Improving I/O Throughput with PRIMACY: Preconditioning ID-Mapper for Compressing Incompressibility

Neil Shah*†, Eric R. Schendel*†, Sriram Lakshminarasimhan*†, Saurabh V. Pendse*†,
Terry Rogers* and Nagiza F. Samatova*†
*Department of Computer Science
North Carolina State University, Raleigh, North Carolina 27695–8206
Email: {nashah3,erschend,slakshm2,svpendse,tdroger2}@ncsu.edu, samatova@csc.ncsu.edu
†Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830–8001

***Abstract*—The ability to efficiently handle massive amounts of data is necessary for the continuing development towards exascale scientific data-mining applications and database systems. Unfortunately, recent years have shown a growing gap between the size and complexity of data produced from scientific applications and the limited I/O bandwidth available on modern high-performance computing systems. Utilizing data compression in order to lower the degree of I/O activity offers a promising means to addressing this problem. However, the standard compression algorithms previously explored for such use offer limited gains on both the end-to-end throughput and storage fronts. In this paper, we introduce an in-situ compression scheme aimed at improving end-to-end I/O throughput as well as reduction of dataset size. Our technique, PRIMACY (PReconditioning Id-MApper for Compressing incompressibilitY), acts as a *preconditioner* for standard compression libraries by modifying representation of original floating-point scientific data to increase byte-level repeatability, allowing standard lossless compressors to take advantage of their entropy-based byte-level encoding schemes. We additionally present a theoretical model for compression efficiency in high-performance computing environments and evaluate the efficiency of our approach via comparative analysis. Based on our evaluations on 20 real-world scientific datasets, PRIMACY achieved up to 38% and 22% improvements upon standard end-to-end write and read throughputs respectively in addition to a 25% increase in compression ratios paired with 3-to-4-fold improvement in both compression and decompression throughput over general purpose compressors.**

***Keywords*-Lossless Compression; Performance Modeling; I/O**

I. Introduction

The recent years have made us aware of the ever-growing gap between the complexity and size of the data produced in scientific applications and the available I/O bandwidth. This gap has lead to a crucial bottleneck in read and write performance for these applications. This issue is further worsened when one considers the increase in frequency of checkpoint writes due to higher potential of node failure at such a scale.

One promising approach for improving I/O rates involves taking advantage of data compression techniques to reduce the amount of data transferred during read and write processes. Previous research in this area has shown the strong potential for data compression techniques to increase effective network bandwidth [22]. However, this study has assumed that the actual compression and decompression of data would be costless given a sufficiently large number of nodes during compression and a dedicated node for decompression. In reality, the improvement of network bandwidth via compression must inevitably come at the non-zero cost of compressing the data before transferring it over the network. This cost cannot be trivialized both due to the inherently large sizes of data that are written as well as the transmission of variable length segments from compute nodes. Unfortunately, the compression throughputs and ratios that standard reduction methods such as `zlib`, `bzlib2` and `lzo` offer actually offset most of the end-to-end gain yielded by improved effective network bandwidth, thus hardly increasing (and in the worst cases even *reducing*) the end-to-end I/O throughput from that of uncompressed reads and writes. The added time cost for compression renders standard reduction algorithms unsuitable for many crucial scientific applications, especially those using simulation checkpoint & restart data. These applications produce massive amounts of data and could benefit significantly from *in-situ* high-performance lossless compression (operating on data during the simulation, while data still resides in memory either in the application or user space).

Furthermore, the complexity of data produced by scientific simulations challenges these general purpose compression frameworks, restricting their capacity to effectively reduce data. Specifically, these *lossless* compressors offer at best 20% reduction on many of the tested scientific double-precision floating-point datasets—these datasets are dubbed *hard-to-compress* due to the high degree of entropy exhibited by the data (the degree of entropy is typically inversely correlated with the extent of gains in data reduction that standard compressors are able to provide).

The ideal solution to this problem of limited I/O rates from the data compression perspective is the use of a high-

performance lossless compressor that offers both fast and efficient data reduction. Unfortunately, reduction techniques in the past have offered either high compression throughput or high compression ratio, as the two have traditionally been in conflict (Equations (1) and (2) give the basic formulas for these metrics). Libraries which deliver relatively high compression and decompression throughputs tend to yield low compression ratios, whereas those that provide high compression ratios tend to be slow.

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

$$Throughput = \frac{Original\ Data\ Size}{Runtime} \quad (2)$$

However, we argue that these goals can be mutually satisfied. Our chief insight is that by modifying the representation of raw, hard-to-compress data to emulate more highly-compressible data, we can achieve gains in both throughput and compression ratio. These improvements can allow us to effectively hide the cost of compression in the I/O pipeline while reducing dataset size by an even greater amount. Furthermore, if our modified data compresses well enough at a high throughput, we can not only avoid reducing the end-to-end I/O throughput due to compression, but rather *increase* it while both augmenting effective network bandwidth and disk storage capacity.

With this motivation, we introduce a preconditioning technique that enables faster and more efficient lossless reduction of scientific floating-point data via the use of standard compression algorithms. Our strategy is intuitively supported by the frequent use of *preconditioners* for improving the convergence of iterative *solvers* in the linear algebra domain (such as Quasi-Minimal Residual [7], Algebraic Multigrid [5], LDL [2]). Preconditioning processes such as QR-factorization and LU-decomposition are widely used by scientific domains to improve the rate of convergence for iterative solvers. To the extent of our knowledge, the use of such preconditioning techniques that optimize input for solvers is relatively unexplored in the context of lossless data compression.

The preconditioner we introduce is called PRIMACY (PReconditioning Id-MApper for Compressing incompressibilitY). PRIMACY enables fast analysis of data and uses the information it finds in order to create an arrangement of data that improves compressibility of input for use with standard entropy-based lossless compression frameworks. Our preconditioner creates a one-to-one mapping between byte-sequences in the original data and a frequency based permutation of identification values (IDs). Specifically, the ID assignment is based on a probabilistic analysis of which patterns (byte-sequences) appear with the most and least frequency in the data. By representing frequently occurring byte-sequences with strategically assigned identification values, we are able to increase the repeatability in data on an individual byte basis, lending well to the abilities of standard byte-level lossless compressors (the "solver" segment of the data compression pipeline). Because standard compressors use entropy encoding techniques, they take utmost advantage of the MDL (minimum description length) principle [9], which states that any *regularity* in a given set of data can be used to compress the data. By modifying the organization of data to increase repeatability, we can achieve improvements in both compression throughput and ratio, and thus improve rates for the end-to-end I/O pipeline.

