An Adaptive Prediction-based Approach to Lossless Compression of Floating-point Volume Data

Nathaniel Fout and Kwan-liu Ma, *Fellow, IEEE*

Abstract— In this work, we address the problem of lossless compression of scientific and medical floating-point volume data. We propose two prediction-based compression methods that share a common framework, which consists of a switched prediction scheme wherein the best predictor out of a preset group of linear predictors is selected. Such a scheme is able to adapt to different datasets as well as to varying statistics within the data. The first method, called APE (Adaptive Polynomial Encoder), uses a family of structured interpolating polynomials for prediction, while the second method, which we refer to as ACE (Adaptive Combined Encoder), combines predictors from previous work with the polynomial predictors to yield a more flexible, powerful encoder that is able to effectively decorrelate a wide range of data. In addition, in order to facilitate efficient visualization of compressed data, our scheme provides an option to partition floating-point values in such a way as to provide a progressive representation. We compare our two compressors to existing state-of-the-art lossless floating-point compressors for scientific data, with our data suite including both computer simulations and observational measurements. The results demonstrate that our polynomial predictor, APE, is comparable to previous approaches in terms of speed but achieves better compression rates on average. ACE, our combined predictor, while somewhat slower, is able to achieve the best compression rate on all datasets, with significantly better rates on most of the datasets.

Index Terms—Volume compression, Lossless compression, Floating-point compression.

1 Introduction

In the scientific visualization community, floating-point data resulting from observational measurements or computer simulations is being generated at an ever-increasing rate. In particular, large volumetric datasets with sizes on the order of terabytes or even petabytes are now commonplace, and such datasets consume massive resources in terms of both bandwidth and storage. Such datasets are typically high-dimensional, being high-precision, high-resolution, multi-variate, and time-varying with thousands of time steps. Consider a 1000 time-step data output from a state-of-the-art direct numerical simulation, where each time step is a 1024^3 volume consisting of five variables stored in floating-point format. In this case, each time step consumes 20 gigabytes, with the entire dataset requiring 20 terabytes. Difficulties related to dataset size inundate the entire data processing pipeline, including generation, transfer, storage, and analysis. Analysis often consists of volume visualization, but most real-time volume visualization is accomplished using graphics hardware, which has limited memory. Even main memory is insufficient for many datasets, with one time step often exceeding the available RAM on current machines.

Similarly, in the medical imaging community massive amounts of volumetric data are produced by CT, MRI, PET and other scanning modalities. There may be multiple studies for each patient, with many thousands of patients in a typical large hospital. The resolution of the scanners continues to improve, as well as the precision of the measurements. This results in many petabytes of data, which must be acquired and archived indefinitely. Furthermore, this data must be immediately available for evaluation by health professionals; such evaluation typically consists of slice browsing, although 3D volume visualization is becoming more popular in certain applications. Thus in medical imaging, archiving of such massive amounts of data and fast transfer to local workstations are critical components of this data management context.

An effective solution to such large data management problems is to reduce the size of the data using compression techniques. However, most data compression methods target integer data of limited dynamic range, with only a few works focusing on floating-point formats. Floating-point compression is more challenging than integer compression, as the least significant bits of the mantissa tend to be very poorly correlated. This is less of an issue for lossy compression, but makes lossless compression significantly more difficult. Although lossy compression is acceptable in many applications, the effort and expense involved in the acquisition of scientific data usually warrants lossless compression. Likewise, for both practical and legal reasons, medical imaging data is almost always compressed losslessly. For visualization, however, a lossy version may be acceptable, so that the optimal configuration would be a lossy-to-lossless compressed format.

Our solution to this problem is to use an adaptive prediction-based lossless compression scheme, similar to those used in lossless audio compression. This is accomplished by using a switched predictor, in which the best predictor out of a small set of candidate predictors is selected for encoding at any given time. Such an approach is able to adapt to varying data statistics by selecting the predictor that best models the local characteristics of the data. Furthermore, such a scheme is very extensible, in that it is able to incorporate existing predictors as well as predictors developed in the future without modifying the basic framework. While our primary goal is to improve the compression rate, we also describe a unique feature of our system that supports progressive analysis of the compressed data, called Progressive Precision. Progressive Precision allows up to three levels of data access, thereby providing a lossy, lower-resolution option appropriate for visualization preview. To summarize, we believe the salient contributions of our work to be:

- An adaptive switched prediction framework for lossless floating-point volume compression, allowing consolidation of existing prediction methods into a single highly-adaptive compressor.
- A fast method for entropy coding of residual leading zeros using a rank table and universal codes.
- A method providing progressive transmission, thereby supporting efficient visualization.

We apply our approach to several large volume datasets, demonstrating lossless compression rates consistently in the 30-50% range. Our approach is able to provide better rates than existing approaches, while offering similar compression/decompression times. This allows fast, efficient compression of volume data at acquisition time, so that the benefit of compression is conferred to data transfer, storage, and

- *Nathaniel Fout is with UC Davis, E-mail: natefout@gmail.com.*
- *Kwan-liu Ma is with UC Davis, E-mail: klma@ucdavis.edu.*

Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

subsequent visualization. Our novel progressive scheme is able to provide lower-precision, lossy representations of the datasets, thereby facilitating real-time visualization.

2 Related Work

Many approaches for scientific data compression focus primarily on combining compression with data synthesis in order to increase throughput and conserve storage. Engelson et al. [5] compress sequences of double-precision floating-point values resulting from simulations based on ordinary differential equations. In their approach, the numbers are treated as integers and then compressed using predictive coding, with residuals being explicitly stored in the case of lossless coding or truncated for lossy coding. Ratanaworabhan et al. [16] propose a lossless prediction-based compression method for double-precision floating-point scientific data using a DFCM (Differential Finite Context Method) value predictor, which is based on pattern matching using a hash table holding the recent encoding context. The bitwise residual is then computed using the XOR operator, with the compressed representation consisting of the number of leading zeroes and the remaining residual bits. Lindstrom and Isenburg [13] also focus on the goal of fast throughput for online encoding, achieving lossless or lossy compression of floating-point sequences by prediction and entropy coding of residuals. A Lorenzo predictor is used with computation of the predicted value by floating-point arithmetic, with the option of bit-wise integer arithmetic. The residual is computed by transforming the actual and predicted values to unsigned integers and taking the integer difference, which is subsequently encoded using a two-level scheme based on fast entropy coding of the sign and leading zero bits, with explicit transmission of the remaining residual bits. Burtscher and Ratanaworabhan [1] [2] propose a lossless compression method for double-precision scientific data with the goal of achieving fast encoding/decoding for high throughput environments. This method employs two prediction methods, FCM (Finite Context Method) and DFCM, with the method producing the smallest residual being chosen. The difference is computed using XOR and the number of leading zero bytes is recorded along with the nonzero residual. The authors compare their method with previous approaches and show competitive overall compression ratios with much higher throughput. Xie and Qin [30] describe a method for compression of seismic floating-point data wherein a differential predictor is used, followed by separate context-based arithmetic coding of each of the sign, exponent, and mantissa fields of the residual, where the exponent switches between contexts. Tomari et al. [23] propose a simple lossless scheme for double-precision scientific data designed with fast hardware decompression in mind. They compress only the exponent by keeping a small table of recent exponents into which they index.

