cuSZ: An Efficient GPU Based Error-Bounded Lossy Compression Framework for Scientific Data

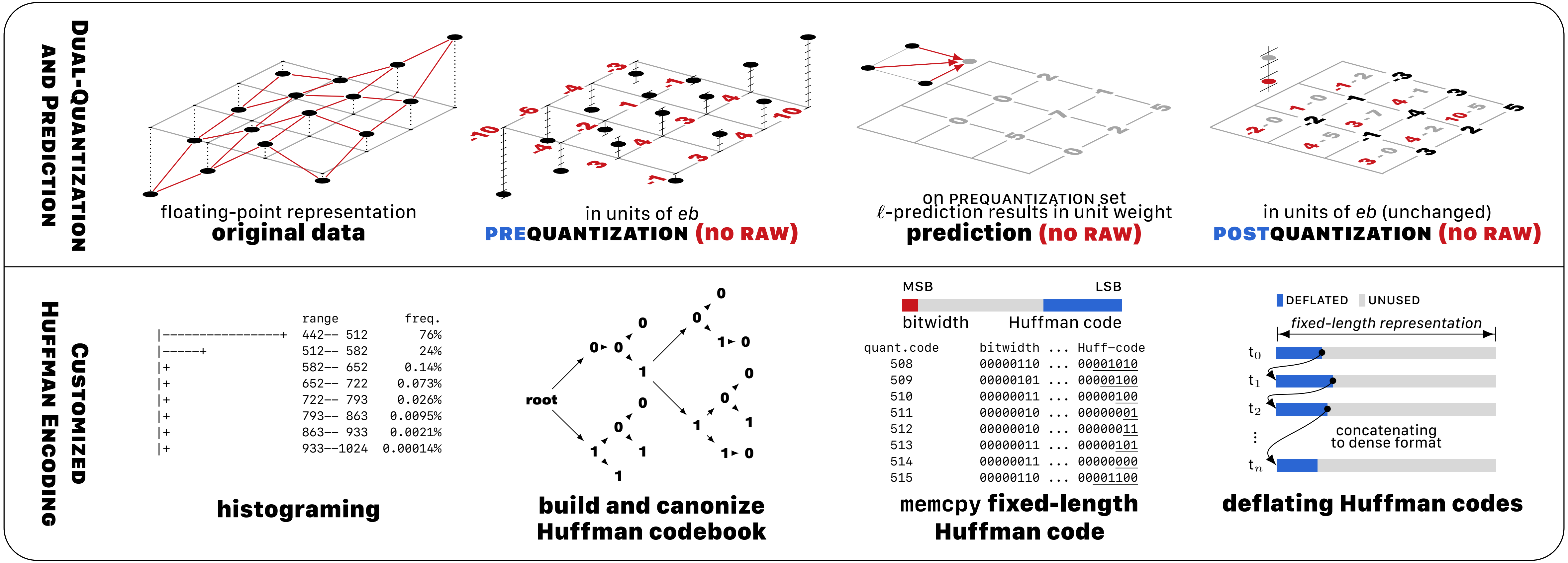
Published in *2020 International Conference on Parallel Architectures and Compilation Techniques (PACT'20)*

Led by **Jiannan Tian** from HiPDAC

System Design

Challenges

- Tight data dependency—loop-carried *read-after-write* (RAW)—hinders parallelization.
- Host-device communications due to only considering CPU/GPU suitableness.



GPU Performance Optimization

Canonical Codebook & Huffman Encoding

ca·non·i·cal *adj.*

A canonical encoding is then generated in which the numerical values of the codes are monotone increasing and each code has the smallest possible numerical value consistent with the requirement that the code is not the prefix of any other code.

[Schwartz and Kallick 1964]

- codebook transformed to a compact manner
- no tree in decoding
- tree build time: 4–7 ms
  update: 0.8 ms
- canonize for 200 us (1024 symbols)
  update: incoporated in tree-building
- Encoding/decoding is done in a coarse-grained manner.
- A GPU thread is assigned to a data chunk.
- Tune degree of parallelism to keep every thread busy.

fine-grained manner:

IPDPS'21: *Revisiting Huffman Coding: Toward Extreme Performance on Modern GPU Architectures*, Tian et al.
IPDPS'22: *Optimizing Huffman Decoding for Error-Bounded Lossy Compression on GPUs*, Rivera et al.

	sequential	coarse-grained	fine-grained	atomic
compression				
dual-quantization			•	
histogram			•	•
build Huffman tree	•			
canonize codebook	•		•	•
Huffman encode (fix-length)			•	
deflate (fix- to variable-length)		•		
decompression				
inflate (Huffman decode)		•		
reversed dual-quantization		•		

Table 2: Parallelism used for cuSZ's subprocedures (kernels) in compression and decompression.

Adaptive Parallelism

Worth noting: in canonizing codebook

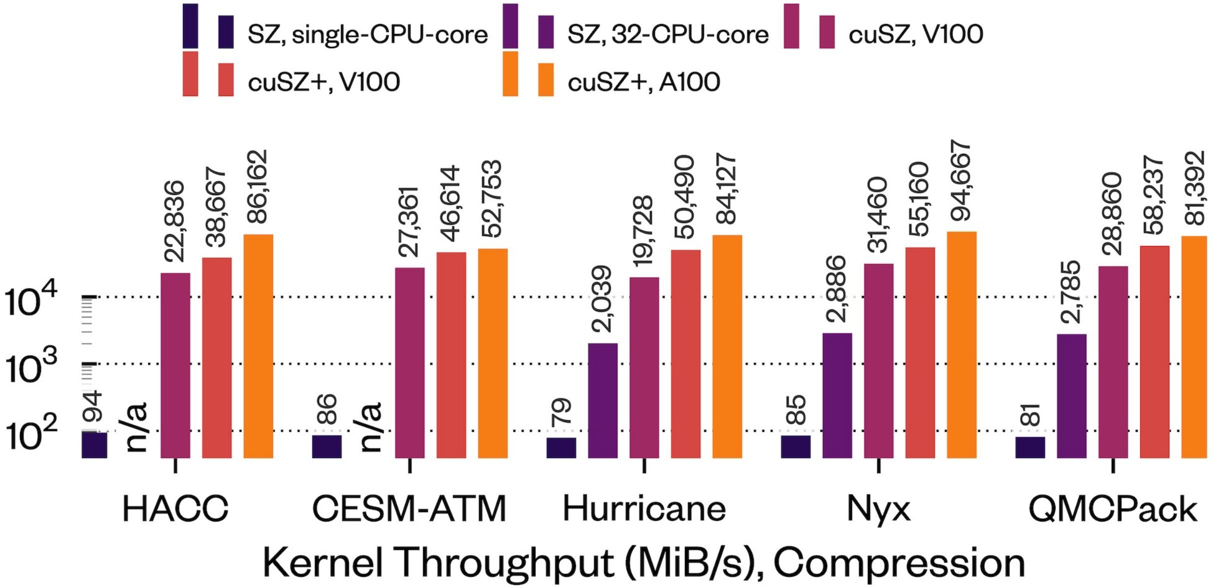
- problem size > max. `block` size (1024)
- utilize `cooperative groups` and `grid.sync()`
- `__syncthreads()`: not able
- `cudaDeviceSynchronize()`: expensive