We tested our method on a total of 20 scientific double-precision floating-point datasets spanning various application domains [3], [4], [8], [16], [20]. When run on the Jaguar XK6 cluster, PRIMACY achieved end-to-end throughput improvements of up to 38% and 22% over standard writes and reads respectively. Furthermore, PRIMACY obtained a better compression ratio on 19 of 20 (95%) datasets when compared to standalone `zlib` compression, one of the most commonly used scientific compression libraries due to it's balance between the very fast but poor compression of `lzo` and the very slow but substantial compression of `bzlib2`. The improvement on compression ratio reached up to 25% in some cases, and averaged at approximately 13% over `zlib`. Throughput improvements were on average $3-4$ times in regards to both compression and decompression.

Lastly, we realize that though we show performance gains when using our compression approach on a leadership-class HPC system, we cannot hope to evaluate the extent of gains on all possible cluster configurations. For this reason, we provide an analytical performance model that can enable prediction of I/O performance on target systems both with and without applied compression and additionally help application developers in choosing particular configurations.

II. Methodology

A. Overview

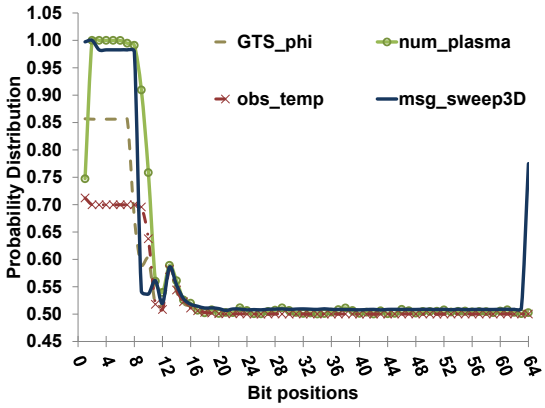


Figure 1. Probability (p) of occurrence of the most frequent bit value (either 0 or 1) at each bit position on 4 representative data sets

Due to the high level of precision required by many scientific applications, floating-point data-types are well-suited for storing data values produced from simulations. However, the established IEEE representation of floating-point numbers inherently gives rise to concerns of overall compressibility of scientific data. In this paper, we showcase examples using 64-bit double-precision data, though the analyses drawn from these examples can be generalized to floating-point data of other precisions as well. The *exponent* portion of each double (within first 2 bytes) stores an integer value, while the *mantissa* portion (approximately 6 bytes) stores the fractional content of the value. While the exponent portion maintains some degree of compressibility due to common bytes in integer parts of numbers, the mantissa is considered to be poorly compressible due to the inherent randomness in fractional parts, machine approximations, and calculation roundoffs. A visual depiction of this phenomenon is given in Figure 1. While the first 2 bytes show regularity in bit value ($p > 0.5$) and imply compressibility, the last 6 bytes show a high degree of randomness ($p \approx 0.5$) and imply poor compressibility.

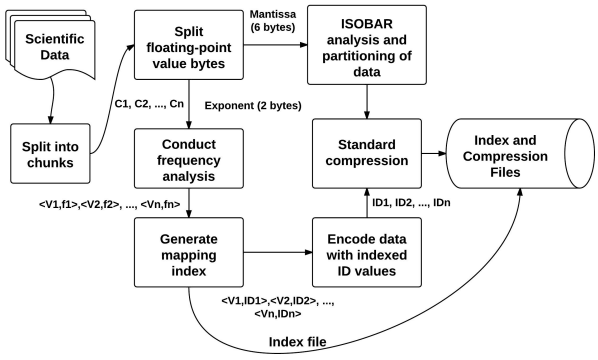


Figure 2. PRIMACY preconditioner workflow

This lack of predictability and the presence of randomness can be considered as *noise* in the data. By isolating this noise from the more repeatable and less-unique portion of each double, we are able to separate the "signal" and "noise" portions of the data to a much greater extent than before [14]. Since the degree of randomness in the mantissa portion of data is generally much higher than in the exponent, it is reasonable to assume that the performance of standard entropy-based encoding frameworks varies between the two. In fact, we find that when compressing most datasets, compression ratio and throughput over the first 2 byte-columns of the data (exponent portion) is often much higher than that of all 8 byte-columns (exponent and mantissa). We propose a method that in many cases improves the overall compression performance of data over current state-of-the-art lossless compression libraries by approaching the problem from this perspective.

In regards to the first two bytes containing the exponent information, we propose a novel method of representation that increases the apparent repeatability in byte values. This allows standard compressors such as `zlib` and `bzlib2` to take full advantage of their byte-level entropy-coding schemes [21]. In general, these types of encoders benefit from application of the MDL principle; thus, any regularity in data can be used to compress the data with fewer bits than needed to express it literally.

Our method revolves around creating a one-to-one mapping between the set of unique byte-sequences (combination of the 2 exponent bytes) found in a set of data, and a specific, statistically determined permutation of identification values. The nuances in the construction of this mapping result in a representation of data that is often more repeatable, and thus more compressible, than the original data. The other six mantissa bytes per double are passed to the `ISOBAR` analysis utility, which uses sampling and statistical analysis to calculate and assess the degree of compressibility of certain bytes in order to optimize compression efficiency of the data [17]. Figure 2 provides an overview of the key steps of the PRIMACY preconditioner. While the above framework can be integrated into any stage of the I/O pipeline, coupling its integration with the data generation phase can result in reduced data movement through multiple stages; for this reason, we explore the use of PRIMACY only on compute nodes in this paper.

B. Data Preprocessing

The data is first split into 3MB sized chunks. Chunking support allows effective in-situ processing and is a crucial step in our preconditioner technique. With chunking, we are able to compress data in a low-memory, high-throughput manner. The 3MB chunk size was chosen after initial experiments confirmed previous studies showing that compressor efficiency begins leveling off at this level [10], [21].

