CEAZ: Accelerating Parallel I/O via Hardware-Algorithm Co-Designed Adaptive Lossy Compression

Chengming Zhang (Washington State University)
Sian Jin (Washington State University)
Tong Geng (Pacific Northwest National Laboratory)
Jiannan Tian (Washington State University)
Ang Li (Pacific Northwest National Laboratory)
Dingwen Tao (Washington State University)

Background: Storage and I/O Issues

supercomputer	year	class	PF	MS	SB	MS/SB	PF/SB
Cray Jaguar	2008	1 PFLOPS	1.75 PFLOPS	360 TB	240 GB/s	1.5K	7.3K
Cray Blue Waters	2012	10 PFLOPS	13.3 PFLOPS	1.5 PB	1.1 TB/s	1.3K	13K
Cray CORI	2017	10 PFLOPS	30 PFLOPS	1.4 PB	1.7 TB/s•	0.8K	17K
IBM Summit	2018	100 PFLOPS	200 PFLOPS	>10 PB••	2.5 TB/s	>4K	80K

PF: peak FLOPS MS: memory size SB: storage bandwidth
• when using burst buffer •• counting only DDR4

Source: F. Cappello (ANL)

supercomputer	year	class	PF	MS	SB	MS/SB	PF/SB
Fujitsu Fugaku	2020	"EXASCALE"	537 PFLOPS•	4.85 PB	≥1.5 TB/s••	≥3.2K	358K
Intel Aurora•••	FUTURE	EXASCALE	≥2 EFLOPS	>10 PB	≥25 TB/s	≥0.4K	80K

• Rpeak, TOP 500 for November 2020 •• DDN Newsroom ••• as of October 2021

The compute capability is ever growing, but storage capacity and bandwidth are developing much **more slowly**

ALCF: Intel Aurora

Background: Why Lossy Compression

➢ **Lossy compression on scientific data**

- Offers much higher compression ratios than lossless compression by trading a little bit of accuracy

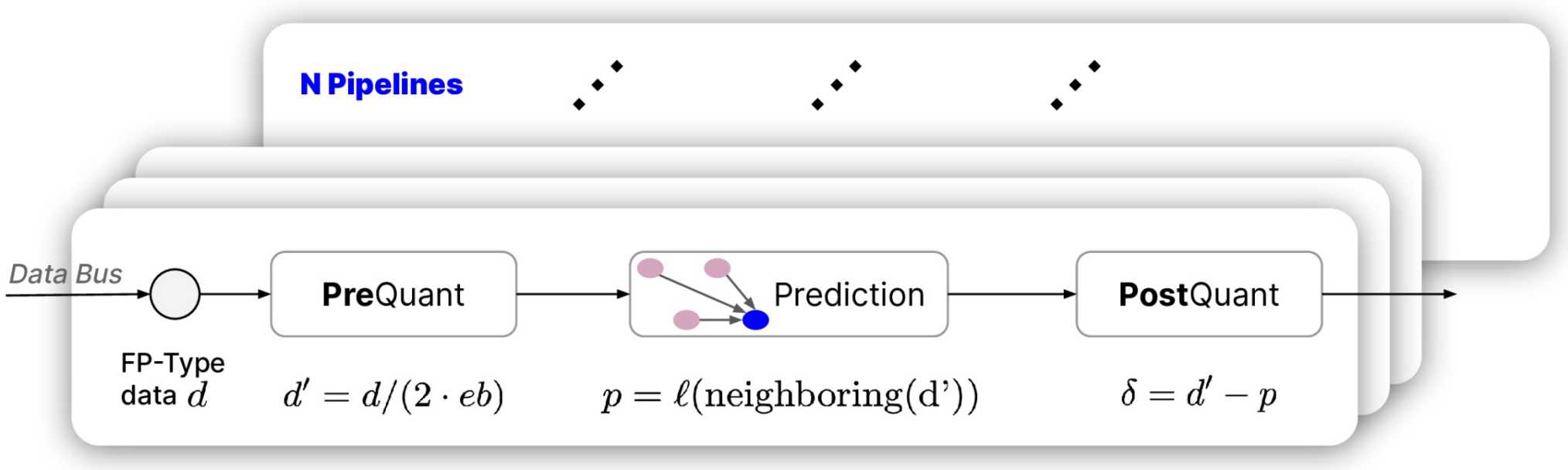
➢ **Data is extremely large**

- One Nyx cosmological simulation with a resolution of 4096^3 cells can generate up to 2.8 TB
- Such a large amount of data is generated in a parallel in many nodes and use collective communication to dump the entire snapshot to the file system
- This process takes an unprecedented challenge to I/O bandwidths and storage systems

Background: Dual-quantization

➢ Dual-quantization algorithm

- Dual-quant consists of two steps: **prequantization** and **postquantization**
- Quantize a float point data based on the user-set error bound and convert it to an integer data then calculate its predicted value
- Difference between the predicted value and the prequantized value will be **compressed** by Huffman compression



Background: FPGA-based Compressor

➢ **FPGA-based lossy compressor**

- waveSZ adopts a wavefront memory layout to to alleviate the data dependency, but
  (1) just alleviates the data dependency
  (2)wavefront memory layout involves rearranging data

- BurstZ can provide a high throughput (8 GB/s), but
  (1) suffers from a significantly higher compression ratio drop compared with the original ZFP algorithm
  (2) 8GB throughput is much smaller than current PCIe3/4 and InfiniBand