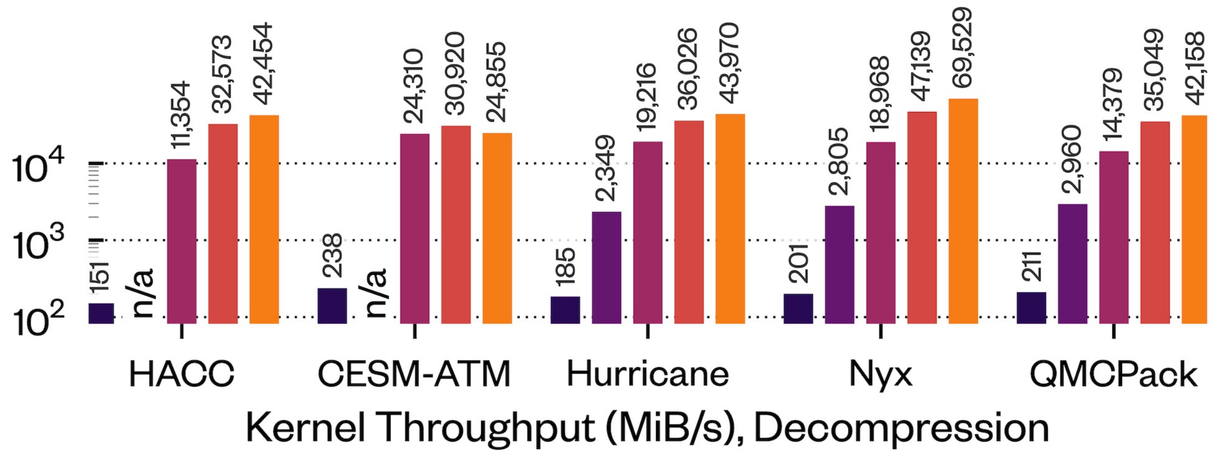
Threads # Tuning

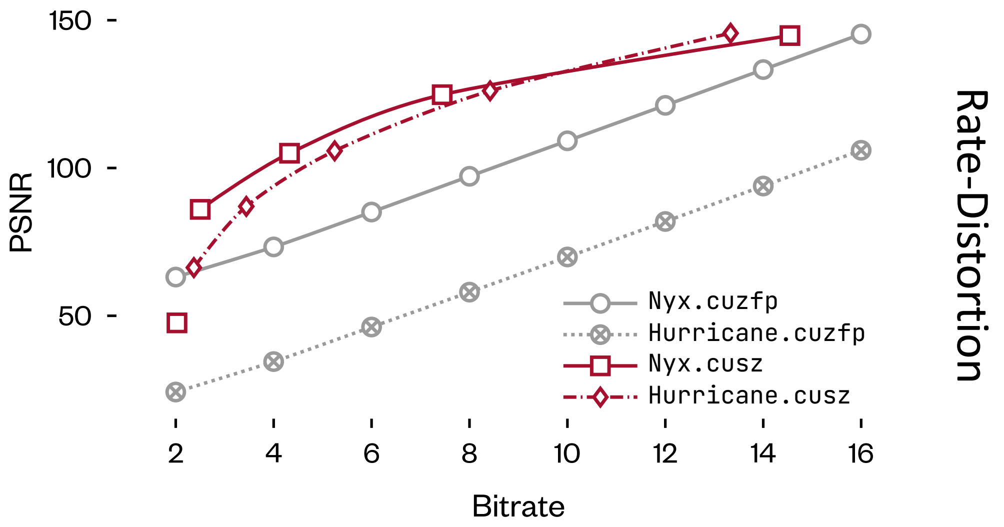
chunk	hacc			cesm			hurricane			nyx			qmcpack		
	1071.8 mb	280,953,867	f32	24.7 mb	6,480,000	f32	95.4 mb	25,000,000	f32	512 mb	134,217,728	f32	601.5 mb	157,684,320	f32
size	#thread	deflate	inflate	#thread	deflate	inflate	#thread	deflate	inflate	#thread	deflate	inflate	#thread	deflate	inflate
2^6	.	.	.	1.0e5	11.3	25.0	.	.	.	.	.	.	.	.	.
2^7	.	.	.	5.1e4	15.5	37.8	.	.	.	.	.	.	.	.	.
2^8	.	.	.	**2.5e4**	**67.1**	**41.6**	9.8e4	5.1	11.0	.	.	.	.	.	.
2^9	.	.	.	1.3e4	55.6	30.7	4.9e4	10.2	9.4	.	.	.	.	.	.
2^{10}	.	.	.	6.3e3	48.2	19.6	**2.4e4**	**64.6**	**34.2**	1.3e5	4.7	5.9	1.5e5	4.7	5.1
2^{11}	1.4e5	4.6	2.8	.	.	.	1.2e4	57.3	27.7	6.6e4	5.7	6.3	7.7e4	5.2	6.2
2^{12}	6.9e4	5.1	5.1	.	.	.	6.1e3	50.7	17.8	3.3e4	25.1	16.1	3.8e4	12.9	11.1
2^{13}	3.4e4	13.6	12.1	.	.	.	.	.	.	**1.6e4**	**69.7**	**52.4**	**1.9e4**	**72.7**	**40.3**
2^{14}	**1.7e4**	**63.1**	**35.0**	.	.	.	.	.	.	8.2e3	72.4	42.6	9.6e3	75.9	29.0
2^{15}	8.6e3	65.8	28.1	.	.	.	.	.	.	4.1e3	50.0	23.1	4.8e3	56.0	16.1
2^{16}	4.3e3	45.9	14.3	.	.	.	.	.	.	.	.	.	.	.	.

Table 3: Throughputs (in GB/s) versus different numbers of threads launched on V100. The optimal thread number in terms of inflating and deflating throughput is shown in bold.

Performance Evaluation: Throughput and Quality







cuSZ (as of October 2021):
For compression kernel,
411× ~ 719× over serial CPU
19.1× ~ 24.8× over OMP CPU

For decompression kernel,
130× ~ 235× over serial CPU
11.8× ~ 16.8× over OMP CPU

Adaptive Configuration of In Situ Lossy Compression for Cosmology Simulations via Fine-Grained Rate-Quality Modeling

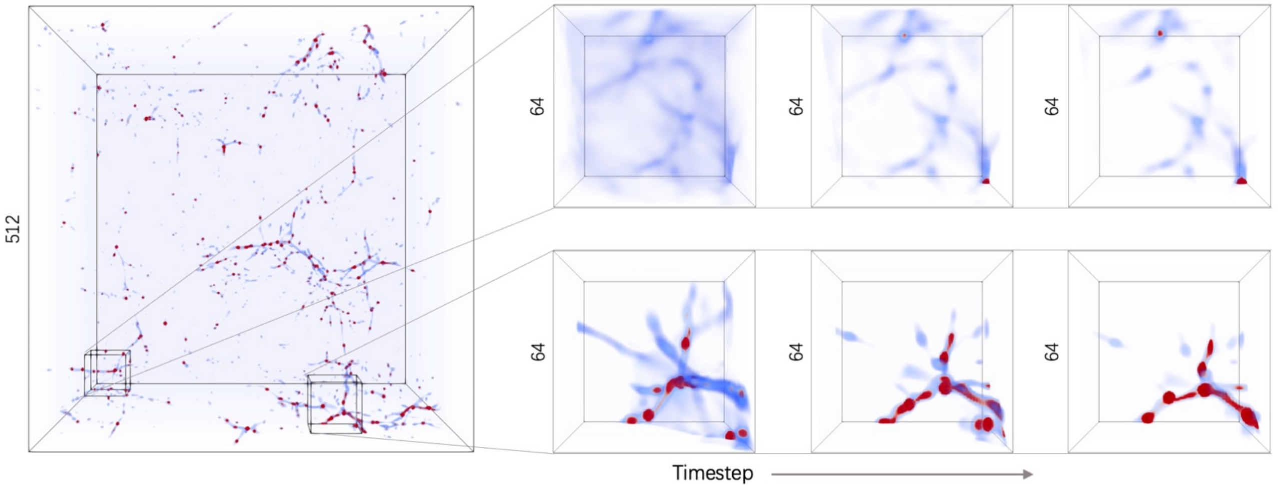
Published in *2021 ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'21)*

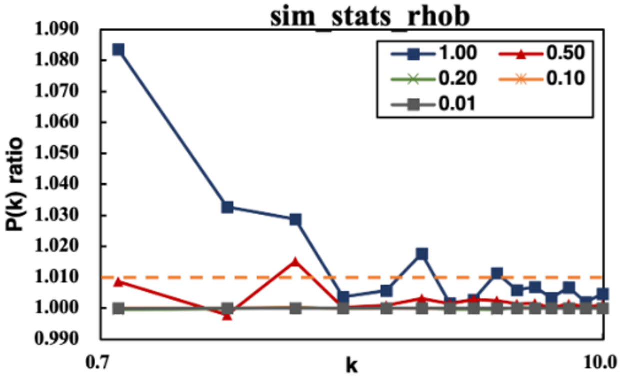
Led by **Sian Jin** from HiPDAC

Nyx Cosmology Simulation Data

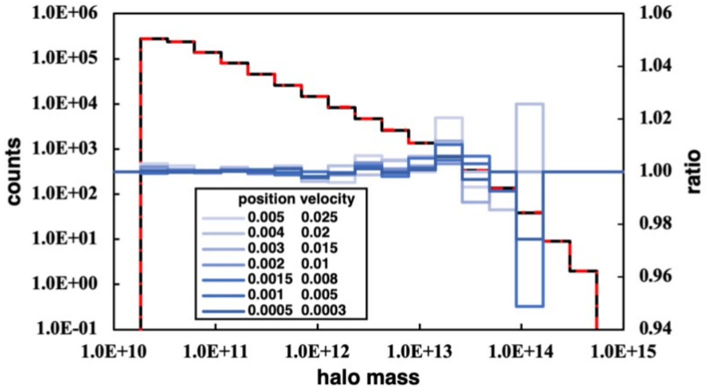
- **Structured Data**
  - Generated by mesh-based simulations in parallel ranks
  - Different ranks/partitions have different **densities of info**
- **Previous Solution** (Jin et al., IPDPS'20)
  - Optimize comp. performance by **trail-and-error** method
  - All partitions use the **same** compression **configuration**
  - Visual metrics (e.g., PSNR) are **insufficient**
- **Our Goals**
  - Guarantee domain-specific analysis quality
    - Power Spectrum
      - FFT-based analysis for Universe's matter distribution
      - **Target**: Ratio of P(k) on reconstructed data and original data remains within 1 ± 0.01
    - Halo Finder
      - Find over-densities in the Mass distribution
      - **Target**: Minimize the mass change of each halo
  - **In-situ** compression towards **optimal** compression ratio

Visualization of Baryon Density in Nyx simulation under resolution of 512 × 512 × 512



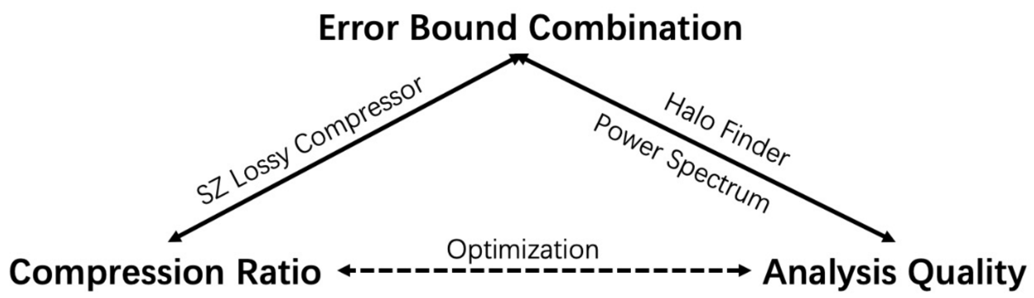


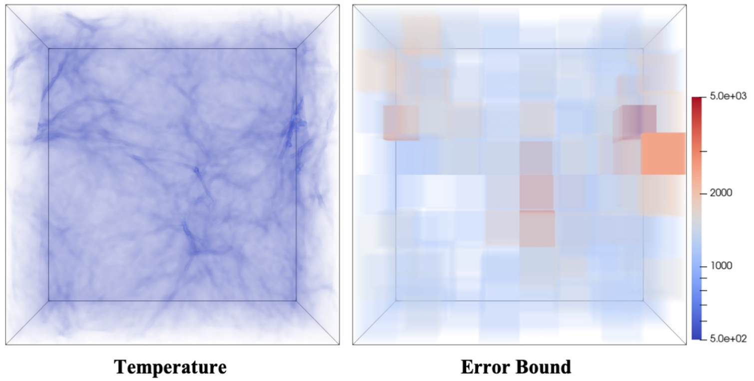
Power spectrum analysis on baryon density.