On the other hand, some researchers attempt to compress in a way that facilitates post-processing and analysis, especially visualization. Tao and Moorhead [21] [22] compress scientific floating-point data by using a biorthogonal wavelet transform followed by entropy coding, with the goal of providing progressive transmission. Trott et al. [24] [25] compress scientific floating-point data in curvilinear grids by using a Haar wavelet transform followed by entropy coding. Lossless coding is achieved by converting the single-precision floating-point numbers to double precision prior to the transform, with the double-precision coefficients being compressed directly via byte-wise application of Huffman coding. Du and Moorhead [4] describe a similar approach using Haar or first-order B-spline wavelet transforms preceded by conversion to double precision. The double-precision coefficients are encoded by separate entropy coding of the exponent and mantissa, with run-length encoding of the least significant mantissa bits. Adaptive coarsening [19] [20] is an alternative method for lossy compression in which the data is sub-sampled according to a permissible level of error.

Most high-quality audio coding approaches are built around a PCM (pulse code modulation) lossless coding scheme. Ghido [7] proposes a method for lossless compression of single-precision audio data by defining a transform from floating-point to integer numbers that is portable and renders the integers amenable to efficient compression. The transform yields an auxiliary stream with encoding parameters and an integer stream, which is then compressed using a PCM lossless entropy coder. Yang et al. [31] encode single-precision audio data by truncating the floating-point numbers to integers and computing the bit-wise residual between the truncated integer version of the number and the original version. The integers are encoded using an established PCM lossless compression method, whereas the floating-point residuals are compressed byte-wise using the lossless coder gzip, excluding zero bytes. Liebchen et al. [12] describe compression of single-precision floating-point audio data in the MPEG-4 ALS specification. Floating-point values are decomposed into a truncated integer component, which is encoded using the prediction-based integer ALS scheme, a common multiplier, which is related to the local dynamic range of the signal, and a floating-point residual, which is compressed using a masked Lempel-Ziv dictionary coder.

The problem of floating-point compression also arises in image, texture, and geometry encoding. Most floating-point image methods are based on JPEG-2000. Lossless variants are described by several authors [6] [26] [27] [28] [29]. Most floating-point texture compression methods are based on block truncation coding and are therefore lossy, being designed for hardware application. In the area of geometry compression most methods are lossy, but some lossless methods based on predictive coding have been described [11] [32] [33].

3 Overview

The primary source of floating-point volume data is scientific simulations, in which simulation code runs on massively parallel supercomputers in an attempt to model some natural or theoretical phenomenon. To begin the simulation, an initialization of the volume is defined, and then the code iteratively modifies the volume as the simulation progresses in order to explore the dynamic behavior of the system. Each iteration defines a time step in the simulation, and these iterations are of great interest to the scientists. A simulation can run for many thousands of time steps, yielding thousands of volumes for subsequent processing and analysis. Managing such large amounts of data continues to be a challenge for simulation centers.

In order to meet this challenge, there has been increasing interest in compression strategies that ameliorate this problem. One such strategy is called in-situ compression, in which the data is compressed in conjunction with the simulation, as it proceeds. This approach is advantageous, because by reducing volume size early savings are gained in transfer of data to storage, storage requirements, and transfer of data to clients for data analysis. The cost of this approach is additional processing time up front to compress the data, as well as any additional cost to decompress the data on the client side. Therefore, in-situ compression methods should ideally maximize the compression rate with the constraint of modest encoding/decoding complexity. Modest decoding complexity allows better integration of decoding with volume visualization.

In the following sections, we present a lossless compression method for floating-point volume data that meets these requirements. In particular, we describe a method that offers fast and efficient coding, thereby allowing in-situ compression. Decoding is even faster, which allows this approach to be integrated with visualization.

4 Predictive Coding Framework

Only a few approaches have been developed for lossless floating-point compression of volume data, most of which are based on predictive coding using various source models. We believe that in certain aspects the challenges of floating-point compression more closely resemble those of audio compression; in particular, audio data formats typically accommodate high dynamic ranges (e.g. 16-bit integer) and require efficient compression/decompression. Based on this observation, we attempt to combine ideas from audio and image coding in order to address the unique challenges of floating-point compression.

The basic approach to lossless compression of correlated data is to first decorrelate the data and then apply entropy coding. There are two general approaches to decorrelation: prediction-based encoding and transform coding (most notably wavelet transform coding).

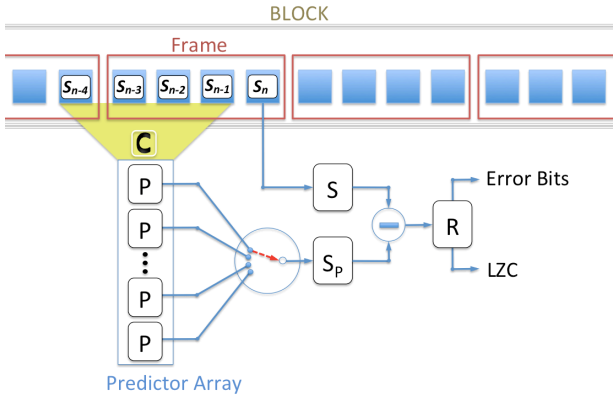
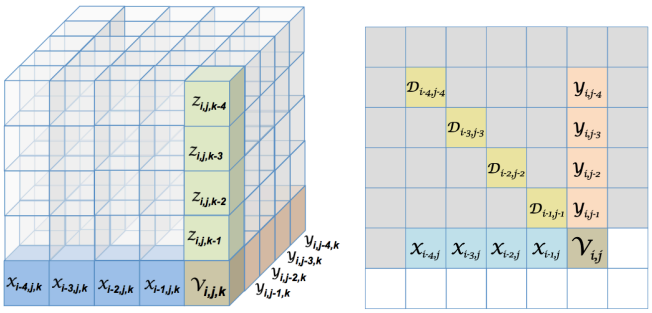