Next, for a chunk of N elements, we process an $N \times 8$ matrix of bytes (8 bytes per double), and split the matrix into two matrices of $N \times 2$ and $N \times 6$ bytes. The $N \times 2$ matrix contains the exponent and first few mantissa bits of each of the N elements, whereas the $N \times 6$ matrix contains the remaining mantissa bits.

These exponent and mantissa byte-matrices are written per chunk to two different files. The file of exponent bytes is passed along to the remainder of the frequency-based encoding pipeline. The file of mantissa bytes shows hard-to-compress characteristics, and is thus passed along to our `ISOBAR` analyzer and partitioner (see Sect. II-G) to improve compressor efficiency in terms of compression and decompression speeds and compression ratio. Algorithm 1 shows the high-level procedural workflow of preprocessing, encoding and compression phases of the PRIMACY preconditioner pipeline.

Algorithm 1 PRIMACY Compression

Input:
X—Set of input elements to be compressed
N—Number of elements in $\{X\}$
C—Number of elements in a chunk

Code:
$to\_process \leftarrow N$
$id \leftarrow 0$
$high \leftarrow 2$ # 2 high order bytes per element for frequency encoding
$low \leftarrow 6$ # 6 low order bytes per element for ISOBAR encoding

while $to\_process > 0$ **do**
  $start \leftarrow id \times C + 1$
  $end \leftarrow min((id + 1) \times C, N)$
  $chunk \leftarrow$ PREP-CHUNK-ARRAY$(X, start, end)$
  $freq \leftarrow$ FREQUENCY-ANALYSIS$(chunk, high)$
  $I \leftarrow$ GENERATE-INDEX$(freq, high)$
  $M \leftarrow$ PREP-HIGH-ORDER-BYTES$(chunk, I, high)$
  $L \leftarrow$ PREP-LOW-ORDER-BYTES$(chunk, low)$
  $M' \leftarrow$ ID-COMPRESS(M, I)
  $L' \leftarrow$ ISOBAR-COMPRESS(L)
  WRITE $\{I\}$
  WRITE $\{M'\}$
  WRITE $\{L'\}$
  $to\_process \leftarrow to\_process - C$
  $id \leftarrow id + 1$
end while

C. Encoding of Byte-Sequences

For a dataset of n doubles, n byte-sequences will be encountered. However, of these n instances, only a fraction will be unique over the entirety of the dataset. This is because almost all scientific data inherently has *locality*. The exponent bytes of data are generally found to be within a range of values. In analyzing our datasets, we found that this range was rather small—in fact, the majority of our data had less than $2,000$ unique byte-sequences from the possible $65,536$. This frequency distribution, skewed towards a limited number of values, implies that there is a relatively high degree of compressibility in exponent bytes from scientific data.

In order to seek improvement in compression of these byte-sequences, it becomes necessary to look at how the sequences are interpreted by standard compressors. The common standard compressors (`zlib`, `bzlib2`, and `lzo`) are 8-bit encoders and look at patterns in data on a *byte*-level [21]. Thus, while high repeatability in byte-sequences is important to the overall compressibility of data, it is still of secondary importance to high repeatability in individual *bytes*. Transforming data to emulate a higher degree of byte-level repeatability improves the compressibility of the data.

This notion causes us to pose the following question: how can we create a bijective mapping between byte-sequences from original data and new byte-sequences to be used in transformed data?

We reasoned that since the mapping must be bijective (unique sequences from the original data must map to unique sequences from transformed data), it is necessary to examine how frequently certain byte-sequences appear when creating a mapping to transformed byte-sequences (identification values, or IDs). This way, we can aim to take advantage of the most repeated byte-sequences in the original dataset when creating a mapping.

Thus, our first step in creating a mapping is to examine each chunk of data and collect information on how frequently different byte-sequences occur within that chunk. We then assign the most frequent byte-sequences an identification value of 0, the next most frequent byte-sequences an identification value of 1 and so on until all unique sequences found in the original data are accounted for.

By traversing *ascending* byte-sequences sorted by *descending* frequency in assigning identification values, we represent the most repeated patterns in the original data with the lowest ID values. On a byte-level, this translates to having the highest possible frequency of 0-bytes in our data. The first identification value (representing the most frequently occurring byte-sequence in the original data) will be represented as a byte-sequence of two 0-bytes, while the next 255 most frequent byte-sequences from original data will be represented as byte-sequences of a single 0-byte paired with a nonzero byte. This mapping on average increased the repeatability of the most frequently occuring data byte by approximately 15% over the 20 datasets used.

D. Byte-level Data Linearization

The previous steps in our process improve compression of data based on increased repeatability and application of the MDL principle for entropy-based coders. However, entropy-based coders often also achieve large compression gains from *run-length encoding*, or intelligent representation of sequentially repeating runs (consecutive occurrences) of the same byte values with fewer bytes.

To further improve the gain in compression of our transformed data, we vary the byte-level linearization of subsequent ID values by compressing the transformed $N \times 8$ byte-matrix column-by-column rather than row-by-row. The driving motivation behind this choice is that because lower ID values represent more frequent byte-sequences, the chance of having runs of the 0-byte in the exponent byte-columns is high. In essence, column linearization equates to passing our compressor the transpose of the transformed byte-matrix instead of the original. This allows us to take advantage of the run-length encoding features of standard compressors and in many cases substantially improve compression throughput and ratio over that of row-linearized data.

E. Standard Compression of Encoded Data

The new representation of the original exponent bytes generally has a higher degree of repeatability than the original data due to the probabilistically modeled permutation of identification values. The final step in our pipeline for compressing exponent bytes is compression on a per-chunk basis using a standard entropy-based lossless compressor (such as `zlib`, `bzlib2` or `lzo`). These compressors become more

effective in the reduction of data in question because of the increased degree of byte-repeatability artificially generated in the new data.

F. Index Generation

Many standard lossless compression frameworks require the use of metadata to reconstruct original data from encoded data. Metadata is generally used by a decompressor to define how to interpret coded symbols and phrases. Similarly, PRIMACY uses an indexing file per each chunk as metadata to associate ID values to their corresponding byte-sequences in the original data. We only assign IDs for those byte-sequences that actually appear in the original data. Identification values start from 0 and continue in increasing order based on the relative frequencies of byte-sequences.

