A Memory-efficient and Computation-balanced Lossy Compressor on Wafer-Scale Engine

Shihui Song[†], Robert Underwood[‡], Sheng Di[‡*], Yafan Huang[†], Peng Jiang[†], Franck Cappello[‡]
[†] Computer Science Department, University of Iowa, Iowa City, IA, USA
[‡] Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
shihui-song@uiowa.edu, runderwood@anl.gov, sdi1@anl.gov, yafan-huang@uiowa.edu
peng-jiang@uiowa.edu, cappello@mcs.anl.gov

Abstract—**Cerebras system has demonstrated immense potential across various scientific domains. However, modern scientific simulations frequently generate vast volumes of data in a short time, leading to bottlenecks in runtime performance and memory footprint. While an ultra-fast error-bounded lossy compressor can mitigate such limitations with high compression ratios and guaranteed data quality, deploying it into Cerebras dataflow architecture poses significant difficulties. Specifically, Cerebras faces memory challenges, such as the absence of shared memory and limited local memory, alongside computational challenges, including specialized parallelism and sensitivity to imbalanced workloads. In this work, we propose CERESZ-II, an error-bounded lossy compressor that computes within Cerebras system. CERESZ-II addresses these challenges with a carefully optimized four-stage compression workflow, consisting of Pre-quantization, Lightweight Prediction, Fixed-size Huffman Encoding, and Spatial-aware Offset Computation, ensuring both memory efficiency and computational balance. Evaluation of several real-world scientific datasets shows that CERESZ-II achieves over 800 GB/s throughput, delivering high compression ratios and reliable reconstructed data quality.**

Keywords—**Error-bounded Lossy Compression, AI Processor, Dataflow Architecture, Wafer-Scale Engine, Parallel Computing**

I. INTRODUCTION

Cerebras [1], powered by its Wafer-Scale Engine (WSE), is an emerging AI system that shows immense potential across various high-performance computing (HPC) domains [2]–[4]. For instance, researchers from three national laboratories have demonstrated strong scaling of molecular dynamics (MD) simulations on the WSE [3]. They achieved a 179-fold improvement in timesteps per second compared to Frontier, the world's leading GPU-based supercomputer [5], along with substantial gains in timesteps per unit of energy.

Meanwhile, modern large-scale HPC simulations frequently encounter challenges related to high datastream intensity and excessive memory footprints [6]–[8]. For example, the particle detectors at the Large Hadron Collider (LHC) at the Europe Center for Nuclear Research (CERN) must process 1 PB data in one second [9]. Another case is Reverse Time Migration (RTM), an advanced seismic imaging technique, producing up to 2,800 TB of data in memory within only a single timestamp on $10\times10\times8$ KM^3 subterranean structure [10]. In those scenarios, in order to benefit from Cerebras-based HPC simulation, a fast and high-ratio lossy compressor with Cerebras WSE will be crucial for efficiently reducing data size while maintaining ultra-fast processing speeds and promising data quality.

However, designing a lossy compressor that achieves ultra-high throughput, promising compression ratios, and effective error control for data quality is challenging. The idiosyncratic memory patterns and unique parallelism of Cerebras WSE add further complexity to the compression algorithm design, where, unfortunately, none of the existing works sufficiently address those. For example, Shah et al. [11] implemented an AI-chip-based compressor using PyTorch [12] to compress training data, achieving moderate throughput due to the reliance on high-level code rather than low-level kernel implementations. In contrast, CereSZ [13], an error-bounded lossy compressor designed for WSE, leverages low-level features[1]. Yet, it suffers from computational imbalance across processing elements (PEs)[2] due to uneven workloads across data blocks. Furthermore, it neglects the unique parallelism of WSE and employs dataflow-unfriendly computations, limiting both its throughput and compression efficiency.

In this work, we propose CERESZ-II, an error-bounded lossy compressor designed for the Cerebras WSE, to achieve computational balance and memory efficiency, ensuring full compatibility with the Cerebras system. CERESZ-II employs a block-wise design to ensure parallel processing across PEs while accounting for the dataflow architecture of the WSE. Specifically, we introduce *Fixed-size Huffman Encoding* method to reduce local memory usage within each PE while maintaining computational balance across different PEs. Through a comprehensive characterization study on several representative HPC datasets, we build a generic fixed-size Huffman codebook. This codebook is generated through a fully offline process, making it only a one-time cost. Moreover, we propose *Spatial-aware Offset Computation*, an on-WSE parallel prefix-sum method designed to efficiently concatenate

[*]Corresponding author: Sheng Di, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA

[1]This is achieved by using Cerebras Software Language (CSL), a C-style language for modeling dataflow. We will explain it in Section VII-A2.

[2]PE is an independent processor with its own local memory. PE is also the minimal unit in WSE dataflow. We will explain in Section II-B.

compressed data blocks. The main contributions of this work are summarized as below:

- We identify several key challenges in deploying a lossy compressor onto Cerebras WSE, including the absence of shared memory, limited local memory, specialized parallelism, and sensitivity to imbalanced computation.
- We propose CERESZ-II to address the above challenges within Cerebras WSE, including four major steps: *Pre-Quantization*, *Lightweight Prediction*, *Fixed-size Huffman Encoding*, and *Spatial-aware Offset Computation*.
- We conduct comprehensive evaluations across seven representative real-world scientific datasets. On average, CERESZ-II achieves 846.20 GB/s for compression and 846.59 GB/s for decompression, even considering the overhead of offset computation. This represents an 85.02% and 45.63% improvement in compression and decompression throughput, respectively, compared to CereSZ, the state-of-the-art Cerebras lossy compressor. Furthermore, CERESZ-II consistently achieved higher compression ratios than CereSZ in all cases, while maintaining superior reconstructed data quality.

II. BACKGROUND

We provide background information on error-bounded lossy compression and the Cerebras CS-2 system in this section.

A. Error-bounded Lossy Compression

Error-bounded lossy compression is a data reduction technique that confines introduced errors within a user-defined threshold [14]. It can offer a significantly higher compression ratio compared with lossless compression methods while still preserving promising reconstructed data quality [6], [7], [15]. Given a user-defined error-bound eb and original data $D = \{d_1, d_2, ..., d_N\}$, where N denotes the length and each d_i denotes the i-th floating point number, the process of error-bounded lossy compression can be formulated as below: In compression, D is compressed into a byte stream B. The **compression ratio** can be computed as $size_D/size_B$. In decompression, B is reconstructed into $D' = \{d'_1, d'_2, ..., d'_N\}$, guaranteeing $|d_i - d'_i| \leq eb$ for each $i \in \{1, ..., N\}$.

Existing exploration of error-bounded lossy compressor designs can be roughly categorized into two directions: CPU-based compressors [16]–[19] and compressors on heterogeneous processors, including GPU [20]–[23] and emerging AI processors [11], [13]. While CPU-based designs can focus on various aspects, including data quality [24], [25], compression ratio [26], [27], or throughput [28], compressors on heterogeneous processors, driven by the massive parallelism of these architectures, *primarily aim to achieve ultra-fast throughput while maintaining an acceptable compression ratio* [13]. With intrinsic error control and this satisfied compression ratio, the quality of the reconstructed data can be reliably ensured. Note that the **throughput**, routinely measured by GB/s for ultra-fast compressors [13], [20], [21], [29], [30], is defined as the volume of data that can be processed during a certain period.

B. Wafer-Scale Engine (WSE) in Cerebras CS-2 System

Cerebras CS-2 is a powerful system for accelerating machine learning training [31] and scientific simulations [2], [3], [32]. Its exceptional performance is driven by the **Wafer-Scale Engine (WSE)**, which consists of a 2D grid of 757×996 **processing elements (PEs)**. Figure 1 illustrates the architecture of the WSE and the PEs. Each internal PE is connected to its neighboring PEs in four directions, with communication facilitated by a Fabric Router. Each PE is also equipped with a local processor and 48 KB of SRAM, linked to its Fabric Router. Operating at 850 MHz, a PE can transmit a 32-bit message, or wavelet, per clock cycle through its own router link to adjacent PEs, while reading up to 128 bits from memory and writing up to 64 bits in the same cycle. This design enables rapid data processing, intensive parallelism, and efficient exchange of information across the entire WSE.

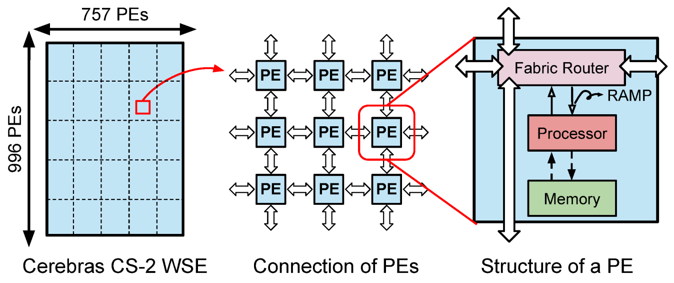


Fig. 1. An overview of Cerebras CS-2 WSE and connection between PEs. Note that adjacent PEs are communicated by their Fabric Routers.