Fig. 1. Datasets are encoded block by block, with each block further subdivided into many frames. Our proposed switched prediction scheme selects the best prediction from a small set of predictors P with given context C for each frame and encodes the residual, which is the difference between the predicted value S_P and the actual value S. The residual R is divided into the Leading Zero Count (LZC), which is further encoded, and some number of error bits, which are transmitted unencoded.



(a) Axis-aligned prediction contexts (b) Diagonal prediction context

Fig. 2. Our prediction context consists of three sets of linear contexts in the X, Y, and Z orientations (a), as well as a diagonal predictor in the X-Y plane (b). Interpolating polynomials of orders one through four in each of these contexts yield a total of 16 predictions of V, the voxel to be encoded. The contexts used by each predictor configuration are given in Table 1.

While wavelet coding is preferred for lossy compression, prediction-based methods generally achieve better rates for lossless compression. Furthermore, prediction-based methods are fast, efficient, and require only one pass over the data. For these reasons, prediction-based methods are popular for both lossless image and audio compression, and form the basis of our proposed solution as well.

A prediction-based compression scheme uses a subset of previously encoded values, called the context, to attempt to predict the next value to be encoded. Then the difference between the actual value and its prediction, called the residual, is encoded instead of the value. If the prediction is accurate then the difference values will be decorrelated and small in magnitude, thereby allowing efficient compression with entropy coding. Central to this approach is the definition of the context and how to compute the prediction. Typically, the prediction calculation is based on knowledge of the structure of the data and is a linear combination of a few previously encoded values. Given a finite context, it is possible to compute the optimal linear coefficients in terms of minimum prediction error; however, in practice this is prohibitively expensive. An alternative is to use a set of coefficients, or equivalently a set of linear predictors, in order to approximate an optimal predictor. If one predictor out of the set is chosen to make the prediction then this is called a switched prediction scheme, and this arrangement forms the basis of our prediction framework. Such a scheme is able to adapt both to different datasets and to varying statistics within the data by choosing predictors that best model the local behavior of the data. This allows more effective and robust decorrelation, thereby yielding good compression rates over a wide range of data.

Our approach operates as follows. We encode the data in blocks consisting of several thousand values (8K in our system), within which we subdivide the data into small frames consisting of only a few values, as shown in Figure 1. Blocks are encoded independently to allow efficient access to subsets of the data. Blocks are processed frame by frame, and within each frame a prediction is computed for each value by each of the predictors based on its respective context. The total prediction error is computed for each prediction method and the predictor with the minimum error is selected to encode the frame. The predictor selection for each frame is sent to a dedicated entropy coder. The added complexity involved in the selection scheme is justified as producing a good prediction is of paramount importance, since all incorrectly predicted bits are transmitted uncompressed.

Once a prediction is generated, the residual is computed. If the prediction is accurate then the predicted and actual values should be close, which allows a residual to be computed that consists of a string of leading zeros followed by a set of significant bits representing the error in the prediction. The leading zero count can be efficiently entropy coded, whereas the error bits are sent verbatim to allow lossless reconstruction.

4.1 Polynomial Predictors

Our basic scheme, which we refer to as APE (Adaptive Polynomial Encoder), uses a set of interpolating polynomial predictors with integer coefficients, as first proposed for audio coding by Robinson [17] and adopted by Hans and Shafer in their AudioPaK coder [8]. A unique $(k-1)$-order polynomial can be constructed from the last k data points and subsequently used to predict the next value, x_n. We have found that the most useful polynomials for encoding floating-point data are orders zero through three, which give the following predictions:

$$P_1(n) = x_{n-1} \quad (1)$$
$$P_2(n) = 2x_{n-1} - x_{n-2} \quad (2)$$
$$P_3(n) = 3x_{n-1} - 3x_{n-2} + x_{n-3} \quad (3)$$
$$P_4(n) = 4x_{n-1} - 6x_{n-2} + 4x_{n-3} - x_{n-4} \quad (4)$$

where $P_i(n)$ is the $(i-1)$th-order prediction of x_n. The prediction residuals for each polynomial predictor can be computed directly using only subtractions based on the following recursive relationship:

$$R_0(n) = x_n \quad (5)$$
$$R_1(n) = R_0(n) - R_0(n-1) \quad (6)$$
$$R_2(n) = R_1(n) - R_1(n-1) \quad (7)$$
$$R_3(n) = R_2(n) - R_2(n-1) \quad (8)$$
$$R_4(n) = R_3(n) - R_3(n-1) \quad (9)$$

We can also use these equations as a more efficient way to determine and compute the best prediction. These four linear polynomial predictors of orders zero through three form the basis of our APE predictor. We extend this 1D approach to 3D by computing parallel predictions in the three cardinal directions, as shown in Figure 2. This results in 12 predictors (4 X-oriented, 4 Y-oriented, and 4 Z-oriented), which requires a four-bit binary code. To take advantage of the full range of the four bits (which selects for 16 predictors), we also use an $X-Y$ diagonal context. This often complements the X and Y predictors well, especially when the inter-plane distance in Z is larger than the resolution in the $X-Y$ plane. We could potentially add more predictors,

requiring additional encoding bits, but this would require more prediction calculations, which would impact the encoding efficiency.

In the case of the X-oriented predictions only, wherein the contexts overlap from one step to the next, we can compute the residuals even more efficiently by reusing residuals from the previous prediction:

$$R'_k(n) = \begin{cases} R_{k-1}(n) - R_{k-1}(n-1) & n = 1 \\ R_k(n-1) & otherwise \end{cases} \quad (10)$$

where R' denotes the new residuals to be computed. This amounts to one subtraction per residual.