Currently, the frequency analysis and indexing process is conducted on a per-chunk basis. Our study of correlations between frequency vectors of subsequent chunks revealed that the "niceness of fit" that a single chunk's index provides on the whole data is data-dependent. While a few of the datasets would compress well using only the index from the first data chunk instead of processing an index for every chunk, many would show a significant decline in compression ratio. In the future, we plan to explore the design of a more intelligent indexing scheme. One possible method of implementing such a design would be indexing a certain chunk only when its frequency analysis correlates poorly with that of the past chunk. This method would likely preserve most of the compression ratio while providing improved throughput over the current implementation.

G. Optimizing Mantissa Compressibility

Thus far, the extent of discussion has revolved around compression of the exponent information per double. However, we propose the use of another technique in regards to the compression of the remaining six bytes containing only mantissa information. Since mantissa bytes have a very high number of unique values, most of the unique values have correspondingly low frequencies and are highly interspersed, resulting in poor repeatability in identification values when applying the same method used on the exponent bytes.

As seen in Figure 3(a), frequent byte-sequences are concentrated in small ranges that comprise the majority of the data in the exponent bytes of each double. Thus, our preconditioner can take advantage of the low number of unique values by assigning them low ID values, effectively increasing byte-level repeatability. However, Figure 3(b) shows the frequency of different values in the mantissa bytes of the same datasets. In the four datasets shown, there are many nonzero frequencies—though each of these may appear very few times, the compressor still needs to consider these bytes when creating the symbol table used to encode each value.

These high numbers of unique values (each with low frequency) translates to poor compressibility, due to the unevenness and lack of skewness in data distribution. The lack of highly repeatable bytes greatly bottlenecks any improvements from the compressor [19]. Our ID value mapping method would not yield desirable results, as PRIMACY would have to process data with almost evenly distributed byte-level frequency. While this method may yield marginally improved repeatability of data on these mantissa bytes, it would likely do so at an unreasonable throughput. Therefore, we propose the use of our `ISOBAR` method to determine compressibility of mantissa bytes.

`ISOBAR` samples and analyzes "high complexity" data to determine the expected gains from applying compressors to the entire data [17]. The method works by first performing a bit-level frequency analysis in regards to whether frequency of bits in certain positions will be adequate to improve performance of standard lossless compression algorithms. If the data is identified as improvable, the data will be partitioned into "compressible" and "incompressible" pieces (the definitions for these are determined by empirically formed threshold values). We are then able to accordingly compress the classified compressible bytes using `zlib` and write out the classified incompressible bytes without wasting operations on compressing those incompressible bytes. This ultimately allows us to achieve throughput improvements without sacrificing almost any data compressibility [15].

III. Performance Modeling

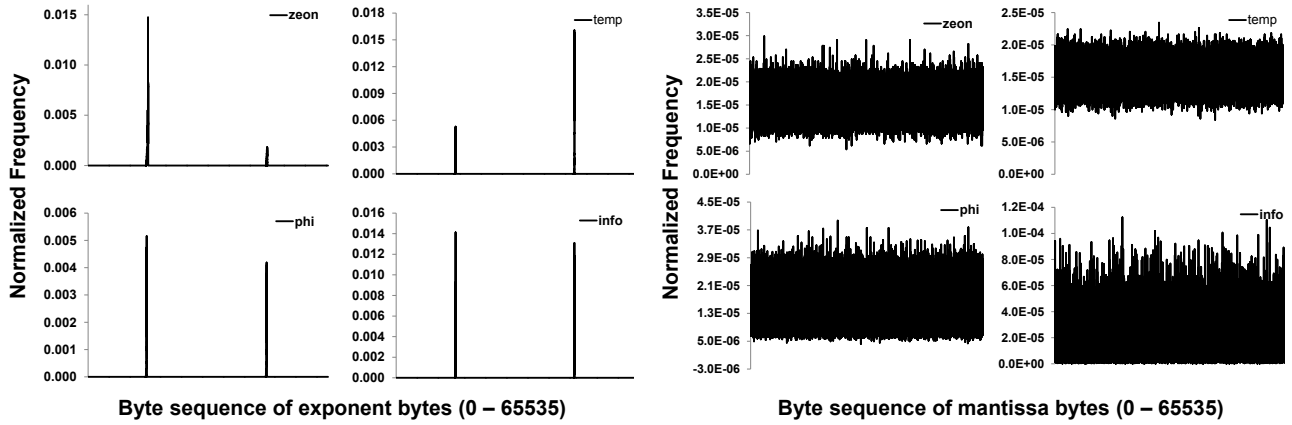
A. Parameters

We develop a generic parameterized model to evaluate the performance of the PRIMACY technique on leadership class systems. The design parameters taken into consideration and the output variables from the model are shown in Tables I and II respectively.

Table I
INPUT SYMBOLS FOR THE PERFORMANCE MODELS

Input Symbol	Description
C	Chunk size
δ	Size of the metadata
α_1	Fraction of the chunk that is compressible
α_2	Fraction of the lower order chunk that is compressible
σ_{ho}	Compression ratio (compressed vs original) on the higher order bytes of the chunk
σ_{lo}	Compression ratio (compressed vs original) on the lower order chunk
ρ	Compute to I/O node ratio
θ	Throughput of the collective network between the compute and I/O nodes
μ_w	Throughput of the disk writes
T_{prec}	Average preconditioner throughput
T_{comp}	Compression throughput

We assume a constant chunk size and compute to I/O node ratio. Data chunks are characterized by their compressible



(a) (b)

Figure 3. (a) Frequency of all possible byte values in the exponent (first two bytes), (b) Frequency of all possible byte values in the mantissa (last six bytes)

Table II
OUTPUT SYMBOLS FOR THE PERFORMANCE MODELS

Output Symbol	Description
t_{prec1}	Time to run the PRIMACY preconditioner on the chunk
t_{prec2}	Time to run the ISOBAR preconditioner on lower order part of the chunk
$t_{compress1}$	Time to compress the higher order bytes
$t_{compress2}$	Time to compress the lower order compressible bytes
$t_{transfer}$	Transfer time across the network
t_{write}	Time to write data to the disk
t_{total}	Total end-to-end data transfer time
τ	End-to-end (aggregate) throughput

fraction as well as the compression ratio on the compressible fraction, and are written following a bulk-synchronous parallel I/O model. This is a typical mode of operation for checkpoint restarts. The throughput of the collective network is measured at the I/O node. It is defined as the mean rate at which the I/O node can receive data sent by the corresponding compute nodes. We also take into account the metadata overhead imposed by PRIMACY for completeness. Finally, we leverage the fact that existing staging frameworks (e.g. ADIOS) provide consistent I/O throughputs.

