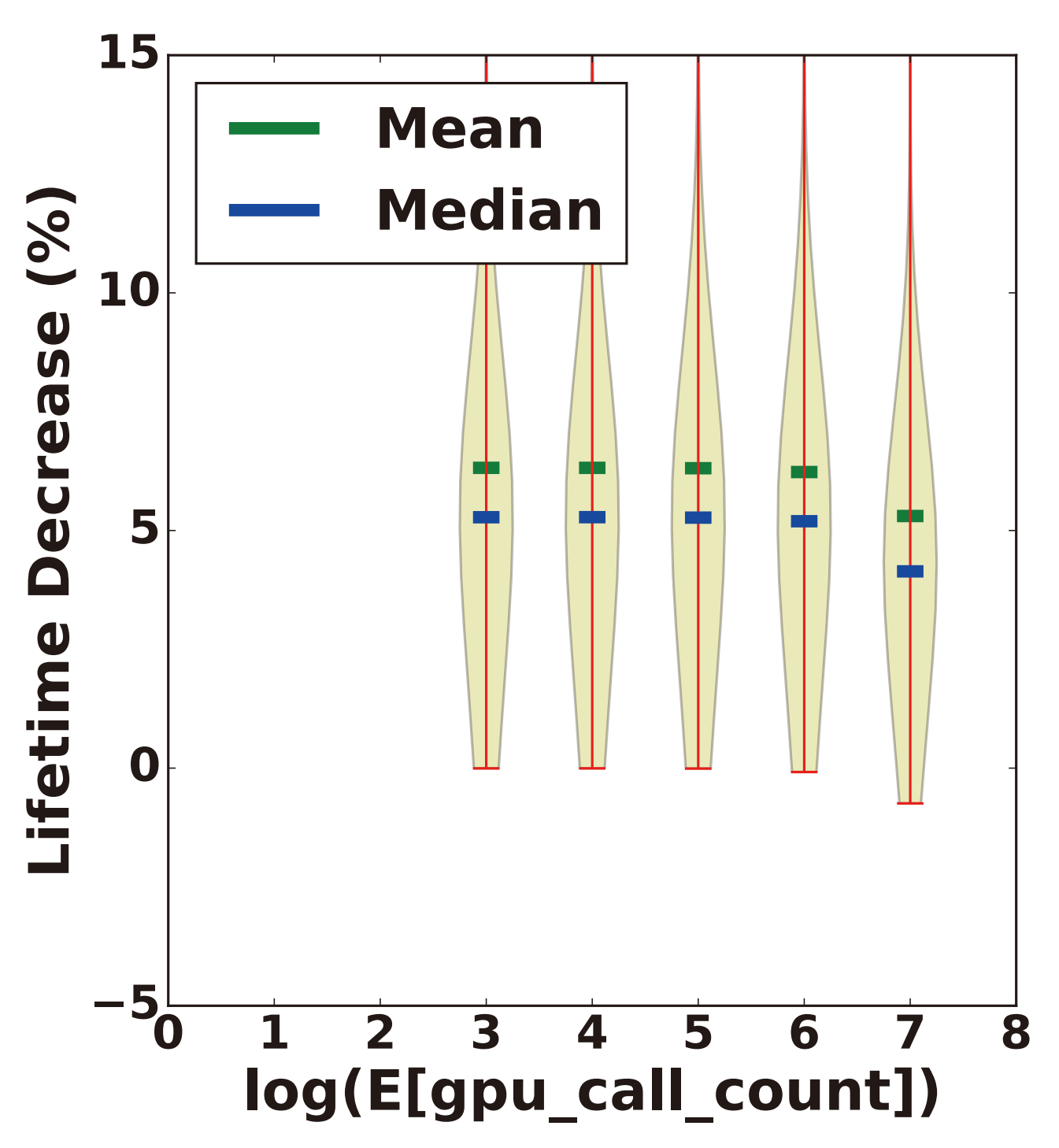


Fig.4: Distribution of jobs' lifetime (waiting + execution time) decrease when using our method compared with FCFS.

Reference: P.Markthub, A.Nomura, and S.Matsuoka, "Serving More GPU Jobs, with Low Penalty, using Remote GPU Execution and Migration," IEEE Cluster 2016.

Acknowledgments. This research was supported by JST, CREST (Research Area: Advanced Core Technologies for Big Data Integration).