Byte-precision Level of Detail Processing for Variable Precision Analytics

John Jenkins [1,2], Eric R. Schendel [1,2], Sriram Lakshminarasimhan [1,2],
David A. Boyuka II [1,2], Terry Rogers [1,2], Stephane Ethier [3],
Robert Ross [4], Scott Klasky [2], Nagiza F. Samatova [1,2,*]
[1] *North Carolina State University, NC 27695, USA*
[2] *Oak Ridge National Laboratory, TN 37831, USA*
[3] *Princeton Plasma Physics Laboratory, NJ 08543, USA*
[4] *Argonne National Laboratory, IL 60439, USA*
[*] Corresponding author: samatova@csc.ncsu.edu

***Abstract*—I/O bottlenecks in HPC applications are becoming a more pressing problem as compute capabilities continue to outpace I/O capabilities. While double-precision simulation data often must be stored losslessly, the loss of some of the fractional component may introduce acceptably small errors to many types of scientific analyses.**

Given this observation, we develop a precision level of detail (APLOD) library, which partitions double-precision datasets along user-defined byte boundaries. APLOD parameterizes the analysis accuracy-I/O performance tradeoff, bounds maximum relative error, maintains I/O access patterns compared to full precision, and operates with low overhead. Using ADIOS as an I/O use-case, we show proportional reduction in disk access time to the degree of precision. Finally, we show the effects of partial precision analysis on accuracy for operations such as k-means and Fourier analysis, finding a strong applicability for the use of varying degrees of precision to reduce the cost of analyzing extreme-scale data.

I. Introduction

Data reduction in extreme-scale, scientific simulations is a quickly emerging necessity to reduce current and especially future I/O bottlenecks, as compute performance continues to increase at a far higher rate than I/O performance [1]. I/O bottlenecks are especially pronounced when running analysis on simulation-generated data, a typically read-only process performed numerous times by multiple application scientists, often on dedicated analysis clusters with less computational power than the machines the data was generated on.

In the context of extreme-scale computing, data reduction technologies face a number of unique architectural, algorithmic, and application-specific challenges which complicate the emergence of an efficient solution that is highly applicable across application contexts. First, scientific data is notoriously hard-to-compress, due to the utilization of double-precision floating-point variables. These variables tend to have highly entropic mantissa bits, leading to data reduction only on the order of $10 - 30\%$. Achieving higher compression ratios with lossless compression methods require the discovery of non-trivial patterns within the typically spatio-temporal data, making these methods unsuitable for *in-situ* processing. While state-of-the-art lossless compression utilities such as ISOBAR [2] and FPC [3] have been making headway into fast lossless compression of scientific data, achieving high degrees of data reduction while retaining full precision is still best suited to a post-processing scenario, where a full-context approach is possible.

Lossy compression, on the other hand, can greatly increase the compression ratio, making it more suitable for alleviating I/O bottlenecks *in-situ*. However, application scientists spend an enormous amount of effort ensuring accurate and precise simulation results. While the loss of precision may be acceptable for some simulations, it will not be for all.

Regardless of method, compression as a data reduction strategy introduces new, less optimal access patterns to the I/O system, on both a software and hardware level. On the software level, non-uniform compressed buffer sizes would necessitate global communication between I/O nodes, introducing additional latency costs. On the hardware level, applications optimized for certain striping patterns may lose performance due to the non-uniform buffer sizes, leading to disk contention and lost I/O bandwidth.

Given the challenges that data reduction impose for simulation codes, instead of focusing on reducing data at *write-time*, we argue it is highly beneficial to reduce data at *read-time* through partial-precision analytics. The key insight here is that many types of analysis functions may produce acceptably accurate results even with a greatly reduced amount of precision. A common invocation of this principle can be found in *multiresolution analysis* (MRA) of wavelet-compressed data, traditionally used in the graphics and visualization communities. However, wavelet-based MRA has no bounds on errors at any resolution, and is technically not lossless for double-precision data (though it can be in some cases for single-precision data [4]). Furthermore, wavelet compression standards such as JPEG 2000 [5] have been successfully used to compress single-precision climate data [6], but requires the quantization of single-precision floating-point data, something which may not work well on double-precision datasets crossing a wide range of exponent values.

To achieve these ends, we propose a *analytics-driven precision level of detail* (APLOD) preprocessing methodology.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00

Our approach is inspired by the bit-level format of double-precision variables, and the fact that truncation of the mantissa component leads to low, bounded maximum errors based on the number of mantissa bits kept (see Table I). To promote high efficiency as well as application-specific tuning based on the accuracy needs of scientific simulation analyses, we enable a generalized partitioning of double-precision data along byte-boundaries, described by a byte-level *component vector* (CV). In other words, based on user preferences, datasets are partitioned into groups of contiguous significant bytes, such as the most significant two bytes. Datasets are stored contiguously by most significant bytes so that only data at a required level of precision can be loaded into memory. There are numerous benefits to this approach that, to our knowledge, have not been utilized by other analysis-level data-reduction methods:

- APLOD processing enables *configurable* byte-level decompositions, providing simple access to a range of precisions, including full precision, based on application needs. Each degree of precision provides a hard bound on per-point relative error, as opposed to wavelet MRA.
- APLOD processing minimally disturbs existing parallel I/O access patterns. If I/O patterns in an application are communication-free, then the patterns with APLOD processing are communication free. Buffer sizes are deterministic, given the original data's buffer sizes. The storage barrier to APLOD processing is low, requiring at most tweaking to disk striping parameters.
- APLOD processing is a low overhead operation in both the *shuffling* of a double-precision buffer to the decomposition defined by a CV as well as the *reconstruction* of the partitioned data back into original (or truncated) form. Even for the finest grain decomposition, the transform operations achieve a throughput of 600MB/s. Wavelet transforms, however, perform at a maximum of 434MB/s, which degrades significantly for very large buffers. When reconstructing partial-precision data from a packed significant byte representation, transform time is decreased in direct proportion with the data reduction. On a per-core basis, the transform throughput far exceeds hard disk bandwidth, making APLOD processing an ideal candidate for *in-situ* integration with applications.
- APLOD processing is orthogonal to existing data reduction and I/O optimization methods. Lossless compression can be applied to each byte-component (oftentimes resulting in higher compression ratios [7]), and I/O optimizations exploiting data layout patterns need not be significantly changed to incorporate APLOD layout changes. Since wavelet MRA has stricter data layout requirements, applying complex layout optimizations to wavelet-transformed data can be nontrivial.
- The programming overhead required to express APLOD operations is minimal and simple to express using well-known I/O libraries, such as ADIOS [8].

