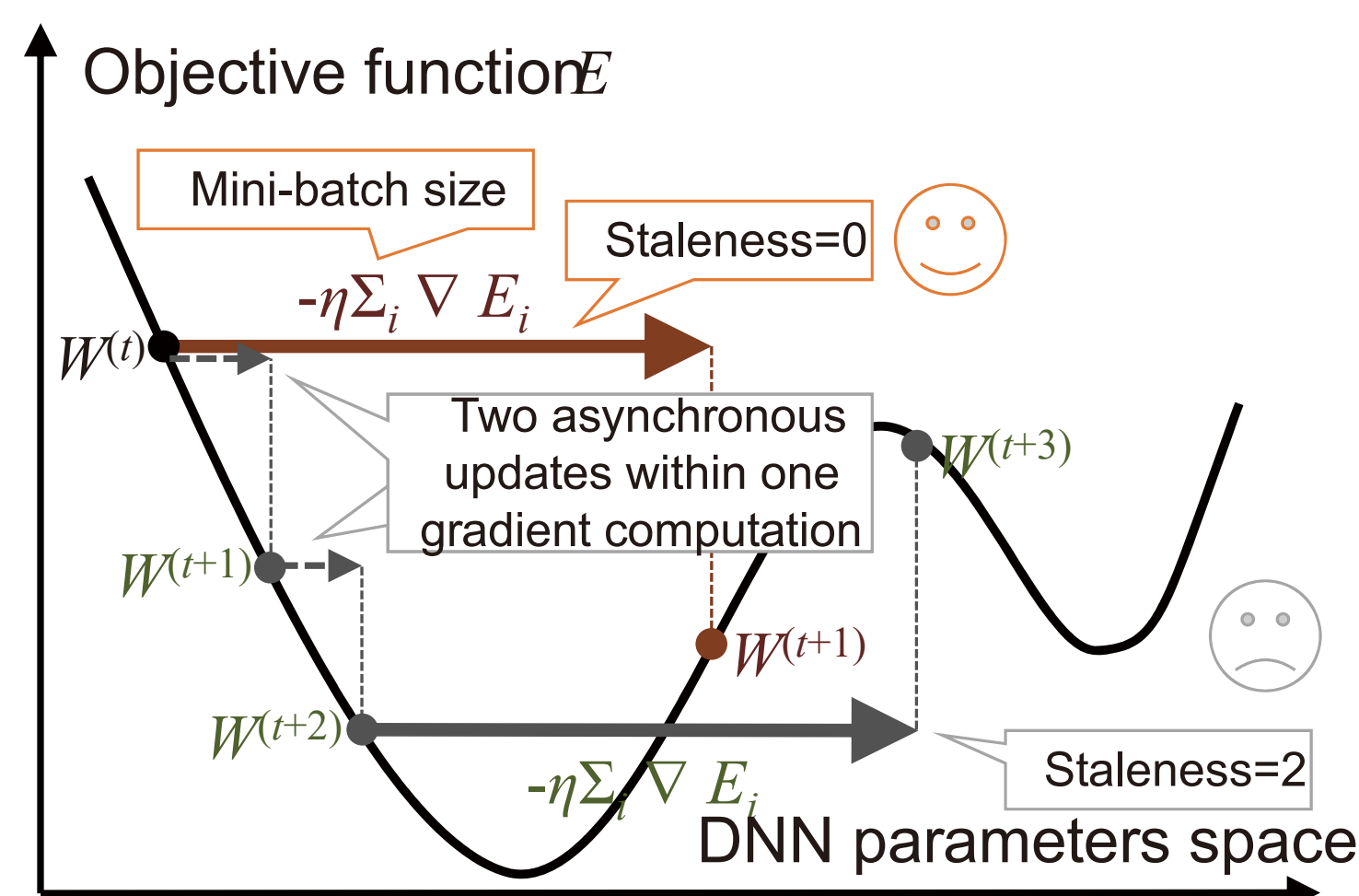# Extreme Big Data and Deep Learning Algorithm Platform
