ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression

Eric R. Schendel [1,2,+], Ye Jin [1,2,+], Neil Shah [1,2], Jackie Chen [3], C.S. Chang [5], Seung-Hoe Ku [4], Stephane Ethier [5], Scott Klasky [2], Robert Latham [6], Robert Ross [6], Nagiza F. Samatova [1,2,*]

[1] *North Carolina State University, NC 27695, USA*
[2] *Oak Ridge National Laboratory, TN 37831, USA*
[3] *Sandia National Laboratory, Livermore, CA 94551, USA*
[4] *New York University, New York, NY 10012, USA*
[5] *Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA*
[6] *Argonne National Laboratory, Argonne, IL 60439, USA*
[*] Corresponding author: samatova@csc.ncsu.edu
[+] Authors contributed equally

***Abstract*—Efficient handling of large volumes of data is a necessity for exascale scientific applications and database systems. To address the growing imbalance between the amount of available storage and the amount of data being produced by high speed (FLOPS) processors on the system, data must be compressed to reduce the total amount of data placed on the file systems. General-purpose lossless compression frameworks, such as zlib and bzlib2, are commonly used on datasets requiring lossless compression. Quite often, however, many scientific data sets compress poorly, referred to as *hard-to-compress* datasets, due to the negative impact of highly entropic content represented within the data. An important problem in better lossless data compression is to identify the hard-to-compress information and subsequently optimize the compression techniques at the byte-level. To address this challenge, we introduce the In-Situ Orthogonal Byte Aggregate Reduction Compression (ISOBAR-compress) methodology as a *preconditioner* of lossless compression to identify and optimize the compression efficiency and throughput of hard-to-compress datasets.**

I. Introduction

In the last ten years of High Performance Computing (HPC), we have seen an increasing imbalance between the FLOPS of the machine and the file system bandwidth. This imbalance necessitates the need to reduce the data before it is written to the file system; but due to the increasing complexity of the data from many simulations, standard compression techniques often become limited in their usefulness. $Lossless$ compression techniques offer no more than 20% reduction on many single and double-precision floating-point scientific datasets that we have tested on. These datasets are considered *hard-to-compress* due to the minute gains obtained from the use of such compression processes to reduce the data size. Moreover, the low throughput of data compression and decompression makes these techniques hardly suitable for *in-situ processing* (real-time processing of the data during simulation run), which is required for applications, such as those with simulation checkpoint and restart data.

To bridge this gap, we introduce a strategy that enables fast, effective, and high-throughput reduction of single- and double-precision floating-point scientific data. The intuition behind this method arises from the use of $preconditioners$ for improving the convergence rate of iterative $solvers$ in linear algebra, such as Algebraic Multigrid (AMG) [13], Quasi-Minimal Residual (QMR) [14], LDL solver [5], and others. Other preconditioning processes such as matrix factorization (e.g., QR factorization) are widely used by applications including aeronautics and fluid dynamics. To the best of our knowledge, such preconditioning techniques optimizing input for solvers have not been used in the lossless data-compression realm.

The preconditioner introduced in this paper is called *ISOBAR-compress*. ISOBAR-compress allows fast analysis of data and is capable of identifying characteristics in data that result in poor compression ratio and compression/decompression throughput. Our preconditioner analyzes the compressibility of data and creates the appropriate pipelines for compression of various datasets. Specifically, it decides how to partition data into compressible and incompressible segments, how to linearize multi-dimensional data, and also which compressor should be used to optimize compression performance in terms of storage or speed (user-specified). ISOBAR-compress essentially circumvents the additional complexities presented by multi-dimensional data by performing roughly the same for original data linearized in different means. Upon testing ISOBAR-compress on 24 scientific datasets of 7 applications summarized in Table I (see Appendix for more details), we found that 19 of them were identified as ISOBAR-compress improvable hard-to-compress datasets. On these datasets, ISOBAR-compress provided both higher throughput (varying from 100MB to 450MB per second) and improved compression ratios than those obtained without its use; 2 datasets had about 40% increase in compression ratio (ΔCR, Eq. 3), 13 of 19 had at least a 20% increase in the same, and the other 6 datasets experienced compression enhancements in the range of $[5\%, 10\%]$. In addition, ISOBAR offered a multi-fold increase in compression and decompression throughputs.

TABLE I
CHARACTERISTICS OF SIMULATION OUTPUT DATASETS FROM SEVEN APPLICATIONS

Applications	Research Area	Variable(s)	Data Type	Reference
GTS	Fusion Plasma Core	density, potential	double	[29]
XGC	Fusion Plasma Edge	igid, iphase	double, integer	[19]
S3D	Combustion	temperature, vmagnitude	float	[11]
FLASH	Astrophysics	velocity	double	[4]
MSG	NAS Parallel Benchmark (NPB) and ASCI Purple	bt, lu, sp, sppm, sweep3d	double	[7]
NUM	Numeric Simulations	brain, comet, control, plasma	double	[9], [26]
OBS	Measurements of Satellite	error, info, spitzer, temp	double	[8]

[See Table II for some examples of ISOBAR-compress performance and Results section for details.]

TABLE II
ISOBAR-COMPRESS PERFORMANCE SUMMARY

Dataset	Δ CR (%)[1]	TP_C[2]	Sp_C[3]	TP_D[4]	Sp_D[5]
GTS	10.15	111.7	8.05	551.90	5.01
XGC	14.09	76.83	21.17	388.87	51.92
S3D	32.56	104.73	31.45	424.79	63.12
FLASH	17.52	455.83	35.89	1617.02	14.19

[1] Δ CR (%): Percentage improvement of compression ratio (see Equation 3) comparing to the best alternative
[2] TP_C: Compression throughput in MB (megabyte) per second
[3] Sp_C: Speed-up of compression (see Equation 2)
[4] TP_D: Decompression throughput in MB per second
[5] Sp_D: Speed-up of decompression (see Equation 2)

We aim to optimize the solver (i.e., compressor) portion of the data reduction pipeline by enabling our preconditioner to function with any type of general-purpose compressors. Thus, a user can specify a preference in compressor to use with little to no change to our preconditioning method. We commonly use `zlib` and `bzlib2` as solvers, but could just as easily use `fpzip`, `FPC`, and various other tools (each may provide a different tradeoff in terms of throughput and compression ratio).

$$CR = \frac{Original\ Data\ Size}{Compressed\ Data\ Size} \quad (1)$$

$$Sp = \frac{Throughput\ of\ ISOBAR\text{-}compress}{Throughput\ of\ Standard\ (De\text{-})Compressor} \quad (2)$$

$$\Delta CR = (\frac{CR_{ISOBAR}}{CR_{Standard}} - 1) \times 100\% \quad (3)$$

II. METHOD

To understand what makes datasets hard-to-compress, we analyzed several double-precision floating-point datasets (64-bit) at the bit-level for their probability distributions (see Figure 1). When a bit position has a probability distribution of 1.0, it means that there is an absolute guarantee that the bit position value will be either 0 or 1 for all the values in the entire dataset. On the other hand, a probability distribution of 0.5 means the bit value for a given bit position has an equal probability of either being 0 or 1 for all the values. Based on this observation and experimentation, *xgc_igid*, *gts_zeon*, and *flash_gamc* (see Figure 1) are considered hard-to-compress datasets, whereas *msg_sppm* is not. Often, the first two bytes have high probabilities due to the representation of double-precision values as exponent and mantissa segments. Exponent values are often close together due to locality of data. Mantissa bytes can, in some cases have high probability based on degree of approximation and amount of precision needed, but are typically not predictable. The assumption is that the 0.5 probability distribution bits made the dataset hard-to-compress due to lack of predictability; this lack of predictability, or presence of randomness can be considered as *noise* in the data. In signal processing, filtering or denoising methods are widely used to improve the *compression efficiency* (a compressor's applied performance on a dataset) of a signal by discarding the noise [23]. Inspired by this idea, we worked with the notion that by identifying and extracting "noisy" data and only compressing the remaining "signal" data, we will obtain a better compression efficiency and throughput for any given general lossless compressor.