Halo Finder analysis on baryon density.

Our Methodology





Fine-grained lossy compression control for different data partitions.

➢ Fine-grained Compression

- Different error bounds for different partitions
- Different eb combinations for different time-steps

➢ Estimation on Post-analysis Quality Loss

- Predict post-analysis error based on eb combination
    - Power spectrum
    - Halo finder

➢ Estimation on Compression Ratio

- Predict compression ratio based on error-bound combination (e.g., SZ compression)

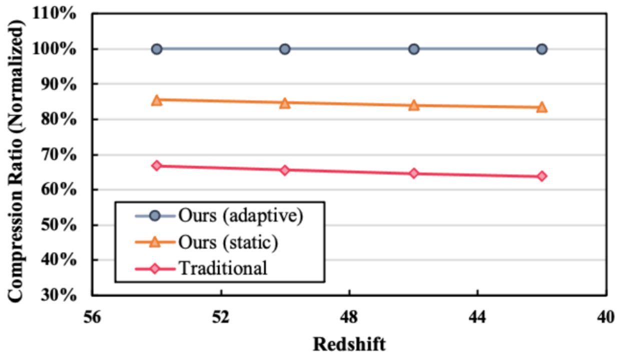
➢ Proposed Optimization Strategy

1. Parameter extraction (to estimate compression ratio)
    - Mean value of given partition
    - Mean value of overall dataset
2. Build Rate-Quality Model
    - EB-quality model
    - EB-rate model
3. Per-partition error bound optimization
    - Derivatives of rate-quality curves are **balanced** for all
4. For baryon density
    - Perform power-spectrum optimization first
    - Perform halo-finder optimization if not satisfied

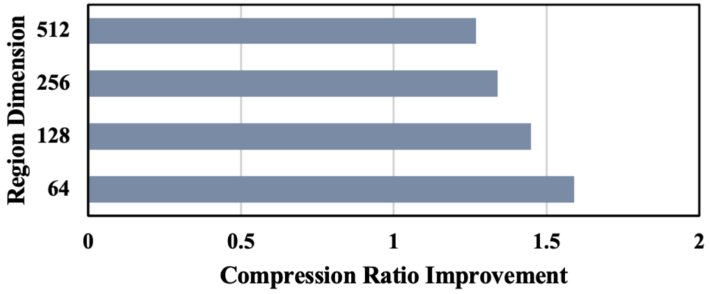
Evaluation

➢ **Compression Ratio Improvement**

- **1.56x** overall improvement (up to **1.73x**)
- Capable across time steps
- Smaller partitions higher improvement
- Capable across simulation with different resolutions

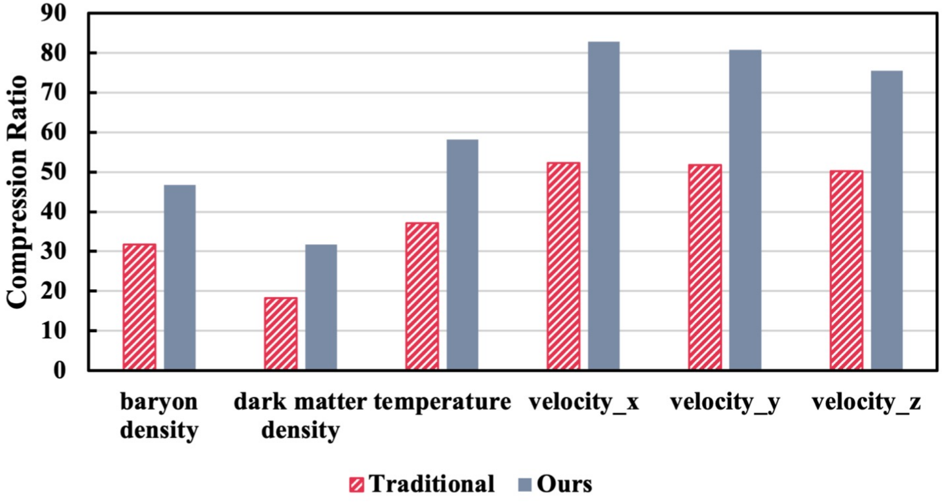


CR comparison between our and trad. methods on multiple redshifts' data using baryon density.

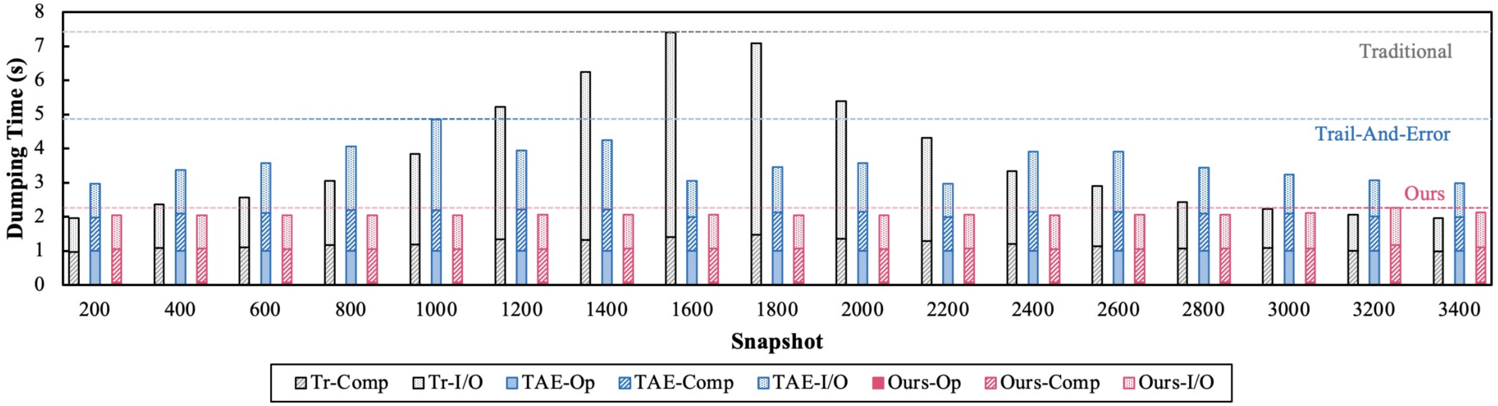


CR improvement with different partition sizes.

Jin *et al.*, submitted to ICDE'22



CR comparison between our and traditional methods on all 6 fields.



We generalize this modeling approach to other HPC applications, such as seismic imaging app. RTM. The above figure shows the overall **data dumping time** of different approaches under a similar post-analysis quality with **parallel HDF5**.

Optimizing Error-Bounded Lossy Compression for Three Dimensional Adaptive Mesh Refinement Simulations

Submitted to *2021 ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'22)*

Led by **Daoce Wang** from HiPDAC

Motivation & Background

➢ **Adaptive Mesh Refinement**

- Increase **resolution** in regions of most **interest**
- Reduce **computational and storage** overhead
- One of the most widely used frameworks for HPC applications

➢ **AMR apps still generate large amounts of data**

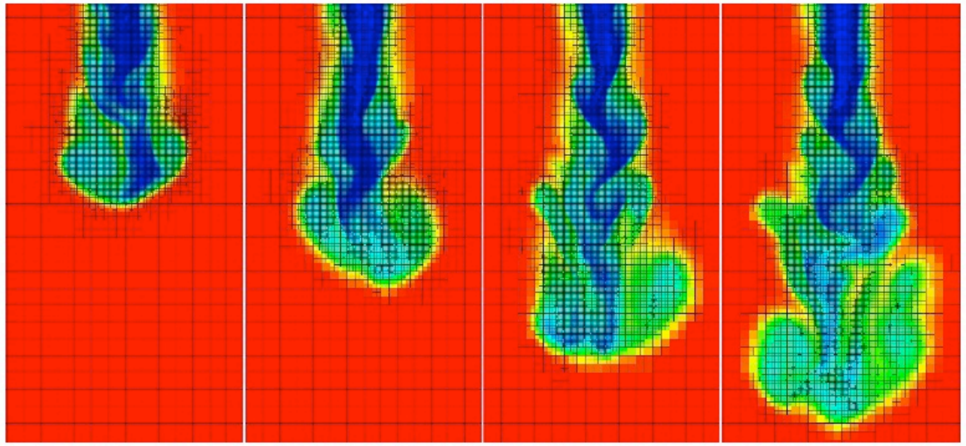
- For example, Nyx with a resolution of 4096^3 (i.e., $0.5 \times 2048^3 + 0.5 \times 4096^3$) generate **1.8 TB** data per snap-shot

➢ **Previous Solution** (Luo et al., IPDPS'21)

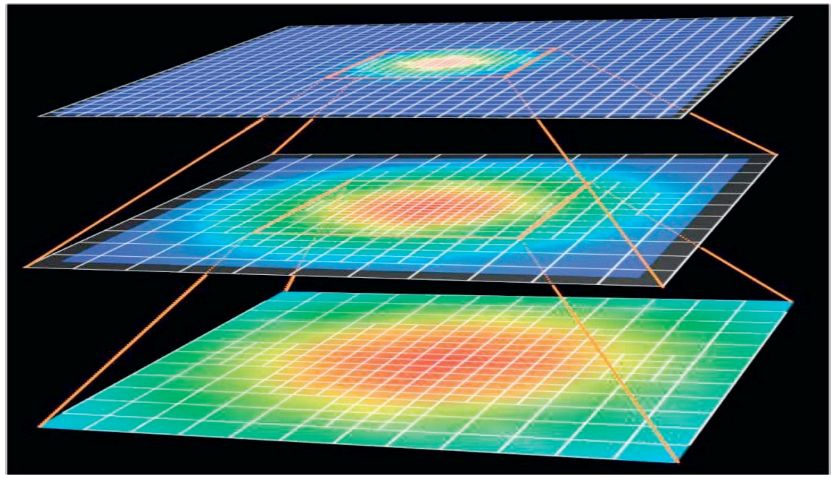
- **Reorder** AMR data in 1D based on geometric coordinates
- Cannot adopt **3D compression**
- Works only for block-structured AMR with **redundant data**

➢ **Our Goals**

- Adopt **3D** compression for each AMR level **separately**
- **Mitigate** separate 2D/3D compression (time/storage) overhead by **pre-processing**



Adaptive Mesh Refinement (AMR) on temperature and velocity during jetting: **grid structure** changes with jet progression.