- ZFP is NOT error-bounded

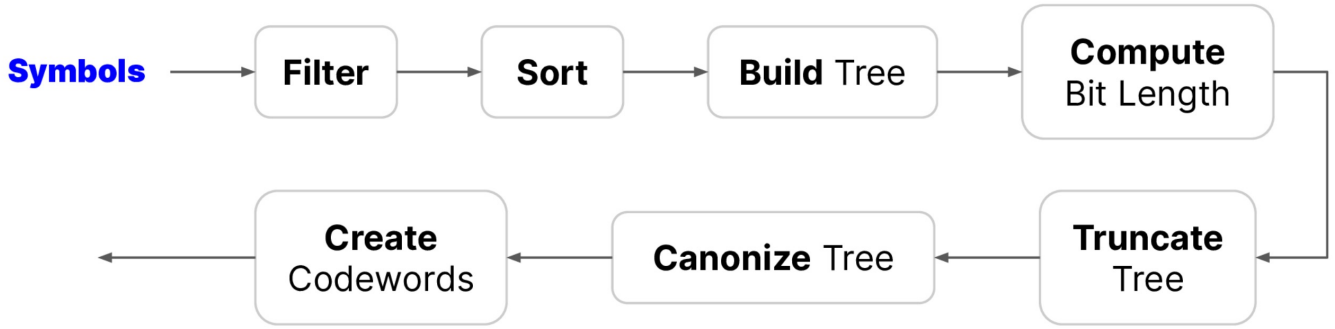
Motivation and Challenge

➢ High Overhead of Huffman Coding

- Huffman coding has high overhead in terms of latency, area, and power
- Hard to build a Huffman tree and generate codewords within limited hardware clock cycles to meet high-throughput requirements

➢ Challenge of Predefined Codewords

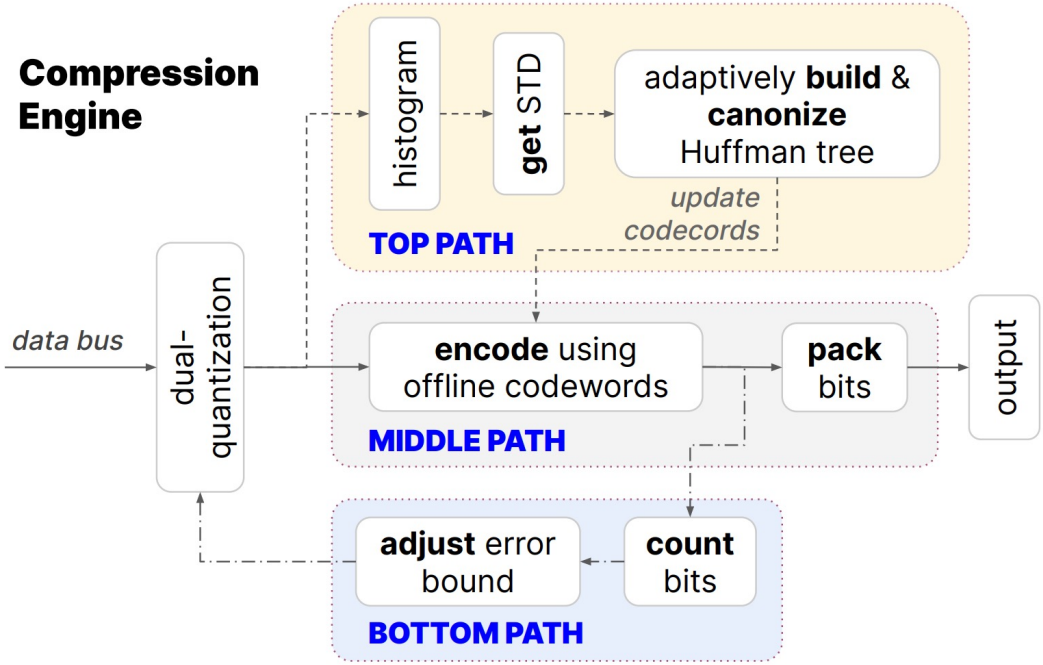
- We use predefined codewords at the beginning and update the codewords during the runtime
- **How to generate suitable codewords?**



Overview of CEAZ

- **FPGA-based lossy compressor CEAZ**

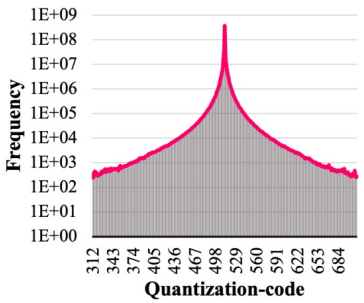
- Top dataflow path, we preprocess float-point data using a dual quantization algorithm
- Middle dataflow path, we directly encode symbols using existing codewords for seeking high throughput
- Bottom dataflow path, we feed back total bits of encoded symbols to estimate compression ratio, and then adjust the error bound