Cerebras WSE also features a **dataflow** control characteristic. In this architecture, tasks are dynamically selected based on the program running on the WSE. Once tasks are bound to their corresponding colors, their execution sequence is determined by the arrival order of the wavelets of the colors. Note that "color" here can be understood as a unique routable identifier of a specific task. This allows a flexible, asynchronous execution model, where tasks can be activated either at compile time or triggered by other tasks. For deploying a program, such as an error-bounded lossy compressor, on Cerebras CS-2, it is crucial to *adapt its algorithm with dataflow designs and achieve compatibility with WSE system.*

III. KEY CHALLENGES

Our goal is to propose an error-bounded lossy compressor for the Cerebras CS-2 system. However, as discussed in Section II, designing such a compression algorithm is non-trivial, requiring (1) aligning with the WSE dataflow architecture and (2) achieving ultra-high throughput as well as promising compression ratios. In this section, we outline two key challenges to achieve this goal, which also motivates the compression algorithm design in the subsequent sections.

A. Memory Constraints

Absence of Shared Memory: In NVIDIA GPUs, memory is managed in a hierarchical manner, where a group of threads has the same shared memory and all threads share the same global memory [33]. In comparison, in Cerebras CS-2 system, there are no such shared memory units across different PEs. This configuration indicates that PEs cannot directly access a unified memory pool, thereby complicating tasks that rely on global coordination, such as device-wide prefix-sum [34]–[36] in block-wise lossy compressors [21], [22], which determines the location of each compressed block in the final compressed array. This limitation requires a novel design for performing synchronization across different PEs.

Limited Local Memory: Recall that, in Cerebras CS-2 system, each PE possesses 48 KB of local memory. While this local memory is sufficient for lightweight compression algorithms (e.g. fixed-length encoding [22], [37]) that operate at block granularity, it presents challenges for encoding methods that require substantially more memory. One notable case is Huffman encoding [38]. Although Huffman encoding is highly effective at reducing data size, it requires much more memory to store its codebook. For example, given a dataset containing 100,000 unique integers (32 bits for storing one integer in computer systems), with an average Huffman code length of 10 bits, the required memory for storing the Huffman codebook can be calculated by $100,000 \times 32 + 100,000 \times 10 = 4,200,000 \approx 513$ KB. With no shared memory units available, this generated codebook must be stored in the local memory of each PE. Unfortunately, such memory requirement far exceeds the capacity of an individual WSE PE. Therefore, it is essential to propose a new design to fit existing encoding schemes within the available memory, more importantly, with minimized degradation of compression performance.

(***C-1***): In Cerebras CS-2 system, the absence of shared memory across PEs and limited local memory in each PE complicate existing compression algorithm designs, such as global coordination and Huffman encoding.

B. Unique Computation Patterns

Specialized Parallelism: While both GPUs and Cerebras CS-2 system utilize parallel processing to enhance runtime performance, the nature of their parallelism is quite different. In a WSE PE, the processor has more capable control units than an individual GPU thread. Coupled with its petabyte-level memory throughput [1], a PE can perform frequent memory accesses and handle tasks that involve both control-flow and data-intensive operations more effectively than a GPU thread, which is primarily optimized for arithmetic operations. For example, the most recent GPU lossy compressor cuSZp [22] transposes the target bit matrix before storing it in global memory. While this step introduces extra bit-shifting operations, it simplifies control-flow logic and increases throughput on GPUs. However, due to the strong control-flow capabilities within PE, this step is unnecessary in WSE. Using a field from CESM-ATM [39], a standard climate dataset, as an example, omitting transposition increases throughput by 80.45% compared to using transposition on Cerebras CS-2 system.

Sensitive to Imbalanced Computation: Cerebras WSE is highly sensitive to computational balance. Specifically, for any computation task on WSE, the results can only be retrieved after all PEs have completed their processing. This limits the throughput of existing parallel compression designs. For instance, a recent Cerebras-oriented compressor, CereSZ [13] adopts fixed-length encoding as its core compression algorithm. During this process, each data block preserves the same number of bits for all its integers, with the bit count determined by the maximum absolute value in the block. However, data patterns in HPC datasets can vary significantly [6]. Some blocks may contain large absolute values requiring more intensive computations, while others may consist entirely of zeros, necessitating minimal processing. This variance leads to an imbalance, where the overall system throughput is constrained by the PE that finishes last. Considering balance is critical in designing compression algorithms for the Cerebras WSE.

(***C-2***): The ability to handle complex control-flow but sensitivity to imbalanced computation make existing designs inefficient on WSE, yet offers new opportunities for a possible ultra-fast compression solution.

IV. CERESZ-II: OVERVIEW OF COMPRESSION ALGORITHM

In this work, we propose CERESZ-II, an error-bounded lossy compressor specifically designed for Cerebras CS-2 system, targeting ultra-fast throughput and high compression ratios.

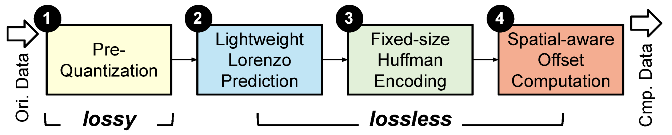

Ori. Data
1
Pre-Quantization
2
Lightweight Lorenzo Prediction
3
Fixed-size Huffman Encoding
4
Spatial-aware Offset Computation
Cmp. Data
lossy
lossless

Fig. 2. A high-level overview of the compression algorithm in CERESZ-II.

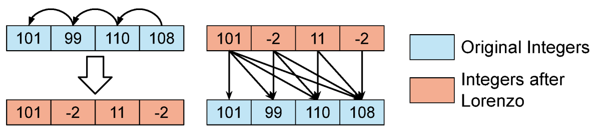
A high-level overview of CERESZ-II compression algorithm is explained in Figure 2. Given an HPC dataset, CERESZ-II processes the input array in a 1D fashion, dividing it into data blocks of equal size (i.e., consecutive floating-point numbers). Within each data block, CERESZ-II compresses the data via four major steps.

(❶) *Pre-Quantization*: This step converts each floating-point number into a 32-bit integer within the proposed error bound, facilitating later lossless compression steps. Assuming the original data block is $\{d_1, d_2, ..., d_L\}$, where d_i is the i-th floating-point number and L denotes the block length, if error bound is eb, ❶ will convert this block into a set of integers $\{n_1, n_2, ..., n_L\}$ by $n_i = \text{round}(d_i/2eb)$. This lossy operation ensures that the reconstructed data $d'_i = n_i \times 2eb$ can satisfy $|d'_i - d_i| \leq eb$.

(❷) *Lightweight Lorenzo Prediction*: Similar to delta compression [40], this step captures data smoothness – a common feature in HPC datasets – by recording the difference between consecutive quantized integers. An example is shown in Figure 3. For the integer 99, instead of storing its actual value, we record the difference between it and its predecessor as $99 - 101 = -2$. It helps improve the efficiency of later compression stages.

(❸) *Fixed-size Huffman Encoding*: Unlike traditional Huffman encoding [38], this step applies lossless compression using a pre-defined Huffman codebook, which is generated based on a thorough characterization study on several HPC datasets. This approach allows the algorithm to run efficiently on each WSE PE, despite the limited local memory, without sacrificing compression ratio effectiveness. Note that the codebook is built fully offline, making it only a one-time cost.

(❹) *Spatial-aware Offset Computation*: Following ❸, each data block is compressed to varying lengths. Merging all compressed blocks into a single, unified byte stream (i.e. output of CERESZ-II) can be formulated as a parallel prefix-sum problem [34]–[36]. This step is designed to efficiently perform this operation on the WSE, where there is no shared memory, while considering the unique dataflow architecture of the Cerebras CS-2 system. Since ❸ and ❹ are key contributions of this work, they will be explained in details in Section V and VI, respectively.



(a) Compression (b) Decompression

Fig. 3. Illustrating the Lightweight Lorenzo Prediction in compression phase (a) and its reversed computation in decompression phase (b).

CERESZ-II effectively addresses the challenges of deploying a fast lossy compressor on the Cerebras WSE. In ❸, the prebuilt fixed-size codebook is optimized for the limited local memory of each PE. In ❹, global coordination is achieved through spatial PE-level computations, avoiding shared memory usage while considering throughput and dataflow computation patterns. These designs overcome the memory constraints in WSE (i.e. ***(C-1)*** in Section II). From the computation perspective, CERESZ-II leverages block-wise operations to exploit the parallelism of the WSE. Additionally, the Huffman-related computations, though requiring more complex control-flow operations, are well within the processing capabilities of each PE. Moreover, CERESZ-II balances the computation overhead between data blocks containing irregular large quantized integers and those consisting entirely of zeros – we will explain more details in Section V. These features ensure the computation compatibility with WSE (i.e. ***(C-2)*** in Section II).

V. FIXED-SIZE HUFFMAN ENCODING (❸)

As discussed in Section III, in Cerebras WSE, existing lossless encoding methods either exceed local memory limits (Huffman encoding) or cause imbalanced computation issues (fixed-length encoding). To address these challenges, we propose the Fixed-size Huffman Encoding method. The key idea is to heuristically create a static Huffman codebook with a fixed tree size, ensuring it fits within local memory and balanced computation. Building this codebook is an offline process, making it a one-time cost. To explain this method in detail, we will introduce the principles of Huffman encoding, define key concepts (e.g. Huffman tree and codebook), explain how we select the Huffman tree size, and outline the process of building the codebook using a characterization study.

