OptZConfig: Efficient Parallel Optimization of Lossy Compression Configuration

Robert Underwood, Jon C. Calhoun, *Member, IEEE*, Sheng Di, *Senior Member, IEEE*, Amy Apon, *Senior Member, IEEE*, and Franck Cappello, *Fellow, IEEE*

Abstract—Lossless compressors have very low compression ratios that do not meet the needs of today's large-scale scientific applications that produce vast volumes of data. Error-bounded lossy compression (EBLC) is considered a critical technique for the success of scientific research. Although EBLC allows users to set an error bound for the compression, users have been unable to specify the requirements on the compression quality, limiting practical use. Our contributions are: (1) We formulate the problem of configuring EBLC to preserve a user-defined metric as an optimization problem. This allows many classes of new metrics to be preserved, which improves over current practices. (2) We present a framework, OptZConfig, that can adapt to improvements in the search algorithm, compressor, and metrics with minimal changes, enabling future advancements in this area. (3) We demonstrate the advantages of our approach against the leading methods to configure compressors to preserve specific metrics. Our approach improves compression ratios against a specialized compressor by up to $3\times$, has a $56\times$ speedup over FRaZ, $1000\times$ speedup over MGARD-QOI post tuning, and $110\times$ speedup over systematic approaches which had not been bounded by compressors before.

Index Terms—Error Bounded Lossy Compression, LibPressio, Parallel Computing, Non-linear Optimization

1 Introduction

With the ever-increasing execution scale and problem size of today's scientific applications, volumes of simulation data are produced that cannot be stored and transferred efficiently because of the limited storage space and I/O or network bandwidth. Consequently, scientific data reduction techniques are studied. Since most scientific datasets are composed of floating-point numbers, traditional byte-stream-based lossless compressors (such as Gzip [1] and Zstd [2]) suffer from low compression ratios (generally $\sim 2\times$ or lower) [1]. Although some lossless compressors such as FPC [3] are designed for floating-point data, the compression ratio is still far lower than the user-required level (often $10\times$ or higher) because of the high entropy of the mantissa of floating-point numbers.

Error-bounded lossy compressors (EBLCs) address this issue by allowing users to control the data distortion by specifying point-wise error bounds. By leveraging the correlation of adjacent data points in either space or time, an EBLC can obtain a compression ratio of $100\times$ or even higher while respecting user prescribed point-wise accuracy requirements on the decompressed data, significantly reducing the data storage and data movement burden. Nevertheless, a significant gap still exists between the EBLCs and the user's compression needs. Users must invest significant effort to understand the effectiveness of each lossy compressor on their specific scientific datasets and the impact of data distortion to their post-hoc analysis. To this end, users have to manually perform many tedious trial-and-error tests for the different compressors with their large datasets, and each compressor may involve setting many parameters. For example, SZ [4], [5] offers more than 25 parameters, including different types of error bounds (absolute error bound, relative error bound), number of quantization bins, block size, and lossless compression technique (e.g., Zlib [1] and Zstd [2]).

In addition to specialized compressors, there have been two major developments in this area: FRaZ [6] and MGARD-QOI mode [7]. The former uses numerical optimization techniques to find a fixed compression ratio. The latter uses general mathematical proprieties to bound compression errors for arbitrary bounded linear functional Quantities of Interest (QoIs) on regular grids – a subset of methods we consider in our paper. However, both share significant shortcomings, namely performance and applicability to a wide array of user defined metrics. In this paper, we propose *OptZConfig* and *Fixed Metric Fidelity Search* (FMFS), which help users select the lossy compressor that help them meet their quality needs with optimized parameter settings in a fully automatic way for more general user defined metrics by building upon the methodology in FRaZ. That is, provided a scientific dataset and the user's specific required compression quality (e.g., some analysis metric and/or target compression ratio), OptZConfig quickly selects the compressor and determines its sufficient parameter setting.

Our major contributions are as follows:

1) We formulate the configuring of EBLCs to preserve a user-defined metrics as an optimization problem improving over current trial-and-error or scientist in-the-loop evaluations.

- *R. Underwood is with Argonne National Laboratory. E-mail: runderwood@anl.gov*
- *A. Apon and J. Calhoun are with Clemson University.*
- *S. Di and F. Cappello are with Argonne National Laboratory.*

Manuscript received February 21, 2022; revised Feburary 14, 2022.

1. early usage of compression ratio defined it as the inverse from what we define as it here; however most modern work on compressors defines it as we do here. Some communities retain the earlier definition

2) We present a framework, OptZConfig, that adapts to improvements in the search algorithm [6], compressor [7], [8], and metrics with minimal changes [9], enabling future innovations. The prior approaches cited are strongly tied to at least one of these three.
3) We present and evaluate on real world data sets and metrics a novel parallel algorithm, Fixed Metrics Fidelity Search (FMFS), that improves over the compression ratio of specialized compressors [8] by up to 3× , is 56× faster than prior black box searching methods [6], over 1000× faster post-tuning than MGARD-QOI post-tuning [7], and 110× faster than systematic approaches to bound metrics that are bounded by prior compressors or frameworks.

2 Research Background and Related Work

This section presents the background and highlights the motivation for this research. First, we provide a discussion of several scientific applications with big data issues and the user's requirements. We discuss several use cases that require automatic lossy compression optimization strategies in practice. Then, we describe the state-of-the-art error-bounded lossy compression (EBLC) techniques fundamental to developing the OptZConfig framework.

2.1 Big Data Applications and Use Cases

Today's scientific applications are producing too much data to efficiently process or store at runtime. Cosmology simulations of the Hardware/Hybrid Accelerated Cosmology Code (HACC) [10], for instance, may produce 21.2 petabytes of data when simulating 2 trillion particles for 500 time-steps. Summit, one of the most powerful supercomputers, at the Oak Ridge National Laboratory [11] provides only hundreds of terabytes of storage for each user. Hundreds of users share the limited total storage space (250 PB in total).

Materials science can produce raw data with a very high rate (e.g. 250 GB/s with Light Coherent Light Source (LCLS-II) [12]). In order to sustain the data acquisition rate, storing the raw data without compression is impractical. The data needs to be reduced by 10 × to fit in the available bandwidth.

Each scientific application brings a unique use case for EBLC in the analysis of high-performance computing simulation data or instrument data. For example, cosmology researchers explore galaxy structures formed by particles coalescing into halos, and the bias of halo masses [13], [14] should be limited to within 3% based on the lossy reconstructed particle data.

A number of metrics can be used to understand the impact of compression errors. These metrics measure the quality of specific statistical outcomes, signal distortion (such as *peak signal-to-noise ratio (PSNR)*), and spatial error. However, not all metrics are supported by all lossy compressors. FPZIP and ZFP do not natively support PSNR or spatial-error-preserving modes requiring workarounds. SZ supports fixed PSNR but suffers from large errors especially for high-compression cases, which is attributed to the assumption of uniform error distribution. This may degrade compression ratios. We refer readers to the survey paper [12] for more use cases of EBLCs. We summarize some key user-analysis metrics and corresponding user-acceptable thresholds in Table 1.

TABLE 1
Summary of Important Lossy Compression User Analysis Metrics

Metric	Domain	Threshold	Range
Pearson Correlation (R value)	Climate	$\geq$ .99999 [15]	[-1, 1]
p value for KS Test	Climate	$\geq$ .05 [15]	[0,1]
Spatial Relative Error	Climate	$\leq$ .05, $\delta = 1e^{-4}$ [15]	[0, 1]
Peak Signal-to-Noise Ratio (PSNR)	Various	Various	[0, ∞)

Users want to guide the compression that they perform by metrics used within their communities [15], [16]. Some robust examples come from the climate community where research has gone into determining thresholds acceptable to the climate community at large. The *Pearson correlation coefficient* is used to show the strength of a linear relationship between two datasets. When used between uncompressed and decompressed data, it measures how well the value in the uncompressed dataset represents the decompressed dataset. The *Kolmogorov Smirnov (KS) test* is a nonparametric hypothesis test that tests whether two samples are from the same distribution. The metric of interest is the p-value, which says how likely the observation is given that the hypothesis is true. It is calculated by computing an empirical cumulative distribution function for each sample and finding the largest difference between these functions. The probability of this distance occurring is computed by using methods from [17]. The *spatial relative error* is the percentage of points that exceed a specified relative error threshold indicating how widely spread a distortion is.

Another important set of metrics comes from the Scientific Data Reduction Benchmarks (SDRBench) [18]. These represent a collection of real-world problems from various domains paired with the metric(s) of interest. All datasets in this paper come from SDRBench.

2.2 Error-Bounded Lossy Compression

Here we describe the EBLC techniques used in the leading error-bounded lossy compressors SZ, ZFP, and MGARD.

SZ [5], [19], [20], [9] is an error-bounded lossy compressor offering multiple error-controlling approaches, including absolute error bound (denoted ϵ_{abs}) [5], value-range relative error bound (denoted ϵ_{rel}) [4], and target PSNR (denoted by ϵ_{psnr}) [9].

SZ adopts a blockwise prediction-based compression model, which involves three key steps: (1) data prediction – each data point is predicted based on its nearby values in space and two major predictors are applied, Lorenzo [5] and linear regression [19]; (2) linear-scale quantization – each data point is converted to an integer by applying an equal-bin-size quantization on the difference between its predicted value and real value; and (3) compression of the generated integer code arrays by a series of custom lossless compression methods including entropy encoding, such as Huffman encoding, and dictionary encoding, such as Zstd [2].

ZFP [21] is another outstanding error-bounded lossy compressor, which is broadly evaluated in many scientific

research studies [22], [6], [23]. Similar to SZ, ZFP supports different types of error controls, such as absolute error bound and precision. The precision mode allows users to set an integer number to control the data distortion with an approximately relative error effect. The higher the precision number is, the lower the data distortion.

Unlike SZ, ZFP adopts a blockwise transform-based compression model, which includes three critical steps: (1) exponent alignment and fixed-point representation, which align the values in each block to a common exponent and performs fixed-point representation conversion; (2) transformation, which applies a near orthogonal transform to each block; and (3) embedded-coding, which orders the transform coefficients and encodes the coefficients one "bit plane" at a time. SZ and ZFP have different design principles, and neither always has the best compression quality on all datasets [22], [6].