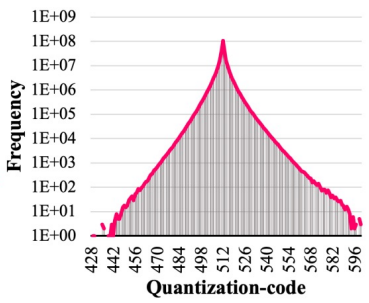
Design Methodology

➢ Fast approximate sort

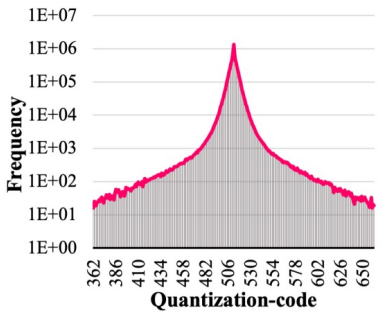
- Frequencies of symbols that are generated by Lorenzo predictor and linear-scaling quantization are symmetric
- our approximate sort saves the sort time by 7.5X on average over the merge sort



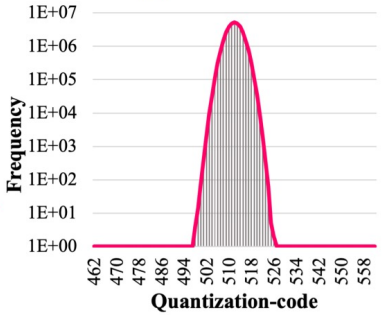
(a) NWChem



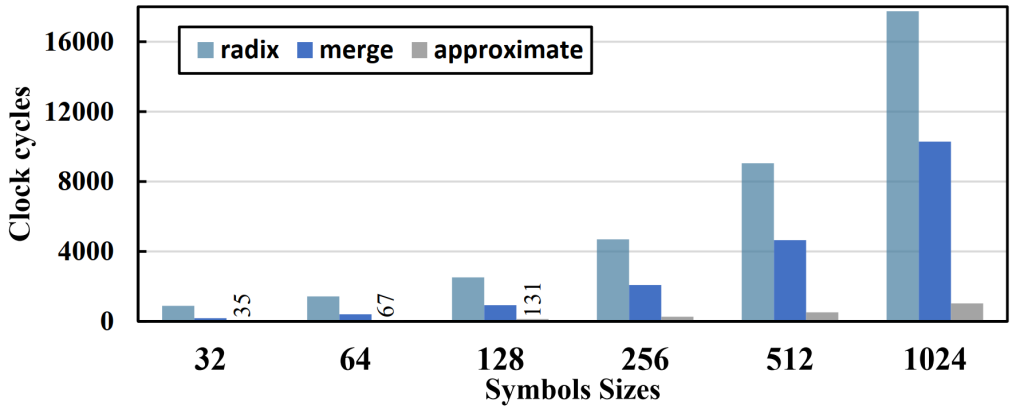
(b) HACC



(c) CESM



(d) Brown



Algorithm 1: Proposed fast sort based on Lorenzo predictor's feature.

Result: A approximately sorted array

```
S: defined structure contains two members: symbol, and its frequency
A: input array, its data type is S, len: length of input array, i: index of A
p: index of symbol 512 in A, m: index of the midpoint of A, l: index, h: index
O: output sorted array, j: index of O
t: loop count
l = p − 1, h = p + 1, j = len − 2, O[len − 1] = A[p]
if p ≤ m then
    t = p
else
    t = len − p − 1
end
for i ← 1 to rows do
    Count if A[l].frequency ≤ A[h].frequency then
        O[j] = A[h]
        O[j − 1] = A[l]
    else
        O[j] = A[l]
        O[j − 1] = A[h]
    end
    l = l − 1, h = h + 1, j = j − 2
end
CopyRemaining(A, O) /* copy remaining data from A to O */
```

Design Methodology

➢ Offline Huffman codewords generation

- To make offline codewords **representative**, we generate corresponding offline codewords for various types (Climate, Cosmology, Molecular, and Physics) of datasets

- When encounter a new type of dataset:
  (1) first use the average offline codewords
  (2) add the offline codewords of this new type into our offline codewords repository for future uses

- Steps for offline codewords:
  (1) let our compressor have a similar compression ratio on different datasets under the same type
  (2) collect symbol frequencies on different datasets under the same type
  (3) calculate the average symbol frequencies from collected frequencies
  (4) use average symbol frequencies to generate offline codewords

Design Methodology

➢ Fixed compression ratio mode

- Assume bit-rate of symbol after Huffman encoding. P is the probability of given code s_i, L is the length of given code s_i

$$\text{mean}(L) = \sum_{i=0}^{n} P(s_i)L(s_i) \approx \sum_{i=0}^{n} P(s_i)\log_2 P(s_i),$$

- Doubling the error bound to $2eb$, total number of symbols is reduced by 2x and the possibility of each symbol is increased by 2X
- New bit-rate should be:

$$B' = \sum_{i=0}^{n/2} P'(s_i)\log_2 P'(s_i) \approx \sum_{i=0}^{n/2} (P(s_{2i-1})\log_2 P'(s_{2i-1}) + P(s_{2i})\log_2 P'(s_{2i})) - 1 = B - 1$$

➢ Observation

- Doubling the error bound, the bit-rate should increase by 1.