A. Principle of Huffman Encoding

Huffman encoding [38], [41] is a lossless compression algorithm that assigns shorter codes to more frequent characters, optimizing compression ratio based on frequency. Figure 4 shows an example of Huffman encoding for integers after the Lightweight Lorenzo Prediction. Given an input array, there are mainly four steps to encode it using Huffman encoding: *Step1:* Identify unique values in the array and compute their frequencies. *Step2:* Build the **Huffman tree** using the Min Heap algorithm, which repeatedly merges the two smallest frequencies into a single node until one root remains. *Step3:* Assign '0' for left branches and '1' for right branches to generate unique Huffman codes, forming the **codebook**. *Step4:* Encode the input array using the codebook. Since directly using Huffman encoding on Cerebras WSE can easily exceed local memory limits, we aim to build a generic codebook that captures HPC dataset features, fully offline. To achieve this, we select a fixed size and propose a method for generating the codebook – this is also the approach detailed in this section.

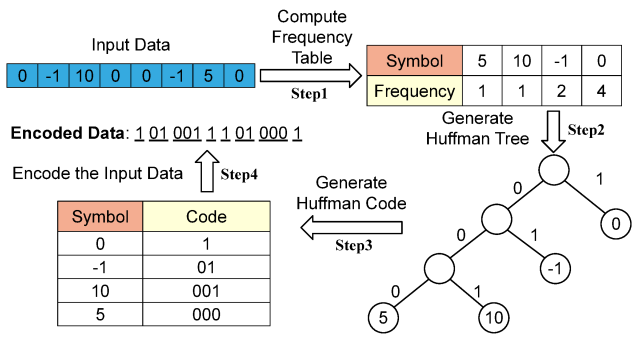


Fig. 4. An example of Huffman encoding.

B. Huffman Tree Size Selection

To fit within the local memory of each PE, a straightforward approach is to construct a fixed Huffman tree by encoding only the most frequent unique values. For integers that exceed the codebook size, we simply retain their original bit patterns without compression. Figure 5 shows an example of how we encode a data block using a fixed-size codebook. For each integer in the block, if it is found in the codebook, it is considered predictable, represented by a leading '1' bit followed by its corresponding Huffman code from the codebook. If the integer is not in the codebook, it is deemed unpredictable and marked with a leading '0' bit, followed by its original 32-bit representation. All encoded bits are then concatenated into 32-bit signed integers, with the first element reserved to indicate the total number of bits used in the block.

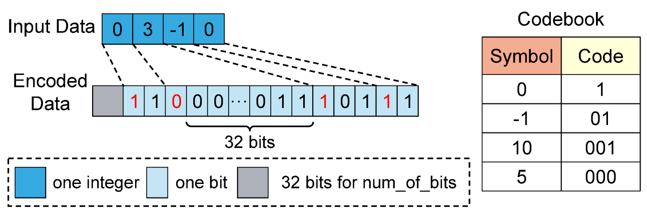


Fig. 5. An example of fixed-size Huffman encoding.

We examine datasets in different fields from various HPC domains to determine the optimal Huffman tree size. For each field, Huffman trees are constructed and evaluated at sizes from 50 to 4,000, and we analyze their impact on the compression ratio. The impact on runtime throughput and data quality is not considered, as the throughput remains consistent in our implementation. Since Huffman encoding is lossless, it does not affect data quality. The experiments involve AEROD_v_1_1800_3600.dat field of CESM-ATM [39], Pf48.bin.f32 field of Hurricane [42], vx.f32 field of the HACC [43], and field einspline_115_69_69_288.f32 of QMCPack [44] with relative error bound (REL) 1E-4[3]. The results are shown in Figure 6.

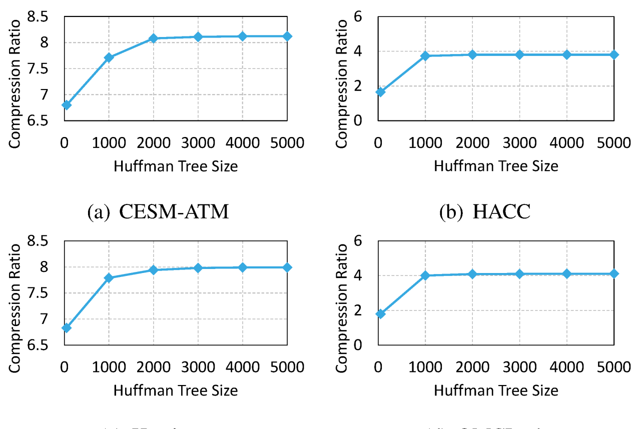


Fig. 6. Compression ratios with Huffman trees of varying sizes: 50, 1000, 2000, 3000, 4000, and 5000.

We can observe that increasing the Huffman tree size from 50 to 1000 significantly boosts the compression ratio across all datasets. Enlarging the tree size from 1000 to 2000 notably enhances the compression ratio for the CESM-ATM and Hurricane datasets, while only yielding a marginal increase for the other two datasets. With a Huffman tree size greater than 2000, the compression ratio shows negligible further increases for all the datasets. Similar observations can be obtained in other fields of the datasets as well. Additionally, in our implementation, we use two arrays to store the Huffman code lengths and the codes themselves as signed integers. Storing the codebook requires only $2000 \times 2 \times 4/1024 = 15.625$ KB, which is manageable within the local memory of one PE (i.e. 48 KB). Therefore, we choose a Huffman tree size of **2000**.

[3]The definition of REL error bound will be explained in Section VII-A4

C. Static Huffman Codebook Generation

Although we know the Huffman tree size, building a codebook requires examining value frequencies, an online process that significantly reduces runtime performance. This raises a natural question for us: can we pre-build a codebook that is generally suitable for all HPC datasets without sacrificing compression ratio, and make this step an offline process? To answer this, we first characterize the data patterns of several representative HPC datasets. Figure 7 presents the distribution of quantized integers (after Lorenzo Prediction) and their MLE lines for two datasets QMCPack [44] and HACC [43]. The error bounds are set at REL 1E-2 and REL 1E-3. The distribution is restricted to the value range of $[-20, 20]$, as the frequencies for values outside this range are significantly lower and can be neglected. As seen, these distributions closely resemble a Gaussian distribution [45], centered around a mean of zero, and the MLE line demonstrates a robust fit to the theoretical distribution. Increasing the error bound from REL 1E-3 to 1E-2 results in a sharper curve, further condensing values near the center.

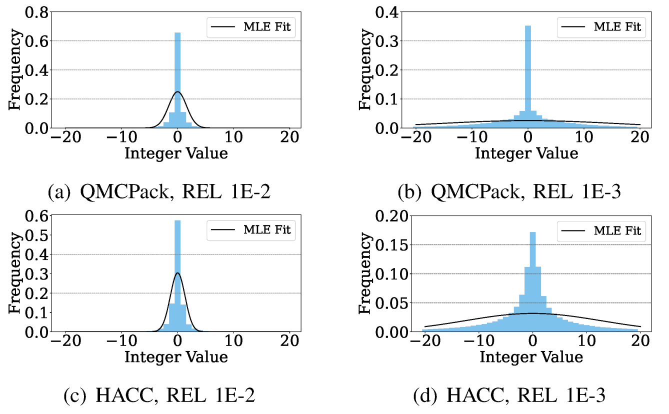


Fig. 7. Distribution of integer values after Lorenzo Prediction on einspline_115_69_69_288.f32 field of QMCPack and vx.f32 field of HACC.

Based on the observation, we propose a proportional method to characterize the distribution of values across multiple datasets. Suppose we have N datasets, for each dataset D_i, we aggregate the Lorenzo data across all fields and generate the distribution P_i. We utilize REL 1E-4 because it results in the sharpest distribution compared with REL 1E-2 and 1E-3, capturing richer information. Then we calculate the average of the distributions by using the following formula:

$$P = \sum_{i=1}^{N} \frac{P_i}{N} \tag{1}$$

We also construct the static Huffman codebook based on this updated distribution. Given the distribution's resemblance to a Gaussian distribution centered at zero, we select a symmetric value range for constructing the Huffman codebook. Consequently, our algorithm encodes values from -1000 to 999, utilizing a Huffman tree size of 2000.

In CERESZ-II, we select three HPC datasets, including CESM-ATM [39], Hurricane [42], and QMCPack [44] to build the static codebook and evaluate its generalization capability using four fields from other four datasets. The details of these datasets will be explained in Section VII-A3. We measure compression ratios by comparing our static method with the traditional Huffman encoding (named Dynamic), which is the optimal codebook by examining the frequency inside each field while considering the fixed size of 2000. The results are presented in Table I. We can see that the compression ratio of our prebuilt codebook is very close to the optimal one (e.g. 29.84 and 29.94 in NYX), demonstrating the effectiveness of our approach in maintaining compression ratio efficiency. Our conclusion remains the same when randomly selecting three of the seven datasets.

Dataset *(Field)*	**NYX** [46] *(b.y._dens.)*	**RTM** [47] *(p._1000)*	**SCALE** [48] *(PRES)*	**HACC** [43] *(vx)*
Static	29.84	25.47	9.09	3.22
Dynamic	29.94	25.59	9.77	3.80

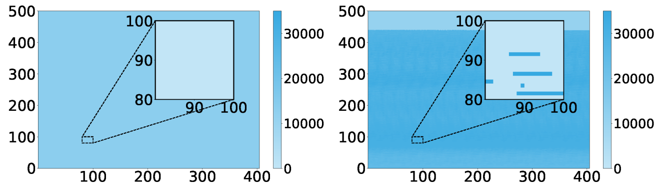
TABLE I
Compression ratios between our prebuilt (i.e. Static) and the optimal (i.e. Dynamic) codebooks in four fields from four HPC datasets.