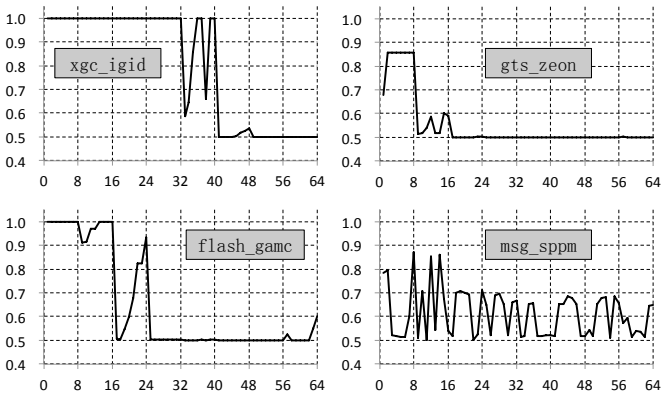

Fig. 1. Bit frequencies of 4 representative datasets; x-axis represents bit position (1 to 64), y-axis represents the probability distribution ranging from 0.5 to 1.0 of the more common bit values at that position (0 or 1)

The following subsections explore, in further detail, the central components that compose the In-Situ Orthogonal Byte Aggregate Reduction Compression (ISOBAR-compress) preconditioner. The preconditioner's objective is to identify the noise-like properties within a dataset that negatively impact compression efficiency and reduce the burden on the compressor from processing such noise. The preconditioner workflow is illustrated in Figure 2. There are two main components that make up ISOBAR-compress: (1) ISOBAR-analyzer, which is

responsible for identifying the "high complexity" data [24] (noise) that makes a dataset hard-to-compress, and (2) the ISOBAR-partitioner, which is responsible for segmenting out the noise that is considered hard-to-compress from the signal-like data, thus improving the compression efficiency [25]. The remaining components (EUPA-selector, general lossless compressor, and merger) of ISOBAR-compress are important for successfully implementing the workflow and are also discussed in the next few subsections.

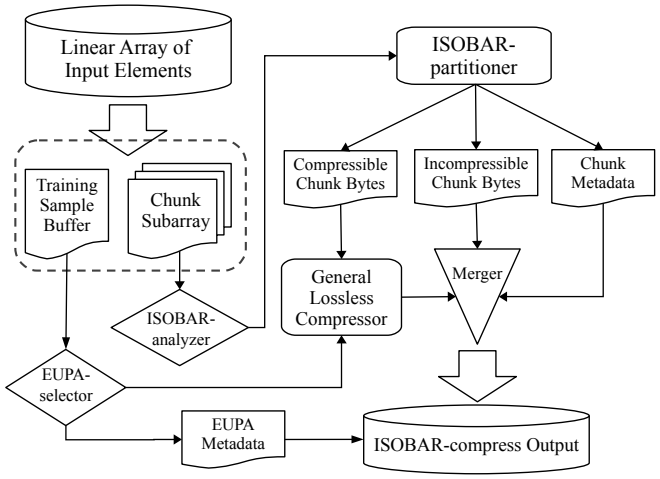

Fig. 2. ISOBAR-compress preconditioner workflow

A. ISOBAR-analyzer

The ISOBAR-analyzer is primarily responsible for analyzing an input dataset consisting of elements of the same type for clusters of bytes that negatively influence compressibility throughout the dataset. This is accomplished by first determining if the input array of elements is a candidate for lossless compression improvement at the *byte-level*. After completing the improvement identification phase, the ISOBAR-analyzer will promote appropriate decisions to other processes in the ISOBAR-compress workflow (see Figure 2).

There are two main reasons that the ISOBAR-analyzer operates at a byte-level: granularity support of general lossless compressors, better entropic complexity [27] identification and computational performance. Firstly, all the general lossless compressors (such as `zlib` and `bzlib2`) operate at the byte-level when doing entropy encoding [1], [2], [31], [6]. Thus, it is important for the ISOBAR-compress workflow to process input data in a manner that promotes pipelining of bytes to a compressor. Secondly, ISOBAR-analyzer can make more accurate and quicker identification decisions in regards to compressibility at the byte-level due to the greater variance of entropy [18] as compared to analysis strictly at the *bit-level*.

When ISOBAR-analyzer receives a dataset of N elements, it starts analyzing each element as an aggregate of ω bytes, where each byte value is in the $[0, 255]$ range. In order to store all N elements of an input dataset, $N \cdot \omega$ bytes are required, for example, each double-precision floating point number needs $\omega = 8$ bytes to store it, storing $N = 1024$ of them will take 8192 bytes on the disk. The values of all required bytes, $Byte_{i,j}$, where $1 \leq i \leq N$ and $1 \leq j \leq \omega$, are represented by the right matrix in Figure 3. Conceptually, the layout for the set of elements can be illustrated as a matrix of bytes, where each row represents an element of ω bytes and each column represents an orthogonal placement of N bytes (one from each element), called a *byte-column*.

$$\begin{pmatrix} Element_1 \\ Element_2 \\ \vdots \\ Element_{N-1} \\ Element_N \end{pmatrix} \Leftrightarrow \begin{pmatrix} Byte_{1,1} & Byte_{1,2} & \cdots & Byte_{1,\omega} \\ Byte_{2,1} & Byte_{2,2} & \cdots & Byte_{2,\omega} \\ \vdots & \vdots & \ddots & \vdots \\ Byte_{N-1,1} & Byte_{N-1,2} & \cdots & Byte_{N-1,\omega} \\ Byte_{N,1} & Byte_{N,2} & \cdots & Byte_{N,\omega} \end{pmatrix}$$

Fig. 3. Element and byte-level representation of an input array

ISOBAR-analyzer will begin its identification phase after receiving an input dataset. The identification phase starts by processing each byte-column separately looking for hard-to-compress properties (illustrated in Figure 4). In order to process all the byte-columns, ω byte-value frequency counters are required. A frequency counter is necessary for generating a frequency distribution of the possible 256 values for a given byte-column. The frequency distribution helps determine the possible reduction of byte sets that negatively influence overall compressibility. If the distribution for each N byte value within a byte-column is less than or equal to $N/256$, then that column-based data set will be considered incompressible for entropy encoding. Byte-columns with incompressible properties negatively affect the compressibility of the original input dataset at the byte-level [27]. Thus, eliminating the incompressible set of bytes represented by a column will improve potential compressibility of the remaining byte-columns as a whole.

After the ISOBAR-analyzer calculates the byte value frequency distribution for each byte-column, the process then selects the columns of bytes for reduction, identifiable as an incompressible byte set. The byte-column selection is based on a frequency distribution tolerance level that is a configurable property of the ISOBAR-analyzer process. If all byte values $[0,\ 255]$ in a byte-column frequency distribution are below the tolerance level, then that byte-column will be selected as incompressible. For the sake of utilizing ISOBAR-compress as a general solution, the frequency distribution tolerance level is defined as $\tau \cdot N/256$ for N elements, where τ is a value between 1 (always incompressible) and 256 (always compressible). We fixed $\tau = 1.42$ based on our experiments and the reason is because when τ varies in the range of $[1.4,\ 1.5]$, the compression ratio's improvement keeps stable for each tested dataset.