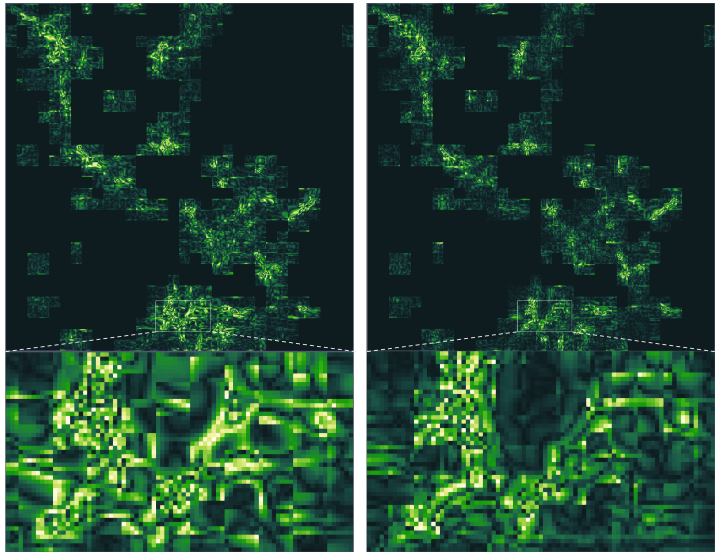


Proposed Approach

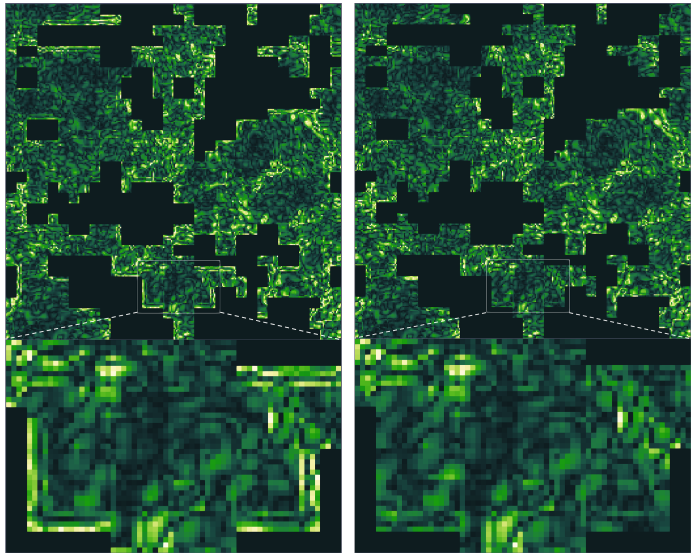
➢ Our Hybrid Pre-process Strategies

- Adaptively select the best-fit pre-process strategy based on data density of each AMR level

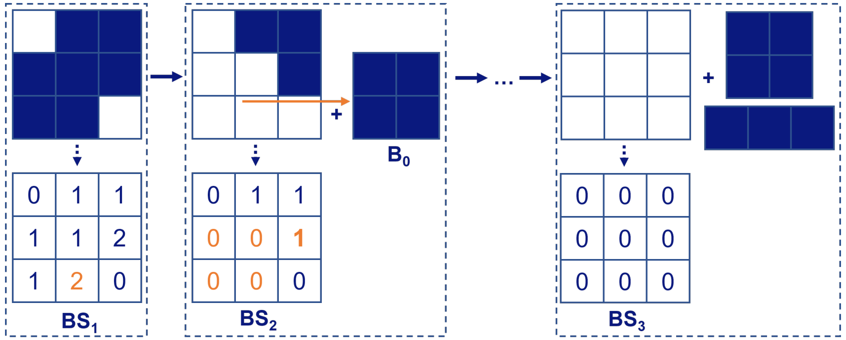
1. Optimized Sparse Tensor Representation (OpST) for **low-density** data
2. Adaptive k-D Tree (AKDTree) for **medium-density** data
3. Ghost-Shell Padding (GSP) for **high-density** data



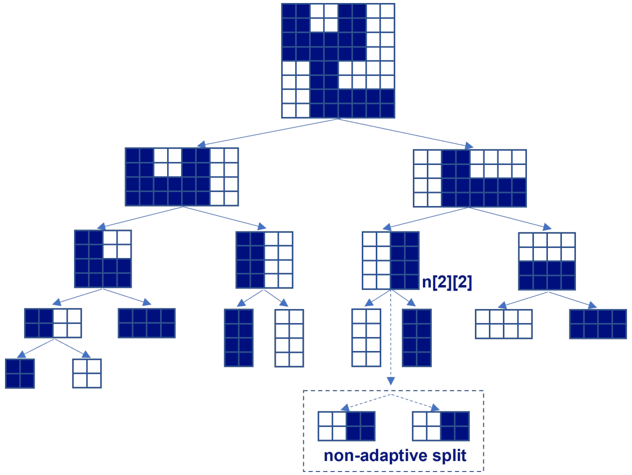
Compression errors of naïve Sparse Tensor (left) and OpST (right). Brighter means higher compression error.



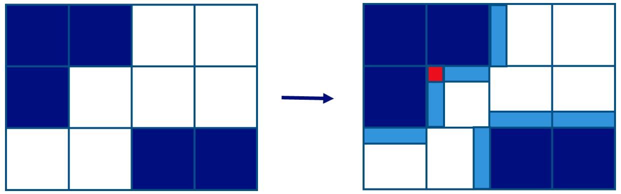
Compression errors of zero filling (left) and GSP (right). Brighter means higher error.



Opst



AKDTree



GSP

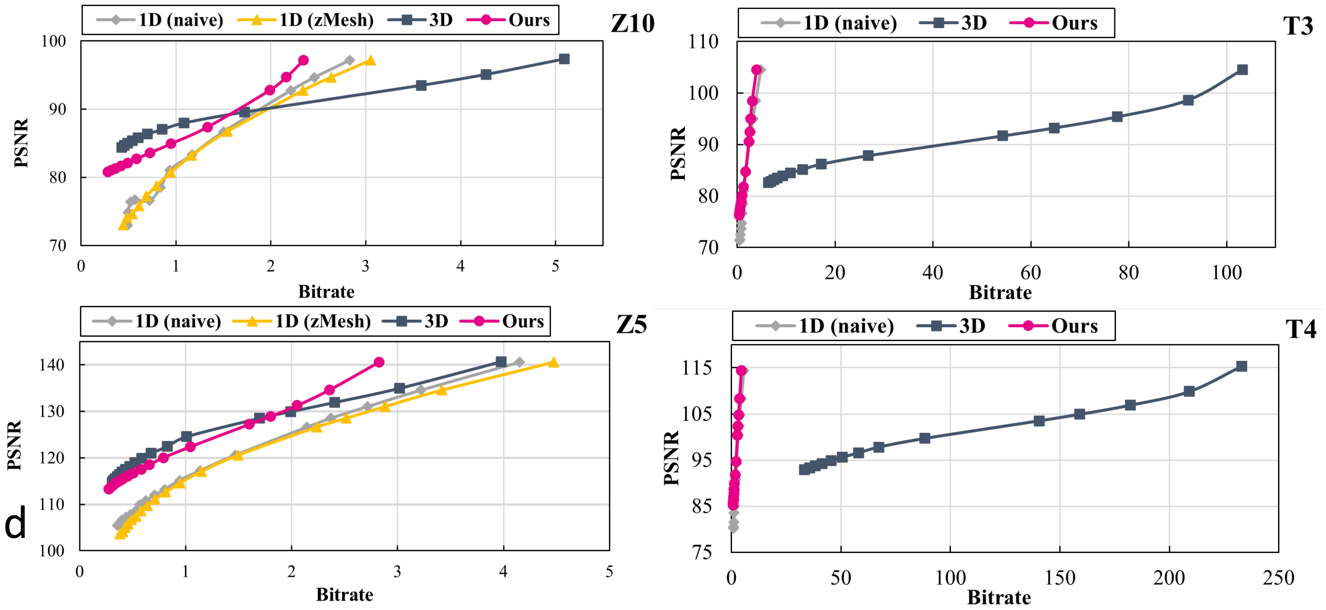
Evaluation

➢ Evaluation on Rate-distortion

- Outperforms naïve 1D baseline & zMesh (up to **3.3x**)
- Perform much better than 3D baseline when
  - (1) finest level has a relatively **low density**, or
  - (2) decompressed data has a **high PSNR**

➢ Evaluation on Time Overhead

- Up to **75x** faster than 3D baseline on Run2 datasets and **2.4x** faster on Run1 datasets
- Throughput degrades on the small datasets (i.e., T3 &T4)



Rate-distortion of timesteps in Run1 (left) and Run2 (right)

Overall compression/decompression throughput (MB/s) of different approaches with different absolute error bounds.