One major challenge in deploying a compression algorithm on Cerebras WSE is the need for computation balance (i.e. ***(C-1)*** in Section II). Imbalanced computation is also the primary limitation of an existing Cerebras compressor [13], which uses fixed-length encoding. To evaluate computation balance, we compare the execution cycles of our proposed Fixed-size Huffman Encoding with those of fixed-length encoding, using the AEROD_v_1_1800_3600.dat field from CESM-ATM [39] as an example. Similar results are observed in other datasets. With 6,480,000 floating points in this field (i.e., ~20K data blocks of length 32 for each), we utilized a 405×500 PE mesh, where each PE processes one data block. The results, shown in Figure 8, depict a heatmap where darker colors indicate a higher number of execution cycles per PE, and the X/Y axes represent PE indices. Although both encoding methods exhibit close total cycles, the right side (fixed-length encoding) shows a significant imbalance – forcing the entire WSE to wait for the slowest PE to complete. In contrast, our method, which assigns bit representations from a prebuilt codebook, ensures a more balanced computation across PEs. This demonstrates that our proposed Fixed-size Huffman Encoding is better suited for Cerebras WSE. More details are provided in Section VII.

VI. SPATIAL-AWARE OFFSET COMPUTATION (❹)

A. Motivation

In ❹ of Figure 2, CERESZ-II compression concatenates the compressed data from multiple blocks into a single array, which can be formulated as a parallel prefix-sum problem. An example illustrating this is provided in Figure 9. In parallel compression, each data block is compressed simultaneously. However, due to varying data patterns, the compressibility of each block differs under a given algorithm, resulting in different compressed data lengths. To concatenate these blocks



(a) Fixed-size Huffman Encoding (b) Fixed-length Encoding

Fig. 8. Heat map of computation overhead (measured by cycle) between our Fixed-size Huffman Encoding (a) and fixed-length encoding (b). The X and Y axes denote the PE index, and the darker color indicates a greater cycle.

in parallel, their locations in the final compressed byte array (defined as **Offset** in this section) need to be calculated. As seen, the starting and ending locations of the j-th block can be calculated as $\sum_{i=0}^{j-1} num_i$ and $\sum_{i=0}^{j} num_i$, respectively.

However, existing Cerebras-oriented lossy compressors, such as CereSZ [13], neglect this problem by performing these computations in CPU. While this approach is straightforward, the inherent linear recurrences make this step extremely slow and difficult to parallelize, becoming a major bottleneck in runtime throughput. Additionally, the unique mesh-like parallelism and the absence of shared memory in WSE introduce new challenges for this process. These motivate us to propose a new solution for computing offsets within CERESZ-II.

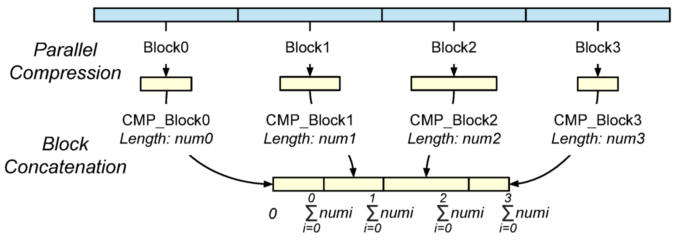


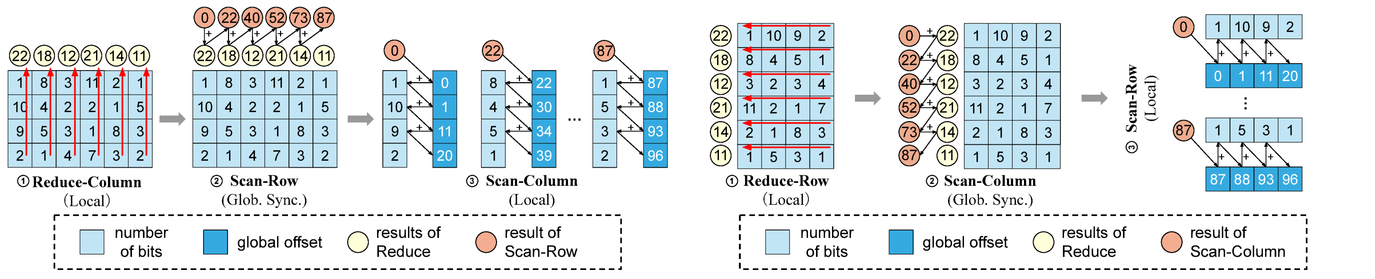
Fig. 9. Illustrating the compressed data block concatenation of a parallel lossy compressor is a parallel prefix-sum problem.

B. Spatial-aware Offset Computation in Cerebras WSE

Spatial-aware Offset Computation[4] calculates the offset for each compressed data block and stores it in the corresponding PE. This proposed approach does not require shared memory and leverages the parallelism of the WSE, ensuring high throughput in the CERESZ-II compression workflow. Given an HPC dataset processed by a mesh of $W \times H$[5] PEs in Cerebras WSE, this step can be performed within three stages after data is compressed and ready to be concatenated. *(1) Reduce-Column/Row (Local)*: For each column/row, aggregate offsets from all PEs and store this value in the first PE in column/row. This is a local reduce operation and is processed in parallel. *(2) Scan-Row/Column (Glob. Sync.)*: After obtaining the local

[4]This step is based on two computation patterns. (1) *Reduce* aggregates all offsets into one value. For example, an array $\{a_1, a_2, ..., a_N\}$ will be reduced into $\sum_{i=1}^{N} a_i$. (2) *Scan* has the same meaning with prefix-sum. In general, we focus on exclusive scan in compression tasks. Given an array $\{a_1, a_2, ..., a_N\}$, the output of it is $\{0, a_1, ..., \sum_{i=1}^{N-1} a_i\}$.

[5]W and H indicates width and height of the used mesh of PEs.



(a) Offest computation when width is greater than height (column-major). (b) Offest computation when height is greater than width (row-major).

Fig. 10. Illustrating the idea of spatial-aware offset computation, where each square with a number refers to a PE with total compressed block length on it.

reduced offsets, the first row/column then performs a scan operation to obtain the offset across different columns and rows. It is important to note that this global synchronization requires no shared memory and only requires PE-level computations. *(3) Scan-Column/Row (Local)*: For each column/row, the last step distributes the synchronized global offsets and performs a local scan in parallel again. This ensures that all data blocks in every PE are assigned their offset (i.e. location) in the final compressed array. To maximize the runtime throughput, CERESZ-II considers the spatial structure of the utilized mesh of PE. If $H > W$, parallelizing across rows (i.e. treating each row as a local computation) provides greater throughput benefits, so CERESZ-II will adopt row-major parallelism and perform global synchronization along columns. Conversely, if $W > H$, this step will prioritize column-major parallelism with global synchronization across rows.

We further illustrate this process with a running example. The column-major computation is explained in detail in Figure 10(a), while the row-major computation, shown in Figure 10(b), follows a similar principle and hence will not be discussed here. Each square with a number represents a PE along with the total length of the compressed block stored in it. First, each column performs reduction and stores the value in the first PE. As seen, for the first column, $\{1, 10, 9, 2\}$ is reduced into 22. After the reductions within each column are finished, the first row of PEs performs a chained scan to retrieve its global offset. Taking 52 as an example, computed by $22 + 18 + 12$, this represents the total length of the first three columns and serves as the starting location for the fourth column. Finally, each column distributes its aggregated value and performs a parallel scan within the column again, allowing each PE to determine its offset in the final compressed array.

To demonstrate the design of Spatial-aware Offset Computation, we test a prefix-sum problem using different mesh sizes of PEs, with results shown in Table II. A smaller number indicates fewer cycles required on WSE, representing better runtime performance. As seen, when $W > H$, column-major is faster than row-major (e.g., a 150.78% speedup on a 100×50 mesh). Vice versa, when $W < H$, row-major outperforms column-major. Additionally, when $W = H$, column-major is slightly faster due to better alignment with the WSE dataflow, which flows from left to right. For this reason, we prioritize column-major when the $W = H$ in the utilized mesh. Note that, in CERESZ-II, this step is automatically conducted and wrapped within the compression pipeline, for better usage.

PE-Size	50x50	50x100	100x50	100x100	50x200	200x50
Column-major	384	731	384	736	1433	401
Row-major	482	485	963	965	480	1924

TABLE II
Clock cycles for different sizes of PEs using column-major (Figure 10(a)) and row-major (Figure 10(b)) Spatial-aware Offset Computation.

VII. EVALUATION

In this section, we introduce the experimental setups and evaluate CERESZ-II on 7 real-world scientific datasets.