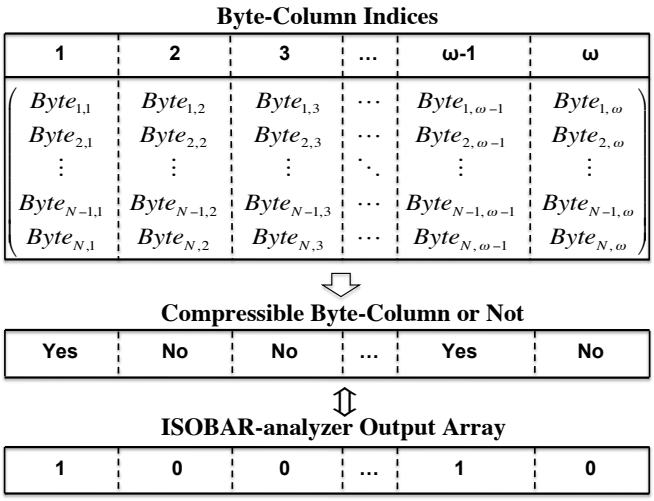

Fig. 4. Byte-column reduction selection example

B. ISOBAR-partitioner

Once the ISOBAR-analyzer identifies all compressible byte-columns for reduction, the next step is to determine whether the input of elements qualifies as a candidate for lossless compression improvements. These improvements are readily available through the ISOBAR-compress workflow's remaining processes. Identification types for compressibility improvement for the entire dataset fall under two categories: improvable, or undetermined. This determination is handled by the ISOBAR-partitioner process. If none or all of the columns are selected for reduction, then the input dataset is considered undetermined, where the entire dataset is then passed to the compressor process. If the dataset is identified as improvable, then the ISOBAR-analysis selected byte-columns are compressed, while the remaining byte-columns are not. Algorithm 1 provides the operational data flow combining the ISOBAR-analysis and ISOBAR-partitioner processes.

Based on the output array of ISOBAR-analyzer, the original dataset is partitioned into two segments: the improvable (compressible) byte-columns, and the hard-to-compress (incompressible) byte-columns (see Figure 5). Following the information passed from the End User's Preference Adaptive Selector (EUPA-selector, see Section C, designed to heuristically choose the linearization strategy and lossless compression technique), the compressible columns will be realigned with the chosen linearization strategy. For example, if we encounter a dataset of 64-bit double-precision floating-point numbers, the EUPA-selector will decide to use `zlib` as the standard compression method (the corresponding row-linearization and metadata provided by the ISOBAR-analyzer is 10000010). The ISOBAR-partitioner will then partition and row-wise linearize the 1^{st} and 7^{th} columns and identify them as the input of the `zlib` standard compression process. In doing so, we only pass the compression algorithm 2 bytes rather than all 8 bytes of the double-precision number as one row in the matrix. This guarantees an increase in compression throughput.

Algorithm 1: ISOBAR-compress

Input:
X—Set of input elements to be compressed
E—End user's preference: throughput or ratio
Output:
X'—Compressed array
Data:
S—ISOBAR-analyzer output array
C—Compressible bytes of input elements
C'—Compressed compressible bytes of input elements
I—Incompressible bytes of input elements
M—Metadata

```
S ← ISOBAR-analyzer(X)
if S = {0, 0, ..., 0} or S = {1, 1, ..., 1} then
    X' ← compressor(X, E)
else
    {C, I, M} ← ISOBAR-partitioner(S, X)
    C' ← compressor(C, E)
    X' ← {C', I, M}
return X'
```

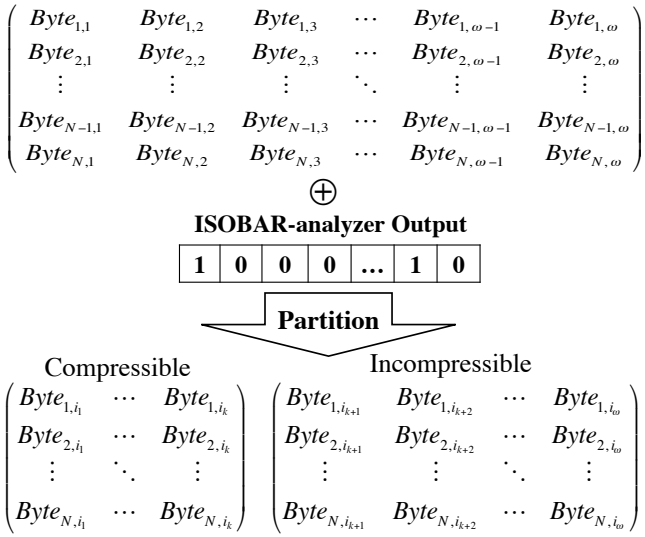

Fig. 5. Example of ISOBAR-partitioner operation

C. EUPA-selector

Since ISOBAR-compress is designed as a preconditioner to improve performance of all *solvers* (lossless compression techniques), the question that arose with this intuition was, "Which type of *performance* is the most desired by the end-user?" While some users may only want to save disk space and hence desire the highest compression ratio rather than compression throughput, most others, including scientists running peta-scale simulation codes (XGC, GTS, etc.), would desire a technique that provides the highest compression throughput with reasonably acceptable (but perhaps not the best) compression ratio. To preserve end-user flexibility in regards to

these choices, we designed the End User's Preference Adaptive Selector (EUPA-selector).

The EUPA-selector is a deterministic process that selects the most suitable lossless compression framework for applying to all the compressible bytes selected by ISOBAR-analysis during the compression process of the workflow. The selector makes a decision based on the evaluation of an input training sample acquired from the input dataset, end-user criterion, and lossless compressor evaluations.

For the purpose of this paper, the ISOBAR-compress workflow will be utilized as a black box solution where the most commonly known compression algorithms (`zlib` and `bzlib2`) are applied by the EUPA-selector. The selector is designed to make a decision on which standard compression method and byte-level linearization will provide the best performance for the end-user's preference. For example, although the EUPA-selector inherently chooses the technique that provides the best compression ratio, the user can instruct the selector to choose a faster method as long as the compression ratio is above a certain, specified threshold. The selector is implemented by first testing each combination of the standard compression method and linearization strategy on sample sets of random elements from the input dataset. Based on the results from the corresponding combinations, the EUPA-selector will make a decision to use either `bzlib2` or `zlib` as the standard compression method and apply either row-based or column-based linearization. Regardless of whether an input dataset to ISOBAR-analysis is identified as improvable or not, the EUPA-selector will choose the optimal standard compression method (`zlib` or `bzlib2`) and linearization strategy for end-users. Based on our experimentations, datasets identified as improvable will have a better compression ratio whether `zlib` or `bzlib2` is used as the standard compression method.

D. Input Array Chunking and Output Merging

Scientific information generated from extreme-scale simulations can easily expand beyond terabytes of archival data, which is impractical for a lossless compressor to handle at a post-processing stage. Compressing an extreme-scale generated dataset as a single input array of elements is problematic due to a single-pass timing cost along with system memory limits. Therefore, it is practical for a lossless compressor to segment an input array into chunks (see Figure 6) of manageable data for pipelining to an *in-situ* process of a workflow.

A minimum chunk size is required by the ISOBAR-compress workflow in order for the ISOBAR-analysis process to function optimally. The reason is because ISOBAR-analysis requires statistical evaluation of all the bytes-columns within an input chunk of elements. This is similar to as why general compressors require minimum block sizes, such as 4 MB used in RCFile [16], and 3 MB demonstrated in the paper [31]. The appropriate dataset chunk size, C, for ISOBAR-compress to efficiently operate is covered in the performance evaluation section of this paper.