This paper is organized as follows. First, we discuss related works in Section II. Next, we discuss the transformation methodology, including different ways of expressing the operations (manually or using MPI datatypes), as well as simple ways of expressing the I/O operations through ADIOS, in Section III. In Section IV, we evaluate the methodology in a number of benchmarks, including ADIOS I/O performance with and without partial precision decompositions enabled (Section IV-A) and transform overhead for both manually coded and MPI-datatypes-based APLOD representations (Section IV-B). Finally, while we do not provide an exhaustive or theoretical treatment of analysis algorithm accuracy, we empirically test a number of algorithms for varying precisions on real-world large-scale scientific applications GTS [9], S3D [10], and XGC-1 [11], in Section IV-C.

TABLE I: Maximum per-point percent errors on partial-precision IEEE 754 doubles, masking the remaining bytes with the quantity 0x7F...FF.

Significant Bytes	Max Underest. Error	Max Overest. Error
2	-1.5e0%	3.1e0%
3	-6.1e-3%	1.2e-2%
4	-2.4e-5%	4.8e-5%
5	-9.3e-8%	1.9e-7%
6	-3.6e-10%	7.3e-10%
7	-1.4e-14%	2.8e-12%

II. Background

The concept of *level of detail processing*, or operating on a subset or approximation of the full context dataset, has been explored in numerous contexts. Examples include statistical sampling techniques, arising from the database community, I/O optimization frameworks in HPC that enable sampling on a spatio-temporal domain, and a wide range of signal and image processing techniques, using transforms such as various families of wavelets that enable MRA. Note that each level of detail processing method discussed is compatible with APLOD processing through applying the operations on each significant byte component, though seek costs, effects on wavelet accuracy, etc. need to be considered from the partitioning of byte-components.

Sampling in databases is a well-studied area, with much early work on general sampling queries [12], [13], sampling on an index [14], and sampling in spatial databases [15]. More recent work has also treated the issue of minimizing sampling error due to data skew [16], [17]. Database sampling methods typically do not change the underlying data layout, leading to a large number of seeks, a problem in HPC environments, where seeking in parallel file systems is a high-latency operation. Furthermore, statistical sampling runs the risk of losing small features or sharp transitions in the data, especially since most scientific datasets exist in some spatio-temporal domain. Finally, statistical sampling, such as random sampling, provides for data analysis errors as a distribution, rather than as a bound; while improbable, there is still the chance for significant skew depending on the data distribution.

To tie data access patterns to level of detail processing and achieve high-performance sampling, Pascucci and Frank [18]

take a different approach and focus on I/O efficiency instead of statistical rigor. They break the data into progressively coarser, mutually exclusive subsets according to the Z-order space filling curve. Each subset is stored contiguously, allowing efficient access to coarse-grained data subsamples, corresponding to the spatial patterns exhibited in the space-filling curve. By retrieving some number of contiguous layers of the hierarchy, data can be returned at a particular sampling rate with only a single sequential read operation. Related methods in the context of visualization are described elsewhere [19], [20], and also use hierarchical data layouts. This method also provides a speed/accuracy tradeoff for data analysis, but has some potential drawbacks. Aside from the mentioned statistical sampling problems, the fixed sampling procedure of hierarchical data layouts is prone to selection bias, though for analysis operations such as visualization this is less of an issue.

Discrete wavelet transforms and MRA [21] are particularly popular for level of detail processing, commonly used in image standards such as JPEG 2000 [5] for high compression ratios while maintaining image integrity. In general, discrete wavelet transforms convert an input signal into detail coefficients (from a *high-pass filter*) and approximation coefficients (from a *low-pass filter*), recursively transforming the approximation coefficients. The hierarchical nature of the transform allows reconstruction of the original signal from a subset of the approximation coefficients with high accuracy. Thus, wavelet MRA is the closest applicable method to our proposed APLOD processing. However, while low average errors from MRA are shown (see Section IV), there is no upper bound on the errors, and accuracy depends on spatio-temporal relationships in the data. Compared to APLOD, wavelet MRA allows for a much finer-grained decomposition of data. However, at least for the double-precision datasets we test on, using even one-half of the coefficients leads to unacceptable errors, rendering this capability mostly ineffective with such datasets.

III. Methodology

As mentioned, the goal of enabling efficient variable-precision analytics requires a modified data representation, one that is capable of being queried by varying degrees of precision. The reduced precision format should directly result in reduced I/O costs; otherwise, there is little tangible benefit to using such a representation aside from perhaps a reduced memory footprint. Furthermore, overhead for performing the transformation to and from the data representation must be small enough to not bottleneck the application.

Based on these restrictions and differing analysis needs of applications, we define a simple, parameterized data decomposition model based on *component vectors* (CVs). We treat a buffer of data as a matrix of bytes, then define column slices (components) based on significant byte boundaries. These slices are transformed to occupy a contiguous buffer, which can then be read into memory separate from other levels of precision. CVs define these column slices, allowing users to choose a data decomposition that makes sense for their analysis needs.

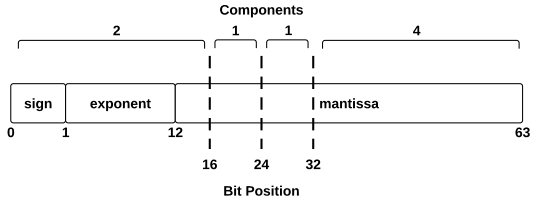


Fig. 1: The partitioning of a IEEE 754 double-precision value by the CV $\{2, 1, 1, 4\}$.

The basis for the component vector is the representation of double-precision floating point data in memory. Recall that the IEEE 754 standard [22] represents double-precision values as a single sign bit, 11 exponent bits and 52 mantissa bits (see Figure 1). The value represented by double-precision data (not including subnormal values) is given by:

$$\text{value} = (-1)^s \times (1 + \sum_{i=1}^{52} (m_i 2^{-i})) \times 2^{e-1023}, \qquad (1)$$

where s is the value of the sign bit, m_i is each mantissa bit in decreasing order of significance, and e is the unsigned integer interpretation of the exponent bits.

Given this representation, we make two observations that drive our partial-precision representation. First, we observe that the mantissa bits represent a fractional component in the overall value. That is, the entire mantissa component with the implicit one bit, between the minimum (of all zeroes) and maximum (of all ones), represents a factor in the resulting double-precision value in the range $[1, 2)$. Second, we observe that each less significant mantissa bit contributes an exponentially smaller amount to this multiplier. Given Equation 1, we can easily bound the effects of truncation or replacement of the mantissa bits, the results of which can be seen in Table I.

Combined with the use of the exponent bits, truncation of the less significant mantissa bits thus provides a good opportunity to reduce the amount of data at small error rates. By comparison, truncation is typically not feasible for integer data since the bits encoding the integer are not exponentiated by a set of exponent bits. Given the double-precision format, the smallest byte-wise CV must always contain the exponent portion; otherwise, unacceptably high error rates would be generated. Hence, since we use byte-boundaries for efficiency reasons, the first component in any CV is restricted to be at least two bytes, as shown in Figure 1. As shown in Table I, truncating to the most significant two bytes, along with masking the discarded mantissa bits, leads to a maximum relative error of 3.1%.