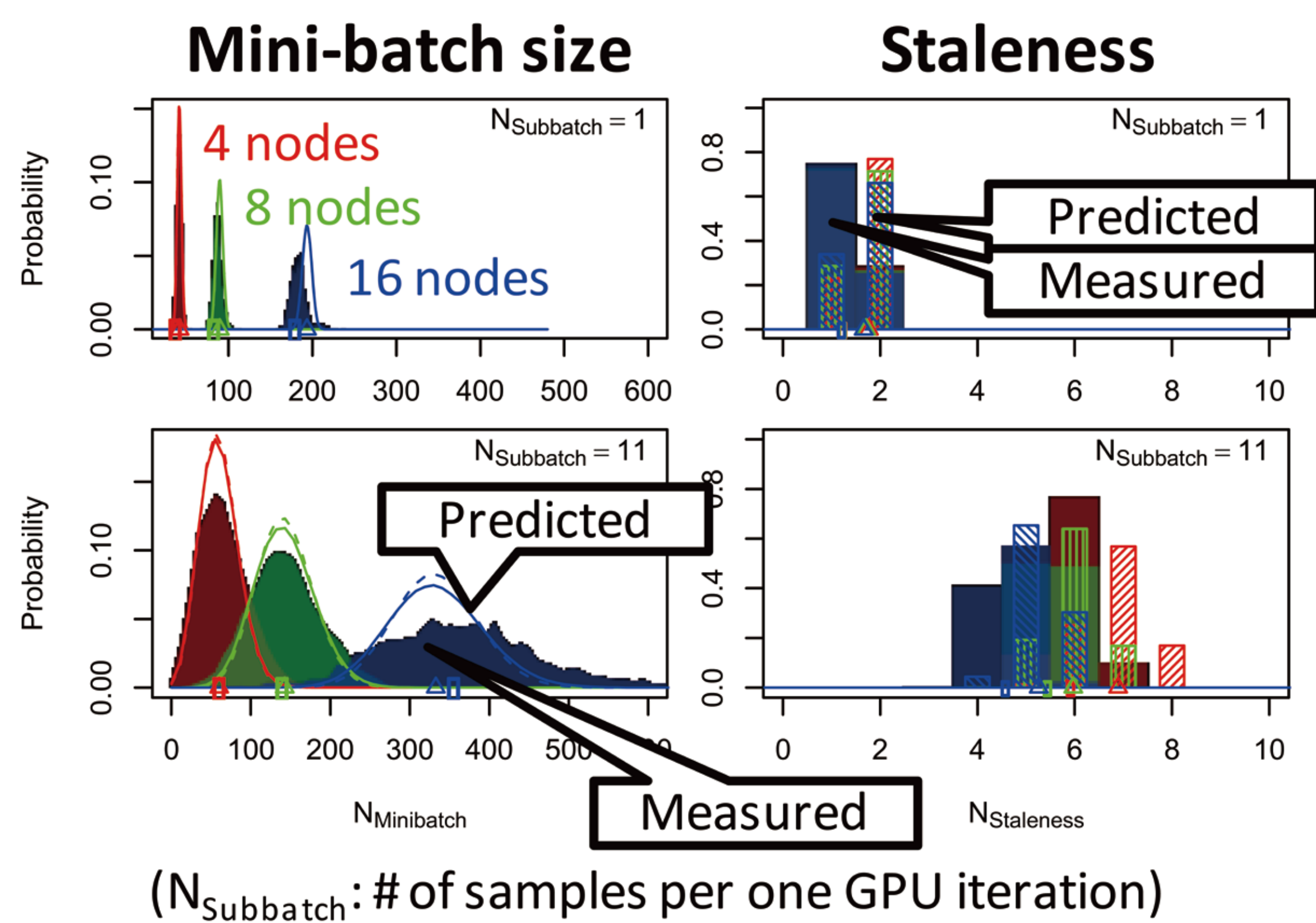
Predicting Statistics of ASGD