Because the prediction context includes the previous four voxels in the Z direction, the memory footprint for encoding must potentially include five slices of the volume. Since our block size is smaller than the size of a slice, blocks will depend on previously-encoded blocks for their Z context, and therefore cannot be coded independently. However, if we consider simulations in which a distributed processing model is used so that each node processes a small subvolume, then we can independently encode blocks as long as the slices of the subvolume are sufficiently small.

In order to support lossless reconstruction, we must be careful that the predictions at the decoder exactly match the predictions made by the encoder. This can be accomplished in several ways. The most direct way, which we adopt in our system, is to use floating-point operations (of the same or greater precision) while using the same platform for the encoder and decoder. Even this is problematic if truly lossless compression is important, as differences in compilers, compiler options, runtime environments, and several other factors may affect floating-point operations. Therefore, the most robust approach is to base the prediction on integer operations only. One integer-based method is to map the values to integers and carry out the operations using integer arithmetic, with an associated reduction in prediction accuracy. A second integer-based method is to use a software implementation of floating-point arithmetic such as SoftFloat [9] that relies on integer operations, which preserves prediction accuracy but is significantly more expensive.

It is possible to implicitly select the predictor, for instance by taking the median prediction or by examining the nearby context; however, we explicitly select the predictor with the best prediction (i.e. the smallest residual) and send the selection as auxiliary information. This choice is based on our observation that the overhead for transmitting the predictor selection is more than compensated by the better prediction accuracy. As we have 16 predictors, binary coding of the selection requires four bits. For our encoder we use frames of size eight, so that the cost of selection is 0.5 bpf (bits per float), since one predictor is chosen to encode the entire frame. For reference, Table 1 lists our encoder configurations based on the constituent predictors. The predictor selection tends to be highly skewed, with some predictors being selected much more frequently than others, allowing for further reduction in cost by entropy coding. In our system, within each block we record the predictor selection using binary codes while simultaneously collecting the predictor pmf (probability mass function). Once we reach the end of the block we compute a Huffman table based on the pmf and encode the selections, which have been recorded in the binary-coded buffer. This buffer serves a dual purpose. In the case of ineffectual Huffman coding, which might result from a more uniform selection pmf, the binary-coded buffer is transmitted instead, thereby guaranteeing at most 0.5 bpf.

4.2 Combined Predictors

Our second scheme combines the polynomial predictors with other types of predictors in order to diversify our framework, and is thus referred to as ACE (Adaptive Combined Encoder). Incorporating other predictors provides additional data models which may aid in achieving effective decorrelation. This not only allows us to build on previous work, but provides a convenient mechanism for extension should better predictors be developed in the future. Based on an analysis of previous work in floating-point prediction, we chose two predictors to incorporate in our system. The first is the Lorenzo predictor, which is a generalization of the parallelogram predictor, as introduced by Ibarria et al. [10]. The second is the FCM/DFCM hash-based predictor pair introduced by Burtscher and Ratanaworabhan [1], which relies on locally repeating patterns in the data. Additionally, we compute a mean prediction as the average of all the constituent predictors. Thus we add a total of four additional predictors: Lorenzo, FCM, DFCM, and Mean.

Scheme	*Predictors*
APE	$P_1(x), P_2(x), P_3(x), P_4(x)$ $P_1(y), P_2(y), P_3(y), P_4(y)$ $P_1(d), P_2(d), P_3(d), P_4(d)$ $P_1(z), P_2(z), P_3(z), P_4(z)$
ACE	$P_1(x), P_2(x), P_3(x), P_4(x)$ $P_1(y), P_2(y), P_3(y), P_4(y)$ $P_1(z), P_2(z), P_3(z), P_4(z)$ Lorenzo, FCM, DFCM, Mean

Table 1. Configurations for the APE and ACE encoders. For each frame the best predictor from the given set is selected. The frame size is set so that the maximum rate for specifying the predictor selection is 0.5 bpf (bits/float).

The ACE predictor configurations are listed in Table 1, with relevant contexts for the polynomial predictors shown in Figure 2. The ACE scheme includes the X, Y, and Z polynomial predictor sets, and the four additional predictors, Lorenzo, FCM, DFCM, and Mean. We use a frame size of eight in order to ensure a worst-case rate of 0.5 bpf. Inspection of the predictor selection histograms for ACE demonstrates that each predictor is selected a variable number of times, depending on the characteristics of the dataset. Again, in most cases the histogram is highly skewed, allowing efficient entropy coding. The strength of our approach is manifest, in that our encoder adapts to each dataset by selecting the predictors that best model the data. Although there is some amount of overhead in calculating several predictions and in the selection process, we show that this modest increase in complexity is offset by significantly better compression rates.

5 Prediction Residual Encoding

Prediction residuals for integer data are usually entropy coded based on a Laplacian model of the error signal. However, this approach is not pragmatic for floating-point residuals, for several reasons. First, consider the pmf of the residuals for a typical dataset, as shown in Figure 3. The pmf shows the probability of occurrence for each possible residual, which aids in the development of a statistical model for encoding. Although from a distance the pmf appears to be Laplacian, when we inspect the distribution more closely we discover that there are discontinuities due to the sparse population of possible values, of which there are over four billion. This structure will be difficult to model without assigning codes to residuals that do not occur. Furthermore, as pointed out by Lindstrom and Isenburg, there are many possible residuals compared to the cardinality of the data to be compressed, so that entropy coding will be inefficient and likely ineffectual. An alternative approach, first proposed by Sayood and Anderson [18] and used in variant forms by both Burtscher and Ratanaworabhan [1] and Lindstrom and Isenburg [13], is to use entropy coding for the number of leading zeros in the residual while simply transmitting the remaining error bits verbatim. Our framework also uses this method, but with a novel fast entropy coding scheme for the leading zero count.