MGARD [24], [7], [25] is a state-of-the-art lossy compressor supporting multigrid adaptive reduction of data. The most important principle of MGARD is a hierarchical scheme that offers the flexibility to produce multiple levels of partial decompression such that users reduce the dataset by either minimizing storage with a required data fidelity or minimizing the data distortion with a target compression ratio. MGARD's Quantity of Interest mode bounds some limited kinds of metrics. We discuss it in detail in Section 7.3.

2.3 Numerical Optimization to Configure EBLC

The relationship between a metric and compressor configuration can be multidimensional, non-monotonic, and non-convex. Therefore, naïve approaches such as binary search are not sufficient [6]. Rather, we use numerical optimization.

Analytic derivatives provide a closed form description of the relationship between the compressor settings. Also, metrics are difficult to construct and can change frequently depending on the implementation of the compressor, making them ill-suited to this task. Numerical methods that estimate the derivative by computing the slope of nearby points are too slow because of the time required to evaluate each point. That is, regularly computing a slope at each point is time prohibitive because it requires running the compressor and any associated metrics.

Derivative-free methods do not rely on having derivatives available during the search. Thus, they are a good candidate for use in this context since one does not have to wait to compute the derivative. A typical example is the FRaZ algorithm [6], which itself is based on [26], [27], [28]. FRaZ is hard-coded to adjust only one compressor setting using a specific kind of derivative-free optimization. It bounds compression ratios and does not require any call-backs to a user-defined metric. Its parallelism scheme is not thread-safe, however. The use of multi-threading in the search function or compressor can cause incorrect results or failure. Most importantly, it has no protocol for communicating early termination between iterations, but only between grid cells. Since some sophisticated metrics that users care about could take hours to compute, one grid of the search might complete successfully, but the algorithm may continue to run for hours until each grid finishes its remaining iterations. This makes FRaZ feasible only for metrics that are quick to compute like compression ratios in offline use cases. Our two novel strategies (OptZConfig and FMFS), significantly outperform FraZ by addressing these and other issues, enabling an online use case. We discuss this in greater detail in Section 6.

2.4 White-Box/Trial-Based Approaches

At least two related works use white-box approaches to autotune compression [29], [8]. In contrast to a specialized compressor, white box methods allow a limited variety of metrics to be bounded on a limited number of compressors that share certain properties which are exploited for performance. They might use internal properties such as the exact methodology of the prediction and quantization scheme or the sampling based on the block size [29] in newer versions of SZ to speed up the search process to maximize bandwidth at a given error bound for a compressor [8]. Notably, these approaches have no means to invoke a user-defined metric. Furthermore, these approaches exploit properties of the compressor and how they are tied to the specific metric(s) they preserve and generally are not transferable to new problems without degrading performance relative to the problem they were designed for or require substantial rewrites if it is possible to adapt the method.

Another attempt was taken by SCIL[30]. SCIL shares properties of both OptZConfig and LibPressio. Like LibPressio, it attempts to abstract across compressors and also has the concept of a meta-compressor. However, it differs in that it has a fixed-function compression pipeline, which limits the chaining of arbitrary meta compressors and some of the more robust configurations supported by LibPressio. It also does not expose the metadata needed for safe multi-threading of compression nor does correctly pass dimensionality information onto the underlying compressor. SCIL also has features like OptZConfig. However, most importantly it only attempts to bound a specific list of compressor-centric measurements such as the relative tolerance or the number of significant digits preserved rather than arbitrary metrics provided by the users. It does so via previous runs which are then converted into an internal decision tree.

These approaches are complemented and extended by the approaches in OptZConfig. For example, if the users know that they are employing SZ, they can use the search interface provided by OptZConfig to define a custom searcher based on the methods in these papers – allowing a white-box style usage when an algorithm is available. For example, one could adopt the approach by [8] to cache the 20 – 30 best configurations and try the best ones at runtime, or use the approach by [29] and sample blocks at the compressor block size to choose between compressors.

3 Problem Formulation

In this section, we formalize our research problem. The overarching goal is to automatically select the lossy compressor with sufficient quality using a tuned configuration based on user analysis metrics. This is a generalization and extension of past methods [6] that only considered a single objective with a single input parameter and a single output parameter. Constructing this problem in the general case enables entirely new classes of problems (see Section 7.4).

Let D be a set of data buffers that involve a set of fields, denoted by F_D, and a set of simulation time-steps, denoted by T_D. We refer to a specific data buffer as $d_{f,t}$ that holds a dataset to compress, where $t \in T_D$ and $f \in F_D$.

3.1 Lossy Compression-Based Parameter Space

State-of-the-art error-bounded lossy compressors use multiple configuration parameters (or error settings) to tune the compression quality and performance. We divide a lossy compressor's configuration parameters into two sets: a set of fixed parameters ($\vec{\theta_c}$) and a set of nonfixed parameters ($\vec{c}$). Fixed parameters' values are specified by users and are not modified during tuning. Nonfixed parameters are modified to optimize performance or compression quality. For instance, SZ controls has an error-bounding type (see Section 2.2), error bound value (a specific positive threshold value), and the number of quantization bins. Each of the three parameters is either set to be a fixed parameter as a constraint by the user, or set to be a nonfixed parameter to be optimized whose values are determined based on the feasible settings of corresponding compressors.

Let U denote the whole set of the nonfixed parameters ($\vec{c}$), and let Ω denote the whole feasible parameter set (including both nonfixed parameters and fixed parameters). Each $\vec{c}$ in U has a value constraint that is bounded by a lower bound vector $\vec{l}$ and upper bound vector $\vec{u}$ such that $\vec{l_i} \leq \vec{c_i} \leq \vec{u_i}$, where $\vec{l_i}$, $\vec{u_i}$, and $\vec{c_i}$ are the ith element of $\vec{l}$, $\vec{u}$, and $\vec{c}$, respectively. We denote the reconstructed data buffer based on lossy compression by $\tilde{d_{f,t}}(\vec{c}, \vec{\theta_c})$ and the corresponding set by D'.

3.2 User-Analysis-Based Parameter Space

Performing high-fidelity analysis on the decompressed buffers (D') for a particular time-step t and field f compared with the one with original uncompressed data may involve some fixed parameters, denoted by $\vec{\theta_m}$. We refer to the user-required fidelity comparison metric on the original uncompressed buffer $d_{f,t}$ and its decompressed buffer as $\mathcal{Q}(d_{f,t}, \tilde{d_{f,t}}(\vec{c}; \vec{\theta_c}); \vec{\theta_m})$. The user expresses a requirement by identifying some threshold for this fidelity comparison metric; we denote this threshold by $\mathcal{Q}_\tau(d_{f,t}; \vec{\theta_m})$. We use an example to further explain the definition of $\mathcal{Q}$ and $\mathcal{Q}_\tau$ in the following text. Unlike the prior work [7] which requires $\mathcal{Q}$ be a bounded linear functional, we do not place any constraints on the fidelity requirement of $\mathcal{Q}$ (or $\mathcal{Q}_\tau$). We summarize all the key notations in Table 2.

3.3 Finding The Sufficient Configuration

Based on the compression parameter space under some compressor and the analysis parameter space defined for some application, we formulate the task of finding the sufficient configuration as the following optimization problem: Given $D, U, \vec{\theta_c}, \vec{\theta_m}, e, \forall\ d_{f,t} \in D$, optimize: $\max_{\vec{c} \in U} \mathcal{Q}\left(d_{f,t}, \tilde{d_{f,t}}\left(\vec{c}; \vec{\theta_c}\right); \vec{\theta_m}\right)$. If a threshold $\mathcal{Q}_\tau\left(d_{f,t}; \vec{\theta_m}\right)$ is provided, the search may terminate early if this constraint is met: $\mathcal{Q}\left(d_{f,t}, \tilde{d_{f,t}}\left(\vec{c}; \vec{\theta_c}\right); \vec{\theta_m}\right) \geq \mathcal{Q}_\tau\left(d_{f,t}; \vec{\theta_m}\right)$. Better quality results may be obtained by increasing the termination threshold, but at the cost of greater run-time. The exact trade-off desired may differ from application to application.

TABLE 2
Key Notations

Notation	Description
D	Set of all uncompressed buffers for all fields and time-steps
D'	Set of all decompressed buffers for all fields and time-steps
T_D	Set of all time-steps for dataset D
F_D	Set of all fields for dataset D
$d_{f,t}$	Buffer for field, f, and time-step, t, in uncompressed form
ϵ	Some threshold; Typically an error bound provided by a compressor
$\vec{c}$	Vector of nonfixed compressor parameters
$\vec{\theta_c}$	Vector of fixed compressor parameters
$\tilde{d_{f,t}}(\vec{c}; \vec{\theta_c})$	Decompressed buffer for field, f, and time-step, t
$\vec{\theta_m}$	Vector of fixed parameters of the user-specified metrics function
U	Set of feasible nonfixed compressor parameters
Ω	Set of feasible fixed and nonfixed compressor parameters
$\mathcal{Q}\left(d_{f,t}, \tilde{d_{f,t}}\left(\vec{c}; \vec{\theta_c}\right); \vec{\theta_m}\right)$	User-specified data fidelity metric function
$\mathcal{Q}_\tau(d_{f,t}; \vec{\theta_m})$	Early termination threshold for user-specified metrics function

We further illustrate the research problem using the following example based on the target metric of the Pearson's correlation coefficent (R). This is a common measure for compression quality which no previous compressor bounds. Our FMFS identifies a sufficient parameter setting of the lossy compressor based on the user-defined target metric.

Here is an example using a real problem set to help illustrate the analysis process. Suppose, in our example, the user wants to use SZ's value-range-based error bound mode to compress as much as possible the Hurricane simulation dataset [18], such that R $\geq$ 0.99999. We tune SZ's relative error bound parameter (denoted by ϵ_{rel}) and the number of quantization bins (denoted by M) to get different quality and performance. The Hurricane simulation dataset has 13 fields with 48 time-steps, for a total of $13 \times 48 = 624$ buffers, each being a $d_{f,t}$. In this case, we set the nonfixed parameter vector $\vec{c}$ as $\{\epsilon, M\}$), with the first element representing the error bound parameter and the second element representing the number of quantization bins. We construct U by determining a vector of lower bounds and upper bounds. According to the SZ documentation [5], ϵ and M should be in the range of [0,1] and [1,65536], respectively. Thus, we have $\vec{l} = \{0, 1\}$ and $\vec{u} = \{1, 65536\}$.[2] This makes $\forall \vec{c}$ such that $\vec{l_i} \leq \vec{c_i} \leq \vec{u_i}, \vec{c} \in U$.