Given the definition of the byte-level partitioning of double-precision data, a high level system overview is given in Figure 2. An application defines a CV suitable for its partial-precision analysis needs, the original data is *shuffled* into the new representation and written to disk. At analysis time, users read partial (or full) precision data on a per-component basis, according to the needed level of precision. If necessary, the data is then *reconstructed* back to the original format, though

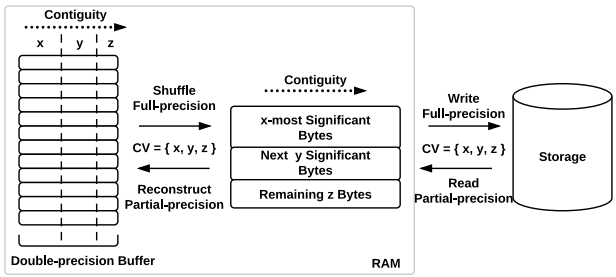


Fig. 2: Byte-precision level of detail partitioning, based on a generic component vector (CV) splitting groups of significant bytes.

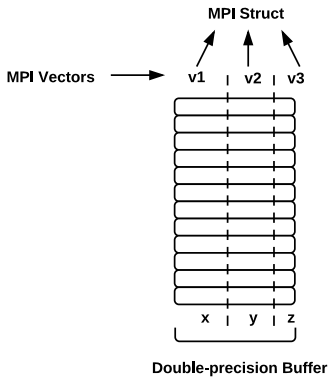


Fig. 3: Partial-precision level of detail transformation using MPI datatypes, for CV $\{x, y, z\}$.

with the missing significant bytes masked with an appropriate value to reduce the average error.

An example CV defined over a double-precision value is shown in Figure 1. For this CV ($\{2, 1, 1, 4\}$), the first two bytes of each double-precision value are stored contiguously. As mentioned, these bytes contain the sign bit, all 11 exponent bits and the four most significant mantissa bits. If a buffer of size 8MB was being shuffled, the result would consist of four buffers: a buffer containing 2MB of most significant two bytes, two 1MB buffers of the next two significant bytes, and a 4MB buffer containing the remaining significant bytes.

A. Component Vector Representation and Operations

The shuffling and reconstructing operators can be defined in two ways, through using the Message Passing Interface (MPI) Standard to specify MPI datatypes and transforming implicitly through MPI communications and I/O routines [23], or as standalone operators. Each component in a CV has a number of semi-regular structures, sharing an element-to-element stride as well as a total number of elements, differing only in the "width" of each datum. These can be simply represented as a set of MPI `vector` types, with `blocklengths`, or number of elements per stride, as the number of bytes in the component. A `vector` type is defined per component, and the set of vectors with necessary offsets are encoded into a single MPI `struct` type, which specifies a set of distinct datatype, offset pairs. Figure 3 shows the overall type for an example CV. Given the three components X, Y, and Z, three MPI `vectors` are used (with possible repetition for components with equal byte-widths), and packaged together as a single MPI `struct`. These types can then be directly used within MPI I/O or communication routines, leaving the MPI library to handle the packaging of the data.

However, there are a number of issues with a purely MPI-based representation. First, the packing algorithm, which places the non-contiguous data into a contiguous buffer given a datatype specification, may not be efficient compared to a low-level implementation which can take advantage of vectorizing and loop unrolling. See Section IV-B. Second, in order to reconstruct at varying degrees of precision, it is necessary to create a different datatype per number of components to load. For example, for the CV $\{2, 1, 1, 4\}$, four MPI `structs` need to be created, one for the first component, one for both the first and second components, and so on. Furthermore, when reconstructing partial-precision data, it is necessary to perform an additional memory setting operation to zero out or mask the significant bytes not loaded in, adding to the overhead. A specialized implementation is able to roll this memory set operation into the reconstruction process.

A manual representation is much simpler, consisting merely of the CV. The shuffling and reconstruction operations can be seen as a generalization of a matrix transpose, where each column is of variable width. This would suggest usage of efficient matrix transpose algorithms. However, in our case, one dimension is always on the order of a few bytes. Such a restricted problem space makes using complex matrix transpose algorithms unnecessary: they break down to following the same access patterns as the naive transpose algorithm. Hence, the naive algorithm is sufficient, consisting of a single loop per component, setting the data in the APLOD format to its correct location in its original format, while taking advantage of component widths and necessary memory setting operations along the way.

B. Partial-precision I/O

While it is difficult to provide a generic I/O analysis for each simulation code, there exist a number of parallel I/O middlewares that abstract file storage details from the application while ensuring high performance. A particularly popular I/O abstraction is the Adaptable I/O System (ADIOS) [8]. ADIOS is a state-of-the-art componentization of the I/O system that, with a simple change to an entry in an XML configuration file, changes codes to use numerous I/O backends, called "transports," without requiring application recompilation. For example, POSIX, MPI-IO, parallel HDF5 [24], PnetCDF [25], and numerous others are supported transports. Given the flexibility of I/O methods to use and the possibility for defining the CV through the XML configuration file, we chose to use ADIOS to show that data reduction through our partial precision reorganization of data can translate directly to improved I/O costs, and can do so under a wide range of application contexts.

```
<!-- original variable -->
<var name="phi"   type="double"          ... />
<!-- variable in APLOD format, CV=2,1,1,4 -->
<var name="phi.c1" type="unsigned short" ... />
<var name="phi.c2" type="unsigned char"  ... />
<var name="phi.c3" type="unsigned char"  ... />
<var name="phi.c4" type="unsigned int"   ... />
```

Fig. 4: A double-precision variable in the ADIOS XML configuration, in both unmodified and in APLOD format.

Enabling APLOD processing using ADIOS requires two simple changes. The first change is at the configuration level, where the ADIOS XML configuration file is modified to support the retrieval of partial-precision data. Figure 4 shows an example. For each variable being reorganized using APLOD, new variables are created for each CV component to replace the original variable. For multivariate data, there are two ways to place the CV components in the configuration file. The first is to interleave the CV components for each variable, optimizing the access of multiple variables at a particular degree of precision in one fell swoop. The other is to place all CV components consecutively for each variable, optimizing for univariate access of data at various degrees of precision.

The second change required is at the code level, where the I/O operations are modified to handle the partial-precision data. Following the data transform, a write/read is made for each APLOD component, depending on the order described in the previous paragraph, replacing the original I/O calls. Since the ADIOS API fetches variables from file using string identifiers, this process can be automatized using an appropriate naming metric, such as *variable-name.component*, and thus can be hidden behind wrapper functions.