5.1 Computation of the Prediction Residual

There are two efficient methods of computing the residual using only integer operations. The first approach, used by Burtscher and Ratanaworabhan [1], is to simply XOR the two values, since floating-point numbers in close proximity will often have identical bit patterns in their most significant bits. The second approach, used by Lindstrom and Isenburg [13], is to map the floating-point numbers into unsigned integers and compute the absolute integer difference, while keeping track of the residual sign explicitly. Regardless of the method, the end

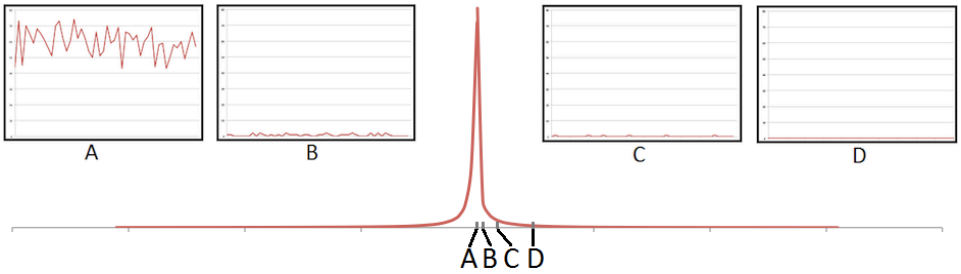


Fig. 3. Bottom: The pmf of the prediction residual for a typical floating-point dataset appears to be Laplacian. Top: Magnification demonstrates how sparsely the pmf is populated; these discontinuities in the pmf, along with the exceptionally large number of possible residuals compared to actual residuals, precludes conventional entropy coding of the residual.

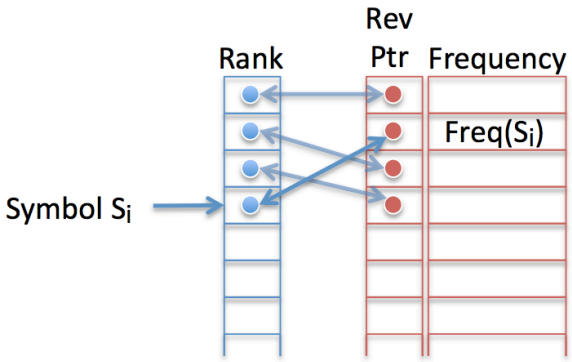


Fig. 4. A rank table keeps a sorted list of symbol frequencies. As symbols are encountered, the frequency array is updated by moving entries up or down in order to maintain a decreasing frequency order. As entries are moved, the pointers to and from the rank array are modified accordingly. This data structure allows the use of Universal codes for fast entropy coding of the LZC' symbols.

result is a residual which consists of some number of leading zeros, followed by a number of significant error bits. The error bits are transmitted uncompressed, so in general the more leading zeros we can produce the better the compression rate. The XOR method of computing the residual is faster than the second method (which we call UINT/SUB), but produces fewer leading zeros when the values are close but separated by an exponent boundary. We offer both methods in our framework, but the default is UINT/SUB. One key observation to make is that the most significant bit of the residual error bits is always 1, and therefore does not need to be transmitted. This small modification improves the compression rate by 1 bpf.

5.2 Entropy Coding of Leading Zero Count

For single-precision floats, the LZC (Leading Zero Count) can range from 0 to 32. In order to more efficiently represent this quantity we compact it into 32 values by the following mapping:

$$LZC' = \begin{cases} 0 & LZC = 0,1 \\ LZC - 1 & otherwise \end{cases} \tag{11}$$

which maps a LZC of 0 and 1 to the same value. This means that for both $LZC = 0$ and $LZC = 1$, 32 bits will need to be transmitted, since the value of the first bit is unknown. Since values of 0 and 1 tend to occur very infrequently, this has negligible effect on the encoding process. LZC', in the range $[0,31]$, can be encoded using a binary code of 5 bits.

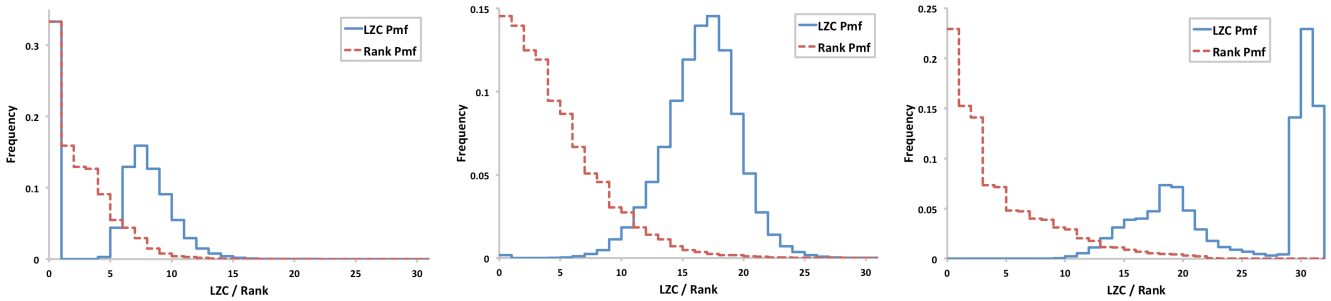
Examining the histogram of LZC' in some typical datasets (see Figure 5) demonstrates that further reduction is possible with entropy coding. In our system we offer adaptive Huffman coding for this purpose, as well as range coding, the arithmetic coding variant described by Lindstrom and Isenburg [13]. In addition, we introduce a third option that is faster than either of these and achieves nearly the same rate. This method uses Universal codes with an associated rank table. Universal codes are variable-length, self-delimiting integer codes that encode a set of integers where smaller values are more likely than larger ones. Codes for the integers can be computed on-the-fly or precomputed and stored if the range of possible values is given a priori, as in the case of the LZC' integers. Using precomputed codes is extremely fast, since only a single table look-up from a very small table is required. The problem with using Universal codes for encoding the LZC' is that the assumption of lower values having a greater probability of occurrence does not hold in general; in fact, this assumption only holds for early-peaking LZC' pmfs, and even then not perfectly.

In order to address this problem, we propose the use of the rank table in conjunction with Universal codes. A rank table is a data structure that is used in some adaptive Huffman coders [15] in order to determine when a revised Huffman table should be recomputed. The basic structure of a rank table is shown in Figure 4. It consists of two arrays, one for the symbol ranks, which is indexed by the symbols, and another for the symbol frequencies. The frequency array keeps a sorted list of the symbol frequencies, along with a reverse pointer which indicates to which symbol each frequency belongs. The symbol rank table is indexed by the symbol value and contains a forward pointer to the corresponding frequency in the frequency array. The rank table operates by updating the symbol frequencies as each symbol is encountered and maintaining them in sorted order by manipulating table pointers.