We evaluate two cases of writing from compute nodes to disk for comparative purposes. The first is a baseline case when no compression is performed and data is directly written to the disk through the I/O nodes. The second uses PRIMACY compression at the compute nodes. With end-to-end throughput being the optimization criterion, it is intuitive that performing compression at the compute nodes is likely to perform better than on I/O nodes due to the parallel compression performed at the compute nodes as well as reduced amount of data being transferred over the network.

In both scenarios, the end-to-end throughput is given by:

$$\tau = \frac{\rho \cdot C}{t_{total}} \tag{3}$$

B. Base Case: No compression

In this scenario, the compute nodes simply transfer the generated simulation data at each time step to the I/O nodes, which subsequently write the data to disk. The end-to-end transfer time and throughput in this case are given by :

$$t_{transfer} = \frac{(1+\rho)C}{\theta} \tag{4}$$

$$t_{write} = \frac{\rho C}{\mu_w} \tag{5}$$

$$t_{total} = t_{transfer} + t_{write} = \frac{(1+\rho)C}{\theta} + \frac{\rho C}{\mu_w} \tag{6}$$

Equation 4 takes into account the network contention, assuming that it scales linearly with the compute to I/O node ratio.

C. PRIMACY at Compute Nodes

In this case, PRIMACY is integrated at the compute nodes. Upon generating the data, the compute nodes execute the PRIMACY preconditioner, thus separating the data into compressible and incompressible parts. The uncompressed data is then passed on the to `ISOBAR` preconditioner, which further compresses a part of it. Thus, the resulting data consists of two compressed buffers, the remaining uncompressed part, and the compression metadata. This cumulative data is then stored on the disk through the I/O nodes. This

case can be modeled as follows:

$$t_{prec1} = \frac{C}{T_{prec}} \tag{7}$$

$$t_{prec2} = \frac{(1-\alpha_1)C}{T_{prec}} \tag{8}$$

$$t_{compress1} = \frac{\alpha_1 \cdot C}{T_{comp}} \tag{9}$$

$$t_{compress2} = \frac{\alpha_2(1-\alpha_1)C}{T_{comp}} \tag{10}$$

$$t_{transfer} = (1+\rho)C\left(\frac{\alpha_1\sigma_{ho} + \alpha_2(1-\alpha_1)\sigma_{lo}}{T_{network}}\right) + (1+\rho)C\left(\frac{(1-\alpha_2)(1-\alpha_1)\sigma_{lo}}{T_{network}}\right) \tag{11}$$

$$t_{write} = (1+\rho)C\left(\frac{\alpha_1\sigma_{ho} + \alpha_2(1-\alpha_1)\sigma_{lo}}{\mu_w}\right) + (1+\rho)C\left(\frac{(1-\alpha_2)(1-\alpha_1)\sigma_{lo}}{\mu_w}\right) \tag{12}$$

$$t_{total} = t_{prec1} + t_{prec2} + t_{compress1} + t_{compress2} + t_{transfer} + t_{write} \tag{13}$$

The read scenarios essentially follow the inverse order of operations relative to writes, and can be easily modeled in a manner similar to those for writes. The model results for both the read and write scenarios are discussed in Section IV-D. In addition, we also present a comparison of these results with the actual empirical results in order to evaluate model correctness on multiple datasets.

IV. Results

A. Environment

Our experiments were conducted on the Cray XK6 Jaguar cluster at the Oak Ridge Leadership Computing Facility. It consists of $18,688$ compute systems, each with a 16-core 2.2 GHz AMD Opteron 6724 processor and 32GB of RAM. It uses the Lustre [18] file system for parallel I/O and a high performance Gemini interconnect for communication. The compute-I/O node ratio for all experiments concerning a staging environment is kept fixed at $8:1$.

B. Datasets

The datasets used in the performance evaluation comprised mainly of hard-to-compress scientific datasets used in various applications — specifically, the datasets were gathered from fusion simulations (GTS), astrophysics simulations (FLASH), parallel benchmarks, numeric simulations and satellite measurements. We compared each compressor on 20 different datasets, all of which are double-precision floats. However, PRIMACY can also perform effectively on floating-point data of higher precisions due to the nature of its mapping scheme. In interest of space, more detailed information about the datasets can be found online [1].

C. Compressor Use

We compared the performance of `zlib` and `lzo` general-purpose lossless compressors for end-to-end data reads and writes on three scientific datasets. We did not compare with `bzlib2` since its compression and decompression speeds are not suitable for in-situ processing applications. We used the `num_comet`, `flash_velx`, and `obs_temp` datasets for our analyses [1]. We specifically chose these datasets to experiment with since they are representative of the entire compressibility spectrum (as seen from Table III).

The results are shown in Figures 4(a) and 4(b). The write results indicate that `zlib` and `lzo` exhibit an end-to-end throughput improvement of 8% and 10% respectively over the null case. For reads, however, both `zlib` and `lzo` perform worse than the null case, exhibiting 7% and 4% respective reduction in performance.