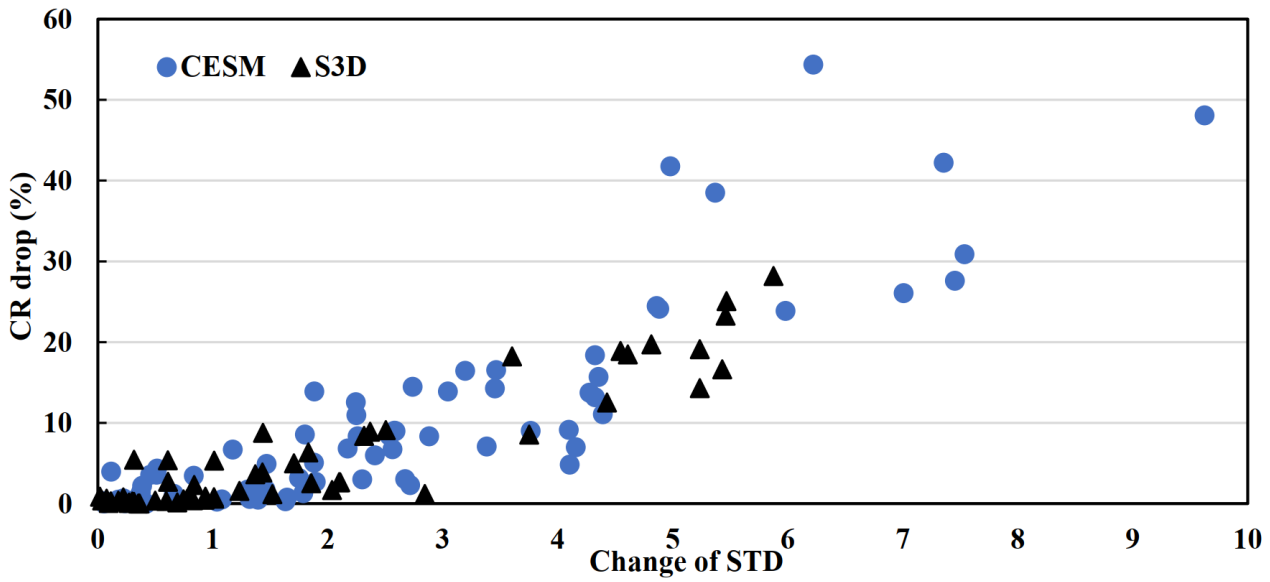
Design Methodology

➢ Adaptive online codewords updates:

- Frequently update the codewords
  Benefit: May obtain high compression ratio
  Drawback: decrease throughput, overhead of codewords
- Symbols generated present a centralized and symmetric distribution
- Use STD to evaluate the similarity of two sets of symbol frequencies
- $\sigma 0$ is the STD of previous data chunk, $\sigma 1$ is the STD of data chunk, and $\chi = |\sigma 0 - \sigma 1|$

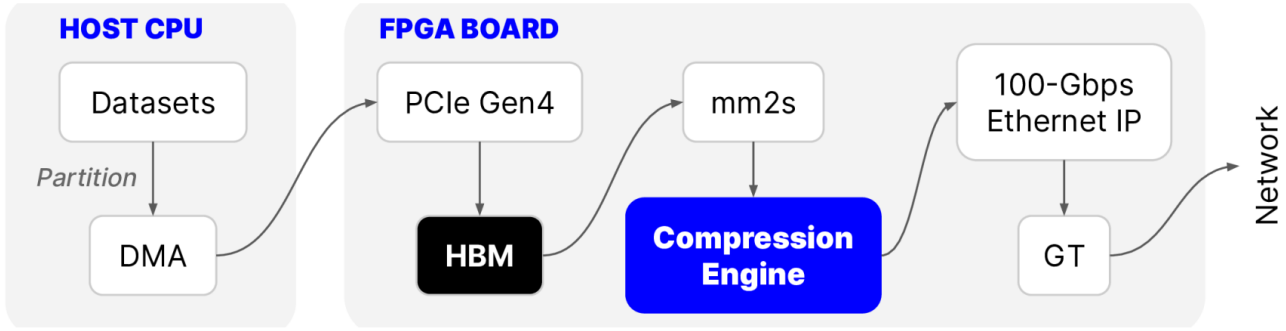
➢ Updates strategy.

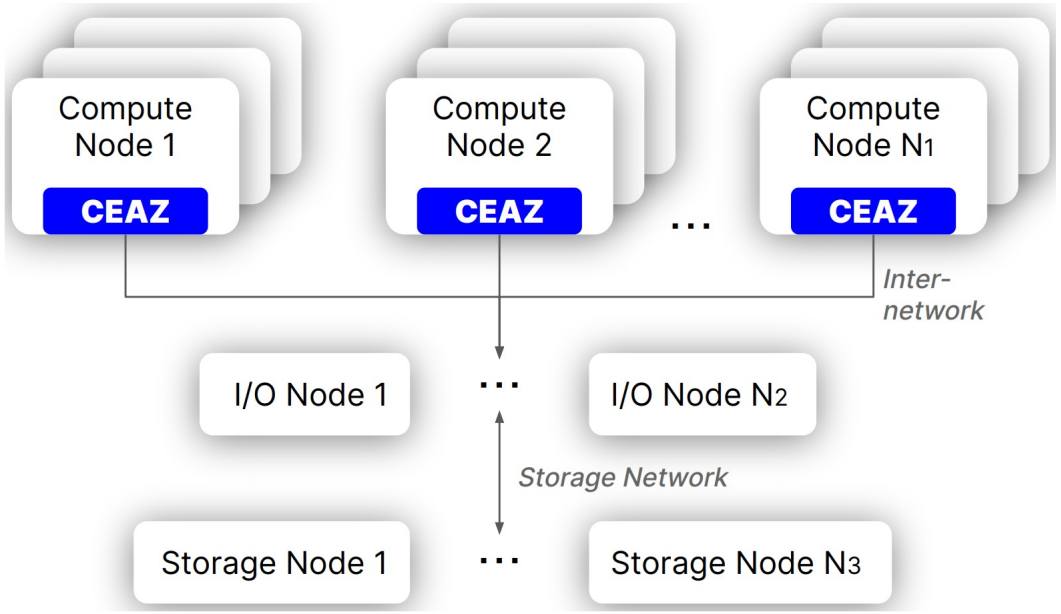
- Not generate new codewords if $\chi \leq \tau 0$ (two frequencies with similar distributions generate almost identical codewords)
- Generate new codewords if $\tau 0 < \chi \leq \tau 1$
- Use the offline Huffman codewords if $\chi \geq \tau 1$ $\chi \geq \tau 1$