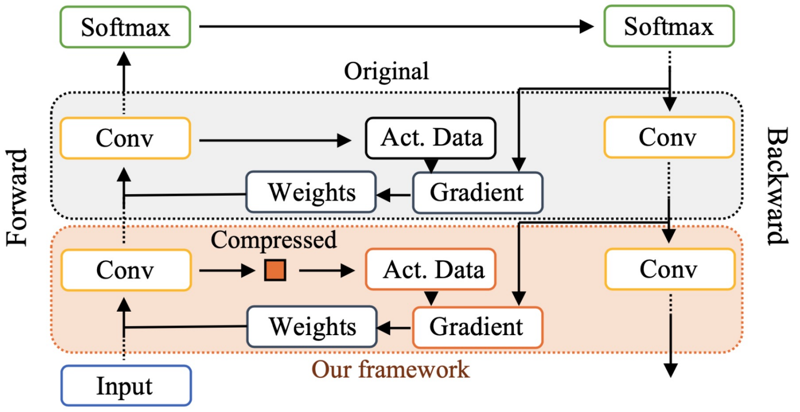
EB_{abs}	Run1_Z2			Run1_Z3			Run1_Z5			Run1_Z10			Run2_T2			Run2_T3			Run2_T4		
	1D	3D	ours	1D	3D	ours	1D	3D	ours	1D	3D	ours	1D	3D	ours	1D	3D	ours	1D	3D	ours
1E+08	169	94	97	166	90	94	161	76	99	160	40	95	152	17	76	143	2.4	60	125	0.4	30
1E+09	219	115	121	213	120	127	208	109	123	208	63	117	193	27	91	184	3.9	66	159	0.5	32
1E+10	259	125	135	256	125	136	253	117	137	250	65	135	242	30	102	229	4.0	72	197	0.5	34

COMET: A Novel Memory-Efficient Deep Learning Training Framework by Using Error-Bounded Lossy Compression

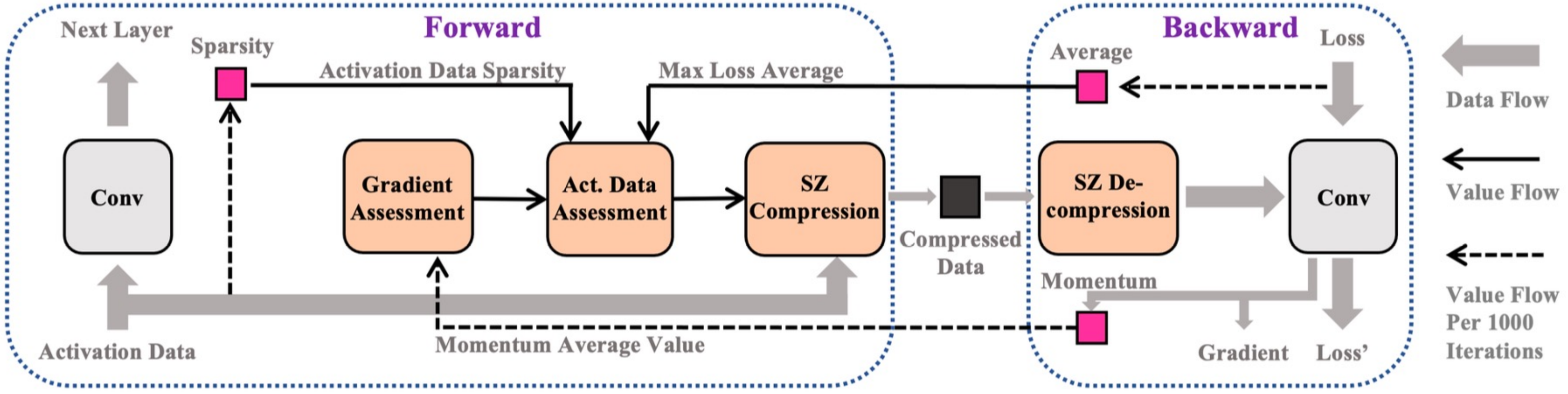
To appear in *2022 International Conference on Very Large Data Bases (VLDB'22)*

Led by **Sian Jin** from HiPDAC

System Design



Data flow in a sample iteration of training CNNs



Overview of our proposed memory-efficient DNN training framework - COMET

Activation Data Storage in Training

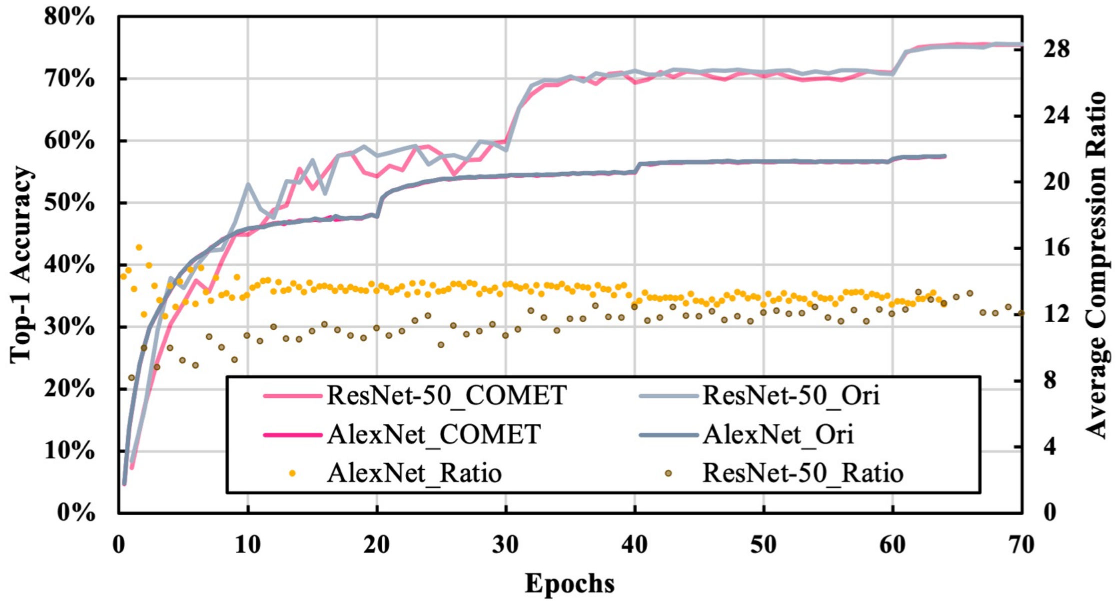
- Must being stored until used in back propagation
- Long waiting period between generating and using the data

- **Parameter collection**: collect parameters for analysis and updating compression configurations
- **Gradient assessment**: estimate acceptable σ in the gradient
- **Activation assessment**: estimate acceptable error bound for compressing activation data
- **Adaptive compression**: deploy lossy compression

Memory Usage Evaluation

➢ **Memory Footprint Reduction**

- High compression ratio, **up to 13.5x**
- **Little/no** testing accuracy loss



Training accuracy curve comparison between the baseline and our proposed framework.

Neural Nets		Top-1 Accuracy	Peak Mem.	Max Batch	Conv. Act. Size	COMET	JPEG-ACT
AlexNet	b.	57.41%	2.17 GB	512	407 MB		
	c.	57.42%	0.85 GB	2048	**30 MB**	13.5×	-
VGG-16	b.	68.05%	17.29 GB	64	6.91 GB		
	c.	68.02%	5.04 GB	256	**0.62 GB**	11.1 ×	-
ResNet-18	b.	67.57%	5.16 GB	256	1.71 GB		
	c.	67.43%	1.37 GB	1024	**0.16 GB**	10.7 ×	7.3 ×
ResNet-50	b.	75.55%	15.57 GB	128	5.14 GB		
	c.	75.51%	4.40 GB	512	**0.46 GB**	11.0 ×	6.0 ×

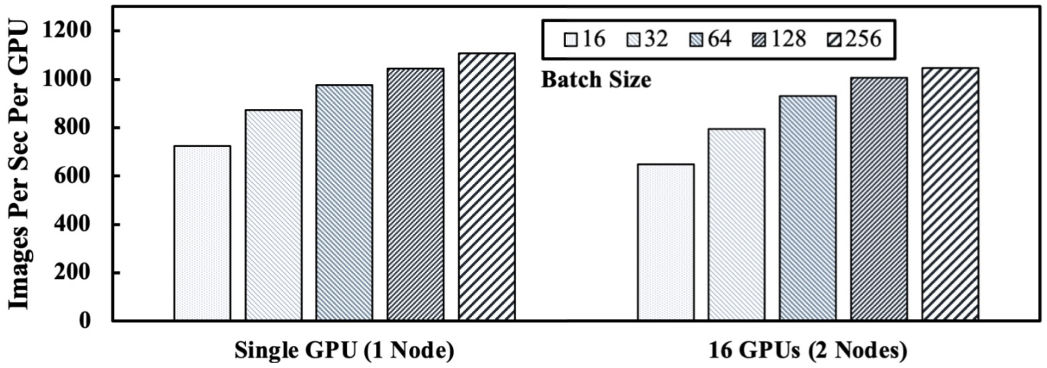
b.= baseline, c.= compressed

Comparison of accuracy and activation size between baseline training and our proposed framework