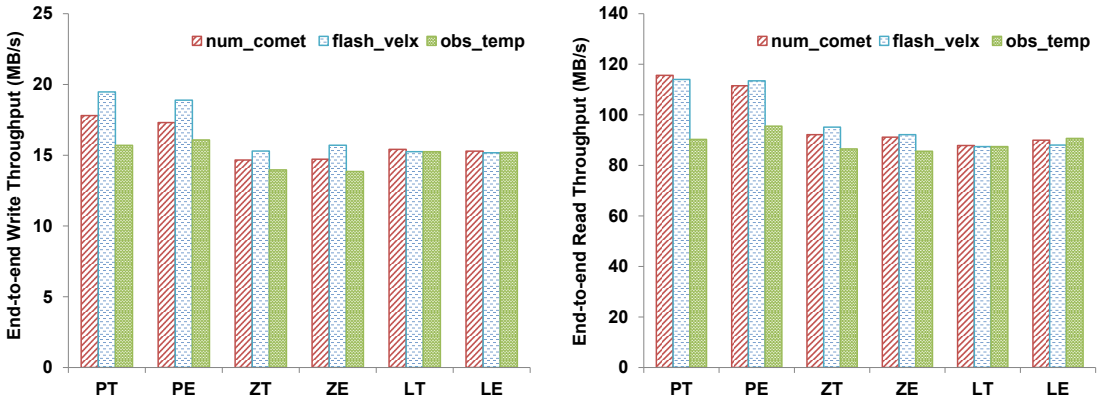
It can be observed that while `zlib` provides approximately equivalent throughput gains as compared to `lzo`, it provides significantly better compression on most datasets. For many users (especially in the in-situ processing community), the balance of compression ratio and throughput that `zlib` attains as compared to `bzlib2` and `lzo` is its primary selling point for use. Based upon these experimental results, we use PRIMACY with `zlib` compression for further evaluation.

D. End-to-end Throughput

We conducted experiments in a staging environment on the Jaguar XK6 cluster to evaluate the end-to-end throughput performance upon using PRIMACY compression on the same three datasets at the compute nodes as compared to using `zlib` and `lzo` (compression of the entire chunk), and the null case (no compression).

Figure 4(a) shows the write micro-benchmark results. It can be seen the using PRIMACY with `zlib` results in an average performance gain of 27% over the null case throughput for the three datasets. In contrast, `zlib` and `lzo` vanilla compression yield an average improvement of 8% and 10% respectively over the null case. The theoretical improvement predictions are consistent with the corresponding empirical values. Thus, PRIMACY with `zlib` outperforms `zlib` and `lzo` vanilla compression for all the three datasets.

The read micro-benchmark results are presented in Figure 4(b). Decompression using PRIMACY with `zlib` results in an average performance gain of 19% over the null case throughput. Comparatively, `zlib` and `lzo` vanilla decompression yield an average reduction of 7% and 4% respectively, in performance over the null case. This suggests that vanilla compression using either of the algorithms is not an optimal strategy for WORM usage patterns. Contrary to this, PRIMACY retains a large fraction of the performance gains observed for the write scenario. Thus it exhibits symmetric characteristics with respect to writes as well as



(a) (b)

Figure 4. Comparison of accuracy of performance model for different compression routines on 3 scientific datasets, for end-to-end write (a) and read (b) throughputs. Prefixes P, Z and L stand for PRIMACY, `zlib` and `lzo`, respectively. Suffixes E and T represent emperically observed and theoretically estimated values, respectively.

reads. Again, the theoretical predictions are consistent with the empirical values.

The results clearly articulate that the PRIMACY compression technique gives significant performance gains even on hard-to-compress datasets for writes as well as reads. Moreover, the consistency between the theoretical and empirical results for both writes and reads suggests that the models are generic and do accurately capture the algorithm modus operandi. These models can be used to predict algorithm performance on a variety of target systems.

E. Compression Ratio Performance

Table III displays the results of our experiments in terms of compression ratio (CR). PRIMACY yields higher compression ratio in all but one dataset (95%) compared to `zlib`. The particular reason for the decline in performance on the msg_sppm dataset is that the data already exhibited certain desirable characteristics of byte-sequence frequency (known as *easy-to-compress*), causing our method to be relatively ineffectual, as the indexing overhead slightly increased the size of original data. The difference in this specific dataset, msg_sppm, is a relatively small amount when considering the magnitude of the compression ratio (as the compression ratio increases, differences in compression ratio translate to less difference in the resulting compressed size). For example, in this case the actual compressed file size differs by less than 3%, while in the case of gts_phi_nl, which has a smaller numeric difference in compression ratio, the compressed dataset sizes differ by nearly 9%. The latter case is representative of the majority of runs on datasets that we tested on, as our method generally surpasses the compression ability of `zlib`. Specifically, our compression ratios are on average 13% higher than `zlib` for the used datasets.

F. (De-)Compression Throughput

Table III shows the compression and decompression throughputs for PRIMACY compared to `zlib`. Compression throughput (CTP) is crucial in determining whether data can be compressed in a time-efficient manner—utilities with high compression throughputs and appropriate algorithmic design are generally strong candidates for in-situ processing. Decompression throughput (DTP) is important in determining the accessibility of data once stored. A compression framework that has very low decompression throughput may take exorbitant amounts of time to recreate original data from compressed data. This time is often very valuable insofar as to ensure the availability of data for analysis and visualization operations once decompressed. High decompression throughput allows data to be reconstructed quickly for such applications.

PRIMACY achieved better compression throughputs than `zlib` on all but one of the datasets we tested on. Though the PRIMACY pipeline utilizes `zlib`, and is effectively bottlenecked by compressor performance, our more optimal arrangement of byte-frequency upon original data improves compressor efficiency in terms of not only compression ratio, but also speed. Additionally, the use of the `ISOBAR` analyzer and partitioner allows us to avoid spending time on compressing mantissa bytes that give a very poor trade-off on compression ratio for time spent, effectively allowing us to perform better than the use of `zlib` on all 8 bytes for each double. The only dataset that did not see an improvement was msg_sppm for the same reasons as described above. On average, PRIMACY shows compression throughputs between 3–4 times faster than `zlib`.