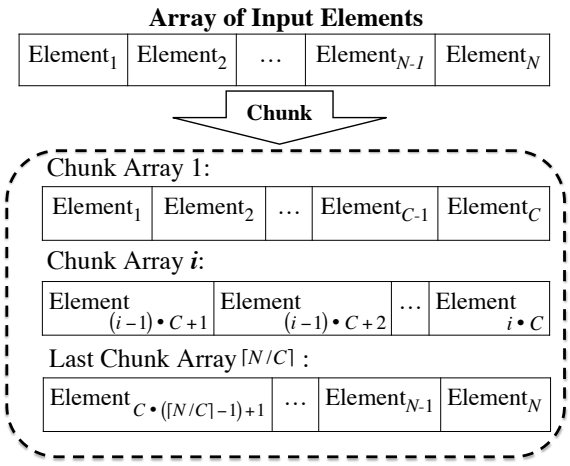

Fig. 6. Chunking a dataset (array of input elements)

At the end of the ISOBAR-compress workflow, an output array of bytes is generated by merging the overall metadata created by EUPA-selector, each chunk's metadata generated by ISOBAR-partitioner, and the incompressible byte-columns along with the compressed byte-columns of each chunk (see Figure 7).

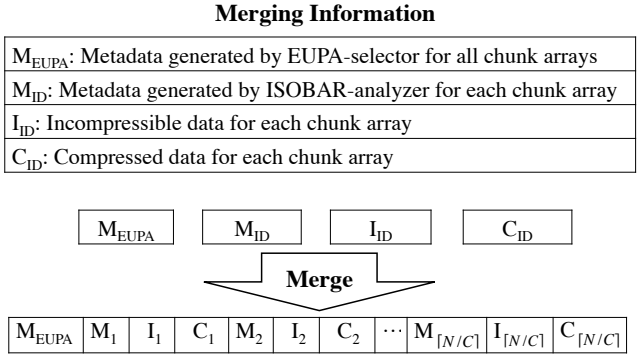

Fig. 7. ISOBAR-compress merged output

III. Performance Evaluation

To demonstrate that ISOBAR-compress has the ability to not only identify hard-to-compress datasets, but also improve the compression efficiency, we apply two major metrics in our tests: Compression Ratio (CR, see Equation 1) and Compression/Decompression Speed-up (Sp, see Equation 2).

All metrics where collected on the Lens Linux cluster at Oak Ridge National Laboratory. The Lens cluster is primarily for data analysis and high-end visualization. Each cluster node consists of four quad-core 2.3 GHz AMD Opteron processessors and 64 GB of memory. All the results for the following experiments were done utilizing only a single core of one processor.

A. Datasets

To calculate these metrics, we tested the ISOBAR-compress preconditioner on 24 scientific datasets of different data types, including single-precision and double-precision floating-point values and 64-bit integers, from various scientific disciplines including combustion, physics, astrophysics, and astronomy.

To briefly introduce those datasets, Table I is provided giving the name of simulation codes generating the datasets (or the capitalized prefix of their name), their data types, applications, variables and referencing materials. Detailed description for each dataset is in Appendix. All tested datasets' statistical characteristics are portrayed in Table III. Most of the datasets consist of unique values (see Equation 4). V is the original vector, V_{Unique} is the array composed of elements from V, with duplicates removed. The datasets also have high degrees of random entropy (see Equation 6); $H(V)$ is the Shannon entropy [28] (see Equation 5) of the input vector, where p_i is the probability of a given element in the vector V, and $Random(|V|)$ is a randomly generated vector consisting of all unique elements with the same size of the original V. The randomness value reflects how close the Shannon entropy of the scientific data is to that of a truly 100% random dataset with the same number of double elements [8].

$$Unique\ Value = \frac{|V_{Unique}|}{|V|} \times 100\% \tag{4}$$

$$H(V) = -\sum_{i=1}^{N} p_i log_2 p_i \tag{5}$$

$$Randomness = \frac{H(V)}{H(Random(|V|))} \times 100\% \tag{6}$$

B. Overall Performance of Identification and Improvement

Table IV and V show that 19 out of 24 (79%) datasets were classified as hard-to-compress, 19 of the 19 (100%) were identified by ISOBAR-compress as improvable, and all datasets showed improvement of both compression ratio and compression/decompression throughput (speed) with ISOBAR-compress.

One such example is the $flash\_velx$ dataset. While using standard compression methods such as `zlib` and `bzlib2` to reduce this data, we achieved compression ratios (CRs) of 1.113 with `zlib` and 1.084 with `bzlib2`. However, when compressing the same data with ISOBAR-compress, the CRs for `zlib` and `bzlib2` were enhanced by 17.52% with 35.89 times compression speed-up and 20.7% with 126.27 times compression speed-up, respectively.

C. Effect of ISOBAR-analyzer

Compression efficiency is sensitive to message length and input chunk size for most lossless compression techniques that adapt based on calculated statistics of the subject data [31]. We did experiments on five datasets and conclude that the proper chunk (or block) size is about $375,000$ elements (doubles), which is about 3 MB (shown in Figure 8), which is consistent with previous conclusions that 3 MB is statistically reasonable [31], [16]. The compression ratios given by ISOBAR-compress start being consistent with the chunking size at this point. With the chunk size guaranteeing consistent compression ratio, the ISOBAR-analyzer identified 19 of 24 datasets as "improvable hard-to-compress" datasets in terms of CR and compression/decompression speed-up by the ISOBAR-compress method. These results are shown in Table V.

TABLE IV
ISOBAR-ANALYZER'S PREDICTIONS

Dataset	HTC†?	HTC Bytes(%)	Improvable?
gts_chkp_zeon	Yes	75%	Yes
gts_chkp_zion	Yes	75%	Yes
gts_phi_l	Yes	75%	Yes
gts_phi_nl	Yes	75%	Yes
xgc_igid	Yes	37.5%	Yes
xgc_iphase	Yes	75%	Yes
s3d_temp	Yes	25%	Yes
s3d_vmag	Yes	50%	Yes
flash_gamc	Yes	62.5%	Yes
flash_velx	Yes	75%	Yes
flash_vely	Yes	75%	Yes
msg_bt	No	0%	No
msg_lu	Yes	75%	Yes
msg_sp	Yes	62.5%	Yes
msg_sppm	No	0%	No
msg_sweep3d	Yes	50%	Yes
num_brain	Yes	75%	Yes
num_comet	Yes	37.5%	Yes
num_control	Yes	75%	Yes
num_plasma	No	0%	No
obs_error	No	0%	No
obs_info	Yes	75%	Yes
obs_spitzer	No	0%	No
obs_temp	Yes	75%	Yes

† Hard-to-compress dataset

By using ISOBAR-compress with `bzlib2`, comparing to standard `bzlib2`, all 19 datasets experienced an improved compression ratio and throughput. The improvement of compression ratio ($\Delta CR(\%)$) was in the range of $[5.24\%,\ 46.6\%]$, while the improvement in speed-up (Sp) was in the range of $[1.319,\ 16.607]$ times faster (see Equation 2).

Using ISOBAR-compress with `zlib`, comparing to standard `zlib`, all 19 datasets showed better compression ratios, and all 19 had gain in throughput (speed). The $\Delta CR(\%)$ was in the range of $[4.7\%,\ 37.0\%]$ and the Sp was in the range of $[1.533,\ 35.89]$.