We set all of the remaining 25+ parameters to their defaults, forming the fixed-parameter vector ($\vec{\theta_c}$). With $d_{f,t}$ and $\vec{\theta_c}$, we compute $\tilde{d_{f,t}}(\vec{c}; \vec{\theta_c})$ for any choice of $\vec{c}$ that the search specifies by setting the appropriate compressor settings and running the compressor on the buffer $d_{f,t}$. Now, we must

2. If more restrictive bounds are known or desired, specifying those makes OptZConfig's search task more efficient.

define $\vec{\theta_m}$, $\mathcal{Q}$, and $\mathcal{Q}_\tau$. We let $\vec{\theta_m} = \{.99999\}$ corresponding to the R threshold of .99999 described above. In this case, $\mathcal{Q}$ is defined as follows:

$$\mathcal{Q}\left(d_{f,t}, \tilde{d_{f,t}}\left(\vec{c};\vec{\theta_c}\right);\vec{\theta_m}\right) = \begin{cases} \mathcal{CR}(d_{f,t},\vec{c};\vec{\theta_c}) & \text{if} \quad \mathcal{R}(d_{f,t},\tilde{d_{f,t}}(\vec{c};\vec{\theta_c})) \geq \vec{\theta_{m0}} \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where $\mathcal{CR}$ is the compressor's compression ratio on a given buffer and compressor configuration, $\mathcal{R}$ is the Pearson's correlation coefficient between original datasets and decompressed datasets, and $\vec{\theta_{m0}}$ = .99999 in this example. Based on this formula, if we reach the target R value in a compression case $\{d_{f,t},\tilde{d_{f,t}}(\vec{c})\}$, then our solution maximizes the compression ratio (CR) for the corresponding data buffer. $\mathcal{Q} = 0$ indicates that our solution skips all the configuration settings that do not meet the user-required R value threshold.

If the user specifies an acceptable value $\mathcal{Q}_\tau$ for all buffers in D, say a compression ratio 20, we terminate the search early if we find a value of $\mathcal{Q}$ that exceeds this value. We further distinguish between successful compression and unsuccessful compression tunings by checking if $\mathcal{Q}$ is 0. This formulation applies to the user's postanalysis metric that needs to be minimized (such as mean squared error or autocorrelation of errors).

It is possible that we are unable to find a configuration that satisfies the user's requirements either because insufficient resources were committed to the search or because the user requested an infeasible configuration. In these cases, we return to the user that the search has failed along with the best point found so far allowing them to determine if they wish to commit additional resources, widen their search area, or stop searching [3].

Often a configuration from the same application can be reused for other fields and time-steps while maintaining a sufficient configuration. How often does this happen in practice? Prior work has shown that on the full hurricane dataset (624 buffers), re-tuning was only required 8 times to compress all buffers to a feasible target [6]. This allows the costs of this method to be amortized over a large dataset.

4 Overall System Design

At a high level, we implement OptZConfig as a meta-compressor in the larger LibPressio ecosystem[4]. A meta-compressor implements all the features of a compressor, allowing it to be used with little change to the user's code, and OptZConfig takes advantages of new compressors as they are developed which may have advantages for specific problems or data-sets [12]. OptZConfig, unlike prior work in [6], is completely embedded within an application to be used online. Figure 1 demonstrates where OptZConfig fits within this larger ecosystem and its major subcomponents.

3. Finding a true optimal configuration while using black-box optimization is not possible because by definition we do not know a relationship between $\vec{c}$ and $\mathcal{Q}$ (c.f. [31]). We can identify the best configuration we observed during our search. When we refer to "best" or "optimal" configurations, this is the sense we intend it

4. LibPressio[32] is a library that provides an abstraction over common lossy and lossless compressors. It supports all of the compressors in this paper and more.

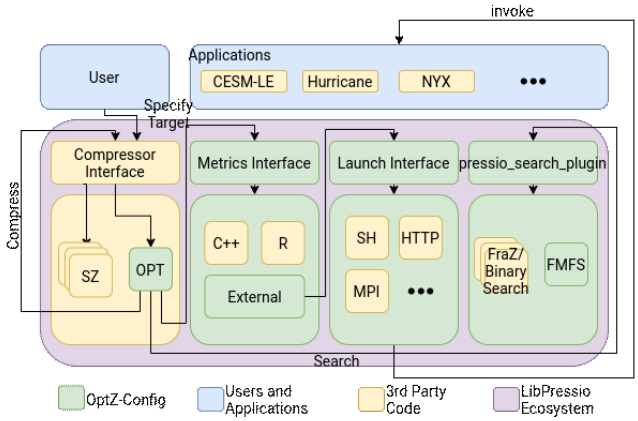


Fig. 1. LibPressio [32] ecosystem with OptZConfig's metacompressor plugin (OPT), search interface (pressio_search_plugin), and several default search modules including FMFS, FRaZ algorithm [6], and binary search.

The OptZConfig library consists of three sets of components: the `opt` metacompressor, the `pressio_search` interface, and many implementations of the search interface including FMFS and FRaZ's algorithm. The `opt` metacompressor provides the interface of a LibPressio compressor including a compress and decompress method. When compress is invoked, the `opt` will conduct a search to find a sufficient configuration to meet the users needs. The `opt` metacompressor acts as a runtime environment provide services to the search implementations such as: query if both the MPI implementation and the compressor supports threading, a callback to notify and check the early termination status, and a callback to invoke the compressor with a particular configuration and return a set of computed metrics for that configuration. The `opt` metacompressor is responsible for invoking the search implementation (such as FRaZ) which implements the `pressio_search` interface.

The `pressio_search` interface is the API that the runtime uses to determine how many and which points to evaluate while searching. It provides a single API `search` which receives as arguments the function object to invoke the callback from `opt` to evaluate the compressor with a given configuration and the callback to check and notify early termination returning the best configuration that it observed. Users can either provide their own implementation of this interface or use one of the builtin implementations.

OptZConfig consists of several builtin implementations of the search interface such as Binary Search, Fraz, and FMFS. These implementations each provide a search function used to identify candidate configurations. There are also "meta" implementations that can provide services for another search method. For example the distributed gird search method partitions a search domain into subdomains so each subdomain can be searched in parallel.

A key improvement we introduce in this paper is the Fixed Metric Fidelity Search (FMFS) search algorithm (see Algorithm 1). FMFS builds on the design of FRaZ's algorithm which builds on classical techniques in numerical optimization and parallelization [33], [34]. However, OptZConfig leverages additional information and cancellation callbacks to allow finer grained parallelism and cancellation, significantly improving performance and quality. The code

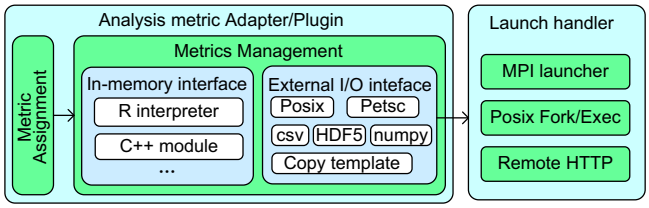


Fig. 2. Components in metric adapter and launch handler.

on lines 6, 14, and 16 represents new parallelism available (see Section 6.2). The code on lines 18-20 represents significant changes to communication (see Section 6.3. The code on lines 6 and 16 evaluates a user defined metric. No metric in Table 1 or Section 7 are possible in FRaZ without OptZConfig and FMFS.

```
1  begin search
2    mpi-parallel for f ∈ F_D do
3      last ← null;
4      for t ∈ T_D do
5        if last ≠ null then
6          q ← Q(d_{f,t,d̃_{f,t}(last.c⃗,θ⃗_c;θ⃗_m)};
             //thread-parallel
7          if good_enough(q) then
8            store(d_{f,t}, last.c⃗, q); continue;
9          end
10       end
11       partitions ← make_partitions(U);
12       mpi-parallel for
           p ∈ partitions; reduce(best : max) do
13         s ←
             derivative_free_search_partition(p, last.c⃗);
14         thread-parallel while ¬done(s) do
15           c⃗ ← next_point(s);
16           q ← Q(d_{f,t}, d̃_{f,t}(c⃗, θ⃗_c; θ⃗_m);
               //thread-parallel
17           best ← max(best, (q, c⃗));
18           if good_enough(q) then
19             global_notify_cancel(); break(2);
20           end
21         end
22       end
23       store(d_{f,t}, best.c⃗, best.q); last ← best;
24     end
25   end
26 end
```

Algorithm 1: FMFS

5 Computing Metrics

Another significant contribution of this work is a comprehensive framework to adapt user developed metrics for use with minimal effort to be used with any compressor. To this end, OptZConfig provides 47 commonly-used builtin metrics, 5 methods for using user provided metrics, and an interface for providing additional methods without recompiling OptZConfig; allowing users flex-ability.

We divide the Metrics types into in-core (dataset in RAM) and out-of-core (dataset communicated through the file-system or network). Figure 2 shows in-core metrics are direct implementations of the metrics interface. The out-of-core metrics are invoked from the "External I/O interface" module which handles process management and serialization to/from the formats used by external metrics applications via a launch method and a set of IO plugins.

In-core techniques are generally faster as they do not require reading/writing from external storage. However, some environments prohibit the spawning of additional processes beyond the initial `mpiexec`. In these cases, only in-core metrics modules may be permitted by the system administrator. The primary disadvantage is that metrics often require substantial rewrites to support embedding. OptZConfig provides two in-core methods: C++ shared library modules and embedded R scripts.

C++ shared libraries are the fundamental implementation of the metrics types. They allow the greatest degree of control over exactly how memory is used, allowing for a zero-copy usage for some metrics. They also provide more possible callbacks into LibPressio than other methods provide, enabling some classes of metrics which are currently not possible using other methods, for example, timing the compression. They can be used in a multi-threaded context, but users can opt out of this behavior by setting a flag.