Collaborative work with DENSO CORPORATION and DENSO IT LABORATORY, INC

In large-scale Asynchronous Stochastic Gradient Descent (ASGD), mini-batch size and gradient staleness tend to be large and unpredictable, which increase the error of trained DNN

Objective function E
Mini-batch size
Staleness=0
$W(t)$
$-\eta\Sigma_i \nabla E_i$
Two asynchronous updates within one gradient computation
$W^{(t+3)}$
$W^{(t+1)}$
$W^{(t+1)}$
$W^{(t+2)}$
$-\eta\Sigma \nabla E_i$
Staleness=2
DNN parameters space

We propose a empirical performance model for an ASGD deep learning system SPRINT which considers probability distribution of mini-batch size and staleness

Mini-batch size

- 4 nodes
- 8 nodes
- 16 nodes

$N_{Subbatch} = 1$
Predicted
$N_{Subbatch} = 11$

Staleness

$N_{Subbatch} = 1$
Predicted
Measured
$N_{Subbatch} = 11$

Predicted
Measured

$N_{Minibatch}$
$N_{Staleness}$

($N_{Subbatch}$: # of samples per one GPU iteration)

Reference: Yosuke Oyama, Akihiro Nomura, Ikuro Sato, Hiroki Nishimura, Yukimasa Tamatsu, and Satoshi Matsuoka, "Predicting Statistics of Asynchronous SGD Parameters for a Large-Scale Distributed Deep Learning System on GPU Supercomputers", IEEE BigData 2016

Evaluating Apps on FPGAs

Nowadays, FPGA can rival CPU/GPU performance and energy efficiency, but also known for its hardness for programming. We compared three high-level programming approaches for FPGAs
- 30-core many-core system (reps. for programmability)
- LegUp High-Level Synthesis (reps for multiple custom accelerators)
- Intel OpenCL for FPGA (reps for Deep-pipeline designs)
We evaluated using Rodinia Benchmark Suite on Stratix V FPGA. We improved memory hierarchy for many-core and multi-accelerator designs through cache multi-banking.