D. Optimization with EUPA-selector

As shown in Table VI with the speed preference chosen and Table VII with the CR preference chosen, combining ISOBAR-compress with general-purpose lossless compression such as `zlib` and `bzlib2` will improve the compression efficiency yielded by using these standard compressors as standalone utilities. Since the EUPA-selector boasts superb flexibility of use, despite inconsistencies in the relative per-

TABLE III
STATISTICAL INFORMATION ABOUT TEST DATASETS

Dataset	Data Type	Set Size (MB)	Number of Elements (millions)	Unique Value (percent)	Shannon Entropy	Randomness (percent)
gts_phi_l	double	42	5.5	99.9	12.05	99.9
gts_phi_nl	double	42	5.5	99.9	12.05	99.9
gts_chkp_zeon	double	18	2.4	99.9	14.68	99.9
gts_chkp_zion	double	18	2.4	99.9	15.12	99.9
xgc_igid	64-bit integer	146	19.2	22.6	13.81	100.0
xgc_iphase	8 doubles	1170	153.4	7.7	12.32	76.4
s3d_temp	single	77	20.2	45.9	12.21	95.4
s3d_vmag	single	77	20.2	49.9	12.81	99.9
flash_velx	double	520	68.1	100	24.34	100
flash_vely	double	520	68.1	100	25.74	100
flash_gamc	double	520	68.1	100	11.26	100
msg_bt	double	254	33.3	92.9	23.67	94.7
msg_lu	double	185	24.2	99.2	24.47	99.7
msg_sp	double	276	36.2	98.9	25.03	99.7
msg_sppm	double	266	34.8	10.2	11.24	44.9
msg_sweep3d	double	119	15.7	89.8	23.41	97.9
num_brain	double	135	17.7	94.9	23.97	99.5
num_comet	double	102	13.4	88.9	22.04	93.1
num_control	double	152	19.9	98.5	24.14	99.6
num_plasma	double	33	4.4	0.3	13.65	61.9
obs_error	double	59	7.7	18.0	17.80	77.8
obs_info	double	18	2.3	23.9	18.07	85.3
obs_spitzer	double	189	24.7	5.7	17.36	70.7
obs_temp	double	38	4.9	100.0	22.25	100.0

TABLE V
PERFORMANCE COMPARISON

Dataset	zlib		bzlib2		TP_A[3]	ISOBAR-CR Preference		ISOBAR-Sp Preference	
	CR[1]	TP_C (MB/s)[2]	CR	TP_C (MB/s)	(MB/s)	CR	TP_C (MB/s)	CR	TP_C (MB/s)
gts_chkp_zeon	1.040	14.10	1.022	3.55	500.40	**1.182**	**24.35**	**1.140**	**104.99**
gts_chkp_zion	1.044	13.87	1.027	3.57	501.60	**1.187**	**24.60**	**1.150**	**111.66**
gts_phi_l	1.041	14.20	1.020	3.55	501.83	**1.186**	**14.92**	**1.160**	**66.36**
gts_phi_nl	1.045	14.11	1.018	3.57	501.26	**1.180**	**15.41**	**1.157**	**65.65**
xgc_igid	3.003	1.12	3.120	5.99	505.33	**3.368**	**5.34**	**2.962**	**100.91**
xgc_iphase	1.362	6.71	1.377	3.63	501.79	**1.589**	**4.21**	**1.571**	**76.83**
s3d_temp†	1.336	7.25	1.452	3.25	513.15	**2.063**	**8.95**	**1.831**	**53.14**
s3d_vmag†	1.190	11.12	1.210	3.33	516.75	**1.774**	**8.50**	**1.604**	**104.73**
flash_gamc	1.289	19.50	1.281	3.87	503.05	**1.557**	**16.40**	**1.532**	**245.19**
flash_velx	1.113	12.70	1.084	3.61	501.34	**1.319**	**17.30**	**1.308**	**455.83**
flash_vely	1.135	12.01	1.091	3.62	501.57	**1.319**	**60.12**	**1.307**	**444.75**
msg_bt	1.131	13.54	1.102	3.79	495.98	NI*	NI	NI	NI
msg_lu	1.057	14.52	1.021	3.55	499.92	**1.298**	**20.19**	**1.246**	**235.21**
msg_sp	1.112	17.52	1.075	3.66	502.61	**1.330**	**5.16**	**1.304**	**106.65**
msg_sppm	7.436	9.14	6.932	1.35	495.02	NI	NI	NI	NI
msg_sweep3d	1.093	13.46	1.277	3.21	501.72	**1.344**	**6.61**	**1.287**	**78.86**
num_brain	1.064	14.01	1.042	3.13	503.51	**1.276**	**10.08**	**1.238**	**226.51**
num_comet	1.160	13.78	1.172	3.66	496.75	**1.236**	**4.83**	**1.215**	**21.13**
num_control	1.057	14.78	1.029	3.11	501.90	**1.143**	**12.52**	**1.126**	**65.10**
num_plasma	1.608	19.50	5.789	0.48	503.44	NI	NI	NI	NI
obs_error	1.448	8.79	1.338	3.58	502.51	NI	NI	NI	NI
obs_info	1.157	15.42	1.213	3.41	503.47	**1.292**	**5.28**	**1.249**	**228.91**
obs_spitzer	1.228	10.42	1.721	4.14	503.25	NI	NI	NI	NI
obs_temp	1.035	14.70	1.024	3.03	502.95	**1.142**	**22.89**	**1.125**	**96.62**

† Single-precision floating-point dataset
* NI: Not Identified—the dataset is identified as non-improvable by ISOBAR-compress
[1] CR: Compression Ratio (see Equation 1)
[2] TP_C (MB/s): Compression throughput in MB (megabyte) per second
[3] TP_A (MB/s): ISOBAR-analysis throughput in MB per second

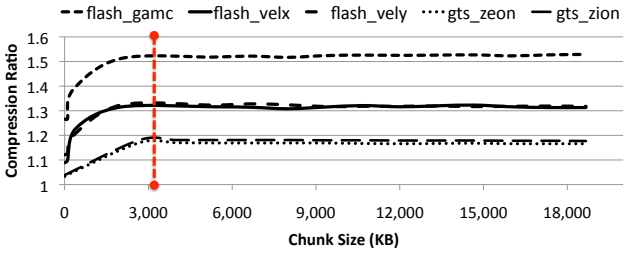

Fig. 8. Chunking size for settled compression ratios

TABLE VI
IMPROVEMENT OF ISOBAR-SP PREFERENCE

Dataset*	LS[1]	ΔCR(%)[2]	Sp[3]
gts_chkp_zeon	Row	9.62	7.447
gts_chkp_zion	Row	10.15	8.050
gts_phi_l	Row	11.43	4.673
gts_phi_nl	Row	10.72	4.653
xgc_iphase	Column	15.35	11.450
flash_gamc	Row	18.85	12.576
flash_velx	Row	17.52	35.899
flash_vely	Row	15.15	37.032
msg_lu	Column	17.88	16.199
msg_sp	Column	17.267	6.087
msg_sweep3d	Column	17.75	5.859
num_brain	Row	16.35	16.168
num_comet	Row	4.74	1.533
num_control	Row	6.53	4.405
obs_info	Row	7.95	14.845
obs_temp	Row	8.70	6.573

[1] LS: Linearization Strategy
[2] ΔCR(%): Percentage improvement of compression ratio (Eq. 3) compared to the alternative with the highest compression throughput
[3] Sp: Compression speed-up (Eq. 2)
* EUPA-selector selected zlib as the better lossless compression technique for all datasets above

formances of `bzlib2` and `zlib`, end-users will be able to get the performance of a (relatively) speedy compression routine with a satisfactory compression ratio. For example, if an end user wishes to compress the msg_lu dataset, prioritizing the compression/decompression throughput (speed) above a higher compression ratio will cause EUPA-selector to choose `zlib` with byte-level column-linearization (the fastest technique, while still offering a compression ratio superior to standard compression). In the worst case scenario, if the dataset to be compressed is not identified as improvable, the EUPA-selector will offer the optimal choice of the standard compression method and linearization scheme. To extend to the user complete flexibility of use, explicit specification of input parameters (fixing the compression method and the linearization strategy) is also permitted by EUPA-selector.