Parallel I/O Accelerator

- **Integrate CEAZ into system.**
- Many scientific applications need dump a huge amount of raw simulation data to the storage
- Even though using InifniBand interconnect (e.g., 200 Gb/s), it can take hours to complete





Evaluation

➢ Experimental Setup

- Datasets: Six real-world datasets from the Scientific Data Reduction Benchmarks
- Platforms: (1) Xilinx Alveo U280 Data Center accelerator card
  (2) Summit supercomputer for I/O simulation
  (3) NVIDIA Tesla V100 GPU for comparison with cuZFP/cuSZ

type	name	# fields	precision	dimensions	size
Climate	CESM	77	float	1,800×3,600	1.86 GB
Cosmology	HACC	6	float	280,953,867	6.28 GB
	NYX	6	float	512×512×512	3.0 GB
Molecular	NWChem	3	double	1,617,048,176	12.05 GB
Physics	S3D	55	double	500×500×500	51.22 GB
Other	Brown	3	double	33,554,433	0.75 GB

Our tested datasets

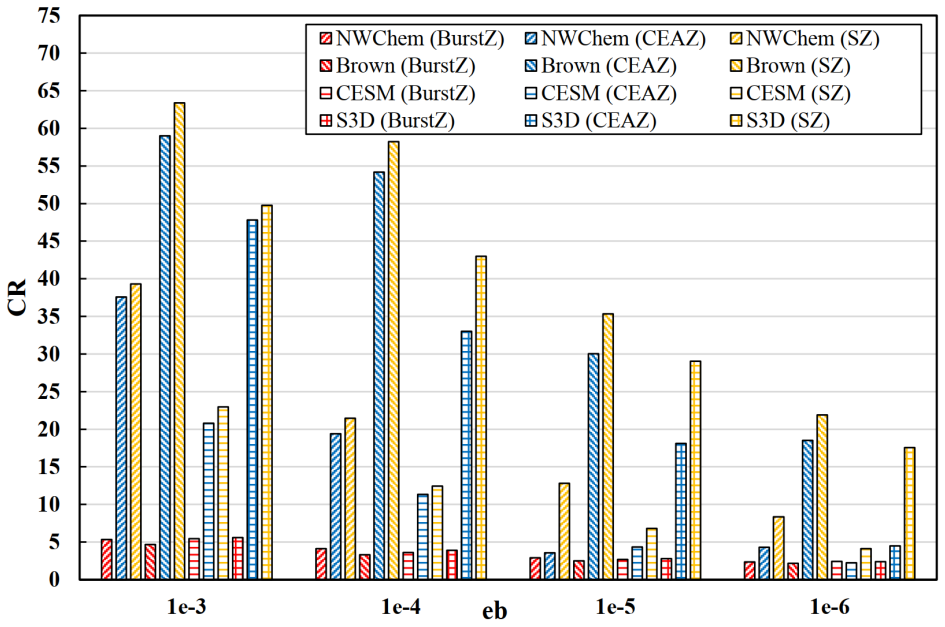
Evaluation

➢ Compression Ratio

- Compared to the BurstZ, CEAZ provides 1.2X~16.4X higher ratio
- Compared to the CPU-SZ, the degradation of ratio is within 23.3%

➢ Distortion (PSNR)

- Compared to CPU-SZ, the degradation of PSNR is within 4 dB
- all PSNRs are higher than 60 dB



Compression ratio

eb	NWChem		Brown		CESM		S3D	
	SZ	CEAZ	SZ	CEAZ	SZ	CEAZ	SZ	CEAZ
1e-3	65.8	75.1	64.7	64.8	65.6	65.4	71.2	68
1e-4	85.8	90.4	84.8	84.8	85.4	84.8	88.8	84.9
1e-5	105.6	107.7	104.8	104.8	105.4	105.3	108.2	104.8
1e-6	125.0	126.0	124.8	124.8	125.5	125.3	127.7	124.8

PNSR

Evaluation

- **Compression ratio and throughput**
- Compared to **lossless** compressors, CEAZ can obviously increase compression ratio and throughput