We can use the same structure as a way to transform the symbols into another set of sorted symbols (i.e. the rank symbols) in which the probabilities are strictly non-increasing, thereby allowing proper use of Universal codes; that is, we encode the *rank* of the LZC' instead of the LZC' itself. As long as the decoder constructs and uses the same rank table then we can unambiguously decode the rank sequence and use the reverse pointer to obtain the LZC' for each rank. Figure 5 shows how each type of LZC' pmf is transformed to a corresponding rank pmf having the necessary Universal coding property. The only remaining issue to resolve is the choice of Universal code. There are several types of Universal codes, of which we tested the Gamma code, the FK1 code, Rice codes, Golomb codes, and Split-sample codes. Rice codes, Golomb codes, and Split-sample codes are more flexible, as they each have a parameter that can be optimized in order to tailor the codes to a particular sequence. Based on our experiments, Golomb codes with the parameter adaptively determined give the best overall coding rate.

6 Adaptive Progressive Precision

The least significant bits of floating-point numbers in scientific datasets often appear to be statistically random. Although it is impossible to determine whether they are truly random or not, we do know that for many datasets some of the least significant bits of the mantissa represent noise resulting from data acquisition (random variation in measurements or model parameters) or from data processing (e.g. quantization noise from floating-point computation/storage). As



(a) Early-peaking *LZC* pmf. (b) Mid-peaking *LZC* pmf. (c) Late-peaking *LZC* pmf.

Fig. 5. In practice we encounter three types of *LZC* pmfs (probability mass functions): (a) early-peaking, (b) mid-peaking, and (c) late-peaking. By using a rank table structure, all three types are converted into a non-increasing pmf appropriate for Universal codes.

a result of the apparent randomness in these last bits, lossless compression of floating-point values is constrained, and we often end up explicitly storing the least significant bits of the residual for all values. However, as these bits may not contain useful information at all, it would be nice to offer a more compact representation of the data in which these bits are initially omitted, in order to facilitate data analysis and/or visualization. We describe such an approach, which we call Progressive Precision (PP). PP provides a lossy preview option for our lossless compression algorithm; however, this method is not intended to compete with more conventional lossy methods. If lossy compression is acceptable, then other methods are more appropriate, such as those based on wavelets.

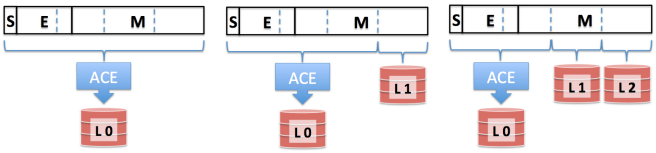
6.1 Variable-precision Prediction

At first glance, it may appear that in order to create a representation supporting progressive precision we can simply partition the error bits in the residual and store them separately. However, prediction-based encoding relies on a context that must be the same for the encoder and decoder, and this requires all residual bits. Instead, we can partition the bits in the original data, with the most significant bits participating in predictive coding and the least significant bits left unencoded. The least significant bits can then be stored separately and accessed as necessary.

As shown in Figure 6, partitioning on byte boundaries results in two options for progressive encoding, based on encoding the first two bytes or first three bytes. This is to be compared to the standard encoding of all four bytes. Ideally we would like to partition the data as much as possible in order to provide greater flexibility in accessing the data; however, partitioning will negatively impact the compression rate if the unencoded bits are not random and are able to be predicted. This means that in practice the efficacy of this approach will depend on the characteristics of the dataset. This method is appropriate in cases where a lossless version of the dataset must be maintained, but yet a more compact, lossy preview of the dataset is desirable for certain types of analysis (i.e. visualization).

6.2 Statistical Modeling of Mantissa

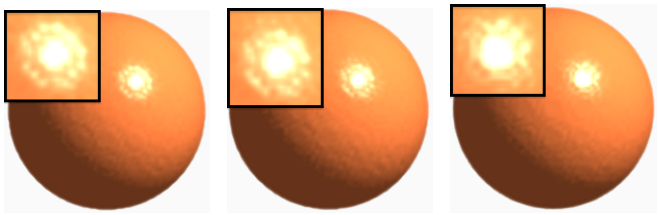
Evaluation of scientific data often involves computation of derived properties, statistical analysis, and/or visualization, especially for 2D/3D data. These tasks may be sensitive to the effects of quantization when using the lossy progressive approximations offered by PP. For instance, visualization of quantized properties may exhibit banding, a distracting artifact that is due to sensitivity of the human visual system to sudden visual discontinuities. In order to address this issue, we propose to keep some statistical information regarding the unencoded bytes along with the encoded data. Specifically, as we process the blocks of data we compute the pmf of the third and fourth bytes of the mantissa and store them with the encoded data if the bytes are partitioned into separate files. This allows us to model the statistical behavior of the last bytes even if they are not entirely uniformly distributed. We find in practice that these pmfs possess varying degrees of



(a) Full encoding (b) Encoding 3 bytes (c) Encoding 2 bytes

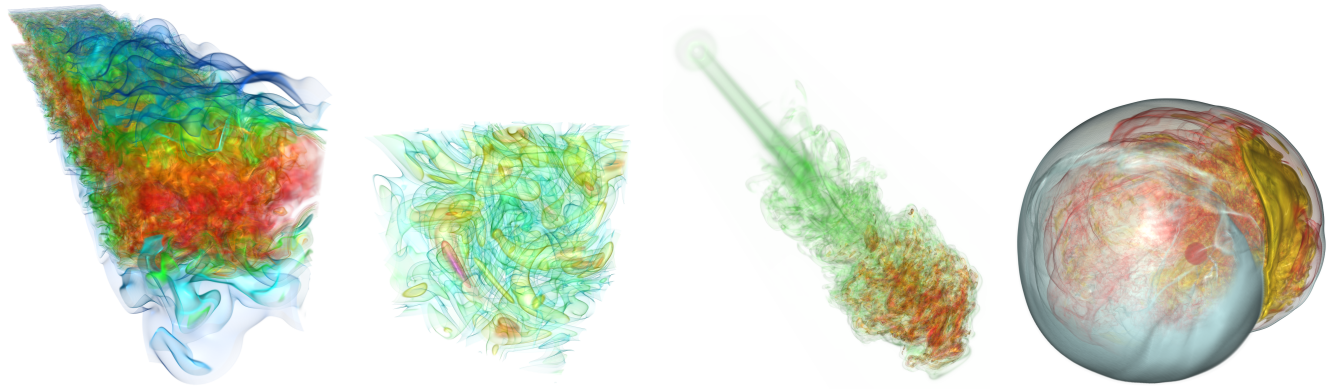
Fig. 6. The Progressive Precision method offers two progressive options for the ACE encoder, by applying the ACE method to the first three bytes and first two bytes of each floating-point value. The remaining bytes are transmitted unencoded. Multiple levels of the data, denoted as L0-L2, are stored separately. In the case of the two-byte and three-byte options, the L0 level provides a compact, lossy version of the data.