E. Single-Precision Data Compression

Although scientific simulations often produce double-precision floating-point data (8-byte elements), for archival and community sharing purposes, the datasets often get converted to single-precision (4-byte elements) format similar to the "s3d" datasets in our experiments. Moreover, in some research activities, such as hurricane prediction in climate studies, original raw datasets for analysis only consist of single-precision floating-point values.

TABLE VII
IMPROVEMENT OF ISOBAR-CR PREFERENCE

Dataset*	LS[1]	ΔCR(%)[2]	Sp[3]
gts_chkp_zeon	Row	13.65	1.727
gts_chkp_zion	Row	13.69	1.774
gts_phi_l	Row	13.93	1.051
gts_phi_nl	Row	12.92	1.092
xgc_iphase	Column	15.39	1.160
flash_gamc	Row	20.79	0.841
flash_velx	Row	18.51	1.362
flash_vely	Row	16.21	5.006
msg_lu	Column	22.80	1.390
msg_sp	Column	19.60	0.295
msg_sweep3d	Column	5.24	1.410
num_brain	Row	19.92	0.719
num_comet	Row	5.46	1.319
num_control	Row	8.13	0.847
obs_info	Row	6.512	1.548
obs_temp	Row	10.34	1.557

[1] LS: Linearization Strategy
[2] ΔCR(%): Percentage improvement of compression ratio (Eq. 3) compared to the alternative with the best compression ratio
[3] Sp: Compression speed-up (Eq. 2)
* EUPA-selector selected bzlib2 as the better lossless compression technique for all datasets above

To support our claim that ISOBAR-compress can be used for data other than double-precision floating-point values, we tested our methodology on 2 single-precision data sets (see Table VIII). Both data sets were identified as improvable by ISOBAR-compress. As shown in Table VIII, both compression ratio and compression/decompression throughput (speed) were enhanced via ISOBAR-compress. For example, while using ISOBAR-compress with CR (compression ratio) preference compressing for the "s3d" temperature dataset, compared to the compressor with better compression ratio, $\Delta CR = 42.08\%$, and compression speed-up ($Sp = 2.758$). If ISOBAR-Sp is chosen, compared to the better compression throughput compressor, for the same "s3d" dataset, $\Delta CR = 37.05\%$ and $Sp = 7.329$.

TABLE VIII
PERFORMANCE ON SINGLE-PRECISION DATASETS

	Dataset*	LS[3]	ΔCR (%)	Sp
ISOBAR-CR[1]	s3d_temp	Column	42.08	2.758
	s3d_vmag	Row	46.67	2.552
ISOBAR-Sp[2]	s3d_temp	Column	37.05	7.329
	s3d_vmag	Row	34.79	9.418

[1] Performance of ISOBAR-CR preference
[2] Performance of ISOBAR-Sp preference
[3] LS: Linearization Strategy selected by EUPA-selector
* EUPA-selector selected bzlib2 for ISOBAR-CR and zlib for ISOBAR-Sp as the better lossless compression technique

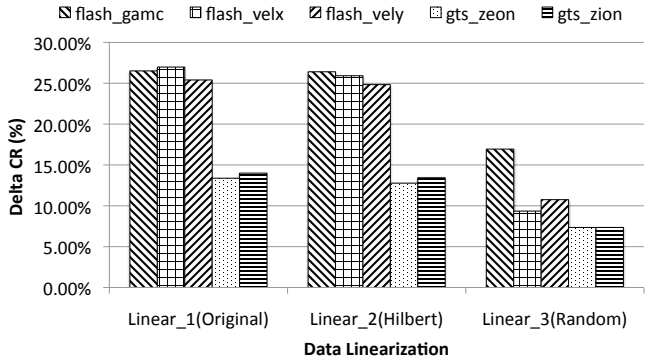

Fig. 9. Compression ratio improvement ($\Delta CR(\%)$, Eq. 3) for multiple datasets with different linearization schemes: original order, Hilbert-linearized order, and random order

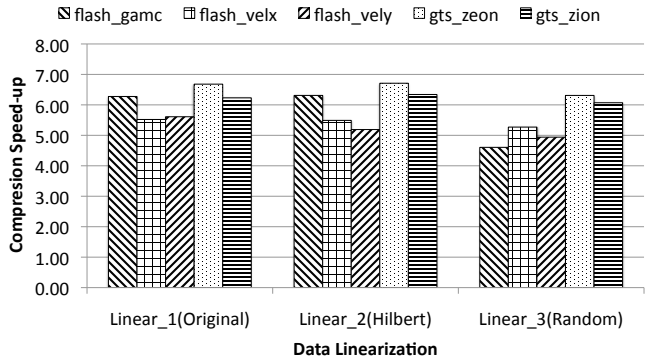

Fig. 10. Compression speed-up (Sp) for multiple datasets with different linearization schemes: original order, Hilbert-linearized order, and random order

F. Consistent Improvement over the Entire Simulation

Typically, scientific simulations generate many intermediate datasets to allow trend analysis and prediction. For example, a single run of the GTS simulation [29] will generate spatial data for approximately $300,000$ time steps. To show that both the EUPA-selector and ISOBAR-analyzer perform consistently well throughput an entire simulation, we tested our technique on GTS simulation data.

Results of the experiments for linear and nonlinear potential fluctuation values show that not only all choices for time steps given by EUPA-selector are the same (using `bzlib2` with row-linearization), but also that all time step datasets are identified as improvable by ISOBAR-compress. For the *linear* regime of potential fluctuation values, the average ΔCR was 14.4% with a standard deviation of 1.8% and the average Sp was 5.952 with a standard deviation of 0.065; for the *nonlinear* regime of potential fluctuation values, the average ΔCR was 13.4% with a standard deviation of 2.7% and the average Sp was 3.749 with a standard deviation of 0.053.

G. Robustness with Different Data Linearization

Data is constantly linearized. When data is generated by simulation codes, it is linearized in memory by the aplication. For example, XGC fusion simulations have multiple means of linearization over the number of variables; particles are mapped onto a mesh and characterized by radial, poloidal, and toroidal dimensions. On disk, data is linearized in various ways, such as Hilbert Space-Filling curves (used in querying multidimensional data) [21], to improve the efficiency of data retrieval.

We argue that by applying ISOBAR-compress as a preconditioner to the general-purpose lossless compression schemes, the performance does not significantly vary based on means of data linearization. As shown in Figure 9, improvement of compression ratios stays almost constant for the data linearized in a specific linearization similar to Hilbert Curve [21] datasets; even in the worst case scenario (where data is totally random), ISOBAR-compress still enhances the compression ratio by approximately 10% as compared to standard compression. Moreover, Figure 10 shows that the compression throughput (speed) is also consistent with various data linearizations.

H. Faster Decompression Throughput

Besides compression ratio and compression throughput (speed), ISOBAR-compress also improves the throughput (speed) of the decompression process. Decompression throughputs of both standard compressor and ISOBAR-compress are listed in Table IX, also the speed-ups of ISOBAR-compress, comparing to the faster one of either standard `bzlib2` or `zlib`, is included. In Table IX, 2 datasets' throughputs increase from about 100 MB/s to more than 1400 MB/s, which means the speed-ups are about 14.0, 15 of 19 had speed-ups greater than 3.0, all the other datasets' decompression throughputs are improved. Test runs on permuted data sets yield approximately the same improvement as the run on the original, non-permuted data.

IV. Related Work

While the idea of preconditioning data for standard compression routines is relatively unexplored (as far as we know), there are many instances in which lossless compression is required in the information storage and retrieval community. For this reason, there is a wealth of related work on the development of other lossless reduction tools. In this section, we acknowledge and address the relevance and comparative performance (in terms of compression ratio and throughput) of these standalone utilities to those obtained via the ISOBAR-compress workflow.