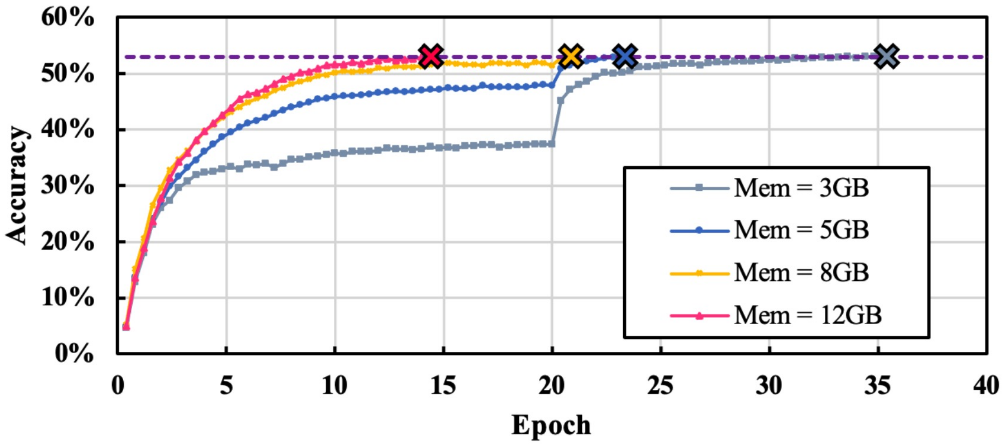
Evaluation

➢ **Performance Improvements**

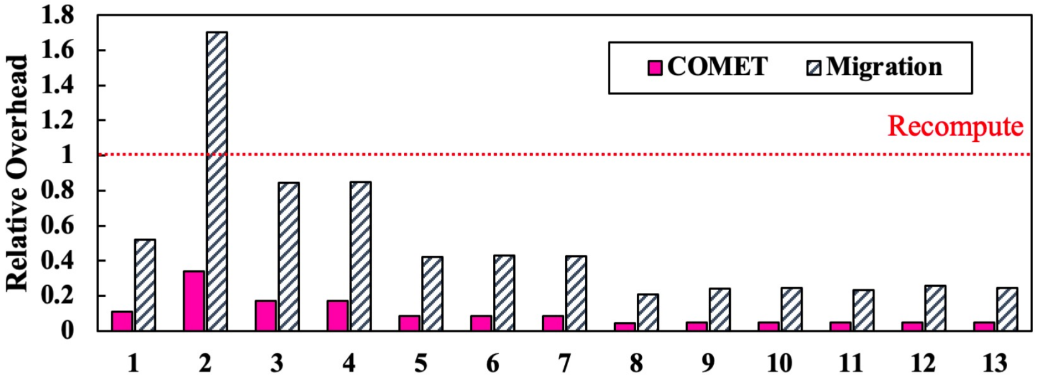
- Low compression overhead, significantly lower than data migration solution (e.g., **7%** on VGG-16)
- High raw throughput (sample/sec) improvement with better resource utilization (e.g., **1.24x** on ResNet-50)
- End-end performance improvement: train model faster (e.g., **2x** on AlexNet)



Training performance on ResNet-50 with different Batch size



Validation accuracy curve of COMET under different GPU memory constraint on AlexNet



Overhead comparison between migration, recomputation

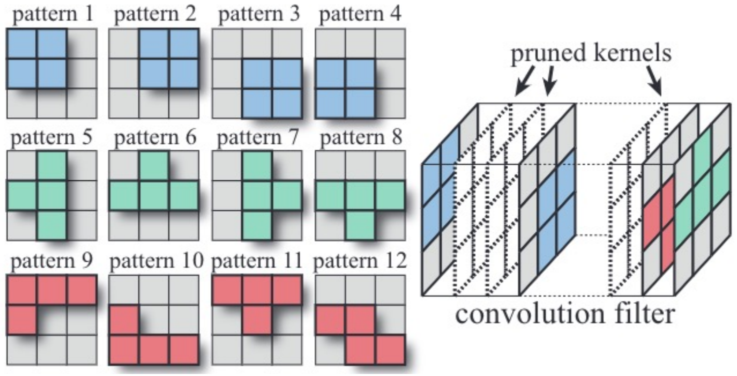
ClickTrain: Efficient and Accurate End-to-End Deep Learning Training via Fine-Grained Architecture-Preserving Pruning

Published in *2021 ACM International Conference on Supercomputing (ICS'21)*

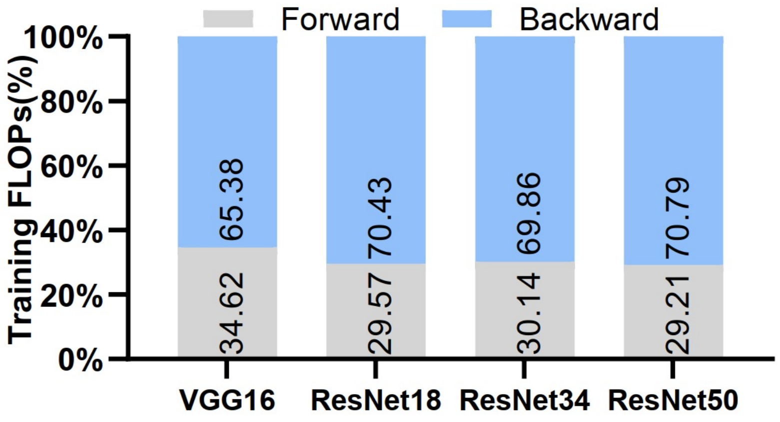
Led by **Chengming Zhang** from HiPDAC

Pattern Based Pruning

- **Fined-grained Pattern-based Pruning**
  - Pruning **intermediate sparsity** between non-structured pruning and structured pruning
- **Why pruning during training?**
  - **Ever-increasing** scale and complexity of DNNs with **large-scale** training datasets, leading to challenges to the cost of DNN training
  - Backward phase consumes **more than 70%** of overall training FLOPs
- **Our Goals**
  - Use pruning during training (PDT)-based method to significantly improve **end-to-end performance**
  - Maintain network architecture for **high accuracy**
  - Fully utilize pattern sparsity via multiple system-level optimizations
    - Library support: fast **sparse matrix conversion**, pattern-accelerated **sparse convolution** & **communication**
    - Compiler support: compiler-assisted **optimized code generation**

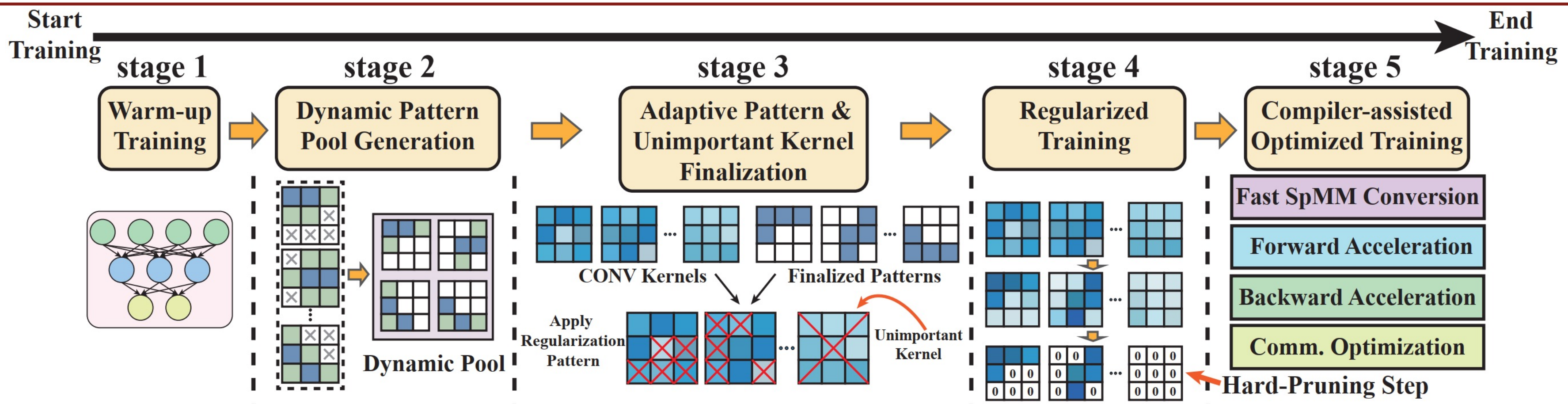


Fined grained pattern-based pruning (gray parts are pruned).

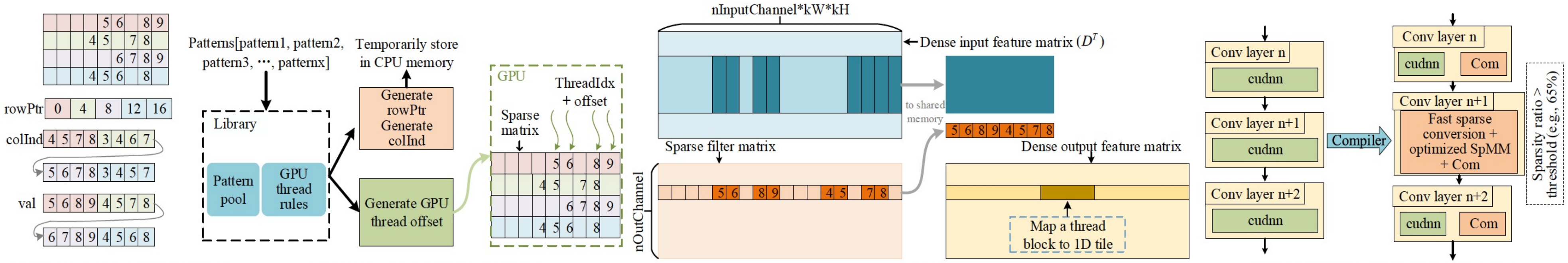


Percentage of FLOPs in forward and backward.