nonuniformity; they are not so nonuniform as to allow entropy coding, but they are usually not completely uniform either. If they do happen to be uniform, which sometimes happens with the least significant byte, then we can simply use a uniform random variable to reconstruct bytes. Otherwise, when analyzing the data we reconstruct these last bytes using the inversion method [3], whereby the inverse cumulative mass function is sampled with a uniform random variable. This effectively achieves a non-deterministic, full-precision lossy reconstruction of the data that reduces the deleterious effects of quantization on subsequent evaluation. Shown in Figure 8 is a synthetic sphere volume rendered using Level 0 with and without statistical modeling. Comparison to rendering of the original full-precision volume indicates that this approach is able to reduce some but not all of the artifacts from limiting the precision.



(a) Level 0 (b) Level 0 w/ modeling (c) Levels 0-2 (Original)

Fig. 8. A synthetic sphere dataset is compressed using our Progressive Precision ACE encoder. Images show volume rendering based on: (a) reconstruction using only Level 0; (b) non-deterministic reconstruction at original precision using statistical models of the least significant bytes; (c) original image reconstruction using all levels. Statistical modeling is able to reduce some but not all of the artifacts of limiting the precision.



(a) Combustion Dataset (b) Vortex Dataset (c) Atmospheric Dataset (d) Supernova Dataset

Fig. 7. Our data suite includes four simulation datasets: (a) A jet flame turbulent combustion ($1200 \times 600 \times 270$, 122 time steps, 5 variables); (b) A CFD simulation of turbulent vortex structures (128^3, 100 time steps); (c) A thermal starting plume descending through an adiabatically-stratified fluid ($256^2 \times 1024$, 400 time steps, 5 variables); (d) A supernova core collapse simulation (864^3, 105 time steps, 5 variables).

7 Results

In this section we evaluate our proposed technique and compare the results to other lossless compression methods for floating-point volume data.

7.1 Floating-point Datasets

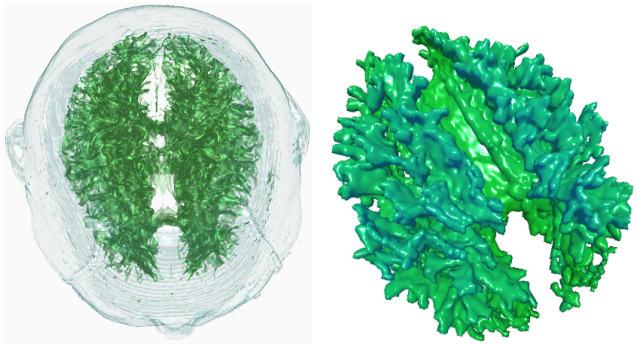
In order to evaluate our proposed compression scheme, we collected a variety of datasets with various dimensions and from various sources. We use four numerical simulations and two measurement datasets, all in single-precision floating-point format. The numerical simulations, shown in Figure 7, include a jet flame turbulent combustion (Sci_Comb), a CFD simulation of turbulent vortex structures (Sci_Vortex), a thermal fluid simulation (Sci_Ohm), and a supernova core collapse (Sci_Super). The dimensions of these datasets are noted in the figure caption. For the jet flame combustion we list results for two of the five variables: the scalar dissipation rate (Sci_Comb_Dr) and the stoichiometric mixture fraction (Sci_Comb_Mx). We also choose two of the five supernova variables, namely density (Sci_Super_D) and pressure (Sci_Super_P).

Our measurement datasets, shown in Figure 9, include the head MRI from the Visible Human Female, but instead of using the original 12-bit dataset we use a registered floating-point version of this data as generated by Muraki et al. [14]. The registration method they employ is based on numerical solution of a nonlinear minimization problem, such that the resulting transformed data is expanded into floating-point precision. This dataset consists of three variables: T1-weighted image (Med_MRI_T1), T2-weighted image (Med_MRI_T2), and proton-density-weighted image (Med_MRI_PD). The other dataset is a Diffusion Tensor MRI (DT-MRI) scan of a healthy person, which consists of the tensor volume (Med_DTMRI_T) as well as the computed anisotropy field (Med_DTMRI_A).

7.2 Implementation

As already mentioned, our encoder processes the data block by block, with each block being independently coded. In our system we use blocks of size 8K, with a frame size of eight, in order to ensure at most 0.5 bpf overhead for predictor selection. Our default APE/ACE encoders use floating-point operations for prediction, compute residuals using the UINT/SUB method, and use Golomb codes for entropy coding the LZC'. The residual sign, produced by the UINT/SUB method, is stored directly, as efforts directed at more efficient encoding were generally unsuccessful. All tests were conducted using a single-threaded program running on a machine equipped with a 2.7GHz Intel i7 processor and 12GB of memory.

We compare our method to two established floating-point compressors for scientific data. The first is FPC [1], which uses FCM and DFCM for prediction. Both are essentially hash tables, with the hash value for FCM based on the previous value and the hash value for DFCM based on the previous stride. In their original form these methods operate on double-precision data, so we modified them in a straightforward way to work on single-precision data as well. Our single-precision implementation follows FPC closely except that we compute the stride using floating-point operations. In particular, we compute residuals using the XOR method and explicitly store both the predictor selection (FCM or DFCM) and number of leading zero bytes (0-3) for each value. We also experimented with different hashing parameters in order to find the best overall settings.



(a) MRI Dataset (b) DT-MRI Dataset

Fig. 9. Our data suite includes two medical datasets: (a) A registered MRI scan ($256 \times 256 \times 187$, 3 channels); (b) A Diffusion Tensor MRI scan ($256 \times 256 \times 64$, 9 channels), along with the computed anisotropy field.