(c) Needleman-Wunsch

(e) LUD

legup-base   legup-imp   niosII-mp   ocl-base   ocl-opt

- Intel OpenCL for FPGA shown highest average performance
- LegUp can remain competitive for good performance and spatial/temporal locality, even without improvement.
- Many-core system offers good programmability, but often does not perform well compared to other approaches

Reference: "Evaluating High-Level Design Strategies on FPGAs for High-Performance Computing", A. Podobas, H.R. Zohouri, N. Maruyama, S. Matsuoka, IEEE FPL 2017

Fast 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 (1, 2, 6)
→ Better utilization of GPU resources
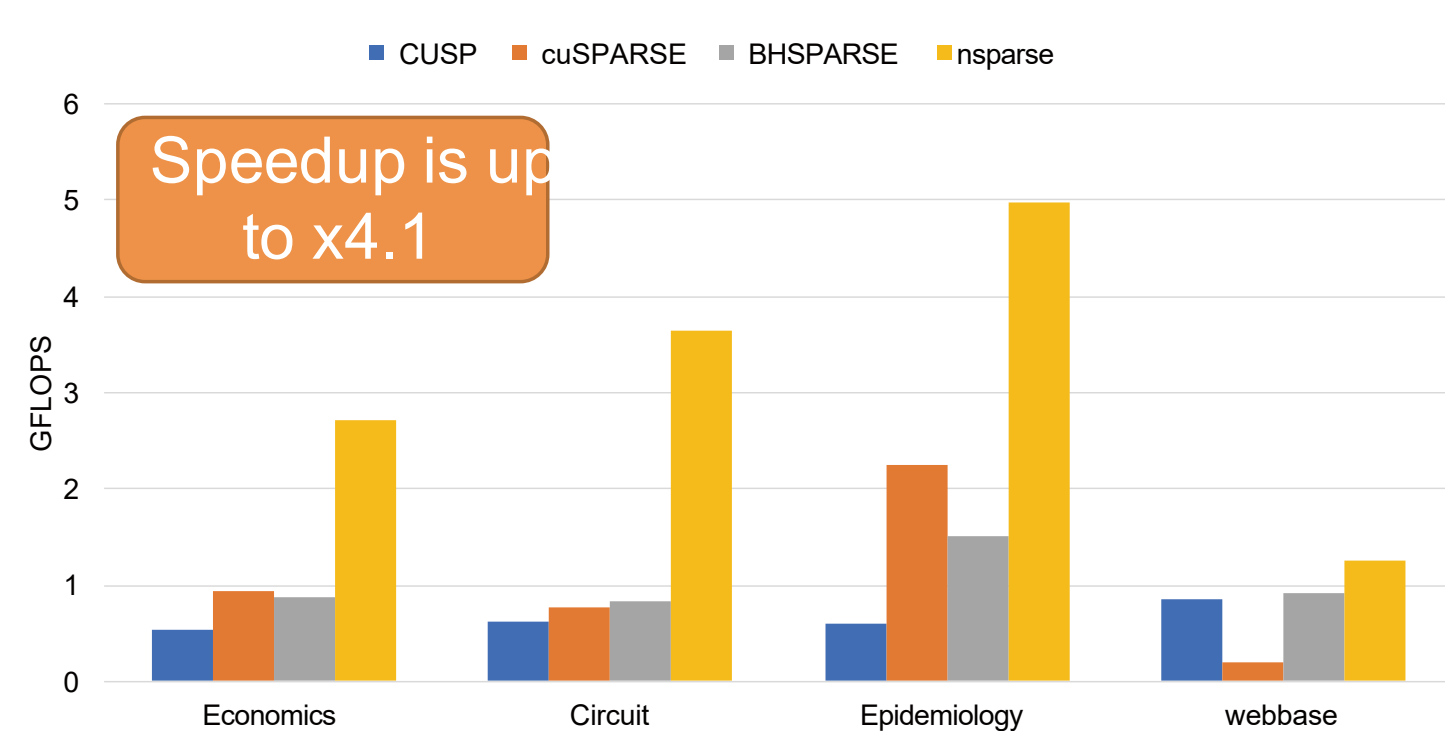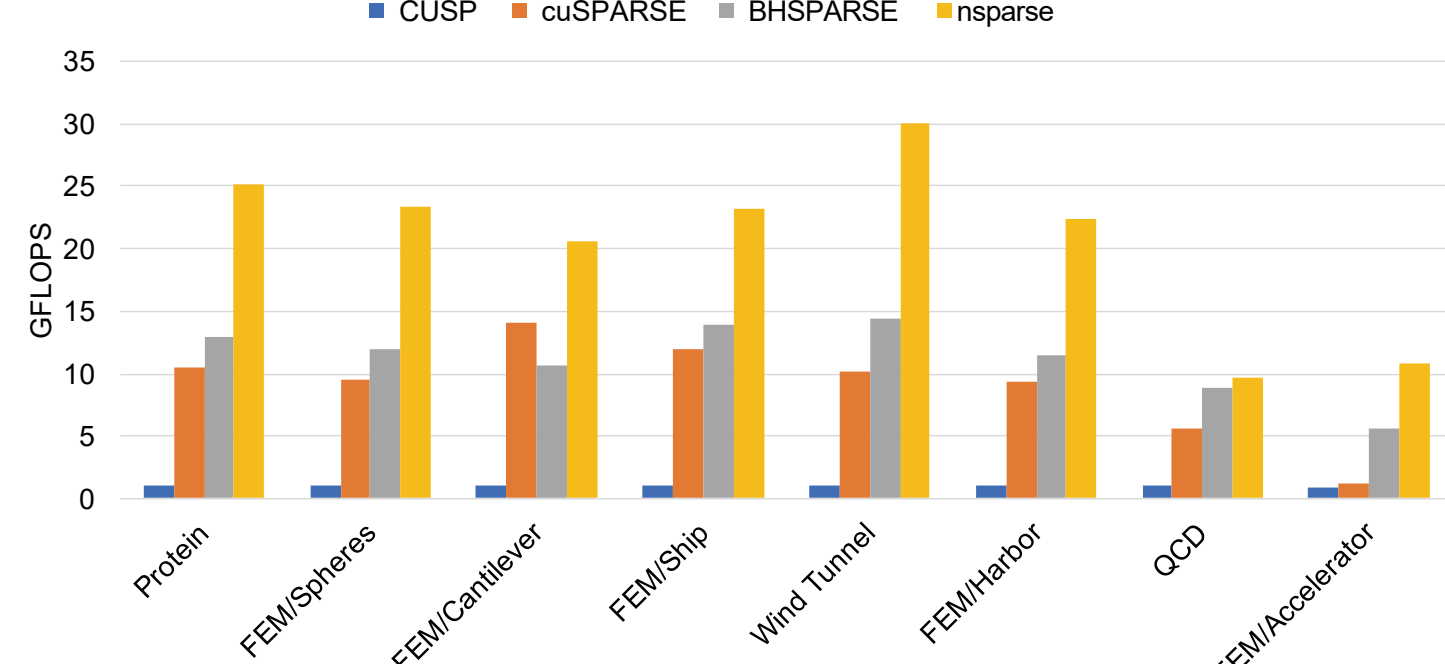**Two ways threads assignments**
→ Improve the load-balance
Hash table on fast shared memory
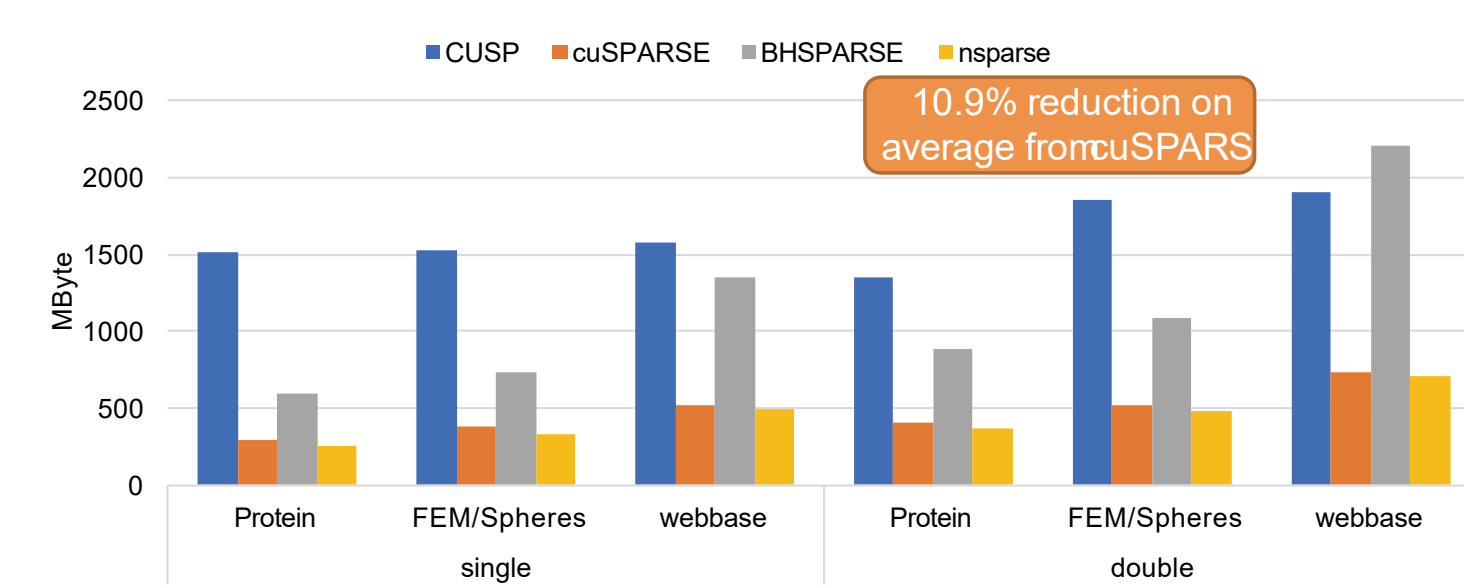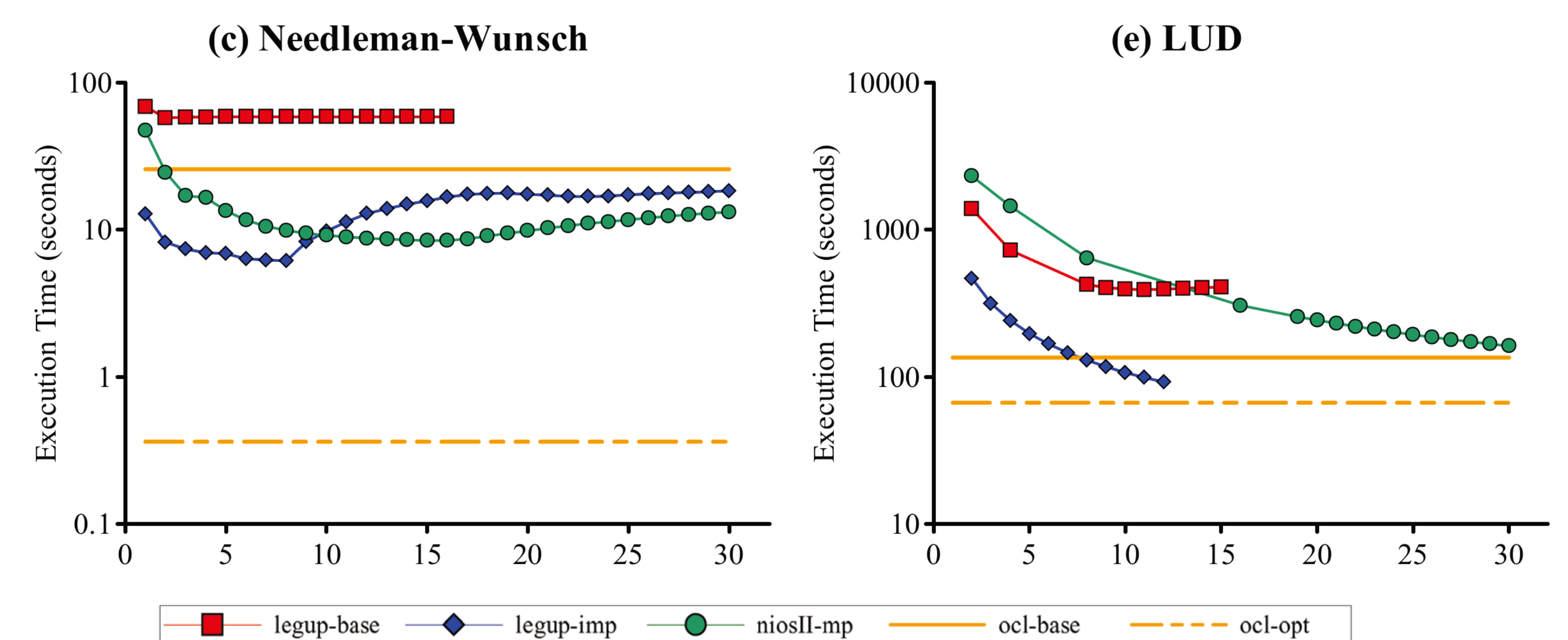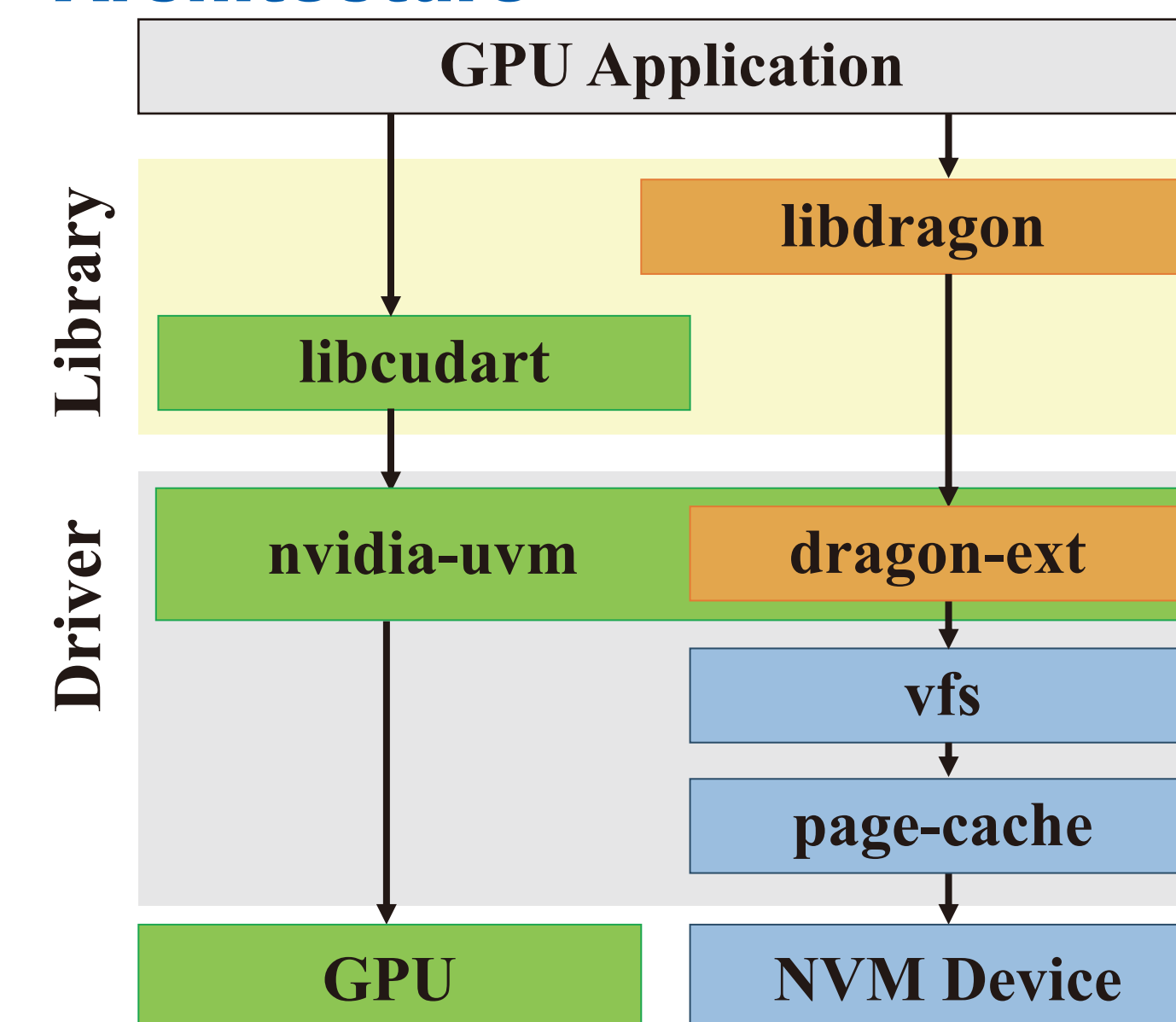→ Accelerate counting part (3) and calculation part (7)