A. Experimental Setup

*1) **Platforms**:* For CERESZ-II and CereSZ, we measure computation time on the Cerebras WSE, We evaluate our algorithm on the Cerebras CS-2 system, of which hardware specification can be found in Section II-B. The system has a 757×996 PE mesh, with up to 750×994 PEs available for computation, as the remaining PEs handle data routing on and off WSE. The CS-2 system supports 24 colors (routable identifiers) for passing wavelets between PEs, and we allocate 8 colors: 2 for data movement and 6 for offset computation. Because the CS-2 has a fixed clock speed [1], we can use hardware cycle counters on each PE to measure runtime, converting counts to seconds by dividing by the clock frequency to obtain runtime.

*2) **Implementation**:* There are two programming interfaces for Cerebras [1]: PyTorch [11] and Cerebras Software Language (CSL) [13] (i.e. a C-like language based on a dataflow programming model). CSL allows for lower-level control of the hardware, leading to higher runtime performance than the PyTorch-based interface. As a result, we implement and test CERESZ-II using the CSL (specifically SDK 0.9).

An example of CSL code for **Reduce-Row**(Local) is shown in Figure 11, where $input\_dsd$ and $output\_dsd$ representing the offsets before and after reduction, and $fabin$ and $fabout$ indicating data received and sent by each PE, respectively. Each PE will run this CSL code: the right-most PEs simply transfer offsets to adjacent left PEs, middle PEs compute a prefix sum by adding offsets received from their right and pass the results leftward, and the left-most PEs store the final results in $output_dsd$ and forward to $fabout$ for host access.

```
fn reduce () void {
  switch (pe_id_width) {
    width-1 => { @fmovs(fabout, input_dsd); },
    0 => { @fadds(output_dsd, input_dsd, fabin);
           @fmovs(fabout, output_dsd); },
    else => { @fadds(fabout, input_dsd, fabin); }
  }
  @activate(activate_global_scan);
}
```

Fig. 11. CSL code demo for performing ① **Reduce-Row** (Local) in 10(b).

One key parameter here is the data block size. Like existing work [13], we set the data block size as 32 in CereSZ-II, balancing compression ratios and runtime throughput. Given an HPC dataset, we distribute it evenly across the PEs before compression. If the dataset is too small to use all PEs, we pad it to a column-major mesh to optimize dataflow.

Datasets	Dims per Field	# Fields	Total Size
CESM-ATM [39]	1800×3600	79	1.91 GB
HACC [43]	280,953,867	6	6.28 GB
RTM [47], [49]	1008×1008×352	3	3.99 GB
Hurricane [42]	500×500×100	13	1.21 GB
QMCPack [44]	33120×69×69	2	1.17 GB
NYX [46]	512×512×512	6	3.00 GB
SCALE [48]	98×1200×1200	13	6.31 GB

TABLE III
Real-world HPC datasets used in this work.

*3) **Datasets**:* We select 7 real-world scientific simulation datasets (single-precision in IEEE-754 standard) from various domains to evaluate CereSZ-II. The details of these datasets can be found in Table III. Most of these datasets come from SDRBench [50] and are widely utilized in recent HPC data reduction works [13], [21]–[23], [51]–[54].

*4) **Experimental Methodology**:* For baseline compressors, we select three state-of-the-art error-bounded lossy compressors from different hardware architectures: SZ3 [55] for CPU, cuSZp [22] for GPU, and CereSZ [13] for the Cerebras CS-2 system. For SZ and cuSZp, we evaluate them with an Intel Xeon Gold 6226R CPU and an NVIDIA A100 (40 GB, 108 SMs) GPU, respectively. Following existing works [13], [21]–[23], we evaluate CereSZ-II and the baselines using three metrics: compression/decompression throughput, compression ratio, and data quality. The first two metrics, defined in Section II-A, are the main focus of CereSZ-II. As for the error setting, we use the value-range-based relative (REL) error bound [16]. Specifically, REL λ ($\lambda \in (0, 1)$) denotes the different between each original data point d_i and its corresponding reconstructed one d'_i should satisfy $|d_i - d'_i| \leq \lambda\text{VR}$, where VR is the value range of this dataset calculated by $d_{\max} - d_{\min}$.

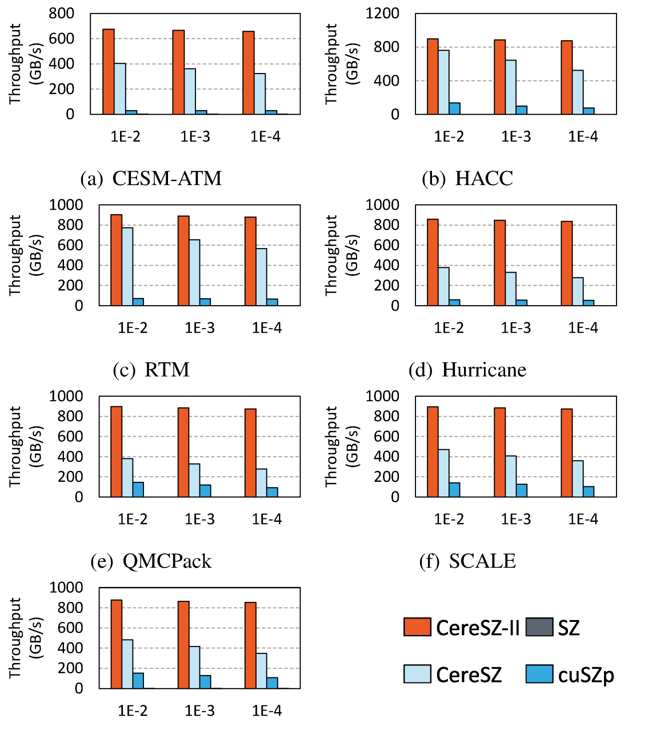
B. Compression and Decompression Throughput

In this section, we evaluate the throughput of CereSZ-II and compare it with other baseline compressors. The compression and decompression throughput is calculated by dividing the raw data size by the measured compression and decompression time (e.g. GB/s). For each dataset, we measure throughput for all fields and report their averages. We use three error settings including, REL 1E-2, REL 1E-3, and REL 1E-4. Such settings are inline with existing works in this literature [13], [22].

Figure 12 reports compression throughput. On average, CereSZ-II has 846.20 GB/s compression throughput, varying from 657.41 GB/s on CESM-ATM dataset with REL 1E-4 to 902.49 GB/s on RTM dataset with REL 1E-2. In contrast, the average compression throughput is only 457.35, 93.63, and 0.18 GB/s for CereSZ, cuSZp, and SZ, respectively. As for the state-of-the-art Cerebras-oriented compressor CereSZ, CereSZ-II consistently outperforms it and achieves 85.02% higher compression throughput. The increased throughput of CereSZ-II is even more impressive because CereSZ-II includes the offset computation for concatenation, which is not performed in CereSZ. CereSZ-II achieves this by addressing the imbalanced issues using our proposed improvements to Fixed-size Huffman Encoding, making the algorithm more suitable for WSE infrastructure.

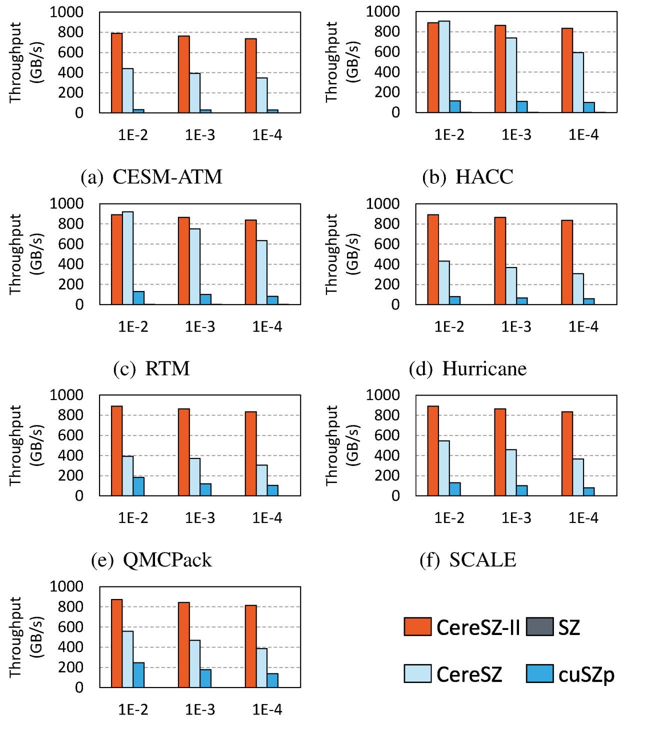
Moreover, we find that the compression throughput of CereSZ-II remains highly stable across different error bounds within the same dataset. For example, in the NYX dataset, the compression throughput under REL 1E-2, 1E-3, and 1E-4 are 876.17, 863.46, and 852.54 GB/s. This can be explained as follows. In Huffman encoding, numbers are compressed by replacing them with their corresponding bit representation from the prebuilt codebook. While we change the error bound from REL 1E-4 to 1E-2, the number of conducted codebook-lookup operations remains unchanged, resulting in stable compression throughput in CereSZ-II. As for GPU compressors, compared with cuSZp, it even achieves up to $\sim$20 times higher throughput on CESM-ATM dataset under REL 1E-3 error bound. However, we agree that comparing the throughput of CereSZ-II with compressors from other platforms is not entirely fair. For instance, the number of transistors differs significantly between the entire Cerebras WSE and an NVIDIA A100 GPU. Nonetheless, this comparison highlights the potential of CereSZ-II in leveraging the unique architecture of Cerebras WSE for high-throughput compression, showing its capacity to perform competitively even across different hardware platforms.