IV. Experimental Evaluation

To evaluate the APLOD methodology in a leadership-class HPC environment, we perform all experiments on Oak Ridge National Laboratory's Jaguar cluster (Cray XK6 architecture), consisting of a single 16-core AMD Opteron 6200 processor and 32GB of memory per node. For I/O benchmarks, we use ADIOS version 1.3.1 on the Lustre parallel file system.

We look at datasets from a number of real-world simulations. GTS [9] and XGC-1 [11] are both particle-in-cell simulations of nuclear fusion devices, with GTS studying microturbulence in the plasma core and XGC-1 studying microturbulence at the edge. We examine the *potential* (ϕ) variable of a single timestep from GTS and the *temperature* varaible of a single timestep from XGC-1. Finally, we look at S3D [10], a direct numerical simulation of reacting flows in combustion. We examine the *velocity* variable of a single timestep in two dimensions (*uvel* and *vvel*).

As a primary source of comparison, we use wavelet multiresolution analysis. Specifically, we use the D4 Daubechies wavelet provided by the GNU Scientific Library (GSL) [26]. Compared to the other wavelet options available in the GSL (such as Haar and Spline), the D4 wavelet was the most accurate for the datasets used in this paper and had minimal performance differences. For the GTS potential variable, which is linearized from a toroidal structure, we use the one-dimensional wavelet transform. For the other two-and-three-dimensional datasets, we divide the data spatially into 32×32 blocks and use the standard two-dimensional transform on each, which interleaves each level of the transform between rows and columns. For comparison against APLOD reorganization, we then load the most significant wavelet coefficients and reconstruct the data, replacing with zeroes the remaining coefficients.

We perform a number of benchmarks to evaluate our methodology. First, we test whether reading reduced data correlates to lower I/O costs using a parallel read of both unmodified double-precision data and APLOD-reorganized data. Second, we examine the APLOD transform overhead for numerous CV configurations, comparing against an MPI datatypes representation as well as against the D4 transform. Finally, we examine analysis functions of varying degrees of complexity to evaluate the effects of partial precision reconstruction and wavelet MRA on analysis accuracy. Note that, as opposed to reconstructing the data as double-precision variables with a masked mantissa portion, it is possible to reconstruct in single-precision format or even using a fixed-point representation, as proposed by Narayanan [27], to optimize analysis performance, energy efficiency, etc. These methods are complementary to the APLOD methodology, using APLOD reorganization to reduce the I/O costs while using the alternate reconstructions to optimize various metrics. However, to evaluate the effectiveness of our partial precision representation with respect to analysis accuracy, we reconstruct the data in double-precision format to remove any additional effects on accuracy that the alternative representations would induce, in effect keeping the data in as close to original form as possible.

A. I/O Performance

Using ADIOS as a driver for our APLOD I/O performance benchmarking, we investigate parallel read performance under a number of scenarios. On a per-reader basis, we chose a partition size of 512MB for two reasons: first, data reading scenarios tend to use many less cores than writing for analysis purposes due to different resource allocation for the respective tasks, and second, we wish to target bandwidth-bound analysis scenarios, and thus minimize the effect of access latency. Furthermore, we chose to test on the CV $\{2, 1, 1, 4\}$, as our accuracy results in Section IV-C indicate that the first four bytes of double-precision variables are sufficient to perform many types of analysis operations, while the latter four bytes are used where full precision is needed (*e.g.*, checkpoint-restart data).

Three scenarios for read performance are shown in Figure 5. Our base case is ADIOS performance without data reorganization, which tops out in our tests at about 46GB/s, or 5.7 billion double-precision elements. This is closely matched by the ADIOS read performance of full precision data using reorganized APLOD data with the CV $\{2, 1, 1, 4\}$, providing evidence that the reorganization of data does little to disturb read access patterns at the disk level. However, for manually

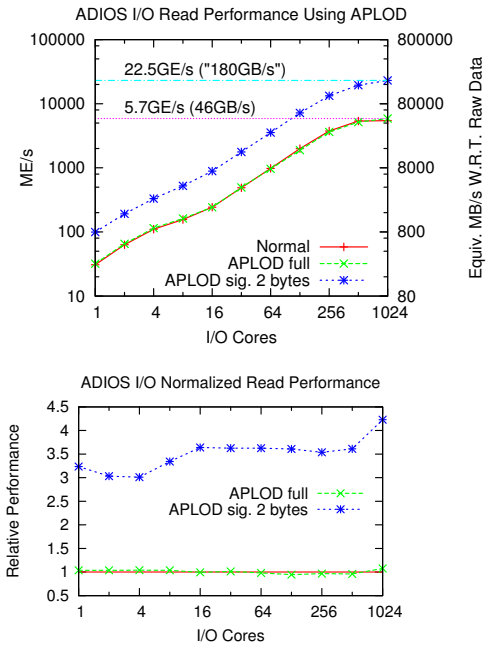


Fig. 5: Parallel I/O read performance (actual and relative) using ADIOS, with and without APLOD-reorganization. ME/s - millions of elements per second. GE/s - billions of elements per second.

tuned I/O read patterns, more care would be needed. Finally, we show the effects of reading only the first APLOD component, corresponding to the two most significant bytes. While the time taken by a single compute node (16 cores) is degraded somewhat, increasing the number of cores shows between 3.5 and 4 times the improvement in I/O performance, relative to the number of "elements" read. At 1024 cores, the partial-precision read operation occurs at 22.5 billion elements per second, or a relative "180GB/s".

B. Transform Performance

For a number of CVs, we test APLOD transformation performance using both our manual implementation and the MPI datatypes representation. For each benchmark, we measure the slowest possible transformation, corresponding to the finest grain CV $\{2, 1, 1, 1, 1, 1, 1\}$, as well as a few others, to show the effect of partitioning with varying degrees of coarseness on transform overhead.

Furthermore, we compare against the one-dimensional transform provided by the GSL for a few reasons. Most notably, the output format of the one-dimensional transform may be directly used for multiresolution analysis and is thus most applicable for comparison against APLOD. The two-dimensional transform, in order to support the contiguous layout of each "level" of the transform, would require additional data reorganization. Combined with the need to transform non-contiguous column data, the overhead becomes unacceptably large.

First, Figure 6 shows both full-precision shuffling and reconstruction operations. Asides from very small sizes (less than one KB), the manual implementation at the finest grain CV shows a throughput of 600MB/s for a 1MB buffer, and a maximum of 744MB/s for an 8KB buffer. For coarser grain CVs, a clear performance advantage is seen due to the implementation taking advantage of larger byte widths. At a very coarse grain partitioning (CV $\{4, 4\}$, where the first component represents approximately the level of precision of a single-precision floating point number), the transform is extremely fast, operating with a throughput of up to 2GB/s. For the reconstruction, the throughput experienced a small drop-off due to the data size exceeding available cache space. MPI datatypes in this case (via a call to `MPI_Pack`) are not able to provide the performance of the manual implementation, operating at an approximately order of magnitude lower throughput. The wavelet transform's performance lies between the two APLOD implementations, showing a throughput of 275MB/s for a 1MB buffer and a maximum throughput of 434MB/s for a 2KB buffer. As evidenced in the graph, the performance of the wavelet transform experiences larger regressions than the other methods for larger buffer sizes.