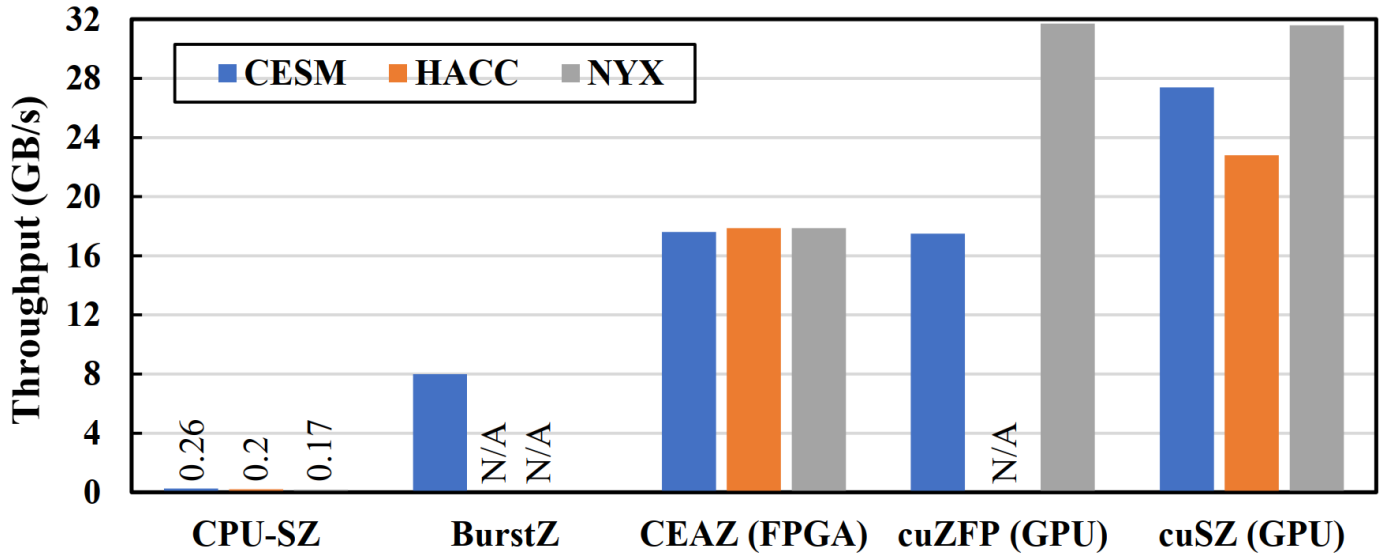
	eb	NWChem	Brown	CESM	S3D	throughput
LZ4	N/A	1.005	1.003	1.182	1.055	1.43
Gzip	N/A	1.056	1.442	1.361	1.181	1.14
CEAZ	1e-4	20.4	58.2	12.3	35.0	17.8
CPU-SZ	1e-4	21.5	58.2	12.5	43.0	0.31

Compression ratio and throughput

Evaluation

➢ Comparation with GPU compressors

- The throughput of CEAZ is 2.3x higher than BurstZ on average
- The throughput of CEAZ is about 56%~78% of cuSZ/cuZFP's throughputs



Throughput comparison among BurstZ, CEAZ, CPU-SZ

Evaluation

➢ **Comparation with GPU compressors**

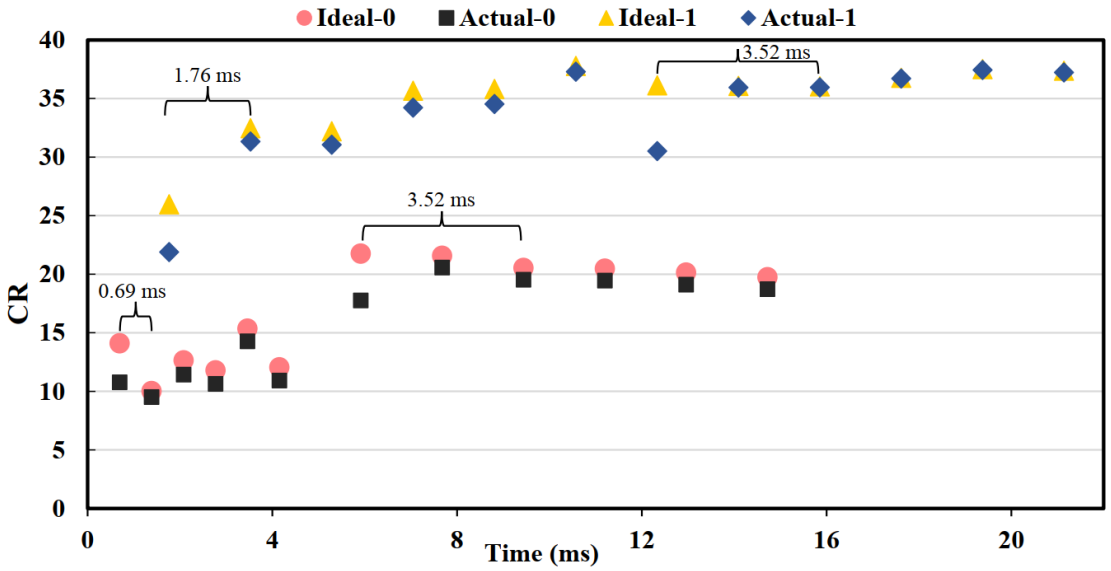
- V100 provides up to 900 GB/s bandwidth (1.96x higher than U280) and 1.25 GHz frequency (4.2x higher than U280)
- A similar FPGA, Intel S10 NX, has the peak performance of 3.96 TF/s (4.0X lower than V100)
- FPGA-based CEAZ is more efficient in resource utilization than GPU-based cuSZ/cuZFP

	Power	Tech.	BW	Freq.	Peak Perf.
V100	250W	12 nm	900 GB/s	1.25 GHz	15.7 TF/s
U280	225W	16 nm	460 GB/s	0.3 GHz	-