Figure 13 reports decompression throughput. We can see that, similar to the compression phase, CereSZ-II always achieves higher throughput than baseline compressors and is non-sensitive to error bounds in throughput. However, there is one interesting observation. In HACC and RTM datasets under REL 1E-2 error bound, CereSZ-II exhibits slightly lower decompression throughput compared with CereSZ. The reasons are twofold. (1) HACC (a smooth 1D array) and RTM (high sparsity) cause a naturally balanced computation due to their unique data patterns. Meanwhile, the number of maximal fixed-length (i.e. number of bits to preserve) of these two datasets under REL 1E-2 is small, making fixed-length encoding occasionally suitable for WSE. (2) As said, CereSZ does not compute the global offset for each compressed block. Although this can sometimes bring higher throughput, this defect will highly limit it from real-world scenarios if users



(a) CESM-ATM (b) HACC (c) RTM (d) Hurricane (e) QMCPack (f) SCALE (g) NYX (h) Caption

Fig. 12. Compression throughput for CERESZ-II and baseline compressors.



(a) CESM-ATM (b) HACC (c) RTM (d) Hurricane (e) QMCPack (f) SCALE (g) NYX (h) Caption

Fig. 13. Decompression throughput for CERESZ-II and baseline compressors.

demand a single and unified byte array on the host.

Throughput of CERESZ-II: Overall, CERESZ-II shows an average throughput of 846.20 GB/s for compression and 846.59 GB/s for decompression across seven HPC datasets. Compared to CereSZ and cuSZp, the state-of-the-art error-bounded lossy compressors for Cerebras and GPU, CERESZ-II demonstrates 85.02% and $\sim$10$\times$ higher compression throughput, as well as 45.63% and $\sim$7$\times$ higher decompression throughput.

C. Compression Ratio

We report compression ratios of CERESZ-II and other baseline lossy compressors in Table IV. Same as Section VII-B, we use REL 1E-2, 1E-3, and 1E-4 error bounds. For each dataset, given a compressor and an error bound, we measure compression ratios of all fields and report their minimal (min), maximal (max), and average (avg) values. The higher average compression ratios between CERESZ-II and CereSZ under each specific setting are highlighted.

As seen, CERESZ-II achieves higher compression ratios than CereSZ in all (21/21) settings. While the prebuilt codebook is generated on CESM-ATM, Hurricane, and QMCPack, the other four datasets also exhibit higher compression ratios in CERESZ-II. This demonstrates our prebuilt Huffman codebook is generic across various HPC domains. Compared to cuSZp, CERESZ-II demonstrates better compressibility in three datasets including CESM-ATM, HACC, and QMCPack. For the other four datasets, cuSZp outperforms CERESZ-II due to the high sparsity and slight non-smoothness in these datasets, which makes cuSZp's algorithm more efficient in handling such data patterns. SZ is the leading compressor in terms of compressibility (e.g. 2.3E+4 ratio for NYX under REL 1E-2). Interestingly, we found that CERESZ-II even achieves a higher compression ratio (6.15) than SZ (5.60) in one field from SCALE. Upon examining the data patterns in this field, we identified two key reasons for this result. (1) The quantized integers after Lorenzo Prediction are highly concentrated around the range $[-5, 5]$, making the prebuilt Huffman codebook particularly efficient. (2) SZ applies linear-scale quantization after prediction, meaning the predicted coefficients need to be stored. Although the spline interpolation in SZ is effective, such coefficients add extra information, unlike CERESZ-II, which only compresses the quantized integers.

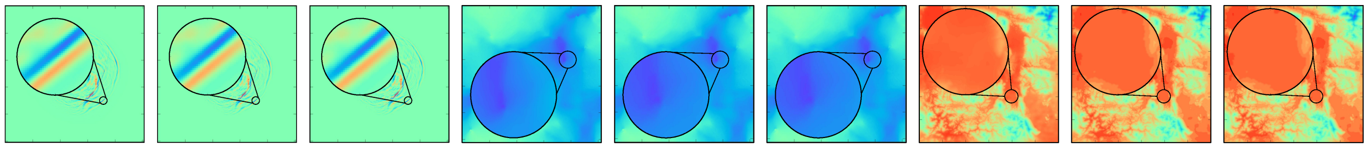
Compression Ratio of CERESZ-II: Even with higher throughput, CERESZ-II exhibits higher compression ratios in all cases (21/21) compared with CereSZ. The prebuilt codebook is generic across various domains.

D. Data Quality

We evaluate the reconstructed data quality of CERESZ-II and CereSZ, two Cerebras-oriented lossy compressors, in Figure 14. Some detailed settings (e.g. which field from which

		CESM-ATM			HACC			Hurricane			NYX			QMCPack			RTM			SCALE		
	REL	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
CereSZ-II	1E-2	17.42	29.16	**23.36**	12.58	23.02	**17.39**	21.73	29.56	**25.90**	23.99	30.12	**27.49**	13.95	25.73	**19.84**	24.82	29.53	**27.09**	21.88	29.87	**26.50**
	1E-3	7.62	25.27	**15.29**	5.69	14.98	**9.44**	9.89	27.54	**18.80**	11.72	30.10	**21.27**	6.45	14.51	**10.48**	12.03	27.84	**19.28**	13.01	28.79	**20.20**
	1E-4	3.83	18.00	**8.34**	3.22	6.59	**4.55**	4.62	24.01	**13.34**	5.12	29.84	**14.26**	3.68	6.74	**5.21**	6.01	25.47	**14.18**	6.15	25.87	**13.15**
CereSZ	1E-2	2.67	21.60	8.73	4.66	9.18	6.82	5.21	28.82	17.10	7.83	31.98	20.22	9.59	19.67	14.63	10.52	31.99	23.46	3.55	30.41	15.71
	1E-3	2.13	16.10	6.49	3.18	4.91	4.05	3.41	24.37	12.57	4.54	31.84	14.05	5.31	9.02	7.16	5.94	31.98	17.73	2.58	26.87	11.29
	1E-4	1.68	13.42	5.11	2.38	3.20	2.83	2.53	19.71	9.64	3.10	29.74	9.61	3.48	4.97	4.23	3.79	31.96	12.87	2.04	21.18	8.15
cuSZp	1E-2	2.84	43.75	12.56	5.24	10.08	7.63	5.94	88.88	38.70	9.60	127.80	66.73	12.44	22.21	17.33	13.97	127.95	66.97	3.87	105.89	37.76
	1E-3	2.25	25.86	8.46	3.43	5.20	4.31	3.71	56.88	22.31	5.09	125.55	38.44	6.08	10.08	8.08	6.90	127.80	42.29	2.74	72.60	21.11
	1E-4	1.75	19.59	6.24	2.53	3.39	2.96	2.70	36.66	14.36	3.35	98.23	22.14	3.79	5.56	4.68	4.17	127.52	27.43	2.14	42.06	12.33
SZ	1E-2	26.13	4.0E+4	2.2E+3	16.58	931.76	217.94	23.76	404.71	110.33	1.3E+3	1.2E+5	2.3E+4	17.10	727.13	372.11	23.57	1.3E+5	4.4E+3	23.49	452.89	127.59
	1E-3	9.30	2.9E+4	941.39	6.11	30.97	15.57	8.81	105.49	35.67	84.55	1.8E+4	3.2E+3	6.37	221.11	113.74	9.27	2.3E+4	894.69	10.59	123.59	34.65
	1E-4	5.04	2.9E+4	825.49	3.74	8.92	5.75	4.63	48.46	18.72	14.38	2.6E+3	471.61	3.88	66.09	34.99	5.30	1.6E+4	548.91	5.60	48.66	15.87

TABLE IV
Compression ratio results between CERESZ-II and three baseline error-bounded CPU, GPU, or Cerebras compressors. The **highlighted** "avg" indicates CERESZ-II *has a higher compression ratio than CereSZ*, the state-of-the-art error-bounded lossy compressor on Cerebras system [13].



(a) Original (b) CERESZ-II (c) CereSZ (d) Original (e) CERESZ-II (f) CereSZ (g) Original (h) CERESZ-II (i) CereSZ

Fig. 14. Visualization data quality from original, CERESZ-II-reconstructed, and CereSZ-reconstructed datasets. All of them are under REL 1E-3 error bound. (a), (b), and (c) are visualized by pressure_2000 field from RTM dataset, where compression ratios for CERESZ-II and CereSZ are 17.97 and 14.67. (d), (e), and (f) are visualized by velocity_x field from NYX dataset, where compression ratios for CERESZ-II and CereSZ are 14.78 and 4.54. (g), (h), and (i) are visualized by PRES field from SCALE dataset, where compression ratios for CERESZ-II and CereSZ are 18.04 and 3.36. *Note that, under the same error bound,* CERESZ-II *and CereSZ have the same reconstructed data quality.*

dataset) are explained in the caption of Figure 14. In CERESZ-II, the only lossy step is Pre-Quantization (❶), whereas the rest are lossless operations. Such a design is also adopted in other recent parallel compressors [13], [21], [22], [28], [56]. As a result, given the same error bound (e.g. REL 1E-3 in Figure 14), those compressors will generate identical reconstructed data. For example, in Figure 14(e) and 14(f), both CERESZ-II and CereSZ exhibit SSIM [57] (0.9604) and PSNR [58] (64.77). However, in this case, CERESZ-II achieves around a $\sim 3\times$ compression ratio (14.68) of CereSZ (4.54). In other words, if we align the compressed length, CERESZ-II can always result in better data quality in CERESZ-II, since it can support a more rigorous error control. This also demonstrates CERESZ-II outperforms CereSZ in terms of data quality.