The R metrics module embeds R in OptZConfig [5] to efficiently access all of the modules in R. The R Metrics module takes an R script and a list of output variables as input to compute metrics. However, due to limitations in R's implementation, access to the interpreter must be protected by a shared lock which limits scalability.

The primary advantage of out-of-core methods are that they require minimal changes to the user's application and avoid some of the threading limitations required for thread safety. OptZConfig currently provides three out-of-core-methods to maximize performance and meet specific environmental requirements: Fork+Exec, MPI_Comm_spawn, and HTTP endpoint. To use an out-of-core method, the user's application and OptZConfig need to agree on a protocol for communicating the decompressed data. To make this easy for users porting their applications, OptZConfig provides routines to write the data in many common formats including a binary blob, an HDF5 file, a numpy array, a PetSc Matrix, and CSV files. Additionally, we support dynamically loaded user defined serialization methods.

The MPI_Comm_spawn method uses MPI-2 features to spawn child MPI programs. Some MPI implementations do not allow calling Fork+Exec explicitly and others do not support nested invocations of MPI programs, and because OptZConfig uses MPI, using Fork+Exec can cause undefined behavior. However, because the MPI-2 standard envisions this routine may talk to a batch scheduler or other runtime system, some system administrators have disabled it, and some MPI implementations never implemented it.

An alternative solution on systems where MPI_Comm_Spawn is not allowed is to use the HTTP endpoint module [6]. It avoids the problem of nested MPI by calling out to a distinct process tree that does not use MPI. Thus, the the lifetime of the metrics program is extended beyond the life of the metrics invocation. This enables programs that are written in languages such as Julia or

5. R is a specialized programming language for statistical computing. It has sophisticated visualization and error analysis tools

6. Some HPC systems also prohibit the creation or use of HTTP endpoints on compute nodes. However at least the Palmetto cluster at Clemson University and Bebop Cluster at Argonne National Laboratory allow this – both top500 machines when they were introduced.

other systems, that have large startup times to be used for metrics.

We choose HTTP and not a more efficient protocol because of its ubiquity and wide support among many programming languages[7]. Additionally, if users want to use a more efficient protocol such as Mochi's Mango, Protocol Buffers, or Apache arrow, a launch plugin could be written and used.

6 Optimization of FMFS and OptZConfig

OptZConfig and FMFS make several important improvements over FRaZ, which is both an algorithm and an implementation [6]. First, the FRaZ implementation does not support being used online or embedded within an application as a metacompressor whereas OptZConfig does. Second, OptZConfig supports custom search methods. This allows OptZConfig to compete with advances in white-box-based methods (see Section 2.4) and adopt new searching techniques as they become available. Third, OptZConfig supports quality constraints on the objective — such as enforcing a specific PSNR or p-value on the KS test — and multidimensional searches allowing users to adjust multiple parameters of the compressor simultaneously to achieve higher compression ratios and higher quality than what is possible in FRaZ, which allows adjusting only the error bound. Finally, OptZConfig enables composable distributed-memory and multithreaded parallel search algorithms and compressors with fine-grained resource allocation by allowing searches to delegate a subdomain of the search to other implementations of the `pressio_search` interface.

FMFS utilizes the additional context and features of OptZConfig to account for up to a $60\times$ speedup over the FRaZ algorithm (see Section 7.2). We take a novel approach to parallelism for a black-box-based compressor auto tuner by allowing multithreading in the inner search algorithms, parallelism in the compressor implementation, and faster cancellation by moving the cancellation check between iterations of the underlying search algorithm rather than after all iterations. Sections 6.1, 6.2 and 6.3 describe the details.

6.1 Composable Parallelism

OptZConfig provides two types of search algorithms: concrete and metasearch algorithms. Concrete search algorithms implement a specific search algorithm. For example, `fmfs`, `fraz` and `binary_search` implement our new algorithm FMFS, the FRaZ algorithm from [6] and a simple binary search, respectively.

Metasearch algorithms (e.g., `guess_first`, `dist_gridsearch` modules) allow common search functionality to be implemented in a reusable way. The `guess_first` module implements the common functionality that attempts to test a prediction first before spawning an expensive search process. The `dist_gridsearch` takes a search request and spreads it out into a user-defined number of subgrids across the cluster, spawning a search on the search grid cell and enabling search methods that do not normally take advantage of distributed-memory parallelism to take advantage of it.

What makes this truly reusable, however, is the interface requirements for sharing resources. Numerous researchers find that when multiple frameworks attempt to control parallelism, parallelism suffers [35], [36], [37], [38]. OptZConfig addresses this problem by requiring well-known options for expressing the allocation of parallel resources such as CPU processes, MPI communicators, and (in the future) GPUs and other accelerators. Thus, users can specify where they would like to allocate different scarce parallel resources, giving them fine-grained control over the degree of parallelism.

6.2 Safe Multithreading

Not all search algorithms or compressors are thread safe in all use cases. For example, SZ 1.X and 2.X safely uses multiple threads if the compression parameters are identical between the threads. But because SZ has global state parameters that are shared between threads, data races are possible if these are modified while another thread is invoking the compressor. However, checking if the compressor is thread safe alone is insufficient. The library must also check the MPI implementation, and ensure each invocation gets its own compressor handle. To support this, OptZConfig's callback to invoke the compressor clones the output buffer and compressor object explicitly for each thread, thereby implicitly cloning the metrics object held by the compressor. These clones ensure that threads do not clobber a compressed buffer produced by other threads. The remaining objects are either stack allocated by each thread or are referenced in a read-only manner, preventing a data race.

Given the clones involved, it is important to consider the memory usage of this design. The outer-most loop of FMFS uses MPI parallelization giving each rank a seperate address space. In order to reduce memory usage at this level, a careful use of public, shared, virtual memory mappings ensures that each input is loaded exactly once and only on the rank that uses it. Additionally, the input buffers for each field and time-step that are not currently being processed can be paged out efficiently when they are not in use. In an HPC usage scenario, processes are typically allocated one thread per hardware core, and OptZConfig uses at most one input buffer per thread at any given time. The memory usage grows linearly with respect to the thread count and $O(1/x^2)$ with respect to the per-thread memory overhead. The largest single buffer used in our experiments is 539 MB. Assuming the worst case of a compression ratio of 1 for the compressed and output buffers, and the memory overhead from SZ (3.004 $\times$), ZFP (.6623 $\times$), or MGARD (4.412 $\times$)[8], the maximum memory usage per thread is at worst 4.0 GB per thread which fits well within the per-node memory limit.

6.3 Inter-Iteration Early Termination Support

The last major performance improvement is inter-iteration early termination support. Cancellation in OptZConfig is

7. While this method was intended for HTTP(S), the implementation uses libcurl and supports any protocol supported by libcurl.

8. measured via the peak resident set size for both compression and decompression, m, from `getrusage` for a data of input size i and compressed size c. Then computed as $(m - 2i - c) \div (i)$

TABLE 3
Which metrics are supported by which prior approaches?

Metric	Specialized Compressors	MGARD-QOI	FRaZ	OptZConfig
compression ratio	✓(ZFP)	×	✓	✓
compression bandwidth	×	×	×	✓
PSNR	✓(SZ, MGARD)	×⋆	×	✓
L_∞ norm	✓(MGARD)	✓	×	✓
Weighted Mean	×	✓	×	✓
Pearson's Coefficient	×	×	×	✓
P value for KS-test	×	×	×	✓
Spatial Relative Error	×	×	×	✓

TABLE 4
Hardware and Software Details

Component	Description	Component	Version
CPU	Intel Xeon 6148G (40 Cores)	Compiler	GCC 8.3.1
RAM	372 GB	OS	CentOS 8
Interconnect	100 GB/s HDR Infiniband	MPI	OpenMPI 3.1.6
GPU	2 Nvidia v100 with NvLink	LibPressio	0.72.2
MGARD	v1.0.0	SZ	v2.1.12
ZFP	v0.5.5		

cooperative rather than preemptive. We now use 1-sided MPI atomic remove direct memory access (RDMA) operations in MPI 2 for cancellation. At the beginning of the search process, the master exposes a RDMA window to each process. If the master wants to terminate, it writes to the memory window atomically. Peroidically the children preform an atomic load to see whether termination is requested. A worker requests a termination by preforming an atomic RDMA write on the window exposed by the master process.

We also implemented a version based on MPI `I_Bcast`. In this version, the workers begin a non-blocking broadcast on a dedicated communicator. If the master wants to terminate, it issues the final call to broadcast completing the operation. The workers use `MPI_Test` to see if the operation has completed. The workers can request termination by sending their next response with a specific tag to the master. The master then issues the termination request.

Micro-benchmarking finds the RDMA based version allows for workers to observe the termination request up to 30% faster. In practice, a larger speedup is seen up to 2×. We attribute this to two factors: 1) MPI implementations differ for how they communicate completion of a non-blocking broadcast which may require more nodes to cooperate. 2) in the RDMA based version, the worker requesting cancellation does not need to wait for the master process to explicitly process the request for cancellation eliminating possible queue-ing for a cancellation request.

7 Experimental Evaluation

We perform the evaluation as comprehensively as possible in four areas. As can be seen in Table 3, a key challenge for our evaluation is that there is not one metric that is supported by all three approaches taken in prior work [6], [7], [19]. Each comparison will need to use different evaluation metrics and compressors. First, in subsection 7.1, we evaluate OptZConfig against specialized compressors which are custom-built to preserve a specific metric. In subsection 7.1, we show that OptZConfig achieves up to 3 × improvements in compression ratio on some data sets over a state-of-the-art specialized compressor while yielding a tighter bounding on the user's metric. Next, in subsection 7.2, we evaluate the performance improvement over the FRaZ algorithm. In subsection 7.2, we show how the three performance optimizations we make over FRaZ contribute to a 56× speedup. After that, in subsection 7.3, we conduct a comparison against MGARD's Quantity of Interest Mode (MGARD-QOI) which is the only other compressor which bounds a subset of of user-defined quantities – bounded linear functional on regular grids. In subsection 7.3, we show over a 1000× speedup when both MGARD-QOI and OptZConfig are tuned. We show our technique reduces tuning time from 23.36 minutes for MGARD-QOI to at most 6 seconds for OptZConfig. We further demonstrate that OptZConfig without tuning is faster than the tuned version of MGARD-QOI 75% of the time and only up to 5× slower in the worst case. Finally, in subsection 7.4, we evaluate the runtime of OptZConfig on problems that are not solvable with any other current compressor. In subsection 7.4, we show a speed up over the prior systematic approach from 24 node hours to 13 node minutes – a 110× speedup.