PRIMACY also resulted in better decompression throughputs than `zlib` on every dataset we tested on. Since not all of the original data merits compression, not all of the compressed data merits decompression. Thus, decom-

Table III
PERFORMANCE COMPARISON BETWEEN ZLIB AND PRIMACY

Dataset	CR		Linearization CR		CTP (MB/s)		DTP (MB/s)	
	zlib	PRIMACY	zlib	PRIMACY	zlib	PRIMACY	zlib	PRIMACY
gts_chkp_zeon	1.04	**1.14**	1.04	**1.12**	18.23	**84.87**	87.13	**275.22**
gts_chkp_zion	1.04	**1.16**	1.04	**1.12**	18.21	**88.93**	90.83	**279.96**
gts_phi_l	1.04	**1.15**	1.04	**1.11**	17.14	**54.19**	95.42	**201.01**
gts_phi_nl	1.05	**1.15**	1.04	**1.12**	17.02	**54.27**	89.25	**202.20**
flash_gamc	1.29	**1.47**	1.16	**1.32**	20.92	**57.06**	64.4	**214.99**
flash_velx	1.11	**1.31**	1.05	**1.15**	19.04	**184.64**	76.47	**382.16**
flash_vely	1.14	**1.31**	1.06	**1.16**	19.14	**183.92**	73.04	**380.74**
msg_bt	1.13	**1.31**	1.08	**1.14**	19.23	**23.64**	85.55	**149.91**
msg_lu	1.06	**1.24**	1.04	**1.12**	17.57	**133.92**	89.57	**317.60**
msg_sp	1.10	**1.30**	1.04	**1.14**	18.80	**76.05**	76.37	**257.28**
msg_sppm	**7.42**	7.17	**2.13**	1.99	**77.35**	66.86	32.11	**198.91**
msg_sweep3d	1.09	**1.31**	1.07	**1.17**	18.29	**24.52**	84.13	**238.22**
num_brain	1.06	**1.24**	1.06	**1.17**	17.69	**134.29**	84.94	**329.86**
num_comet	1.16	**1.27**	1.13	**1.17**	17.13	**19.73**	83.02	**117.76**
num_control	1.06	**1.13**	1.02	**1.08**	17.50	**21.11**	93.6	**193.97**
num_plasma	1.78	**2.16**	1.37	**1.50**	28.31	**37.32**	67.15	**157.42**
obs_error	1.44	**1.59**	1.16	**1.26**	24.21	**26.37**	69.13	**137.68**
obs_info	1.15	**1.25**	1.06	**1.15**	19.82	**130.02**	86.59	**335.65**
obs_spitzer	1.23	**1.39**	1.23	**1.38**	18.65	**22.07**	65.39	**113.98**
obs_temp	1.04	**1.14**	1.04	**1.14**	17.76	**89.40**	88.99	**305.78**

pression routines are generally applied to data that is efficiently processed with zlib's decompressor and avoided on practically incompressible and compressor inefficient data. On average, PRIMACY shows decompression throughputs between 3–4 times faster than zlib and is approximately in the 150–300MB/s range.

G. Effects of User-controlled Linearization

A key requirement for general lossless compression frameworks is the capacity to provide robust performance without consideration to the particular data model that a scientific application uses. For example, XGC fusion particles are mapped to a toroidal geometry and defined by their radian, toroidal, and poloidal dimensions [11]. With regard to disk access, disks may linearize data to fit a space-filling curve to improve access times, such as a Hilbert curve [12]. In many scenarios, users can choose to linearize data in a specific format, often to make operating on the data easier.

To show that PRIMACY has applicability to various linearizations of scientific data, we conducted an experiment to assess the relative performance of PRIMACY to zlib on permutations of the original datasets. The results are portrayed in Table III. The results are consistent with those of compression ratio on the original data. Though the average compression ratios of permuted data are less than those of original data (due to loss of compression through run-length encoding), the permutation did not noticeably change the improvement that PRIMACY achieves over zlib on all but one (95%) dataset.

H. Effects of Byte-Level Organization

Since we utilize a byte-level compressor (zlib) that uses repeatability to compress data, byte-level linearization is an important factor in influencing compression ratio.

Our experiments show that column-based linearization yield superior compression results when compressing identification values with zlib, due to run-length encoding gains. Compressing these bytes in column-order instead of row-order yields an 8-10% improvement in compression ratios and roughly a 20% increase in compression throughput of the identification values.

The impact of byte-level linearization on ISOBAR-identified compressible mantissa bytes is data-dependent, since the entropy of mantissa bytes is generally very high due to the large number of unique values. Our experiments revealed that compression ratio gains were seen in almost equivalent number of datasets using row and column linearization, but were trivial. Additionally, throughput gains obtained from zlib by column linearizing the mantissa bytes were generally offset by the cost of linearization.

V. RELATED WORK

Though data compression has been explored for use in order to reduce data in post-simulation contexts, the notion of making compression a viable tool to reduce I/O activity has only recently been given attention.

Filgueira et al. presented the Two-Phase Compressed I/O technique, which involves compression of data on compute nodes before transferring over the netwok, with the hopes of reducing total execution time [6]. They use lzo instead of zlib, due to the higher compression/decompression speeds. However, the results obtained from this study show that while compression can yield improvements in execution time when used on integer data, it tends to substantially worsen execution time (in some cases over 200%) when applied to floating-point data. In fact, only one dataset showed improvement in execution time for floating-point data. This characteristic is undesirable when considering that a majority of scientific applications produce floating-point data.

Welton et al. considered the use of compression as a potential means for improving network bandwidth [22]. Upon conducting a study using several compressors (`zlib`, `bzlib2` and `lzo`), the conclusion was drawn that compression (using `zlib`) was suitable at a compute-node stage for improved network transfer bandwidth and rates. Using a theoretical model, the study additionally projected that such a technique could be incorporated in scientific applications to yield improved end-to-end throughput. However, this model makes the assumption that compression and decompression costs would be negligible with sufficient nodes and does not factor in these costs when calculating performance gains. We argue that due to the sheer size of the data as well as the communication bottleneck involving transmitting varying length offsets, the overhead due to compression/decompression cannot be trivialized when considering end-to-end throughput. Both our empirical results and performance model substantiate this claim.

To the best of our knowledge, data preconditioning to optimize compression is a rather unexplored area of study. However, as a result of the importance of lossless compression techniques and their applications to the information storage and scientific communities, there is a substantial amount of work that has been put into the development of other notable data reduction frameworks that aim to improve compression ratio or throughput. In this section, we reference and compare the performance of these other top-ranked utilities to PRIMACY.