Data Quality of CERESZ-II: CERESZ-II outperforms CereSZ in data quality due to stronger compressibility in Fixed-size Huffman Encoding (❸).

VIII. DISCUSSION

A. Pipeline Parallelism on WSE

There are two ways of structuring computations for execution on Cerebras: *(1) Single Instruction/Multiple Data (SIMD) Execution*: Each PE performs the same computation for different data chunks. CERESZ-II follows this pattern. *(2) Pipelined Execution*: Computation is sub-divided into steps which are executed on adjacent PEs in a pipeline design leveraging the low latency of the routers connecting PEs. (1) and (2) can be combined by repeating the same computational pipeline across the WSE. CereSZ follows this mixed pattern by dividing compression into multiple stages with similar cycle counts, distributing each stage across three PEs. These PEs are then treated as a group and processed using SIMD. While pipelining is powerful enough to utilize the WSE architectures, we do not utilize this in CERESZ-II because lossy and lossless stages feature large load imbalance issues, reducing efficiency in tasks such as offsets computation, which makes more PE remain idle. Future work could explore having faster PEs assist slower ones to improve pipeline parallelism. However, in existing WSE architectures, this is complicated by the limited communication patterns between different PEs. As a result, we stick with only adopting SIMD in proposing and implementing CERESZ-II in this work.

B. Offset Computation with Other Optimization

In CERESZ-II, we compute Offset for each compressed block by ❹ in Figure 2. In principle, this follows a computation pattern called *Reduce then Scan* [34], where "then" denotes global synchronization and routinely becomes the performance bottleneck due to its serial implementation. In CERESZ-II, we can see that the scan in Step 2 of ❹ is only performed within the first column/row of PEs in serial. When the scan operation has not yet reached the later PEs, those PEs remain idle. Merrill et al. [35] proposed a strategy called *decoupled* *lookback* to accelerate this process. However, we argue that this design is not suitable for the WSE, as each PE can only access data from its immediate neighbors, and the absence of shared memory makes it inefficient for a PE to retrieve data from its multi-hop predecessors. This demonstrates the rationale of our proposed ❹ in CERESZ-II.

C. *Portability of* CERESZ-II *across Other Architectures*

Our compression algorithm is designed with broad architectural compatibility, extending its application beyond Cerebras CS-2 to other specialized hardware such as the Graphcore IPU and NVIDIA's DPU platforms. These architectures, like the CS-2, feature a 2D core mesh and possess limited local memory per core, and lack shared memory, which facilitates efficient global offset computations. The robustness of our CERESZ-II, which adopts a static approach not only simplifies the implementation but also enhances performance consistency across platforms, thereby increasing its utility and scalability in heterogeneous computing environments. We regard extending CERESZ-II to other architectures as our future work.

IX. RELATED WORK

We present related works from two perspectives: ultra-fast parallel error-bounded lossy compression and applications for Cerebras dataflow architecture.

A. *Ultra-fast Error-bounded Lossy Compression*

In the past decade, several error-bounded lossy compressors have been proposed to target ultra-fast throughput [20]–[22], [28], [30], [56], [59]–[61]. Gruyzmacher et al. [62] presented a lightweight in-register GPU compressor for the GMRES iterative solver, significantly reducing memory footprint by compressing Krylov basis vectors. Zhang et al. [21] introduced a pure-GPU compressor that features an innovative lossless encoding, achieving high performance in both throughput and compression ratio. Huang et al. [22] proposed the first single-kernel, pure GPU compressor featuring lightweight fixed-length encoding and bit-shuffling, which demonstrated a significant throughput improvement over prior GPU compressors.

B. *Applications on Cerebras Dataflow Architecture*

Recently, Cerebras has emerged as a critical role in advancing applications within both machine learning [31], [63] and scientific computing domains [3], [11], [13], [32]. Thangarasa et al. [31] leveraged the Cerebras system to accelerate training with unstructured weight sparsity for pre-trained biomedical language models. Chiley et al. [63] utilized the unique capabilities of the Cerebras hardware to address memory and scalability challenges in training large-scale neural networks, achieving high efficiency in both training memory requirements and computational cost. Luczynski et al. [32] presented the first systematic investigation of Reduce and AllReduce on WSE, outperforming existing solutions. Song et al. [13] proposed the first error-bounded lossy compressor CereSZ on Cerebras.

X. CONCLUSION AND FUTURE WORKS

In this work, we propose CERESZ-II, a memory-efficient and computational-balanced error-bounded lossy compression pipeline on Cerebras Wafer-Scale Engine. Experiments on seven real-world scientific datasets demonstrate that CERESZ-II can achieve on average 846.20 GB/s and 846.59 GB/s throughput for compression and decompression, which is 85.02% and 45.63% faster than state-of-the-art, respectively, with also higher compression ratios and data quality.

In the future, we aim to conduct our research for CERESZ-II in two directions. (1) Support for Other Platforms: We aim to extend CERESZ-II to other heterogeneous AI training infrastructures, such as Graphcore IPU and NVIDIA's DPU. (2) Domain Application Accelerations: We will utilize CERESZ-II to benefit domain applications, such as quantum simulation [64]–[66] and deep learning training [67]–[69].

ACKNOWLEDGEMENTS

The material was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357. This material was also supported by the National Science Foundation under Grant Nos. CSSI/OAC# 2311875 and CSSI/OAC# 2104023. We acknowledge the technical support of Leighton Wilson from Cerebras Systems Inc., and Mei-Yu Wang and Julian Uran from the Pittsburgh Supercomputing Center, whose expertise and assistance were instrumental in the development of this work.

REFERENCES

[1] J. Selig, "The cerebras software development kit: A technical overview," *Technical Report. Cerebras*, 2022.

[2] H. Ltaief, Y. Hong, L. Wilson, M. Jacquelin, M. Ravasi, and D. E. Keyes, "Scaling the "memory wall" for multi-dimensional seismic processing with algebraic compression on cerebras cs-2 systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–12.

[3] K. Santos, S. Moore, T. Oppelstrup, A. Sharifian, I. Sharapov, A. Thompson, D. Z. Kalchev, D. Perez, R. Schreiber, S. Pakin *et al.*, "Breaking the molecular dynamics timescale barrier using a wafer-scale system," *arXiv preprint arXiv:2405.07898*, 2024.

[4] J. Tramm, B. Allen, K. Yoshii, A. Siegel, and L. Wilson, "Efficient algorithms for monte carlo particle transport on ai accelerator hardware," *Computer Physics Communications*, vol. 298, p. 109072, 2024.

[5] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach *et al.*, "Frontier: exploring exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–16.

[6] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X.-C. Wu, Y. Alexeev, and F. T. Chong, "Use cases of lossy compression for floating-point data in scientific data sets," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1201–1220, 2019.

[7] F. Cappello, S. Di, R. Underwood, D. Tao, J. Calhoun, Y. Kazutomo, K. Sato, A. Singh, L. Giraud, E. Agullo *et al.*, "Multifacets of lossy compression for scientific data in the joint-laboratory of extreme scale computing," *Future Generation Computer Systems*, 2024.

[8] D. Hoang, H. Bhatia, P. Lindstrom, and V. Pascucci, "High-quality and low-memory-footprint progressive decoding of large-scale particle data," in *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 2021, pp. 32–42.

[9] A. Scialdone, R. Ferraro, R. G. Alía, L. Sterpone, S. Danzeca, and A. Masi, “Fpga qualification and failure rate estimation methodology for lhc environments using benchmarks test circuits,” *IEEE Transactions on Nuclear Science*, vol. 69, no. 7, pp. 1633–1641, 2022.

[10] E. Robein, “Eage e-lecture: Reverse time migration: How does it work, when to use it,” 2016.

[11] M. Shah, X. Yu, S. Di, M. Becchi, and F. Cappello, “A portable, fast, dct-based compressor for ai accelerators,” in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, 2024, pp. 109–121.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, 2019.

[13] S. Song, Y. Huang, P. Jiang, X. Yu, W. Zheng, S. Di, Q. Cao, Y. Feng, Z. Xie, and F. Cappello, “Ceresz: Enabling and scaling error-bounded lossy compression on cerebras cs-2,” in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, 2024, pp. 309–321.

[14] S. Di, J. Liu, K. Zhao, X. Liang, R. Underwood, Z. Zhang, M. Shah, Y. Huang, J. Huang, X. Yu *et al.*, “A survey on error-bounded lossy compression for scientific datasets,” *arXiv preprint arXiv:2404.02840*, 2024.

[15] S. Li, P. Lindstrom, and J. Clyne, “Lossy scientific data compression with sperr,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 1007–1017.

[16] S. Di and F. Cappello, “Fast error-bounded lossy hpc data compression with sz,” in *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE, 2016, pp. 730–739.