One such tool is PFOR [33], which aims to reduce the I/O bottleneck in the compression routine by extracting maximum instructions per cycle from modern CPUs. PFOR [33] (and some similar variants including PDICT [33] and PFOR-DELTA [33]) are specifically designed with the aim of eliminating *if-then-else* constructs and value dependencies in prediction and encoding of data. This strategy makes the data being compressed or decompressed fully loop-pipelineable by

TABLE X
COMPARISON AMONG ISOBAR-COMPRESS, FPC [8] AND FPZIP [22].

Data Set	ISOBAR-Sp			FPC			fpzip		
	CR[1]	TP_C[2]	TP_D[3]	CR	TP_C	TP_D	CR	TP_C	TP_D
gts_chkp_zeon	1.140	104.99	517.89	1.018	38.22	39.24	1.096	35.80	29.48
gts_chkp_zion	1.150	111.66	551.90	1.025	38.42	38.98	1.100	36.47	30.20
gts_phi_l	1.160	66.36	366.25	1.077	24.06	23.93	1.182	38.66	31.44
gts_phi_nl	1.157	65.65	358.21	1.072	24.10	24.06	1.177	38.38	31.23
xgc_igid	2.962	100.91	341.50	1.960	87.84	87.85	2.736	13.63	12.84
xgc_iphase	1.571	76.83	388.87	1.360	17.66	17.43	1.535	44.89	36.12
flash_gamc	1.532	245.19	940.91	1.416	91.64	90.06	1.620	38.70	31.72
flash_velx	1.308	455.83	1617.02	1.265	49.61	49.70	1.342	36.77	31.39
flash_vely	1.307	444.75	1538.98	1.294	53.80	53.37	1.435	38.47	32.11
mean	1.476	185.80	735.73	1.276	47.26	47.18	1.469	35.75	29.61

[1] CR: Compression Ratio (see Equation 1)
[2] TP_C: Compression throughput tested on Lens system (MB per second)
[3] TP_D: Decompression throughput tested on Lens system (MB per second)

TABLE IX
DECOMPRESSION THROUGHPUT COMPARISON

Dataset	zlib (MB/s)	bzlib2 (MB/s)	ISOBAR[1] (MB/s)	Sp[2]
gts_chkp_zeon	115.22	10.48	**517.89**	4.5
gts_chkp_zion	110.38	10.57	**551.90**	5.0
gts_phi_l	114.41	10.00	**366.25**	3.2
gts_phi_nl	117.97	9.90	**358.21**	3.0
xgc_igid	177.69	21.08	**341.50**	1.9
xgc_iphase	138.99	7.49	**388.87**	2.8
s3d_temp†	113.80	6.26	**250.46**	2.2
s3d_vmag†	103.69	6.73	**424.79**	4.1
flash_velx	113.95	10.51	**1617.02**	14.2
flash_vely	112.03	10.53	**1538.98**	13.7
flash_gamc	113.41	12.02	**940.91**	8.3
msg_lu	112.51	10.51	**866.21**	7.7
msg_sp	106.77	10.68	**527.18**	4.9
msg_sweep3d	114.43	6.89	**446.49**	3.9
num_brain	114.47	6.55	**908.65**	7.9
num_comet	123.08	7.69	**145.73**	1.2
num_control	122.13	7.28	**373.63**	3.1
obs_info	118.61	7.27	**910.12**	7.7
obs_temp	114.10	6.59	**511.98**	4.5

† Single-precision floating-point dataset
[1] ISOBAR: Decompression throughput using ISOBAR-compress with speed preference
[2] Sp: Decompression speed-up comparing ISOBAR throughput to the faster alternative of either bzlib2 or zlib

modern compilers and allows for out-of-order execution on modern CPUs while achieving high Instructions Per Cycle efficiency. Based on the experimental results provided in the paper, PFOR performs approximately 4 times faster than `zlib` and `bzlib2` for most data sets tested, though its compression ratios hardly beat those obtained with `zlib` and `bzlib2` (in some cases, the ratio is even 3 times worse). ISOBAR-compress was designed to improve both compression efficiency and throughput for general purpose lossless compression techniques, such as `zlib` and `bzlib2`. In the results section, we showed that ISOBAR-compress's performance almost universally surpassed the performance of these compressors in terms of compression ratio and compression/decompression throughput.

General purpose lossless compression tools are also used with MapReduce techniques. For example, RCFile [16] is another reduction utility focusing on efficient storage of data produced from Hadoop MapReduce [3], [12] based applications. RCFile compresses data tables sequentially column-wise, but uses no intelligent technique to determine how the input data should be optimally compressed, defaulting to `zlib` for all data columns. For this reason, compression yielded with RCFile varies in effectiveness, because it does not consider datatype of columns and blindly relies on `zlib` compression and a particular linearization, whereas other combinations of these two factors may be more fruitful in compression ratio and throughput. ISOBAR-compress's EUPA-selector takes care of this problem, and the workflow involves applying general-purpose lossless compression techniques, particularly after the ISOBAR-analyzer phase because of the varying sizes of values in the data columns (for example, a column storing numerical data may occupy 3 bytes per element in a MySQL database, whereas some string type columns may require approximately 100 bytes per element).

Tools, such as `fpzip` [22] and `FPC` [8] are also widely used in scientific database compression applications. Although `fpzip` and `FPC` are tools designed only for compression of 32 (`fpzip`) or 64 (`fpzip` and `FPC`) bit floating point data , their performance is also competitive (when compared to other lossless compression techniques) on other types of variously-sized scientific data. Hence, it is worthwhile introducing and comparing these utilities with ISOBAR-compress. Both `fpzip` and `FPC` are based on context modeling and prediction. `Fpzip` traverses data in a coherent order and then uses the corresponding n-dimensional (where n is the dimensionality of the data) Lorenzo predictor [17] to predict the subsequent values. It next maps the predicted values and actual values to their integer representations, and encodes the XORd́ residual between these values. Similarly, `FPC` first predicts values sequentially using two predictors (`fcm` [32]

and dfcm [15]), and subsequently selects the closer predicted value to the actual. Lastly, FPC XORs the selected predicted value with the actual value, and compresses the leading-zero result. We compared the performance of ISOBAR-compress, fpzip, and FPC on identified scientific datasets in Table X.

It is briefly worth addressing that there has also been a wealth of research done on the lossy compression front. Methods using DCT (Discrete Cosine Transform) [30] and wavelets [10] have been actively researched over the last few decades in the lossy compression realm. These works have primarily been applied in the context of visualization and geometric modeling applications. Lakshminarasimhan *et al.* explored the use of B-spline modeling to exploit monotonic properties in sorted data and reduce size by as much as 85% of the original [20]. Multi-dimensional histogram binning has also been used in the lossy reduction of simulation data. However, these techniques are not applicable in the reduction of certain types of simulation data and checkpoint/restart data as inaccuracies in these values can completely throw off a simulation run and yield incorrect results.

V. Conclusion

In this paper, we proposed a new technique called ISOBAR-compress, which is a *preconditioner* to identify hard-to-compress datasets and improve compression efficiency for all general-purpose lossless compression *solvers*.

Given a dataset requiring lossless compression, our system first applies the ISOBAR-analyzer to determine whether ISOBAR-compress will improve the performance of lossless compression algorithms such as zlib. If performance can be improved, then the ISOBAR-partitioner will segment the dataset into two pieces: compressible and incompressible data. EUPA-selector will choose the optimal combination of standard lossless compression methods and proper multi-dimensional data linearization to meet the end-user's preference in terms of better compression ratio vs. much higher compression speed. Finally, ISOBAR-compress will merge the metadata, compressed data, and incompressible data into an output file. When tested on 24 scientific datasets, ISOBAR demonstrated high compression and decompression throughput as well as an improved compression ratio upon standard compressors.