Evaluation

➢ **Robustness**

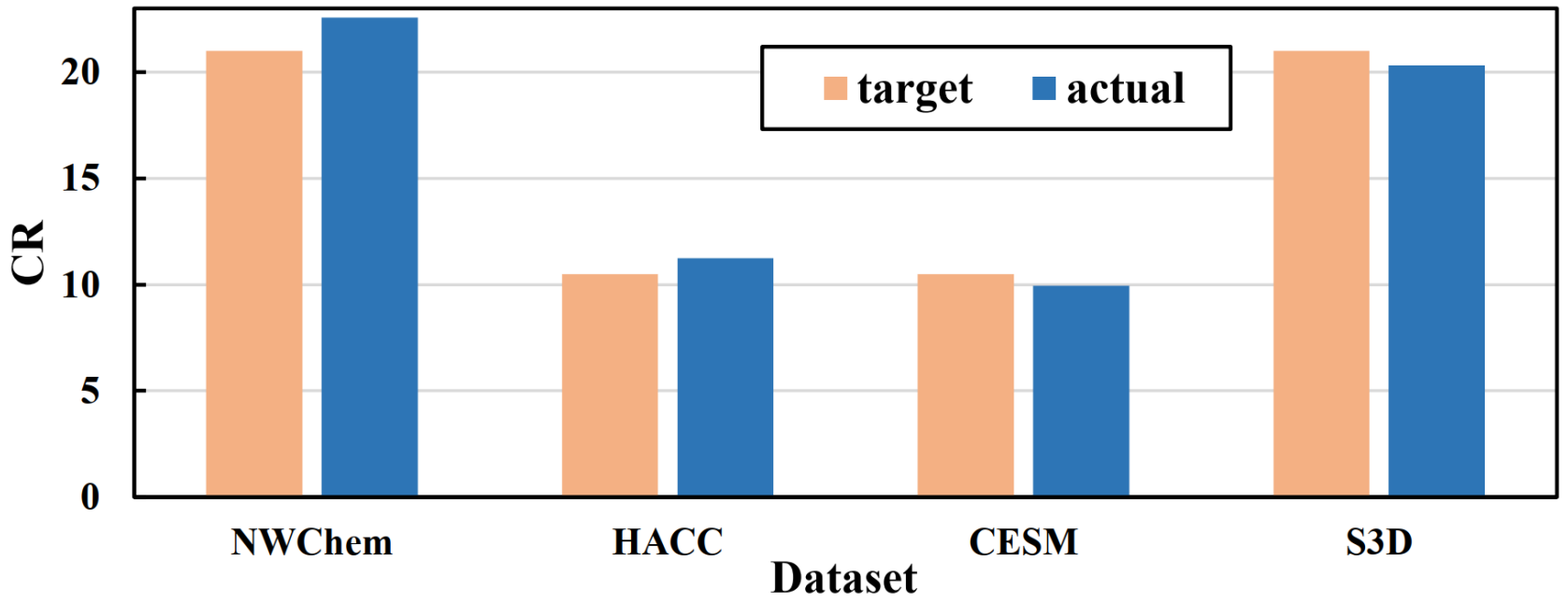
- Codewords change drastically
  (1) offline codewords may be largely different from the actual codewords
  (2) the statistics of input data chunk suddenly change
- Evaluate response time by concatenating two different datasets with different types



Evaluation

- **Fixed-ratio mode**

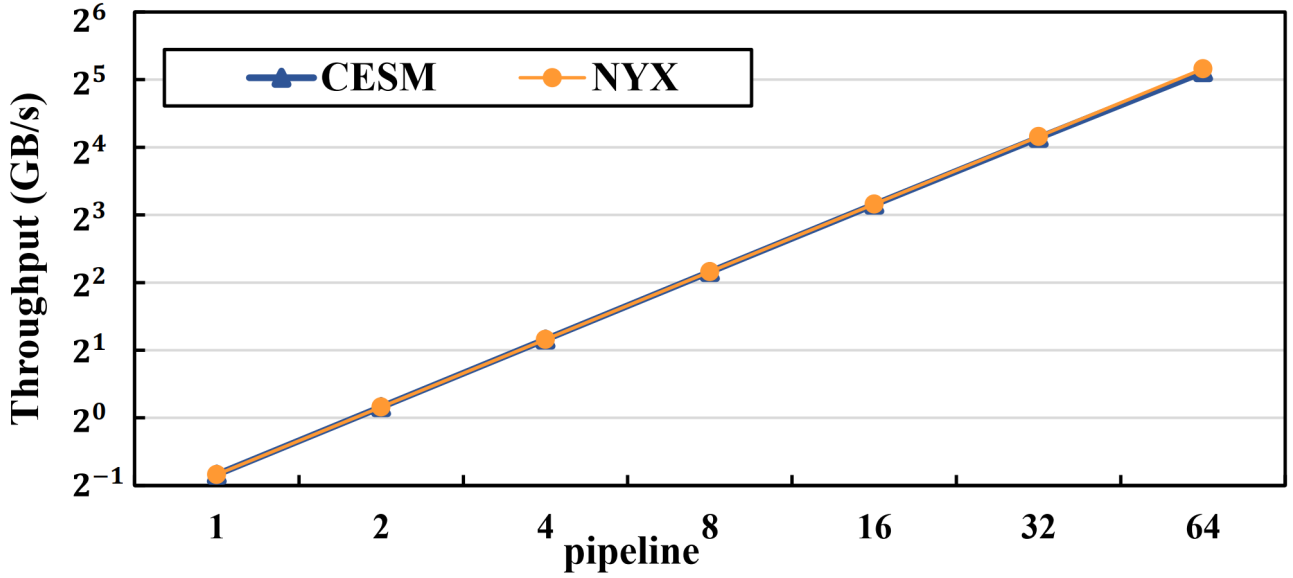
- We set the target compression ratios of 10.5 and 21 for single and double floating-point data, respectively
- Difference is within 7.5%, which is acceptable in our use case