Next, Figure 7 shows the throughput of the reconstruction process, in terms of the amount of output data produced. The rate of transformation is increased in proportion to the precision of data transformed. For instance, reconstructing a buffer from the most significant two bytes produces the data at a rate of up to 4.3GB/s. Similar trends can be seen in reconstructing the most significant three and four bytes, including performance increases from using coarse-grained CVs. As with the full-precision shuffling and reconstruction, the MPI packing methodology achieves approximately five to ten times slower performance. Performance metrics for wavelets are not shown here, as the inverse transform performs equivalent operations regardless of the proportion of coefficients used, meaning that wavelet multiresolution analysis only takes advantage of a reduced data representation for I/O operations.

C. Partial Precision Analysis Accuracy

1) Basic Measures: While the benefits of partial precision analysis to I/O costs are clear and intuitive, it must be verified that partial precision analytics is a viable methodology to pursue. More specifically, it is necessary to determine at which levels of precision analysis functions provide adequately accurate results. As the scope of analysis functions on scientific data is much too large for an exhaustive survey, we choose a number of simple and more complex analysis scenarios to determine the potential for large-scale partial-precision analytics.

First, we ensure that the general structure of the data is consistent, which we expect to see from the low maximum errors presented in Table I. To that end, we calculate mean and maximum per-point relative error in the dataset, Pearson correlation, and the relative errors of mean/standard deviation between the original datasets and the partial precision

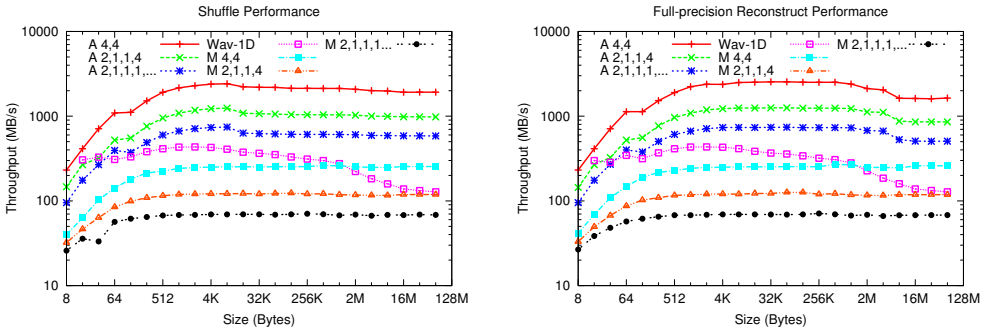


Fig. 6: Performance of transforming from original data layout to component-level contiguous chunks, and vice versa. *A* - APLOD. *M* - MPI datatypes. *Wav-1D* - one-dimensional wavelet transform.

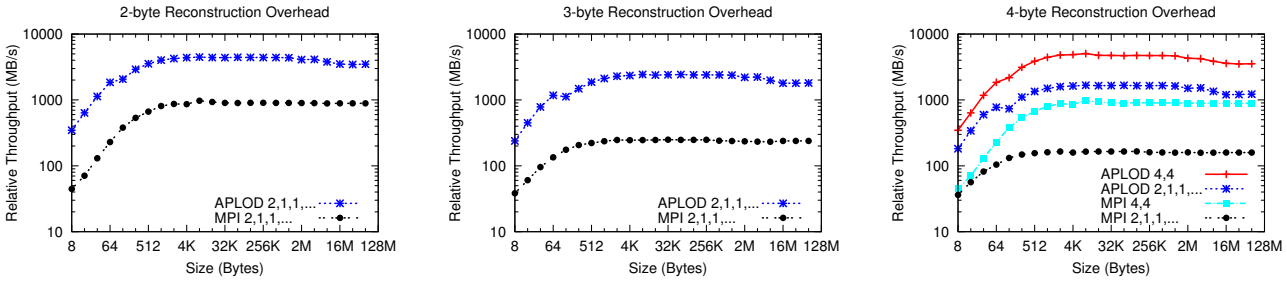


Fig. 7: Performance of partial-precision value reconstruction.

datasets. Each of these metrics are shown in Tables II, III and IV, respectively. These results show that the datasets are roughly equivalent in the aggregate, with low errors for the mean and standard deviation values and near equivalent (1.0) Pearson correlations. Furthermore, the average absolute relative error per-point is low, though the maximum error for two bytes of precision reaches the maximum possible. In comparison, the wavelet-reconstructed data has much more variability. The most significant difference between the D4 wavelet and APLOD is the presence of large relative errors in the wavelet-reconstructed data, that persist even for a large number of coefficients. Also, the errors seen in the wavelet-regenerated data see a lesser degree of change when moving from lower precisions to higher precisions, both due to the "vanishing" nature of the less significant coefficients as well as the persistence of high maximum errors throughout.

2) Distributional Analysis: k-means and Histogram: Since there is little error between points of data, and that error is distributed across the dataset, the previous metrics were expected to be highly accurate. However, small changes to a variable may change the global behavior of analysis algorithms that, for example, partition the data. In the worst case, edge conditions between the partitions can be sensitive to changes in precision, producing different results. To test the possibility of small local changes yielding large global changes, we performed two experiments on the data: generating an equal-interval histogram based on the data with a constant number of bins, and clustering the data via the k-means algorithm with randomized centroids.

TABLE II: Per-point relative errors (absolute values).

		Per-point relative error (%, absolute)			
		Median		Maximum	
Variable	**Parameter**[1]	**APLOD**	**D4**	**APLOD**	**D4**
potential	2	1.08e0	1.10e1	3.12e0	2.12e6
	3	4.24e-3	-	1.22e-2	-
	4	1.65e-5	4.55e0	4.76e-5	5.57e5
temp	2	1.02e0	1.30e-1	3.12e0	7.10e1
	3	4.24e-3	-	1.21e-2	-
	4	1.63e-5	5.34e-2	4.73e-5	4.08e1
uvel	2	1.08e0	4.44e-2	3.12e0	3.84e5
	3	4.15e-3	-	1.22e-2	-
	4	1.62e-5	1.23e-2	4.77e-5	1.31e5
vvel	2	1.08e0	6.34e-1	3.12e0	1.30e7
	3	4.24e-3	-	1.22e-2	-
	4	1.66e-5	1.87e-1	4.77e-5	2.26e6

[1]Proportion of data used. For APLOD, the number of significant bytes. For the D4 wavelet, the proportion of coefficients brought in for multiresolution analysis, as a fraction of eight.