We conduct all of these evaluations on nodes of Palmetto cluster (see Table 4). We select these nodes to have a large number of CPUs since at the time of writing most compressors GPU support is still maturing for all modes.

For our analysis we use a number of datasets from SDRBench[18]. These are summarized in Table 5. We choose these datasets because they represent regular-grids and have metrics of interests defined by their communities. Regular grids are the type of data structure that MGARD is designed to protect. SZ and ZFP work on other datasets as well, but we focus on regular grids for the purpose of comparison with MGARD [9]. In particular we present only Cloudf48 from Hurricane, Prec from CESM-LE, pressure from miranda, baryon_density from NYX and volume from SCALE-LETKF because other fields we observe are similar.

Additionally, our implementation and evaluation code is

9. It is possible to treat unstructured girds as if they are structured for purposes of compression by treating them a 1d structured grid of dimension N where N is the number of points. This has been done with SZ to compress data from HACC [19]. However some compressors (e.g. some versions of MGARD) do not allow any dimension to be less than 3 and be at least 2d so this is not universally true. Additionally doing so violates assumptions used by the compressors and can result in poor compression both in terms of quality and compression ratio.

TABLE 5
Datasets

Dataset	Description	total size	buffer size
CESM-LE	Climate Earth Science Model Ensemble	17GB	643MB
Hurricane	Weather data from Hurricane Isabelle	48GB	96MB
Miranda	Hydrodynamics turbulence simulation	1.87GB	288MB
NYX	Cosmology simulation	2.7GB	512MB
SCALE-LETKF	Local Ensemble Transform Kalman Filter	4.9GB	539MB

open source and can be found online [10]. Additional details such as threading configurations can be found in this code.

OptZConfig requires an underlying compressor be used. We present results here from SZ, ZFP, and MGARD the three leading lossy compressors listed on SDRBench[18] that currently have integration with LibPressio which OptZConfig uses to invoke the compressors at the time of development[11]. LibPressio provides an extensible interface to adopt other compressors as they are developed.

We focus here on CPU-based compressors rather than GPU based compressors largely because the GPU based compressors are still maturing. ZFP's GPU implementation supports only fixed-rate mode, cuSZ lacks a stable API, MGARD's GPU implementation only compiled after submitting patches and does not accelerate the norm finding operation for MGARD-QOI. Finally, at time of writing only ZFP's and MGARD's GPU implementation is supported in LibPressio which we rely upon for interfacing the compressors. We expect even greater speedups using GPU versions of the compressors when they mature. However, with the current implementations, there is only minimal impact from moving to the GPU. Including required data movement, MGARD-GPU compress/decompress cycle takes 11% longer than SZ-CPU and ZFP is slightly faster at 24% faster. In our experiments computing the user's metrics takes 90% of the time per iteration for a metric like the p-value of the KS-test, and more complex metrics such as those from HACC can take substantially longer limiting the scale-ability gained by using GPU based compressors and methods.

7.1 Comparison to Specialized Compressors

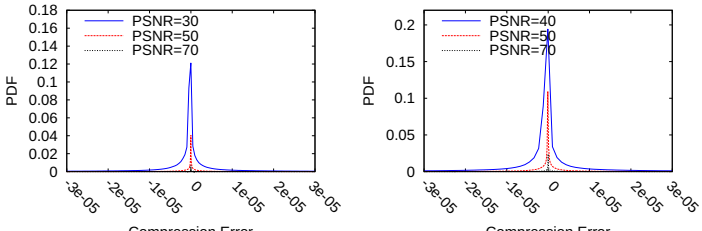
7.1.1 Background on Specialized Compressors

The needs of some applications have led to the development of specialized compressors which are designed specifically to enforce a particular error bound. Developing these can be expensive. Among leading lossy compressors there are only a few specialized natively supported user-defined metrics: PSNR (supported only by SZ; used in image analysis, climate science, and other domains), metrics supported in MGARD's QOI-mode (which are discussed in Section 7.3) and the L_∞ norm (supported only by MGARD; used in mathematics and finite element methods); the other modes preserve some point-wise bound which are not widely adopted in the analysis used by users. They offer high runtime performance but at the cost of development effort.

To evaluate OptZConfig against specialized compressors with built-in metric support, we use the PSNR mode. We choose PSNR because it is commonly used in the literature, and SZ and MGARD natively support it. Additionally, both SZ and PSNR are significantly faster to compute presenting a worst case for our approach. We exclude ZFP in this test because it lacks a native PSNR mode.

SZ fixes PSNR by relating the absolute error bound to the PSNR, which was proposed by Tao et al. [9]: $20 \cdot log_{10}(value\_range(d_{f,t})/\epsilon_{abs}) + 10 \cdot log_{10}12$ However in order to make it hold, there is a fairly strong assumption that the distribution of compression errors induced by the compressor must be uniform. However, much prior work has verified that other compressors often have a non-uniform distribution [22] meaning this approach does not necessarily generalize to other compressors. Moreover, the error distribution can vary significantly with error bound [19], so that inevitably PSNR cannot be controlled accurately based on the above formula [9]. Based on our observation with various application datasets, distribution of errors are non-uniform in most cases, as exemplified in Figure 3, which also explains well why SZ's PSNR mode suffers inferior compression ratios, to be shown later.

According to [7] standard MGARD can bound the PSNR using the configuration $s = 0$, $tolerance = (value\_range\,(d_{f,t}) \times \sqrt{3} \times 10^{(-p \div 20)}$ where p is the PSNR by bounding the L_2 norm. Note, this does not use the QOI functionality considered in detail in section 7.3 because PSNR can be infinite, it is not a bounded functional.



(a) SZ (b) ZFP

Fig. 3. Distribution of Compressor Errors of SZ and ZFP for Miranda:pressure with different data distortion levels

7.1.2 Metric for Evaluation

We compare six different configurations: OptZConfig+sz (uses OptZConfig to set an absolute error bound for SZ); OptZConfig+zfp (uses OptZConfig with ZFP to set the ZFP accuracy parameter); and sz+psnr (specialized version of SZ which bounds PSNR), OptZConfig+sz+tuned (uses OptZConfig to set the absolute error bound for SZ with a good guess for a sufficient configuration), OptZConfig+sz+tuned (uses OptZConfig to set the accuracy parameter for zfp with

10. Our implementation is spread between https://github.com/robertu94/libpressio_opt (opt, pressio_search, OptZConfig), https://github.com/robertu94/libpressio (changes made to libpressio), and https://github.com/robertu94/libdistributed (distributed early termination communication implementation) the experimental codes are https://github.com/robertu94/libpressio_opt_experiments.

11. Since development of this paper other SDRBench compressors such as tthresh, bitgrooming, and digit rounding have gained LibPressio support. Our design can use these compressors. We see similar overall behavior, but the compressors are less effective and less widely used than SZ, ZFP, and MGARD so we only present results from them.

a good guess for a sufficient configuration), and MGARD (using the native support described above). The goal of these configurations is to show how OptZConfig compares with the states of the art for bounding PSNR – SZ's PSNR mode and MGARD's PSNR mode.

We run the six configurations with several possible PSNR tolerances of 30 – 90 db as a set of plausible thresholds that a user might desire. *However*, for space and legibility, we only present the results from 60 – 90 db. The higher the PSNR threshold, the more challenging the problem is for OptZConfig to find a feasible solution because of the smaller number of nonfixed compressor settings that satisfy the bounds. Additionally, we allow OptZConfig to terminate early if it finds a solution with a compression ratio such that most PSNR tolerances require searching – greater than 60× – and the quality thresholds are met. During this evaluation, we find a solution for every configuration, slice, and tolerance desired.

On each run, we record the achieved compression ratio, compression time (including search time for OptZConfig-based methods), and the achieved PSNR. Compression time and compression ratio are two common measures to evaluate compressors. We consider the achieved PSNR to understand how much over-preservation occurs. For example, if asked to target a PSNR of 40dB, did the compressor get a higher PSNR like 80dB? Over-preservation is undesired because it will cause a much lower compression ratio and higher compression time than expected.

7.1.3 Results

We present our results in Figure 4. First, we consider compression ratios in Figure 4 (a). In general, OptZConfig achieves a higher compression ratio than what is possible using the state-of-the-art with the same underlying compressor, because the state-of-the-art compressors often over-preserve the PSNR by imposing a strict global bound rather than allowing individual points to vary when the overall constraint is met. Even with the high PSNRs in Figure 4 we can see improvements in compression ratios as high as 1.5×. With at a lower PSNR target of 30 (not shown), we achieve a compression ratio improvement of 3.2×. This over-preservation is highlighted in Figures 4(b) with the red horizontal lines demonstrating the PSNR target. MGARD is especially aggressive in preserving error and this is reflected in its comparatively lower compression ratios.

After that, we look at compression time in Figure 4 (c). In the case of SZ, the leading state-of-the-art method is faster than using OptZConfig without tuning because the OptZConfig tuning process invokes the compressor multiple times. Even given multiple invocations of the compressor, the runtimes of OptZConfig without tuning are often within an order of magnitude between SZ and OptZConfig because we minimize the number of search iterations; OptZConfig allows trade-offs of higher quality or/and compression ratios that may mitigate the differences in runtime in some use cases. Additionally, using the `guess_first` module, the user can specify the parameters as the results of the tuning as a prediction. With these tuned configurations, OptZConfig using SZ is very similar to run-time of SZ+PSNR since no tuning is performed. Again, prior work shows that one often can reuse predictions from prior time-steps or similar fields, resulting in even lower overhead [6].