The most commonly used standard lossless compressors for information storage purposes are `zlib`, `bzlib2` and `lzo`. In the context of scientific data management, `bzlib2` is used less frequently than `zlib` because of its extremely low throughput, despite often improved compression; `lzo` is used less frequently than `zlib` due to its almost negligible compression, despite extremely high throughput. Given the results achieved from using these techniques, it is apparent that their performance leaves much to be desired. In fact, these compressors offered no more than a few percent reduction of many of the original datasets we tested upon. By including our method as a precursor to standard compression, we are able to improve overall compression throughput and ratio with marginal overhead as a result of our preconditioning technique. Though results in this paper have been limited to improvements upon the compressor with the best vanilla compression results (`zlib`), PRIMACY shows substantial improvements on both compression ratio and throughput using `bzlib2` and `lzo`. Throughput figures, though improved upon standalone `bzlib2`, are still too low for in-situ processing.

`fpzip` [13] and `fpc` [3] are two other notable tools used for scientific double-precision data compression. These tools use predictive coding rather than entropy coding techniques to reduce data. Predictive coding does not consider byte-level frequency of data, but relies on the accuracy of data prediction algorithms (`fpzip` uses an n-dimensional Lorenzo prediction algorithm where n is the data dimensionality while `fpc` uses both the `fcm` and `dfcm` algorithms) to forecast subsequent data values in order to compress data, and thus does not lend well to the use of our entropy-based preconditioning approach. In our experiments, we found that PRIMACY achieves a better compression ratio than `fpc` on all but four (80%) of the datasets, and better than `fpzip` on all but seven (65%) of the datasets. Comparatively, PRIMACY achieves superior compression throughputs than both `fpzip` and `fpc` each on all but seven of the datasets (65%), though the sets that it performs better on are different between `fpzip` and `fpc`. On average, PRIMACY achieved throughput improvements of almost 2 times in regards to `fpzip` and 3 times in regards to `fpc`.

While both `fpzip` and `fpc` offer competitive compression ratios and throughputs, they are more prone to poor performance due to the nature of the prediction algorithms used. These algorithms rely heavily on dimensional correlation of data and predict poorly in turbulent data, resulting in poor compression ratio. Furthermore, varying data organization can have a significantly negative impact on the degree of compression attained. In experiments conducted upon reorganized data, we found that PRIMACY performs better than `fpzip` on all but one (95%) of the datasets and better than `fpc` on all (100%) of the datasets. In these cases, PRIMACY shows improvements of approximately 9% over `fpzip` and 14% over `fpc` on compression ratio. Summarily, PRIMACY shows improved and more consistent compression ratio and throughput performance over both the `fpzip` and `fpc` compressors.

VI. Conclusion

Improving data compression technology is a promising approach in order to lessen the burden on I/O in HPC systems. Fast and effective compression techniques can yield gains in not only end-to-end I/O throughput, but also in effective network bandwidth and disk storage capacity. In this paper, we introduced a new technique called PRIMACY that acts as a *preconditioner* for reduction of floating-point scientific datasets via standard lossless compression *solvers*. PRIMACY conducts a statistical analysis to determine the frequencies of occurrence of various byte-sequences in the exponent bytes of the original data. Based on frequency of occurrence, a probabalistically-aware mapping function assigns identification values to byte-sequences, modifying their representation to emulate a higher degree of compressibility due to repeated byte values. This procedure allows standard lossless compressors such as `zlib` to extract a greater degree of compression given their byte-level entropy coding schemes at speeds suitable for in-situ processing.

When tested on an array of scientific datasets on the Jaguar XK6 cluster, PRIMACY improved upon standard end-to-end write and read speeds by up to 38% and 22%

respectively and showed high compression and decompression throughput improvements (as much as 160 MB/s in compression and 307 MB/s in decompression) in addition to an improved compression ratio (up to 25%) compared to the `zlib` standard compressor.

VII. Acknowledgement

We would like to thank ORNL's OLCF leadership class computing facility for the use of their resources. We would also like to thank the development teams of the XGC-1, S3D, and GTS simulations for the data used in this paper. This work was supported in part by the U.S. Department of Energy, Office of Science 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] Datasets, 2012. http://www4.ncsu.edu/~nashah3/PRIMACY/datasets.html.

[2] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182(2):418–477, November 2002.

[3] M. Burtscher and P. Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58:18–31, 2009.

[4] 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.

[5] R. D. Falgout. An introduction to Algebraic Multigrid. *Computing in Science and Engineering*, 8:24–33, November 2006.

[6] R. Filgueira, D. E. Singh, J. C. Pichel, and J. Carretero. Exploiting data compression in collective I/O techniques. In *International Conference on Cluster Computing*, CLUSTER '08, pages 479–485. IEEE, 2008.

[7] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60(1):315–339, 1991.

[8] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, 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.

[9] P. D. Grünwald. *The minimum description length principle*. The MIT Press, 2007.

[10] 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 *Proceedings of the 27th International Conference on Data Engineering*, ICDE '11, pages 1199–1208. IEEE, 2011.

[11] 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*, 49:115021, 2009.

[12] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert Space-Filling Curve. *ACM SIGMOD Record*, 30(1):19–24, March 2001.

[13] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[14] B. K. Natarajan. Filtering random noise via data compression. *Data Compression Conference*, pages 60–69, 1993.

[15] 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(878):414–427, April 2009.

[16] J. M. Prusa, P. K. Smolarkiewicz, and A. A. Wyszogrodzki. Simulations of gravity wave induced turbulence using 512 PE CRAY T3E. *International Journal of Applied Mathematics and Computational Science*, 11(4):883–898, 2001.

[17] E. R. Schendel, Y. Jin, N. Shah, J. Chen, C.S. Chang, S-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. ISOBAR preconditioner for effective and high-throughput lossless data compression. In *Proceedings of the 28th International Conference on Data Engineering*, ICDE '12, pages 138–149. IEEE, 2012.

[18] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, pages 400–407, July 2003.

[19] Y. Sehoon and W. A. Pearlman. Critical encoding rate in combined denoising and compression. In *International Conference on Image Processing*, volume 3 of *ICIP 2005*, page 341. IEEE, September 2005.

[20] 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. Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments. *Physics of Plasmas*, 13:092505, 2006.

[21] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

[22] B. Welton, D. Kimpe, J. Cope, C. Patrick, K. Iskra, and R. Ross. Improving I/O forwarding throughput with data compression. In *International Conference on Cluster Computing*, CLUSTER '11, pages 438–445. IEEE, 2011.