Table V show the *misclassification rate* of running the k-means algorithm on the *uvel* and *vvel* variables. The misclassification rate is computed by using the same randomly chosen centroids for both the full and partial precision data and computing the proportion of partial-precision points assigned to different clusters than the corresponding full precision points. For two bytes of precision, 5.41% of points are assigned to a different cluster than when using full-precision values. These points are likely near the "edges" of the original clusters, which are vulnerable to membership switching through small changes in the cluster centers. The error quickly disappears

TABLE III: Pearson Correlation between full and partial-precision data.

		Pearson Correlation	
Variable	**Parameter**[1]	**APLOD**	**D4**
potential	2	1-8.59e-5	0.972
	3	1-1.34e-9	-
	4	1-0.00e0	0.996
temp	2	1-5.75e-04	0.954
	3	1-9.04e-09	-
	4	1-1.34e-13	0.984
uvel	2	1-6.59e-04	0.992
	3	1-9.97e-09	-
	4	1-4.77e-14	0.998
vvel	2	1-9.27e-05	0.993
	3	1-1.42e-09	-
	4	1-0.00e0	0.998

[1]Proportion of data used. For APLOD, the number of significant bytes. For the D4 wavelet, the proportion of coefficients brought in for multiresolution analysis, as a fraction of eight.

TABLE IV: Partial-precision relative errors for mean and standard deviation.

		Mean Error (%)		Std. Dev. Error (%)	
Variable	**Parameter**[1]	**APLOD**	**D4**	**APLOD**	**D4**
potential	2	3.88e0	1.60e-11	5.53e-2	2.77e0
	3	2.88e-2	-	2.70e-5	-
	4	2.06e-4	9.78-11	1.54e-7	4.24e-01
temp	2	6.87e-2	7.91e-14	2.92e-1	4.64e0
	3	6.46e-6	-	1.87e-4	-
	4	4.57e-8	9.10e-13	1.56e-9	1.64e0
uvel	2	1.90e-2	8.47e-12	9.85e-2	8.16e-1
	3	3.81e-6	-	1.95e-5	-
	4	4.64e-9	1.27e-11	1.39e-8	2.09e-1
vvel	2	4.26e-2	2.68e-12	6.30e-2	6.90e-1
	3	1.04e-5	-	6.80e-6	-
	4	2.30e-8	3.64e-12	5.81e-8	1.55e-1

[1]Proportion of data used. For APLOD, the number of significant bytes. For the D4 wavelet, the proportion of coefficients brought in for multiresolution analysis, as a fraction of eight.

when using a higher degree of precision, however. Once again, in the wavelet-reconstructed data we see a relatively consistent level of error due to the existence of outliers. At the lowest degree of precision tested, the error is less than that of APLOD, but the persistent presence of outliers regardless of the proportion of wavelet coefficients keeps the errors relatively consistent, while increasing the bytes of precision with APLOD quickly overcomes such errors.

Figure 8 superimposes the partial-precision histogram on the full-precision version. We display only the XGC-1 temperature histogram as it is the most illustrative of the trends shown in the datasets considered (S3D has similar patterns, but to a lesser degree, and the exponential component of GTS data differs by far too much for a equal-interval histogram to capture useful distributional information). Note that the selection of data we are using has little difference in the exponential component, meaning that the fractional component is a relatively more important component of the data.

TABLE V: Clustering errors, measured as the misclassification rate compared to full-precision. The *uvel* and *vvel* variables from S3D are partitioned into 10 clusters.

	K-means Error (%)	
Parameter[1]	**APLOD**	**D4**
2	5.41e0	3.27e0
3	2.52e-1	-
4	1.30e-3	3.31e0

[1]Proportion of data used. For APLOD, the number of significant bytes. For the D4 wavelet, the proportion of coefficients brought in for multiresolution analysis, as a fraction of eight.

For APLOD-based reconstruction, as Figure 8 shows, the problems with losing precision (truncating the six least significant mantissa bytes) are apparent. There is a clear "clustering" effect whereby there is not enough precision to sufficiently separate values into bins when equal-interval binning is used, hence the oscillatory pattern between empty bins and much larger bins. One solution is to use different parameters based on the degree of precision used, which is undesirable; ideally, reduced precision should only introduce noise into the analysis, instead of requiring a revisiting of the analysis function. In any case, with three bytes of precision, these problems all but disappear, suggesting that three bytes is accurate enough to represent this dataset. Data with small ranges that primarily occupy the mantissa portion of the double-precision data are likely to need additional precision.

The wavelet-reconstructed data also has distributional problems, though not of the variety that the APLOD data has. Figure 8 shows that the existence of outliers shift the range of the data, leading to a different distribution of values among them. However, even if the bins were "shifted" from lower to higher bins in Figure 8, skewed results would still appear, as evidenced in the spikes in values compared to the three bytes of precision in APLOD, which matches perfectly against the full-precision data.

3) Fourier Analysis Accuracy: Finally, some analysis functions involve transformations of the data into a different space. In particular, many applications rely on signal processing techniques for data analysis, built on transformations such as wavelet and Fourier transforms. The data in this case may be especially prone to *error propagation* since new points of data are produced by sometimes complex operations on the full or subselected data set, which are then analyzed.

For GTS simulations in particular, Fourier Transforms (FFT) of the grid-based electrostatic potential are carried out to analyze the spectral characteristics of turbulence in the fusion core. Gradients in the hot fusion plasma generate so-called "drift waves", which couple with each other through non-linear effects. Charged particles forming the plasma continuously exchange energy with these waves as turbulence develops. Spectral analysis through FFTs allows the identification of the most important modes in the system (i.e., fastest growing modes) and how they couple with each other to form other modes.

To examine the effect of running FFTs on partial-precision data, we perform the transform on the full-precision and partial-precision GTS potential data. With the resulting set of complex numbers, we generated three metrics for both

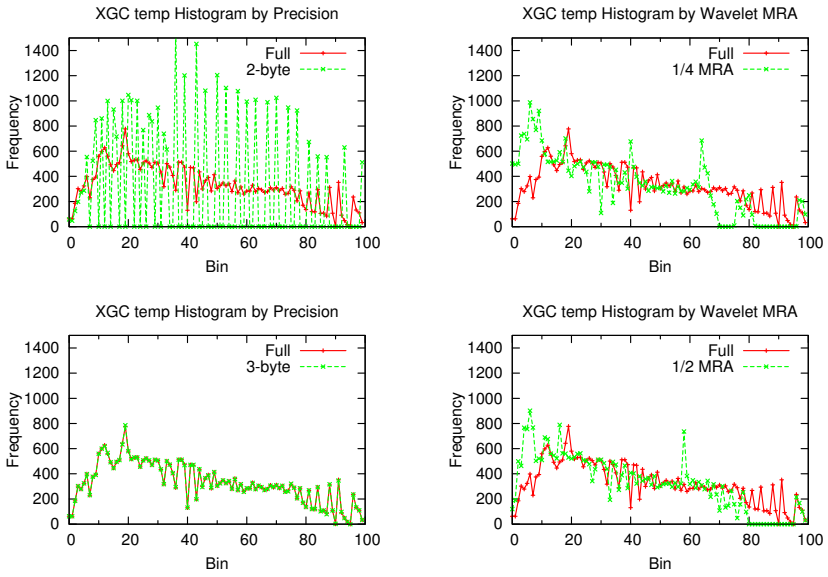


Fig. 8: XGC-1 100-bin histograms.