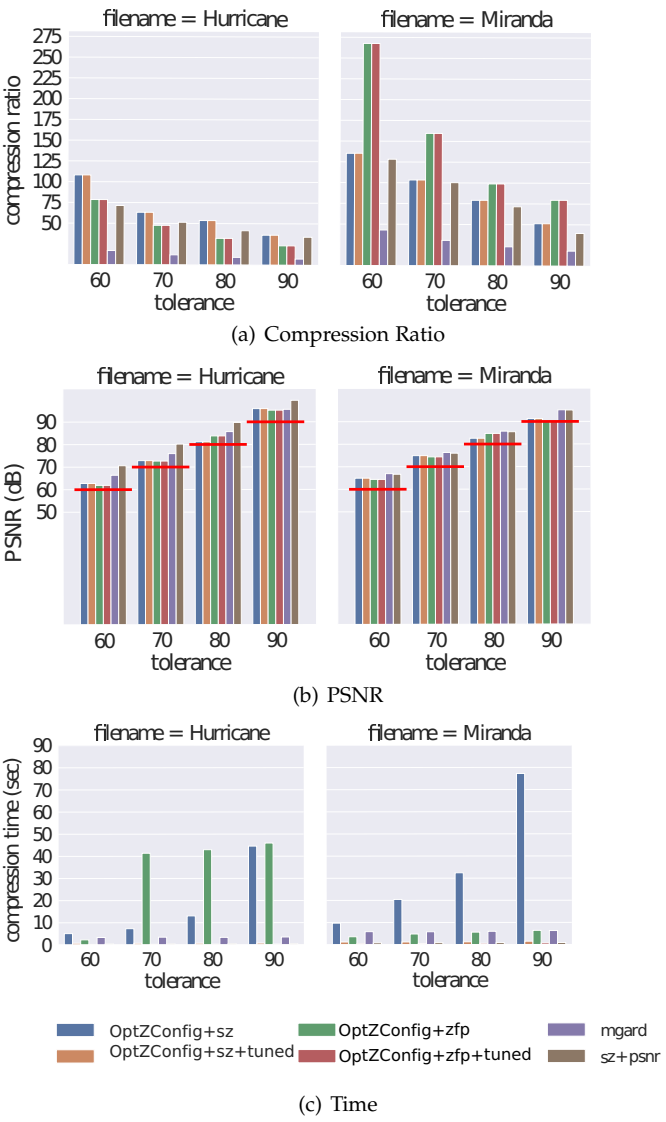


Fig. 4. Various methods attempting to maximize the compression ratio while maintaining a specified PSNR tolerance.

What accounts for the differences between SZ+PSNR and LibPressio+SZ+tuned? When a good guess is provided, OptZConfig needs to run compression just like SZ+PSNR, but it also needs to run decompression and the PSNR calculation since the PSNR cannot be computed from just the original and compressed buffer. If OptZConfig is preserving a metric such as compression ratio, the decompression step can be disabled. After verifying the configuration is within tolerance, no search is conducted. As for how to determine a good guess, we find that a run of OptZConfig from a similar experiment is often sufficient to get a good guess, but other methods may exist for specific metrics [12].

How much time does this metric evaluation and decompression time compare to the compression time? Compres-

12. A good guess of the PSNR, p, can be obtained for a compressor with an absolute error bound α and value range v is $\alpha = \frac{v}{10^{p/20}}$, this guess used by MGARD [7] is often too conservative (see Figure 4) and benefits from the refinement that OptZConfig can provide.

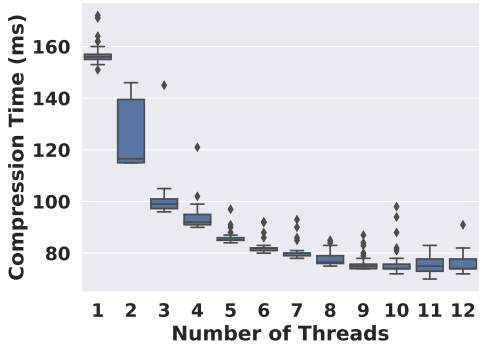


Fig. 5. Improved search time by 2.07× on average (a speedup over the FRaZ algorithm) when using 12 threads within ZFP

sion using SZ with a PSNR of 60 for a field from Hurricane can take about 460ms. Decompression takes an additional 220ms, and computing the PSNR takes an additional 100ms. Collectively this amounts to an additional 41% overhead over just performing compression. For this reason the number of iterations drives the execution time of our approach.

Why does the performance of our approach decrease as the PSNR decrease as the PSNR increases from 60 to 90dB? A target of 90dB is harder to achieve than a target of 60DB – that is fewer of the $\vec{c} \in U$ result in $\mathcal{Q}_\tau(d_{f,t}; \vec{\theta_m})$ being satisfied. Because in section 7.1.2 we define $\mathcal{Q}$ in a similar way to equation 1 (except with $\mathcal{R}$ being the PSNR) and the underlying derivative free search algorithm searches randomly while exploring U, we expect search times to be geometrically distributed where p is the proportion of $\vec{c}$ that results in $\mathcal{Q}_\tau(d_{f,t}; \vec{\theta_m})$ being satisfied. We leave exploration of different penalty function schemes that may result in faster convergence to future work.

7.2 Comparison versus FRaZ Algorithm

Allowing parallel compressors, parallel search techniques, and inter-iteration early termination improves the performance of the search over the techniques used in FRaZ algorithm. We first consider the impacts of each of these optimizations separately.

7.2.1 Improvements from Threading

First, we evaluate using multi-threading in the compressor. At time of writing: ZFP is the compressor which has the greatest support for multi-threaded compression, MGARD has no support for multi-threading compression, and SZ does not have multi-threaded implementations of all of its modes. Therefore, we run ZFP with the same default configuration and error bound except to allow increasing numbers of threads. Since threading performance for ZFP is variable, we run each configuration of threads 30 times using defaults for all other parameters. Figure 5 shows the decrease in compression time for increasing numbers of threads in the ZFP compressor on the CLOUDf48 buffer from the Hurricane dataset. Similar results were seen for other datasets using ZFP. Enabling threading in the compressor decreases the time spent evaluating each point during the search progress and improves search performance on average by 2.07× for ZFP by going from 1 to 12 threads.

Next we consider only threading the search, we again use ZFP since it is the only EBLC compressor that supports being called from a multi-threaded context with different configurations. As we increase the threading in the search only from 1 to 4 threads performance improves on average by 2.03× for different PSNR targets ($40 \rightarrow 1.88\times, 50 \rightarrow 1.97\times, 60 \rightarrow 2.24\times$). We again show results using buffers from the Hurricane dataset, but other results from other datasets are similar. Allowing multithreading of the search itself also has impacts on the execution time—as much as an additional 2.03× improvement in our tests. These improvements are similar to what we observe by using a distributed grid search. However, in the multithreaded implementation, the implementation of the search algorithm shares knowledge of the points as they are searched by other threads each iteration. This allows a better guiding of the search process. Currently, only ZFP supports being called from a multithreaded context, but this situation is expected to change with improvements to SZ and MGARD.

7.2.2 Benefits of Inter-iteration Early Termination

The benefits of allowing inter-iteration early termination are the most dramatic. Figure 6 shows the speed up when using FRaZ over OptZConfig with two different compressors on different slices from the Hurricane CLOUDf48 dataset to a user-defined PSNR tolerance. FRaZ does not support user-defined metrics, only compression ratio. For sake of comparison, we took the FRaZ algorithm and re-implemented it in OptZConfig to enable us to compare only the improvements from early termination making only the changes required to provide a user-defined metric. In this example, we allow FMFS and FRaZ to terminate early as soon as a feasible solution is found by setting the acceptable PSNR threshold to 0 to represent an upper bound for the effectiveness of this technique when used with PSNR and these compressors. PSNR was chosen because it can be evaluated in linear time, and is easy to implement. We then compute the speedup of the inter-iteration early termination over the intra-iteration early termination from earlier approaches [6]. When used with MGARD, inter-iteration early termination offers improvements as great as 15×. In contrast, when used with SZ, the improvements are closer to 2×. The key explanation of the difference here is the relative fraction of the search time spent invoking the compressor and computing the metrics. In its current implementation, MGARD is nearly 1000× slower than SZ. However, this time would effectively be increased with additional metrics. These metrics from the perspective of the search code are "part of" the compression time, meaning the more metrics or the more complex the metrics being computed, the more SZ would also benefit from inter-iteration early termination.

7.2.3 Overall Improvements

The three FMFS improvements affect different aspects of the search process, and could be used in conjunction if supported by the compressor and sufficient hardware is available. We ran an experiment that used the ZFP compressor with each of the various speedups enabled in order to find a configuration with a PSNR greater than 105 for the SCALE-LETKF V data buffer on a single node. SCALE-LETKF has the largest buffers of the datasets we consider.

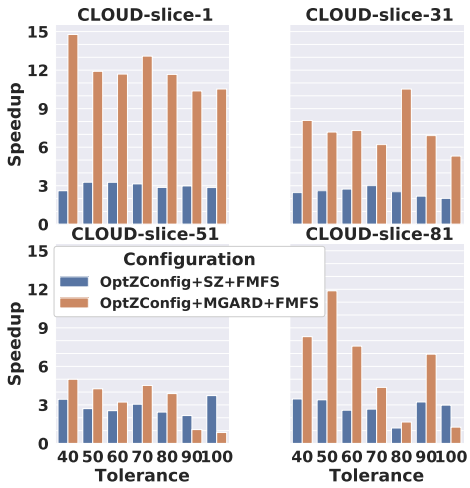


Fig. 6. Inter-iteration early termination speedup over FRaZ algorithm on a search for a desired PSNR. We include two underlying compressors for OptZConfig: SZ and MGARD. Because MGARD's implementation is slower than SZ's implementation for each invocation of the compressor, it benefits more from being able to terminate early.

We found that a configuration of 3 MPI processes (2 workers and 1 leader), where the workers spawned 5 search threads which in turn spawned 14 compressor threads[13] maximized performance on a single node[14]. In this configuration, we measured a 56.03× speedup over FRaZ. 83% of the time is spent in tuning the search, the remaining 17% corresponds to a single compress, decompress, compute metrics cycle to compute the final results. A single cycle of compression, decompression, and computing metrics took 780ms (17%), 900ms (19.7%), and 2900ms (63.3%) on average respectively. In total, only 6 invocations were used vs the 101 for FRaZ.

7.3 Comparison Versus MGARD-QOI

7.3.1 Background on MGARD-QOI

MGARD's quantity of interest mode (MGARD-QOI) introduces an analytic approach that relies on mathematical properties of supported metrics being computed to solve for an error bound that is mathematically guaranteed to preserve the metrics[15]. It works by computing a scaling factor called the "norm of the quantity of interest" and properties of metrics it supports to bound the error in the metric in terms of the H_s norm chosen by the user.