ClickTrain Design



- Stage 1, 2, 3, 4 are **algorithm-level design**: focusing on high compress ratio and high accuracy
- Stage 5 is **system-level design**: focusing on improving computation efficiency



(a) Sparse matrix and CSR format (b) Pattern-driven CSR conversion (c) Optimized SpMM library (d) Compiler-based code transformation

1. Fast sparse matrix conversion: through **pre-selected sparsity pattern**
2. Workload balancing: limit all filters in same layer with **same number** of un-pruned (non-zero) weights
3. Sparse convolution on GPU: **1D tiling** strategy - map each thread block to a 1D row tile of output matrix

Evaluation

Dataset	Model	PDT Method	Base. Acc.	Valid. Acc. Δ	Comp. Ratio	Train./Inf. FLOPs	Hard Pr. Epoch
CIFAR10	ResNet32	PRT	93.6%	−2.1%	2.2×	53% / 66%	N/A
		CLK	93.6%	0±0.05%	8.6×	41.3% / 85.1%	98
		CLK	93.6%	**0±0.07%**	**10.7×**	**43.0% / 85.7%**	95
	ResNet50	PRT	94.2%	−1.1%	2.3×	50% / 70%	N/A
		CLK	94.1%	0±0.04%	8.5×	37.5% / 74.3%	95
		CLK	94.1%	**−0.2±0.05%**	**10.8×**	**41.2% / 77.6%**	90
	VGG11	PRT	92.1%	−0.7%	8.1×	57% / 65%	N/A
		CLK	92.1%	−0.1±0.04%	8.7×	41.2% / 81.5%	96
		CLK	92.1%	**−0.3±0.06%**	**11.5×**	**43.9% / 85.3%**	94
	VGG13	PRT	93.9%	−0.6%	8.0×	56% / 63%	N/A
		CLK	93.8%	0±0.08%	8.6×	41.3% / 81.3%	95
		CLK	93.8%	**−0.2±0.04%**	**10.9×**	**42.5% / 84.9%**	96
CIFAR100	ResNet32	PRT	71.0%	−1.4%	2.1×	32% / 46%	N/A
		CLK	71.0%	0±0.05%	8.3×	41.7% / 82.9%	95
		CLK	71.0%	**−0.2±0.05%**	**10.4×**	**45.2% / 85.6%**	90
	ResNet50	PRT	73.1%	−0.7%	1.9×	53% / 69%	N/A
		CLK	73.1%	0±0.04%	8.2×	36.7% / 73.6%	96
		CLK	73.1%	**−0.2±0.07%**	**9.7×**	**38.9% / 77.3%**	95
	VGG11	PRT	70.6%	−1.3%	3.0×	47% / 57%	N/A
		CLK	70.6%	0±0.1%	6.7×	40.1% / 78.6%	95
		CLK	70.6%	**−0.2±0.06%**	**8.4×**	**43.1% / 82.0%**	92
	VGG13	PRT	74.1%	−1.4%	2.9×	42% / 52%	N/A
		CLK	74.1%	−0.1±0.05%	7.4×	40.5% / 79.7%	95
		CLK	74.1%	**−0.2±0.08%**	**9.2×**	**41.7% / 83.3%**	96
ImageNet	ResNet50	PRT	76.2%	−1.9%	1.6×	40% / 53%	N/A
		CLK	76.2%	**−0.6±0.07%**	4.3×	**36.9% / 66%**	40

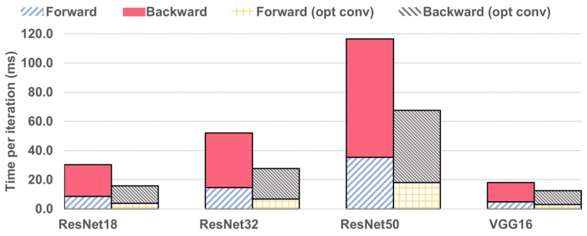
➢ **Comparison with SOTA PDT-based Approach**

- ResNet32/50: **10×** compression ratio with only up to **0.2%** accuracy drop
- VGG11/13: **8.6×~11.5×** compression ratio with up to **0.3%** accuracy drop

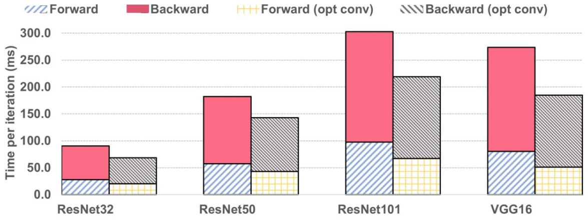
Model	PAT Method	Base. Acc.	Valid. Acc. Δ	Comp. Ratio	Total Epochs
ResNet-18	TAS [12]	70.6%	−1.5%	1.5×	120
	DCP [74]	69.6%	−5.5%	3.3×	well train + 60
	CLK	69.6%	**−0.9%**	**4.1×**	**90**
ResNet-50	GBN [65]	75.8%	−0.6%	2.2×	well train + 60
	GAL [29]	76.4%	−7.1%	2.5×	well train + 30
	CLK	76.2%	**−0.6%**	**4.3×**	**90**
ResNet-101	RSNLIA [63]	75.27%	−2.10%	1.9×	well train + tune
	CLK	76.4%	**−1.2%**	**4.2×**	**90**
VGG-16	NeST [8]	71.6%	−2.3%	6.5×	N/A
	CLK	73.1%	**−0.8%**	**6.6×**	**90**

Comparison with SOTA PAT-based Approaches

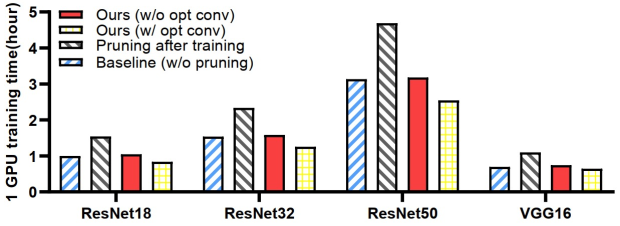
- Save up to **67%** computation time with up to **1.2%** accuracy drop



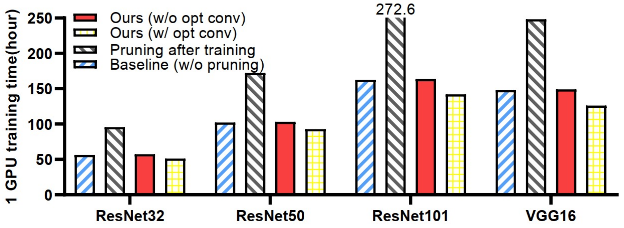
(a) CIFAR



(b) ImageNet



(a) CIFAR10



(b) ImageNet

- Speedups of **2.2×**, **2.1×**, **1.9×**, **1.6×** on ResNet18, ResNet32, ResNet50, VGG16
- Saves **0.16**, **0.29**, **0.59**, and **0.15** hours on ResNet18/32/50 and VGG16 on CIFAR10

HuffMax: Optimizing Memory Efficiency for Parallel Influence Maximization on Multicore Architectures

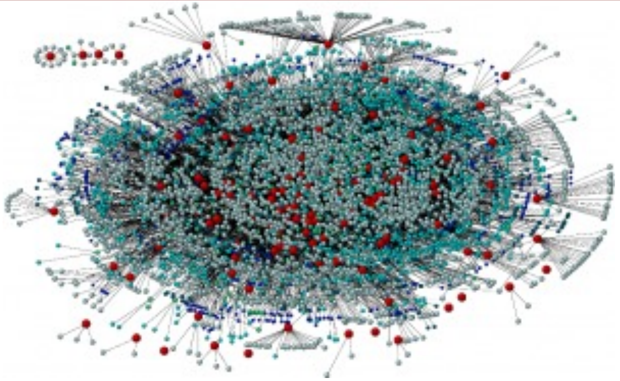
Submitted to *ACM International Conference on Supercomputing (ICS'22)*

Led by **Xinyu Chen** from HiPDAC

Motivation

➢ Influence Maximization (IM) Problem

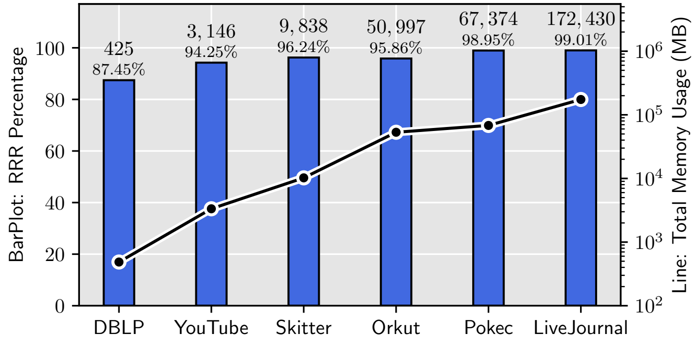
- Given a graph G=(V,E), find k vertices that can activate maximal number of vertices in G (**NP-hard** problem)
- Use MC simulation to get approximate solution
- Both **computation and memory** intensive on large graphs



IM has wide applications in *viral marketing*, *politics*, *public health*, *sensor networks*, *bioinformatics*, etc.

➢ SOTA Solution (Minutoli *et al.*, Ripples)

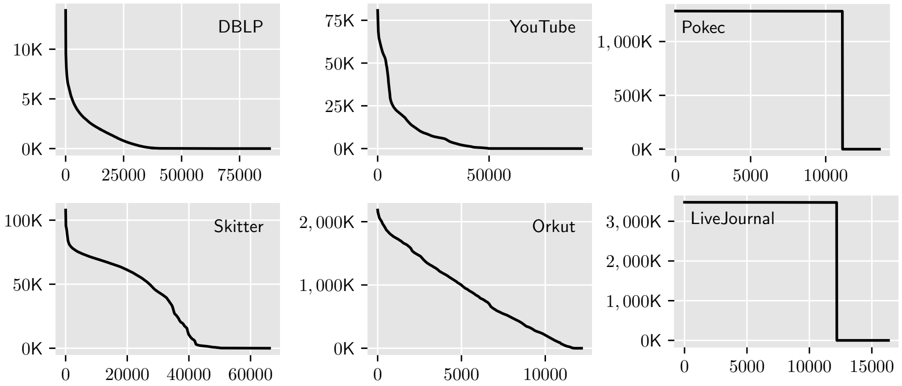
- Improved performance by parallelization on shared-and distributed-memory systems
- Huge **memory inflation** (30x~165x) during computation