(1) Count #intermediate products
(2) Divide the rows into groups by #intermediate products
(3) Count #nonzero elements
(4) Set row pointers of output matrix
(5) Memory allocation of output matrix
(6) Divide the rows into groups by #non-zero elements
(7) Compute the output matrix
  a. Calculate values and column indices on hash table
  b. Shrink the hash table
  c. Store to the memory with sorting

Double Precision Performance

CUSP   cuSPARSE   BHSPARSE   nsparse

Protein, FEM/Spheres, FEM/Cantilever, FEM/Ship, Wind Tunnel, FEM/Harbor, QCD, FEM/Accelerator

Speedup is up to x4.1

Economics, Circuit, Epidemiology, webbase

Memory Usage

CUSP   cuSPARSE   BHSPARSE   nsparse

10.9% reduction on average from cuSPARS

Protein, FEM/Spheres, webbase (single)
Protein, FEM/Spheres, webbase (double)

[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

Code available at:
https://github.com/EBD-CREST/nsparse

Reference: Yusuke Nagasaka, Akira Nukada, Satoshi Matsuoka, "High-performance and Memory-saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU", ICPP 2017.

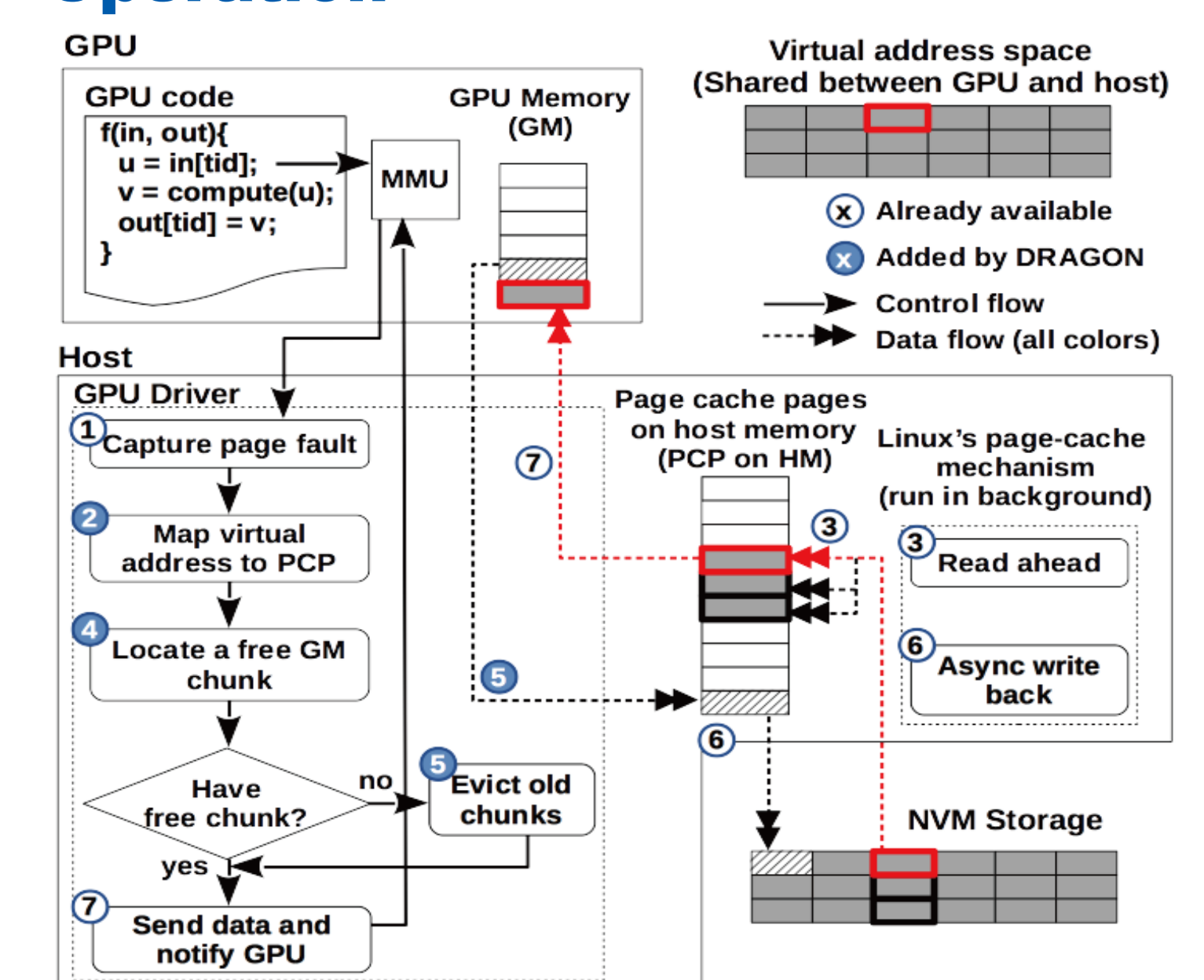## Unlimited GPU Memory with DRAGON

Collaborative work with Oak Ridge National Laboratory

Problem sizes grow larger than GPU and even host memory capacity. We proposed DRAGON, a framework that seamlessly enables all classes of GPU applications to directly access terabytes of data on storage with user-oblivious addressing. DRAGON leverages GPU hardware page-faulting mechanism and two-level prefetching with direct page-cache access to achieve this functionality while bestowing up to 2.3x speedup compared with using Unified Memory + POSIX IO.