Specifically, MGARD-QOI requires that the metric be a bounded linear functional computed on a regular grid. The term bounded linear functional implies that the distributive property hold for the metric. An example of a bounded linear functional is the arithmetic mean. If one scales the dataset or add two datasets together, the mean is the same regardless of the order of these operations. The term regular grid applies to many HPC applications where a domain is discretized into equal-sized and equally-spaced distinct chunks to compute the effects inside the domain.

13. Our implementation requires that the thread count be a multiple of the data dimension in order to reduce copies

14. Over-subscription in this case overlaps serial and highly parallel portions of ZFP's compression and decompression code from different thread teams maximizing performance

15. What we call metrics in this paper are referred to as quantities of interest in [7]. The reason is that in the mathematics community the term metrics has a strict mathematical definition. We choose to use it in the broader computer science usage here.

While MGARD-QOI places restrictions on the metrics and problems supported, these restrictions allow the quantities of interest for many scientific codes. The complete proof of how this works is in their paper [7]. In contrast, the approach offered by OptZConfig supports a strict superset of metrics supported by MGARD-QOI by supporting non-regular grids inputs, and metrics that are not bounded linear functional. The spatial error metric is an example of a metric that is not a bounded linear functional because it allows unbounded error on a point by counting the percentage of points that exceed a threshold. In fact, **all** of the other metrics mentioned outside this section are not bounded linear functionals and cannot be used with MGARD-QOI.

7.3.2 Metric for Evaluation

The *weighted* mean is appealing for evaluation because it is a bounded linear functional but less trivial to bound. This is because some entries may have a weighting near zero [16]. Therefore, an implementation that uses the absolute error bound times its weight for a cell over-preserves information if the cell has small compression error.

We attempted a comparison between MGARD-QOI mode and OptZConfig using the weighted mean metric using a full-sized buffer; however this was performance prohibitive. As written, however, the MGARD-QOI calculation takes $O(\|d_{f,t}\|)$ evaluations of the metric function. The weighted mean itself takes $O(\|d_{f,t}\|)$ time, making the time to find the norm of the quantity of interest $O(\|d_{f,t}\|^2)$. Even if parallelized (current implementation is serial), this requires $O(\|d_{f,t}\|^2/p)$ time where p is the number of processors. If run on a full-size problem, finding the norm of the quantity of interest is expected to take between 17 and 19 days. We confirm this analysis by timing the first 250,000 basis values (1% of the basis values of the full dataset) on a full sized problem, which takes nearly 4.5 hours.

Instead we focus on the four 3% slices of the input data. We present results from only four $3 \times 500 \times 500$ slices from the 48th time-step of the CLOUDf48 field in the Hurricane dataset from SDRBench[18]. Results from other datasets and their full sized counterparts are similar. This approach is consistent with the slice-by-slice analysis used in the Climate community for CESM [16]. We choose the four slices to represent different problem difficulties: slice-1 is sparse and therefore easy to compress to a specified tolerance; slice-81 is a little bit harder; slice-31 is harder still; and slice-51 is less sparse, making it the hardest to compress.

Empirically, the time required to compute the norm of the quantity of interest on a 3% sample MGARD-QOI mode takes on average 23.36 minutes ± 0.04. This is scaled down from the full computation in two ways. First, in this problem, we are computing only the first 3% of the basis values. Second, in this problem, each basis value takes 3% of the time that it takes on the full problem.

7.3.3 Experimental Results

For fair comparison, this expensive norm-finding operation does not have be performed with every evaluation of the

16. a weighting near zero is useful when you want to ignore "ghost cells" or the edges of detectors for analysis

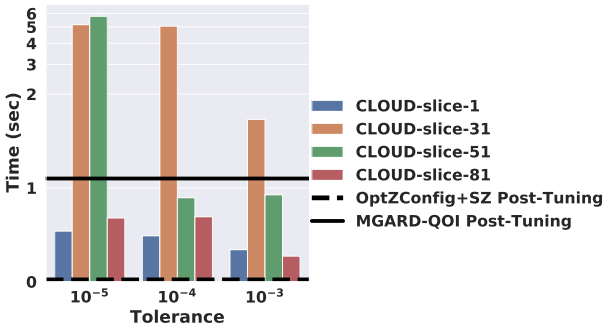


Fig. 7. OptZConfig vs MGARD-QOI for weighted mean: OptZConfig is 83% faster 75% of the time without tuning OptZConfig; with tuning OptZConfig+SZ is over 1000× faster in all cases.

TABLE 6
Runtime for systematic sampling vs OptZConfig

method	samples (1 node)	runtime	CR found
systematic	10	17s	no solution
systematic	100	33s	no solution
systematic	1000	3m45s	no solution
systematic	10000	35m40s	no solution
systematic	100000	6h (4 nodes)	7.264
OptZConfig	n/a	13m	7.3046

compressor. The norm can be reused as long as the dimensions of the input data remain constant and the metric does not change. Thus, for some quantities of interest that are not dependent on the original data, one does not have to recompute the norm of the quantity of interest even if the values of the input data are changed. Additionally, quantities of interest with this property converge to a constant times the number of bases computed as the number of bases increases. Thus, users can often bootstrap their problem with a smaller version for faster computing of the norm. However, because the weighting changes with the size of the matrix, this kind of bootstrapping is not viable in this case.

There are two possible cases for fair comparisons between MGARD-QOI and OptZConfig for the metrics that MGARD-QOI supports: (1) MGARD-QOI with-tuning vs OptZConfig with-tuning, and (2) MGARD-QOI with-tuning vs OptZConfig without-tuning. Case (1) is most appropriate when there is the potential for substantial variation between the compressor configurations required to preserve a user error metric – such as a turbulent simulations or shock codes. Case (2) is most appropriate when the configuration is likely to be similar between time-steps for fields as prior work has shown is true for many HPC codes [32].

To evaluate the differences between these two cases, we used MGARD-QOI and OptZConfig with SZ in absolute error bound mode to bound the weighted average for each of the different slices listed above. We attempted to bound the absolute error in the weighted average to $\pm 10^{-3}, 10^{-4}$, and 10^{-6} with both compressors and measured the tuning time for OptZConfig, compression times, and compression ratio. We attempted to include OptZConfig+MGARD to have a OptZConfig+MGARD vs MGARD-QOI evaluation as well, but were unable to do so due to implementation flaws in MGARD. OptZConfig+ZFP results were similar to the OptZConfig+SZ results.

Figure 7 summarizes the results for Case 1 and Case 2 of this experiment. In the figure, the black line represents the time that MGARD-QOI took post tuning and was identical for all tolerances and files. The dashed line represents the post-tuning for OptZConfig and was the same for all tolerances and files. The bars represent the time for compression and tuning using OptZConfig. First, let's consider Case 1: We are able to tune SZ compression faster than MGARD compresses the same buffer using QOI mode when the QOI norm is already found in 75% of the cases we test. In those cases where OptZConfig is faster, the average time taken by OptZConfig using SZ is 0.6 seconds compared to the 1.1 taken by MGARD-QOI mode. In the other 25% of cases, OptZConfig was $2-5\times$ slower than MGARD with its tuning already complete. In Case 2 after OptZConfigs' tuning was performed, OptZConfig with SZ was over 1,000 times faster than MGARD-QOI with tuning in all cases.

7.4 Performance on Non-trivial Metrics

To provide more evidence of OptZConfig's effectiveness on previously un-automated problems, we consider a real-world set of constraints from the climate community that cannot be bounded by any other currently available compressor including MGARD-QOI and FRaZ. Section 2 states metrics and bounds for PSNR, the p-value of the KS test, the spatial error percentage, and the R value for the raw data using the thresholds from Table 1. We add PSNR to this evaluation as it is discussed in [16]. However, the threshold we use is arbitrarily determined since a bound for PSNR was not identified. Figure 8 shows that PSNR never binds over the search space.

For this evaluation, we use a buffer from the CESM application. Table 1 shows the thresholds we use. For this evaluation, we use SZ in its absolute error bound mode and consider one parameter, the error bound, for the sake of tuning time. OptZConfig finds a solution that meets all of the constraints with a compression ratio of $7.3046\times$ in 13.1642 minutes using 40 cores of one node. Of this time, 95% is spent computing metrics on the decompressed data. Figure 8 shows a graphical summary of that evaluation.

Figure 8 shows that over the search range, the KS-test p-value was the only binding metric[17]. Most of the other values were not close to their allowed tolerance. The R value is not able to even detect loss in this error bound range. The spatial error percentage is able to detect at most 0.0017% spatial error at its most extreme value. Over the search range, the spatial error percentage, KS Test p-value, and the PSNR are not monotonic. This nonmonotonic behavior defeats more naïve approaches such as binary search [6].

Because binary search-like approaches are not applicable to this problem, users would have to resort to an approach similar to OptZConfig or some kind of systematic evaluation. For a performance comparison, we conducted a systematic evaluation of various numbers of points between the error bounds we provide to OptZConfig 10^{-18} and 10^{-12}. For the systematic evaluation, we divide the state space into evenly sized bins and evaluate the compressor on the midpoint of each bin, and record all points that meet

17. binding means that a metric crossed it's allowable threshold

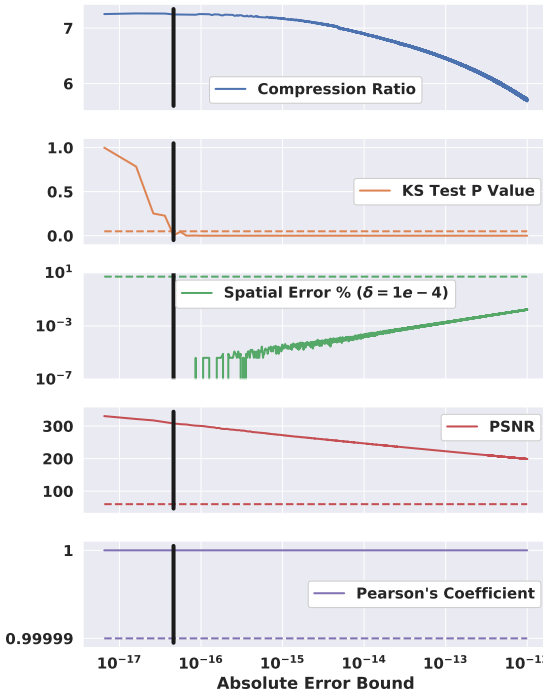


Fig. 8. Evaluation metrics at 100,000 error bounds between 10^{-18} and 10^{-12}. The PSNR, KS-test p-value, and R value must be above their threshold in order to satisfy the constraints. The spatial error % must be below its threshold. All points to the left of the black line satisfy the constraints.

the constraints outlined in this section. The results of this process are in Table 6. With even a moderate number of points, the systematic approach was not able to identify a valid solution. This is because a wide range of allowed error bounds result in infeasible requests given the user's constraints. If the user had placed a time limit on the search shorter than the time required, our approach would have failed and returned the result which satisfied the most constraints. The only systematic process which found a solution evaluated 100,000 points and took a little over 6 hours running in parallel on 4 nodes compared to the 13 minutes for OptZConfig on one node. The best result from the systematic search has a compression ratio of 7.264×, which is about .5% worse than using OptZConfig's solution taking 27× more time on 4× as much hardware.