Memory usage of intermediate result on different graphs

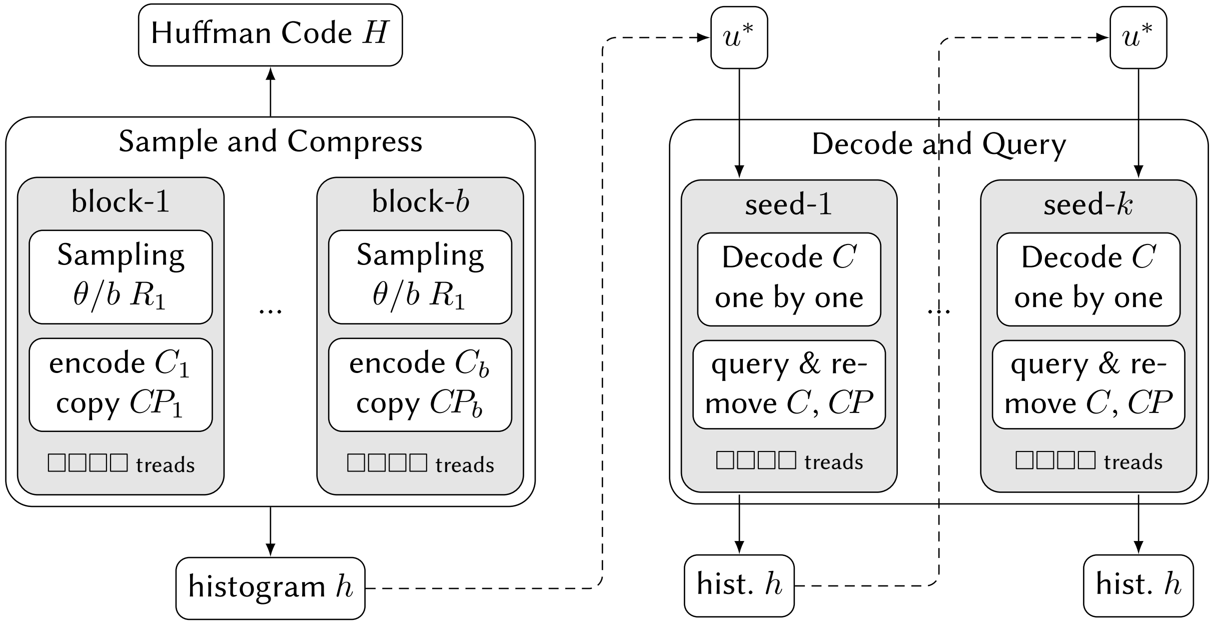
➢ Our Goals

- Characterize memory footprint based on **graph characteristics**
- Use **compression techniques** to reduce memory footprint.
- **Analysis on compressed data** to preserve memory saving.

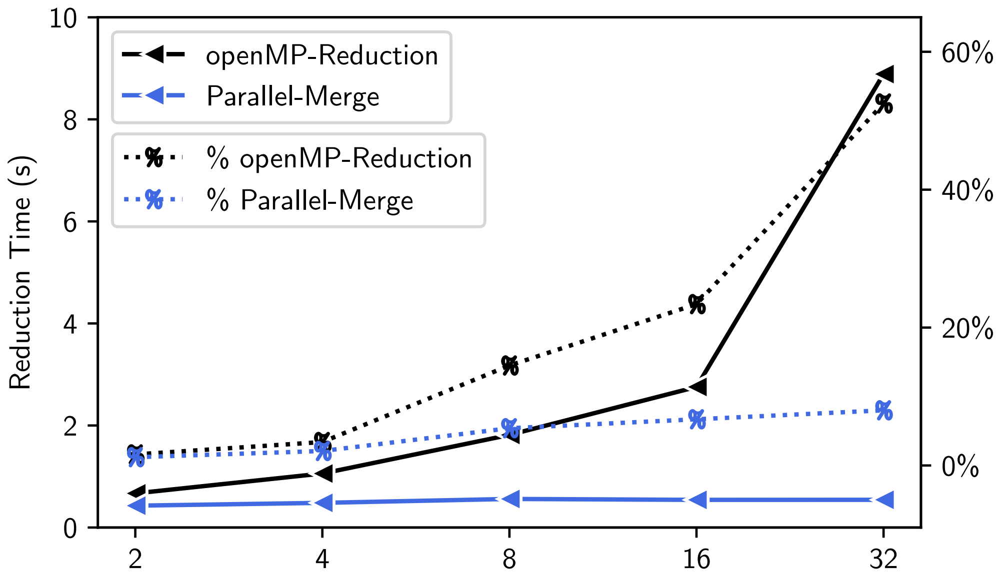


Characterization of intermediate result's distribution on different graphs

System Design



HuffMax workflow: sampling-and-encoding, decoding-and-selection



Scalability of Parallel Merge and OpenMP reduction

➢ Block-based sampling-and-encoding

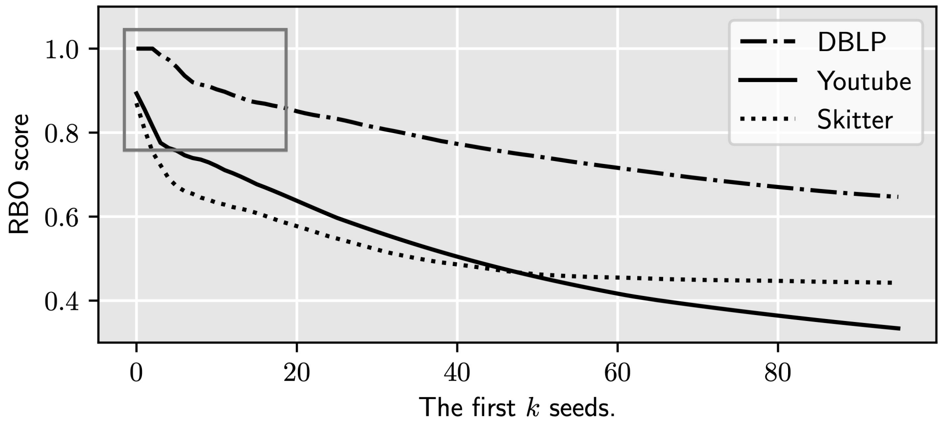
- Use 1st portion of MC to characterize graphs
  - Kurtosis K for **Adaptive Sampling** (increase threshold σ)
  - Skewness S to trigger **Huffman Coding** or fall back to Ripples
- Use OpenMP to parallelize
  - **Swap** potential seed to the front

➢ Decoding-and-selection

- Query decoded data
  - Leverage data locality for **partially decoding**
- Parallel merge
  - Reduce **global maximum** from **local maxima** (p<<n)
  - Nearly **constant time** compared with OpenMP reduction

Evaluation

- **Effectiveness of adaptive sampling**
  - Top-1 seeds is **NOT affected** by increasing $\sigma = 10$
- **Reduction of memory footprint**
  - Up to **45.7%** (Skitter) w/o adaptive sampling
- **Shorten time-to-solution**
  - Up to **28.0%** (Youtube) w/ adaptive sampling
- **Strong scalability**
  - **9.45x speedup** on 64 cores

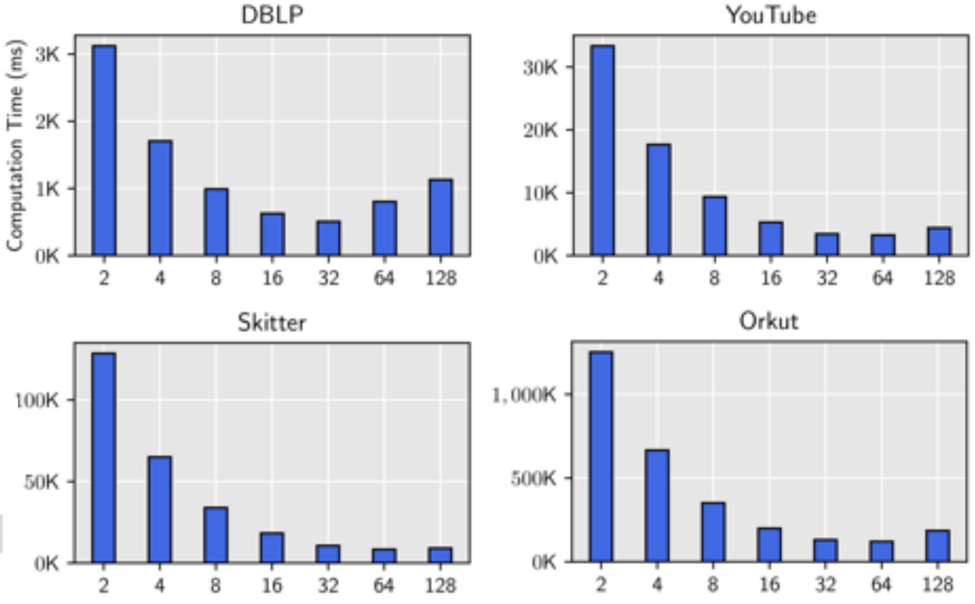


Graph	DBLP	YouTube	Skitter	Orkut
Ripples	423 (1.00)	3,143 (1.00)	9,888 (1.00)	45,784 (1.00)
HuffMax1	342 (1.24)	1,780 (1.77)	5,365 (1.84)	30,001 (1.53)
HuffMax2	258 (1.64)	1,943 (1.62)	6,447 (1.53)	30,088 (1.52)

Average 36% memory reduction on skew distributed graphs

Graph	DBLP		YouTube		Skitter		Orkut		Pokec		Journal	
Ripples	0.77	(0%)	6.16	(0%)	13.52	(0%)	172.23	(0%)	164.47	(0%)	514.44	(0%)
HuffMax1	1.02	(32%)	4.93	(−20%)	12.89	(−5%)	152.08	(−12%)	164.84	(0.2%)	515.55	(0.2%)
HuffMax2	0.68	(−12%)	4.44	(−28%)	12.72	(−6%)	-	-	-	-	-	-

Time-to-solution on tested graphs. Average time shortened is 14.5% on skew-distributed graphs.



Scalability of Parallel Merge and OpenMP reduction