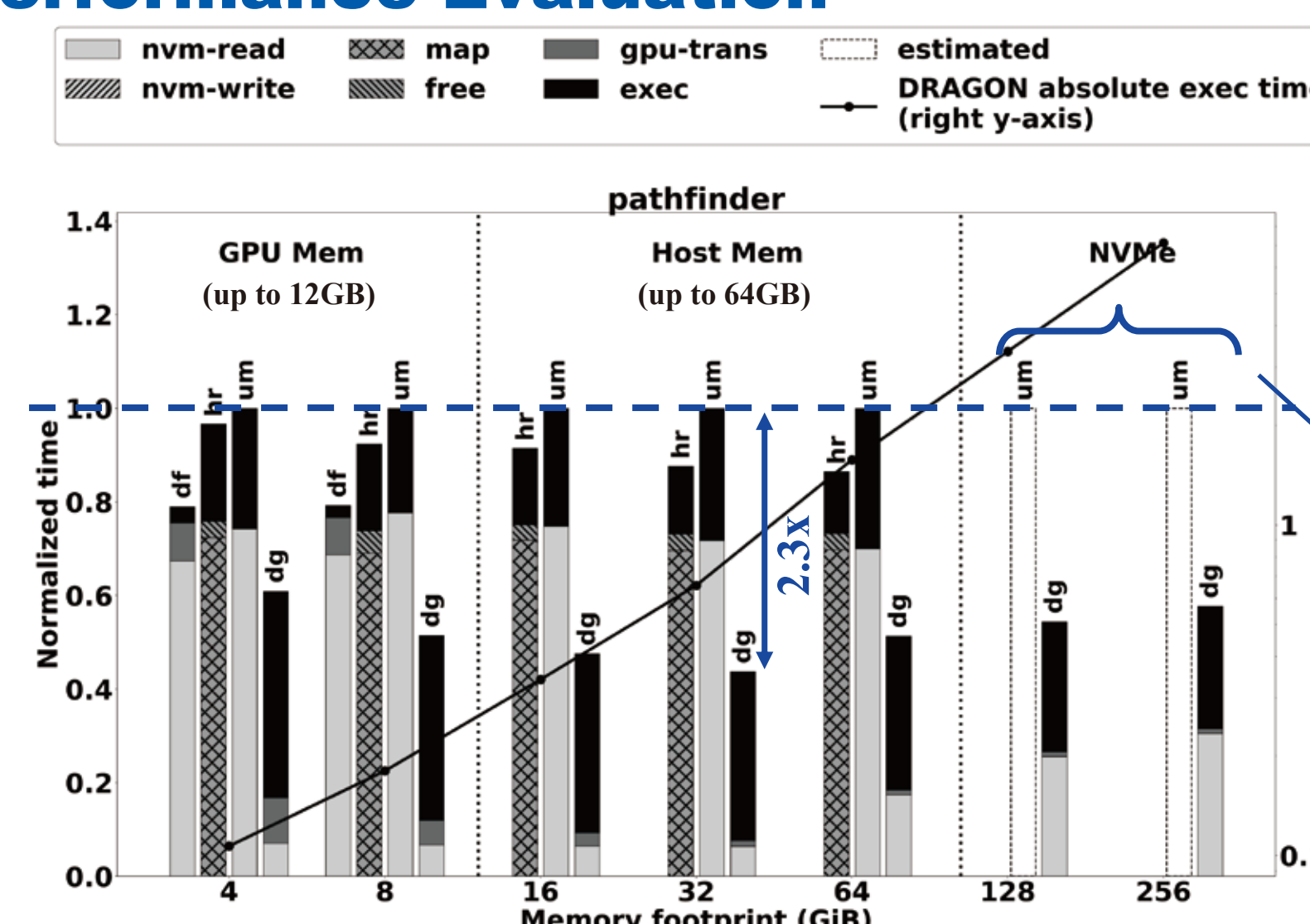
Architecture

GPU Application
libdragon
libcudart
nvidia-uvm   dragon-ext
vfs
page-cache
GPU   NVM Device

Library / Driver

Operation

GPU
GPU code
f(in, out){
  u = in[tid];
  v = compute(u);
  out[tid] = v;
}
GPU Memory (GM)
MMU

Virtual address space (Shared between GPU and host)
⊗ Already available
⊕ Added by DRAGON
Control flow
Data flow (all colors)

Host
GPU Driver
1 Capture page fault
2 Map virtual address to PCP
4 Locate a free GM chunk
Have free chunk?
no → Evict old chunks
yes
7 Send data and notify GPU
Page cache pages on host memory (PCP on HM)
Linux's page-cache mechanism (run in background)
Read ahead
Async write back
NVM Storage

Performance Evaluation

nvm-read   nvm-write   map   free   gpu-trans   exec   estimated   DRAGON absolute exec time (right y-axis)

pathfinder
GPU Mem (up to 12GB)
Host Mem (up to 64GB)
NVMe

Normalized time
Execution time (mins)
Memory footprint (GiB)
4, 8, 16, 32, 64, 128, 256
2.3x

df, hr, um, dg

Data transfer methods

df	cudaMemcpy + IO
hr	cudaHostRegister + mmap
um	Unified Memory + IO
dg	DRAGON

Out-of-core was faster than the extrapolated baseline (um)

DRAGON is the only existing out-of-core solution with user-oblivious addressing

Presentation at SC18 Technical Program:
11:00-11:30, Wednesday, Room C141/143/149

Reference: Pak Markthub, Mehmet E. Belviranli, Seyong Lee, Jeffrey S. Vetter, and Satoshi Matsuoka, "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access", SC18