8 Conclusions

Techniques such as OptZConfig and FMFS have great promise in helping applications cope with moving and storing ever-increasing volumes of data. It offers a higher-performance method of setting error bounds to preserve user metrics than do prior methods such as FRaZ or MGARD-QOI. OptZConfig also offers new capabilities to bound arbitrary user metrics and provide fine-grained control over the search. We demonstrate that OptZConfig can get up to a 3× improvement in compression ratio for equivalent PSNR requirements as SZ's specialized mode. We further show up to a 56× speed up over FRaZ when using compressors that support being used from a threaded context. OptZConfig further offers a 233× improvement over MGARD-QOI tuning time, and are 1000× faster than MGARD-QOI post-tuning. Lastly, we demonstrate our method on 3 metrics from the climate community achieving a 110× performance improvement over the systematic approach used previously for these metrics.

For future work, we would like to consider approaches that bound more complex metrics that are multi-valued such as spectral metrics and/or require multiple buffers to be tuned simultaneously to preserve the desired in-variants. Further study is needed on the trade-offs between speed and quality of the results obtained by a method like OptZConfig is warranted. Additionally techniques that would further improve the performance of the search such as proxy metrics (using a cheaper metric in place of a more expensive one if you can show that the cheaper one bounds the others) and better penalty functions than the one proposed in equation 1 to give the search algorithm additional guidance.

References

[1] L. P. Deutsch, "RFC 1952 GZIP File Format Specification Version 4.3," 1996. [Online]. Available: http://www.zlib.org/rfc-gzip.html
[2] I. Facebook. Zstandard - Real-time data compression algorithm. [Online]. Available: https://facebook.github.io/zstd/
[3] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," vol. 58, no. 1, pp. 18–31, 2009.
[4] S. Di, "Error-bounded lossy data compressor (for floating-point/integer datasets): szcompressor/SZ," 2019. [Online]. Available: https://github.com/szcompressor/SZ
[5] S. Di and F. Cappello, "Fast Error-Bounded lossy hpc data compression with SZ," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 730–739.
[6] R. Underwood, S. Di, J. C. Calhoun, and F. Cappello, "FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-Point Data," in *34th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2020.
[7] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—quantitative control of accuracy inderived quantities," vol. 41, no. 4, pp. A2146–A2171, 2019.
[8] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen, and F. Cappello, "Significantly improving lossy compression for HPC datasets with second-order prediction and parameter optimization," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 89—-100.
[9] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Fixed-PSNR lossy compression for scientific data," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 314–318.
[10] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley *et al.*, "HACC: extreme scaling and performance across diverse architectures," *Communications of the ACM*, vol. 60, no. 1, pp. 97–104, 2016.
[11] S. supercomputer at ORNL, https://www.olcf.ornl.gov/summit/, online.
[12] 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," vol. 33, no. 6, pp. 1201–1220, 2019.
[13] S. F. Shandarin and N. S. Ramachandra, "Topology and geometry of the dark matter web: a multistream view," *Monthly Notices of the Royal Astronomical Society*, vol. 467, no. 2, pp. 1748–1762, 01 2017.
[14] ——, "Dark matter haloes: a multistream view," *Monthly Notices of the Royal Astronomical Society*, vol. 470, no. 3, pp. 3359–3373, 06 2017.
[15] A. H. Baker, H. Xu, D. M. Hammerling, S. Li, and J. P. Clyne, "Toward a multi-method approach: Lossy data compression for climate simulation data," in *High Performance Computing*, ser. Lecture Notes in Computer Science, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Springer International Publishing, 2017, vol. 10524, pp. 30–42.

[16] A. H. Baker, D. M. Hammerling, and T. L. Turton, "Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data," vol. 38, no. 3, pp. 517–528, 2019.

[17] J. L. H. Jr., *The Signifigance Probablity of the Smirnov Two Sample Test*. Arkiv fiur Matematik, 1958, vol. 3, no. 43.

[18] F. Cappello, M. Ainsworth, J. Bessac, M. Burtscher, J. Y. Choi, E. Constantinescu, S. Di, H. Guo, P. Lindstrom, and O. Tugluk. (2018) Scientific data reduction benchmarks. [Online]. Available: https://sdrbench.github.io/

[19] 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)*, 2017, pp. 1129–1139.

[20] X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello, "An efficient transformation scheme for lossy data compression with pointwise relative error bound," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 179–189.

[21] P. Lindstrom, "Fixed-rate compressed floating-point arrays," vol. 20, no. 12, pp. 2674–2683, 2014.

[22] ——, "Error distributions of lossy floating-point compressors," 2017.

[23] A. Fox, J. Diffenderfer, J. Hittinger, G. Sanders, and P. Lindstrom, "Stability Analysis of Inline ZFP Compression for Floating-Point Data in Iterative Methods," vol. 42, no. 5, pp. A2701–A2730, 2020. [Online]. Available: https://epubs.siam.org/doi/10.1137/19M126904X

[24] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," vol. 19, no. 5-6, pp. 65–76, 2018.

[25] ——, "Multilevel techniques for compression and reduction of scientific data—the multivariate case," vol. 41, pp. A1278–A1303, 2019.

[26] D. King. (2018) Dlib C++ Library - Optimization. [Online]. Available: http://dlib.net/optimization.html#global_function_search

[27] M. J. D. Powell, "The NEWUOA software for unconstrained optimization without derivatives," in *Large-Scale Nonlinear Optimization*, G. Di Pillo and M. Roma, Eds. Springer US, 2006, vol. 83, pp. 255–297.

[28] C. Malherbe and N. Vayatis, "Global optimization of Lipschitz functions," 2017. [Online]. Available: http://arxiv.org/abs/1703.02628

[29] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP," vol. 30, no. 8, pp. 1857–1871, 2019.

[30] J. Kunkel, A. Novikova, E. Betke, and A. Schaare, "Toward Decoupling the Selection of Compression Algorithms from Quality Constraints," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham: Springer International Publishing, 2017, vol. 10524, pp. 3–14.

[31] S. Hudson, J. Larson, J.-L. Navarro, and S. Wild, "libensemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[32] R. Underwood, "CODARcode/libpressio," Codesign Center for Online Data Analysis and Reduction, 2019. [Online]. Available: https://github.com/CODARcode/libpressio

[33] A. Zhigljavsky, A. Zilinskas, and J. Birge, *Stochastic Global Optimization*. New York, NY, UNITED STATES: Springer, 2007.

[34] "Covering methods," in *Global Optimization*, ser. Lecture Notes in Computer Science, A. Törn and A. Žilinskas, Eds. Berlin, Heidelberg: Springer, 1989, pp. 25–52.

[35] A. Malakhov, "Composable multi-threading for Python libraries," in *Python in Science Conference*, 2016, pp. 15–19.

[36] M. P. Matijkiw and M. M. K. Martin, "Exploring coordination of threads in multi-core libraries," p. 8, 2010.

[37] H. Ribic and Y. D. Liu, "AEQUITAS: Coordinated energy management across parallel applications," in *Proceedings of the 2016 International Conference on Supercomputing - ICS '16*. ACM Press, 2016, pp. 1–12.

[38] Y. Tanaka, K. Taura, M. Sato, and A. Yonezawa, "Performance evaluation of OpenMP applications with nested parallelism," p. 14, 2000.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant numbers: NRT-DESE 1633608, 1619253, and 1910197. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The material was also supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

Robert Underwood is a PhD from Clemson University now at Argonne National Laboratory. His research interests involve using approximate computing methods such as lossy data compression to accelerate HPC while ensuring that scientific data integrity is preserved. He is currently working on using optimization based approaches to configure lossy compression. Email: runderwood@anl.gov

Jon C. Calhoun is an Assistant Professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. He received a B.S. in Computer Science and a B.S. in Mathematics from Arkansas State University in 2012, and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2017. His research interests lie in fault tolerance and resilience for HPC systems and applications. Email: jonccal@clemson.edu

Sheng Di (Senior Member, IEEE) received his master's from Huazhong University of Science and Technology in 2007 and Ph.D. from the University of Hong Kong in 2011. He is currently a computer scientist at Argonne National Laboratory. Dr. Di's research interest involves resilience on HPC and cloud computing. He works on multiple projects, such as detection of silent data corruption, characterization of failures and faults for HPC systems, and optimization of multilevel checkpoint models. Email: sdi1@anl.gov.

Amy Apon (Senior Member, IEEE) is the C. Tycho Howle Director of the School of Computing and Professor of Computer Science, Clemson University. Her research interests include data intensive computing systems and analytics, scalable machine learning methods, and commercial cloud technologies. She received an MA in mathematics and an MS degree in computer science from the University of Missouri, Columbia, MO, USA and a Ph.D. degree in computer science from Vanderbilt University, Nashville, TN, USA. Email: aapon@clemson.edu

Franck Cappello (Fellow, IEEE) is the director of the Joint-Laboratory on Extreme Scale Computing gathering seven of the leading HPC institutions in the world. He is a senior computer scientist at Argonne National Laboratory and an adjunct associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He is an expert in lossy compression, resilience, and fault tolerance for scientific computing and data analytics. Email: cappello@mcs.anl.gov.