APLOD and wavelet MRA. First, we organize the data by drift wave number and calculate the mean/median relative error vs. the full-precision-generated FFT data, to examine if there is a distributional relation between the errors. Second, we plot the absolute values of the full-precision FFT coefficients against the corresponding error seen to examine whether there is a relation between exponent value and error. Figures 9 and 10 show these metrics for the wavelet-generated and APLOD data, respectively, while Table VI gives a numerical distribution of the errors. The real component is shown for these metrics; in comparison, the complex component shows similar trends and the magnitude of the complex numbers shows order-of-magnitude improvements in accuracy.

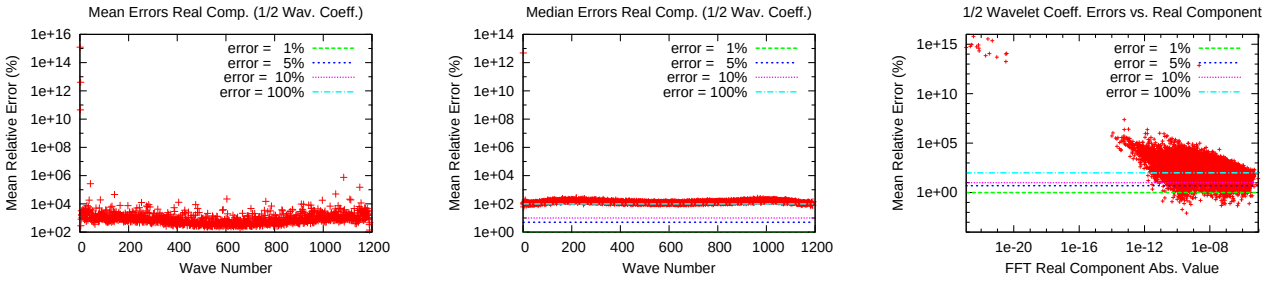
Based on these error measurements, we observe two trends. First, the wavelet-generated data, even when using one-half of the wavelet coefficients, shows unacceptable error in all metrics. As noted in the previous sections, we believe this to be the result of outliers skewing the results across every FFT component. Second, errors seen for the APLOD data are much higher for the lowest precision representation than in the previous metrics. The combination of each point in FFT space being a linear combination of all others and the huge differences in the exponent component leads to the propogation of errors. While the largest of outliers are persistent for multiple bytes of precision (where the real and complex component are in the 10^{-20} range), the remaining data approaches acceptable error at four bytes of precision. For this particular application, approximate measures based on aggregate values at wave numbers may be able to use a lesser degree of precision (three bytes).

TABLE VI: Distribution of relative errors for real, complex, and magnitude components of FFT data generated from the GTS phi data. Total number of points is 191751. *A* refers to APLOD, *W* refers to wavelets, and the number refers to the proportion of the full dataset used (as a fraction of eight).

Comp.	Rel. Err. (%)	A-2	A-3	A-4	W-4
Real	$x \leq 1$	27912	177681	191558	752
	$1 < x \leq 5$	36193	10235	92	3041
	$5 < x \leq 10$	22306	1688	10	3632
	$10 < x \leq 100$	77202	1789	10	66114
	$100 \leq x$	28138	358	81	118212
Complex	$x \leq 1$	32420	179648	191547	719
	$1 < x \leq 5$	43955	8887	88	2969
	$5 < x \leq 10$	23652	1322	14	3802
	$10 < x \leq 100$	67067	1525	8	65491
	$100 \leq x$	24657	369	94	118770
Magnitude	$x \leq 1$	52841	190283	191751	1445
	$1 < x \leq 5$	60856	1410	0	6086
	$5 < x \leq 10$	26606	44	0	7534
	$10 < x \leq 100$	46830	12	0	122837
	$100 \leq x$	4618	2	0	53849