[17] P. Lindstrom, “Fixed-rate compressed floating-point arrays,” *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[18] D. Tao, S. Di, Z. Chen, and F. Cappello, “Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization,” in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 1129–1139.

[19] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, “Multilevel techniques for compression and reduction of scientific data—the multivariate case,” *SIAM Journal on Scientific Computing*, vol. 41, no. 2, pp. A1278–A1303, 2019.

[20] P. Lindstrom, “cuzfp,” https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp.

[21] B. Zhang, J. Tian, S. Di, X. Yu, Y. Feng, X. Liang, D. Tao, and F. Cappello, “Fz-gpu: A fast and high-ratio lossy compressor for scientific computing applications on gpus,” in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 2023, pp. 129–142.

[22] Y. Huang, S. Di, X. Yu, G. Li, and F. Cappello, “cuszp: An ultra-fast gpu error-bounded lossy compression framework with optimized end-to-end performance,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–13.

[23] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. Pugmire, M. Wolf, N. Podhorszki *et al.*, “Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction,” *IEEE Transactions on Computers*, vol. 71, no. 7, pp. 1522–1536, 2021.

[24] M. Xia, S. Di, F. Cappello, P. Jiao, K. Zhao, J. Liu, X. Wu, X. Liang, and H. Guo, “Preserving topological feature with sign-of-determinant predicates in lossy compression: A case study of vector field critical points,” in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4979–4992.

[25] L. Yan, X. Liang, H. Guo, and B. Wang, “Toposz: Preserving topology in error-bounded lossy compression,” *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[26] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello, “Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1643–1654.

[27] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, “Error-controlled lossy compression optimized for high compression ratios of scientific datasets,” in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 438–447.

[28] X. Yu, S. Di, K. Zhao, J. Tian, D. Tao, X. Liang, and F. Cappello, “Ultrafast error-bounded lossy compression for scientific datasets,” in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 159–171.

[29] B. Zhang, J. Tian, S. Di, X. Yu, M. Swany, D. Tao, and F. Cappello, “Gpulz: Optimizing lzss lossless compression for multi-byte data on modern gpus,” in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 348–359.

[30] Y. Huang, S. Di, G. Li, and F. Cappello, “cuszp2: A gpu lossy compressor with extreme throughput and optimized compression ratio,” in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–18.

[31] V. Thangarasa, M. Salem, S. Saxena, K. Leong, J. Hestness, and S. Lie, “Mediswift: Efficient sparse pre-trained biomedical language models,” *arXiv preprint arXiv:2403.00952*, 2024.

[32] P. Luczynski, L. Gianinazzi, P. Iff, L. Wilson, D. De Sensi, and T. Hoefler, “Near-optimal wafer-scale reduce,” in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, 2024, pp. 334–347.

[33] R. Farber, *CUDA application design and development*. Elsevier, 2011.

[34] S. Yan, G. Long, and Y. Zhang, “Streamscan: fast scan algorithms for gpus without global barrier synchronization,” in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2013, pp. 229–238.

[35] D. Merrill and M. Garland, “Single-pass parallel prefix scan with decoupled look-back,” *NVIDIA, Tech. Rep. NVR-2016-002*, 2016.

[36] S. Maleki and M. Burtscher, “Automatic hierarchical parallelization of linear recurrences,” *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 128–138, 2018.

[37] Y. Li, C. Chasseur, and J. M. Patel, “A padded encoding scheme to accelerate scans by leveraging skew,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1509–1524.

[38] A. Moffat, “Huffman coding,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–35, 2019.

[39] J. E. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. Bates, G. Danabasoglu, J. Edwards *et al.*, “The community earth system model (cesm) large ensemble project: A community resource for studying climate change in the presence of internal climate variability,” *Bulletin of the American Meteorological Society*, vol. 96, no. 8, pp. 1333–1349, 2015.

[40] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, “Potential benefits of delta encoding and data compression for http,” in *Proceedings of the ACM SIGCOMM’97 conference on Applications, technologies, architectures, and protocols for computer communication*, 1997, pp. 181–194.

[41] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[42] W. Wang, C. Bruyere, B. Kuo, T. Scheitlin, and others at NCAR, “Hurricane dataset: Weather simulation,” http://vis.computer.org/vis2004contest/data.html.

[43] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, “Hacc: Extreme scaling and performance across diverse architectures,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–10.

[44] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley *et al.*, “Qmcpack: an open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids,” *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, 2018.

[45] N. R. Goodman, “Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction),” *The Annals of mathematical statistics*, vol. 34, no. 1, pp. 152–177, 1963.

[46] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, “Nyx: A massively parallel amr code for computational cosmology,” *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[47] E. Baysal, D. D. Kosloff, and J. W. Sherwood, “Reverse time migration,” *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983.

[48] G.-Y. Lien, T. Miyoshi, S. Nishizawa, R. Yoshida, H. Yashiro, S. A. Adachi, T. Yamaura, and H. Tomita, “The near-real-time scale-letkf system: A case of the september 2015 kanto-tohoku heavy rainfall,” *Sola*, vol. 13, pp. 1–6, 2017.

[49] Y. Huang, K. Zhao, S. Di, G. Li, M. Dmitriev, T.-L. D. Tonellot, and F. Cappello, “Towards improving reverse time migration performance by high-speed lossy compression,” in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023, pp. 651–661.

[50] K. Zhao, S. Di, X. Lian, S. Li, D. Tao, J. Bessac, Z. Chen, and F. Cappello, “SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors,” in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 2716–2724.

[51] ——, “Sdrbench: Scientific data reduction benchmark for lossy compressors,” in *2020 IEEE international conference on big data (big data)*. IEEE, 2020, pp. 2716–2724.

[52] R. Underwood, J. Bessac, D. Krasowska, J. C. Calhoun, S. Di, and F. Cappello, “Black-box statistical prediction of lossy compression ratios for scientific data,” *The International Journal of High Performance Computing Applications*, vol. 37, no. 3-4, pp. 412–433, 2023.

[53] E. Sobolev, P. Schmidt, J. Malka, D. Hammer, D. Boukhelef, J. Möller, K. Ahmed, R. Bean, I. J. Bermúdez Macías, J. Bielecki *et al.*, “Data reduction activities at european xfel: early results,” *Frontiers in Physics*, vol. 12, p. 1331329, 2024.

[54] V. A. Magri and P. Lindstrom, “A general framework for progressive data compression and retrieval,” *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[55] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao *et al.*, “Sz3: A modular framework for composing prediction-based error-bounded lossy compressors,” *IEEE Transactions on Big Data*, vol. 9, no. 2, pp. 485–498, 2022.

[56] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao *et al.*, “Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data,” in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 3–15.

[57] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[58] A. Hore and D. Ziou, “Image quality metrics: Psnr vs. ssim,” in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 2366–2369.

[59] J. Liu, J. Tian, S. Wu, S. Di, B. Zhang, R. Underwood, Y. Huang, J. Huang, K. Zhao, G. Li *et al.*, “Cusz-i: High-ratio scientific lossy compression on gpus with optimized multi-level interpolation,” in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–15.

[60] P. Ratanaworabhan, J. Ke, and M. Burtscher, “Fast lossless compression of scientific floating-point data,” in *Data Compression Conference (DCC’06)*. IEEE, 2006, pp. 133–142.

[61] M. Burtscher and P. Ratanaworabhan, “Fpc: A high-speed compressor for double-precision floating-point data,” *IEEE transactions on computers*, vol. 58, no. 1, pp. 18–31, 2008.

[62] T. Grützmacher, R. Underwood, S. Di, F. Cappello, and H. Anzt, “Frsz2 for in-register block compression inside gmres on gpus,” *arXiv preprint arXiv:2409.15468*, 2024.

[63] V. Chiley, V. Thangarasa, A. Gupta, A. Samar, J. Hestness, and D. DeCoste, “Revbifpn: the fully reversible bidirectional feature pyramid network,” *Proceedings of Machine Learning and Systems*, vol. 5, pp. 625–645, 2023.

[64] B. Zhang, B. Fang, F. Ye, Y. Gu, N. Tallent, G. Tan, and D. Tao, “Overcoming memory constraints in quantum circuit simulation with a high-fidelity compression framework,” *arXiv preprint arXiv:2410.14088*, 2024.

[65] M. Xu, S. Cao, X. Miao, U. A. Acar, and Z. Jia, “Atlas: Hierarchical partitioning for quantum circuit simulation on gpus,” in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–17.

[66] H. Shang, Y. Liu, Z. Wu, Z. Chen, J. Liu, M. Shao, Y. Li, B. Kan, H. Cui, X. Feng *et al.*, “Pushing the limit of quantum mechanical simulation to the raman spectra of a biological system with 100 million atoms,” in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–12.

[67] S. Song and P. Jiang, “Rethinking graph data placement for graph neural network training on multiple gpus,” in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–10.

[68] A. Choudhury, Y. Wang, T. Pelkonen, K. Srinivasan, A. Jain, S. Lin, D. David, S. Soleimanifard, M. Chen, A. Yadav *et al.*, “{MAST}: Global scheduling of {ML} training across {Geo-Distributed} datacenters at hyperscale,” in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 563–580.

[69] Z. Mo, H. Xu, and C. Xu, “Heet: Accelerating elastic training in heterogeneous deep learning clusters,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 499–513.