Evaluation

➢ Scalability with multi-pipeline

- Increase the compression pipelines from 1 to 64
- Throughput (in log-scale) increases linearly as the number of pipelines increases
  (1) HMB2 with a very high bandwidth of 460 GB/s
  (2) Adopts dual-quant to remove the data dependency



Compression throughputs with multiple pipelines.

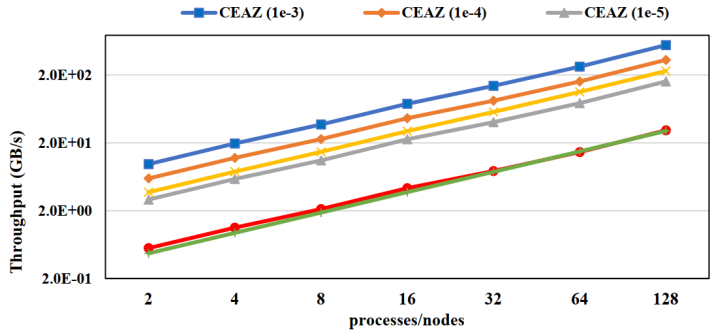
Evaluation

- **Scalability with multi-node**

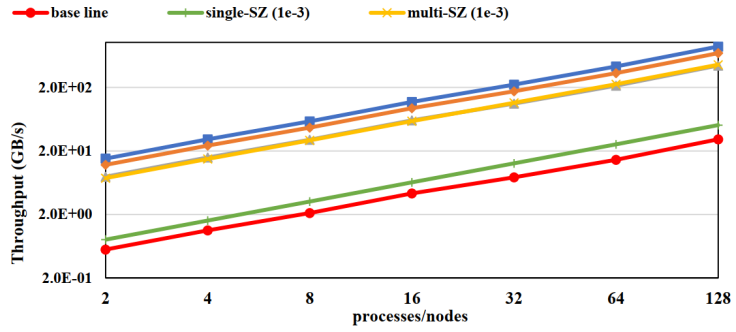
- CEAZ-supported MPI_File_write can improve the overall throughput (including compression time andtime to write compressed data) by 18.0X and 28.9X on NYX and S3D, respectively

- CEAZ-supported MPI_Gather can improve the overall throughput by 21.0X and 37.8X on NYX and S3D

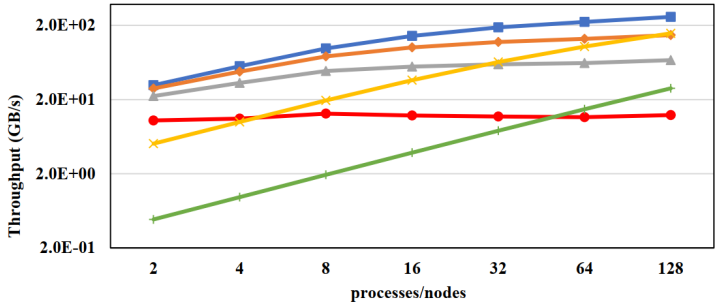
	NYX		S3D	
	CPU-SZ	CEAZ	CPU-SZ	CEAZ
1e-3	26	23.7	49.8	47.8
1e-4	14.7	12.8	43.0	33
1e-5	7.7	5.7	29.1	18.1



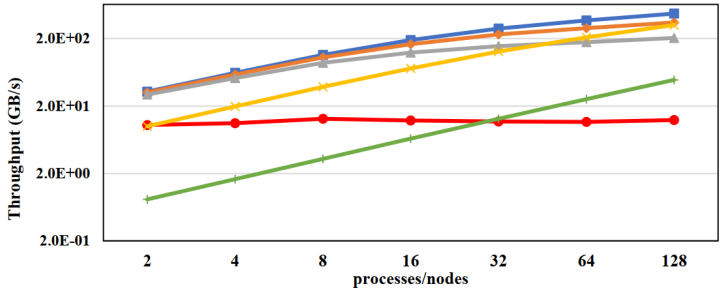
(a) MPI_File_write on NYX



(b) MPI_File_write on S3D



(c) MPI_gather on NYX



(d) MPI_gather on S3D

Conclusion & Future Work

➢ Conclusion

- CEAZ is a hardware-algorithm co-design of efficient and adaptive lossy compressor for scientific data
- It adaptively updates Huffman codewords online based on our offline generated Huffman codewords
- It can accurately control compression ratio
- It outperforms the second-best FPGA-based error-bounded lossy compressor by 2.3X of throughput and 3.0X of compression ratio

➢ Future work

- Deploy our system to FPGA-based clusters, like PNNL's "Junction" cluster
- Extend CEAZ to DPU-based systems

Thank you!

All questions and ideas are welcomed

Contact: Dingwen Tao: dingwen.tao@wsu.edu
Chengming Zhang: chengming.zhang@wsu.edu