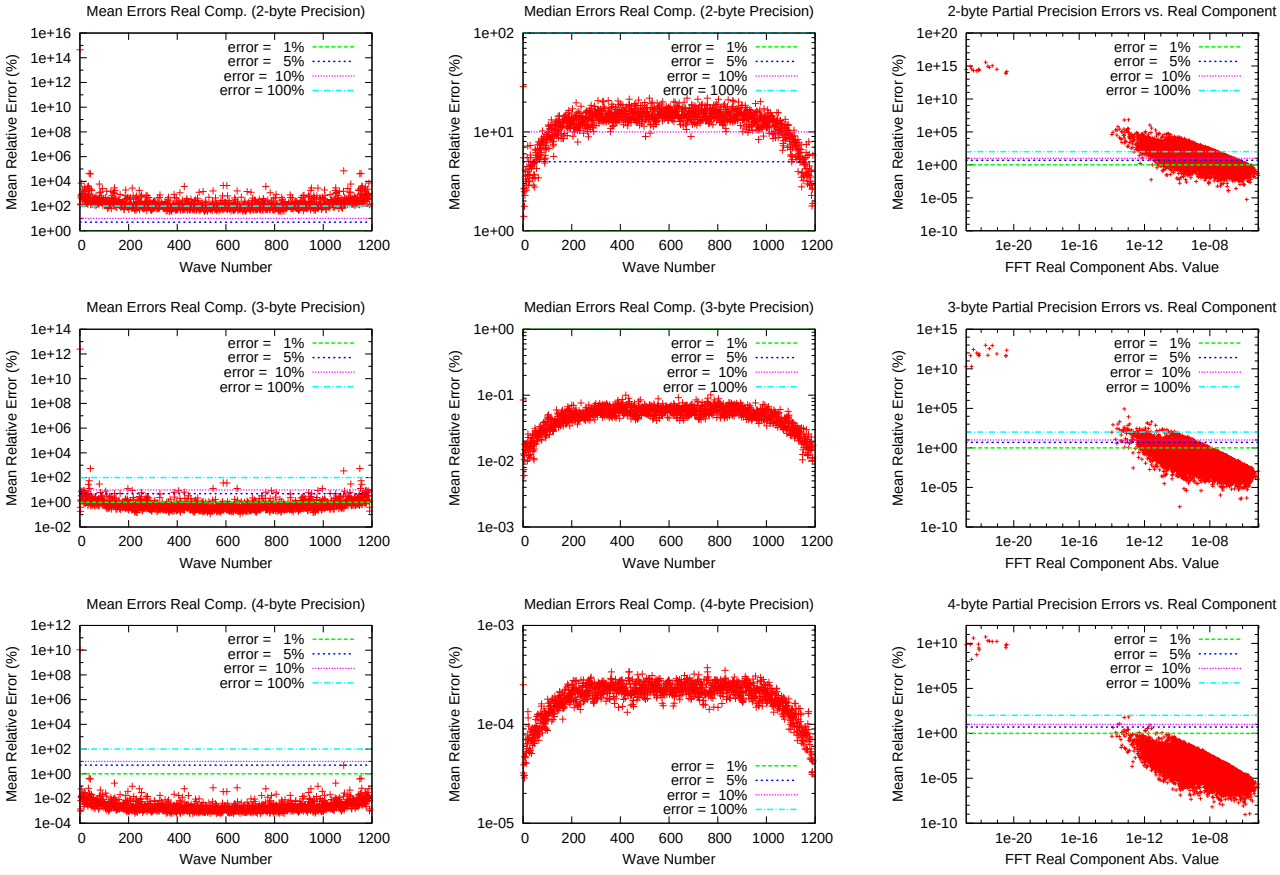
V. Conclusion

Extreme-scale scientific applications follow the write-once, read-many paradigm, following a cycle of production-level simulation runs followed by analysis by multiple scientists. Rather than focusing on optimizing the writing of simulation data or reducing it through compression, which may disrupt I/O patterns, we instead chose to reorganize the data to optimize read-level performance by trading off variable data precision for data reduction. Through our method of APLOD processing, this functionality is provided in a simple and low-overhead manner. Perhaps most importantly, it avoids the primary problems of data reduction through compression by providing a globally deterministic partitioning of data that respects existing data layouts and avoids non-uniform buffer sizes. We believe our approach is a low-barrier, high-applicability solution for applications that wish to speed up their analysis processes without sacrificing performance with



a *b* *c*

Fig. 9: For the D4 wavelet, *a,b* - Mean, median errors of FFT data along each drift wave (real component), and *c* - FFT errors plotted against real component value.



a *b* *c*

Fig. 10: For varying APLOD precisions, *a,b* - Mean, median errors of FFT data along each drift wave (real component), and *c* - FFT errors plotted against real component value.

respect to the original data layout or spend significant effort redesigning I/O and data layout methodologies. Our claim to applicability is strengthened by the use of ADIOS, a widely-used I/O library in the extreme-scale sciences.

ACKNOWLEDGMENTS

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] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*. New York, NY, USA: ACM, 2009, pp. 40:1–40:12.

[2] 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 IEEE International Conference on Data Engineering (ICDE)*, 2012, pp. 138–149.

[3] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2009.

[4] A. Trott, R. Moorhead, and J. McGinley, "Wavelets applied to lossless compression and progressive transmission of floating point data in 3-D curvilinear grids," in *Proceedings of the Conference on Visualization*, 1996, pp. 385–388.

[5] D. Taubman and M. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

[6] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens, "Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision," in *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2011, pp. 31–38.

[7] E. R. Schendel, S. V. Pendse, J. Jenkins, D. A. Boyuka, Z. Gong, S. Lakshminarasimhan, Q. Liu, H. Kolla, J. Chen, S. Klasky, R. Ross, and N. F. Samatova, "ISOBAR hybrid compression-I/O interleaving for large-scale parallel I/O optimization," in *Proceedings of the ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2012, pp. 61–72.

[8] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*. ACM, 2008, pp. 15–24.

[9] W. Wang, Z. Lin, W. Tang, W. Lee, S. Ethier, J. Lewandowski, G. Rewoldt, T. Hahm, and J. Manickam, "Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments," *Physics of Plasmas*, vol. 13, no. 092505, 2006.

[10] J. Chen, A. Choudhary, B. Supinski, M. DeVries, E. Hawkes, S. Klasky, W. Liao, K. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. Yoo, "Terascale direct numerical simulations of turbulent combustion using S3D," *Computational Science & Discovery*, vol. 2, no. 015001, 2009.

[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*, vol. 49, no. 115021, 2009.

[12] F. Olken and D. Rotem, "Simple random sampling from relational databases," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 160–169.

[13] F. Olken, "Random sampling from databases," Ph.D. dissertation, University of California, 1993.

[14] F. Olken and D. Rotem, "Random sampling from B+ trees," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 269–277.

[15] F. Olken and D. Rotem, "Sampling from spatial databases," in *Proceedings of the International Conference on Data Engineering (ICDE)*, 1993, pp. 199–208.

[16] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya, "Overcoming limitations of sampling for aggregation queries," in *Proceedings of the International Conference on Data Engineering (ICDE)*, 2001, pp. 534–542.

[17] P. J. Haas and C. König, "A bi-level bernoulli scheme for database sampling," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2004, pp. 275–286.

[18] V. Pascucci and R. J. Frank, "Global static indexing for real-time exploration of very large regular grids," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*. New York, NY, USA: ACM, 2001, p. 2.

[19] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl, "Level-of-detail volume rendering via 3D textures," in *Proceedings of the IEEE Symposium on Volume Visualization*. New York, NY, USA: ACM, 2000, pp. 7–13.

[20] E. LaMar, B. Hamann, and K. I. Joy, "Multiresolution techniques for interactive texture-based volume visualization," in *Proceedings of the Conference on Visualization*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1999, pp. 355–361.

[21] M. Frazier, *An Introduction to Wavelets through Linear Algebra*. Springer-Verlag, 1999.

[22] "IEEE standard for floating-point arithmetic," *IEEE Standard 754-2008*, 2008.

[23] MPI Forum, "MPI-2: Extensions to the message-passing interface," Univ. of Tennessee, Knoxville, Tech. Rep., 1996.

[24] M. Yang, R. E. McGrath, and M. Folk, "HDF5 - a high performance data format for earth science," in *Proceedings of the International Conference on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, 2005.

[25] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A high-performance scientific I/O interface," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*. New York, NY, USA: ACM, 2003, p. 39.

[26] M. Galassi *et al.*, *GNU Scientific Library Reference Manual*, 3rd ed., ISBN:0954612078.

[27] R. Narayanan, B. Özışkyılmaz, G. Memik, A. Choudhary, and J. Zambreno, "Quantization error and accuracy-performance tradeoffs for embedded data mining workloads," in *Proceedings of the International Conference on Computational Science (ICCS)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 734–741.