Acknowledgment

We would like to acknowledge the use of resources at ORNL's and ANL's leadership class computing facilities, OLCF and ALCF, respectively. Also, we appreciate the use of the datasets available from the Flash Center for Computational Science. This work was supported in part by the U.S. Department of Energy, Office of Science (SciDAC SDM Center, DE-AC02-06CH11357, DE-SC0004935, DE-FC02-10ER26002, DE-FOA-0000256, DE-FOA-0000257) and the U.S. National Science Foundation (Expeditions in Computing). Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725.

References

[1] Bzip2, 2010. http://www.bzip.org/.
[2] Zlib, 2010. http://www.zlib.net/.
[3] Hadoop MapReduce, 2011. http://hadoop.apache.org/mapreduce/.
[4] B. Fryxell and K. Olson and P. Ricker and F. X. Timmes and M. Zingale and D. Q. Lamb and P. MacNeice and R. Rosner and J. W. Truran and H. Tufo. FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131:273–334, November 2000.
[5] M. Benzi. Preconditioning techniques for large linear systems: A survey. *J. COMPUT. PHYS*, 182:418–477, 2002.
[6] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. *HP Labs Technical Reports*, 1994.
[7] M. Burtscher and P. Ratanaworabhan. High throughput compression of double-precision floating-point data. In *IEEE Data Compression Conference*, pages 293–302, 2007.
[8] M. Burtscher and P. Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58:18–31, 2009.
[9] M. Burtscher and I. Szczyrba. Numerical modeling of brain dynamics in traumatic situations - Impulsive Translations. In *Mathematics and Engineering Techniques in Medicine and Biological Scienes*, pages 205–211, 2005.
[10] C. Chang and B. Girod. Direction-adaptive discrete wavelet transform for image compression. *Image Processing, IEEE Transactions on*, 16(5):1289 –1302, may 2007.
[11] J. H. Chen, A. Choudhary, B. De. Supinski, M. DeVries, S. Klasky E. R. Hawkes, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, S. Shende R. Sankaran, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science and Discovery*, 2(1):015001, 2009.
[12] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.
[13] R. D. Falgout. An introduction to Algebraic Multigrid. *Computing in Science and Engg.*, 8:24–33, November 2006.
[14] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–339, 1991. 10.1007/BF01385726.
[15] B. Goeman, H. Vandierendonck, and K. D. Bosschere. Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In *Seventh International Symposium on High Performance Computer Architecture*, pages 207–216, 2001.
[16] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1199–1208, April 2011.
[17] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22:343–348, 2003.
[18] K. Konstantinides and K. B. Natarajan. An architecture for non-linear noise filtering via piecewise linear compression. *HP Labs Technical Reports*, 1994.
[19] S. Ku, C.S. Chang, and P.H. Diamond. Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic Tokamak geometry. *Nuclear Fusion*, 49(11):115021, 2009.
[20] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Euro-Par*, 2011.
[21] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert Space-Filling Curve. *SIGMOD Record*, 30:2001, 2001.
[22] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12:1245–1250, 2006.
[23] B. K. Natarajan. Filtering random noise via data compression. *Data Compression Conference*, pages 60 –69, 1993.
[24] B. K. Natarajan. Occam's razor for functions. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 370–376, New York, NY, USA, 1993. ACM.
[25] W. D. Pence, R. Seaman, and R. L. White. Lossless astronomical image compression and the effects of noise. *Publications of the Astronomical Society of the Pacific*, 121:414–427, Apr 2009.

[26] J. M. Prusa, P. K. Smolarkiewicz, and A. A. Wyszogrodzki. Simulations of gravity wave induced turbulence using 512 PE CRAY T3E. In *International Journal of Applied Mathematics and Computational Science*, pages 101–115, 2001.
[27] Y. Sehoon and W. A. Pearlman. Critical encoding rate in combined denoising and compression. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III – 341–4, Sept. 2005.
[28] C. E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.
[29] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam. Gyrokinetic simulation of global turbulent transport properties in Tokamak experiments. *Physics of Plasmas*, 13(9):092505, 2006.
[30] A. B. Watson. Image compression using the discrete cosine transform. *Mathematica Journal*, 4:81–88, 1994.
[31] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.
[32] S. Yiannakis and J. E. Smith. The predictability of data values. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 30, pages 248–258, Washington, DC, USA, 1997. IEEE Computer Society.
[33] M. Zukowski, S. Hman, N. Nes, P. A. Boncz, M. Zukowski, S. Hman, N. Nes, and P. Boncz. Super-scalar RAM-CPU cache compression. In *Proceedings of the International Conference of Data Engineering (IEEE ICDE)*, page 59. IEEE Computer Society, 2006.

APPENDIX

Datasets with the prefix "gts" and "xgc" in name are generated from the scientific applications: Gyrokinetic Tokamak Simulation (GTS) [29] and full-function X-point included Gyrokinetic Code (XGC) [19].

1) *gts_phi_l*: linear potential fluctuation variable values of particle-based simulations of fusion plasmas to study plasma micro-turbulence in reactor core and edge.
2) *gts_phi_l*: nonlinear potential fluctuation variable values of the same simulations of fusion plasmas.
3) *gts_chkp_zeon*: values for zeon variable's checkpoint restart data for each 10th time-step of GTS simulation.
4) *gts_chkp_zion*: values for zion variable's checkpoint restart data for each 10th time-step of GTS simulation.
5) *xgc_igid*: ID number of each particle on the fusion plasma edge during XGC simulation.
6) *xgc_iphase*: indicates 8 phase variables of each ion during XGC simulation.

In the application of velocity in the field of astrophysics, there are three datasets for 3 variables respectively generated by code development at the Flash Center: *flash_velx, flash_vely* and *flash_gamc* [4]. Here is the brief illustration of the three datasets.

1) *flash_velx*: fluid velocity x variable values for FLASH.
2) *flash_vely*: fluid velocity y variable values for FLASH.
3) *flash_gamc*: fluid velocity $gamc$ variable values for FLASH.

Two single floating-point datasets *s3d_temp* and *s3d_vmag* are generated by the three-dimensional solver application (S3D) [11] used for direct numerical simulations of turbulent combustion.

1) *s3d_temp*: temperature values of S3D simulation.
2) *s3d_vmag*: magnitude of vectors sensed by the toroidal devices.

These dataset names starting with "obs" [8] and "num" [7] comprise measurements from scientific observational instruments and numeric simulations:

1) *obs_error*: data values specifying brightness temperature errors of a weather satellite.
2) *obs_info*: latitude and longitude information of the observation points of a weather satellite.
3) *obs_spitzer*: data from the Spitzer Space Telescope showing a slight darkening as an extra-solar planet disappears behinds its star.
4) *obs_temp*: data from a weather satellite denoting how much the observed temperature differs from the actual contiguous analysis temperature field.
5) *num_brain*: simulation of the velocity field of a human brain during a head impact.
6) *num_comet*: simulation of the comet Shoemaker-Levy 9 entering Jupiter atmosphere.
7) *num_control*: control vector output between two minimization steps in weather-satellite data assimilation.
8) *num_plasma*: simulated plasma temperature evolution of a wire array z-pinch.

Parallel messages datasets have the prefix "msg" [8]. These 5 datasets contain the numeric messages sent by a node in a parallel system running NAS Parallel Benchmark (NPB) and ASCI Purple applications:

1) *msg_bt*: NPB computational fluid dynamics pseudo-application `bt`.
2) *msg_lu*: NPB computational fluid dynamics pseudo-application `lu`.
3) *msg_sp*: NPB computational fluid dynamics pseudo-application `sp`.
4) *msg_sppm*: ASCI Purple solver `sppm`.
5) *msg_sweep3d*: ASCI Purple solver `sweep3d`.