The second method we use for comparison is the Lorenzo predictor as described by Lindstrom and Isenburg [13]. Predictions are computed in floating-point arithmetic and residuals are computed based on the UINT/SUB method. Entropy coding of the LZC' is accomplished using a C translation of the C++ range coder implementation provided by the authors. Our implementation differs only in that the residual sign is stored separately, instead of being folded in with the LZC' as in the original implementation.

7.3 Compression Analysis

Of paramount importance in achieving good compression rates for lossless floating-point compression is computing accurate predictions. In order to evaluate the different prediction methods we can analyze the predictor selection histograms for ACE, since this selec-

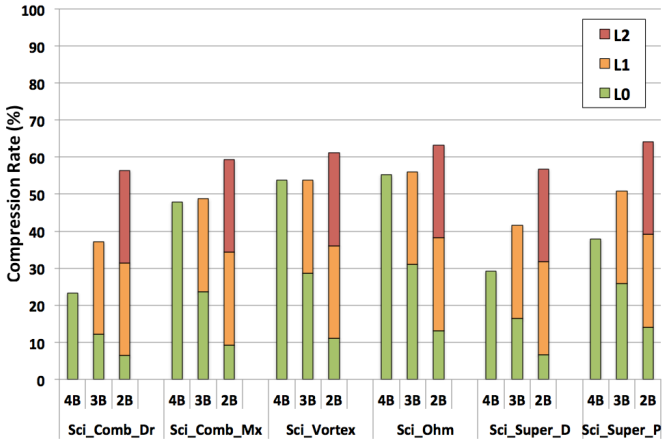


Fig. 10. We apply the PP scheme to scientific datasets using the ACE encoder, with compression of two bytes (2B), three bytes (3B), or all four bytes (4B). The results indicate that providing a lossless, progressive version of certain datasets does not greatly impact the compression rate. Regardless, the L0 level of the 3B and 2B options provides a compact lossy version of the dataset for preview.

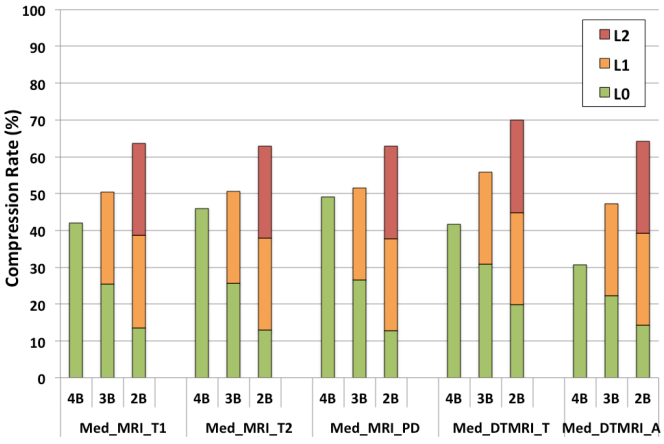


Fig. 11. We apply the PP scheme to medical datasets using the ACE encoder, with compression of two bytes (2B), three bytes (3B), or all four bytes (4B). Although compression using the 3B version results in a small to moderate increase in compression rate, the 2B version requires a significantly higher rate. However, the L0 level of the 2B option offers a lossy version of the dataset at about 10% compression rate.

tion is based on minimum prediction error with all the candidate predictors participating. We observe that the polynomial predictors and the Lorenzo predictors are almost exclusively selected, with the Lorenzo predictor being selected much less frequently, except in the Sci_Super_D dataset. However, we point out that the Lorenzo predictor generally incurs less prediction error than any one polynomial predictor.

The efficiency of entropy coding the predictor selection is related to the relative frequency of selection for all predictors, which varies greatly from dataset to dataset. For most datasets, entropy coding of the predictor selection using adaptive Huffman coding results in a rate of 0.2-0.3 bpf. Experiments with entropy coding of the *LZC'* are shown in Table 2, where our rank-based Golomb coding scheme is compared to adaptive Huffman coding and range coding. The proposed method is able to offer faster encoding, typically twice as fast as Huffman coding and four times as fast as range coding, with slightly lower compression factors. This explains why encoding times for the Lorenzo method and APE are very close; the switched predictor in APE is slower than the Lorenzo predictor, but some of this cost is offset by using a faster entropy coder. Overall, we believe that this method offers a good compromise between encoding speed and coding quality.

The results of applying our PP (Progressive Precision) scheme to the datasets are shown in Figures 10 and 11. Compression rates are shown for encoding the first two bytes (2B) and the first three bytes (3B), with comparisons provided for conventional encoding of all four bytes (4B). For the 3B and 2B options, a breakdown of the space required by each of the constituent levels (L0, L1, L2) is shown in stacked form. For some of the datasets, the 3B and/or 2B options can provide a lossless progressive version of the data with minimal increase in compression rate. However, for several other datasets, these options significantly increase the rate. The primary advantage of using PP in these datasets is the relatively small size of the L0 level, which can provide a very compact, lossy version of the dataset, often at a rate of about 10% for the 2B option.

Dataset	***Universal Coding*** (Compression Factor / Coding Time)	***Huffman Coding*** (Compression Factor / Coding Time)	***Range Coding*** (Compression Factor / Coding Time)
Sci_Comb_Dr	1.5 / 1.0	1.6 / 1.7	1.8 / 4.1
Sci_Ohm	1.2 / 1.0	1.3 / 2.2	1.4 / 5.0
Sci_Super_D	1.5 / 1.0	1.6 / 1.9	1.8 / 4.3
Med_MRI_PD	1.2 / 1.0	1.5 / 2.0	1.6 / 4.5

Table 2. The proposed Universal coding scheme using Golomb codes is compared to adaptive Huffman coding and Range coding for representative datasets. Results are reported as compression factor, followed by relative encoding time, expressed in terms of the time required for Universal coding. The results show that the proposed method is about four times faster than Range coding and about twice as fast as Huffman coding, at the cost of some coding efficiency.

Tables 3, 4, and 5 list the main results of this work, in which our proposed methods are compared against the two previously published methods for our given data suite. Results are reported for a single representative time step, with other time steps giving similar results. Compression is lossless, so the results are reported in terms of the compressed data size relative to the original data size. We provide two metrics of compression performance, namely the compression rate (compressed size / original size) and the compression throughput, reported as mean bpf (bits per float). Based on these tests, we conclude that the best encoder is ACE, which achieves the best rate on all datasets, regardless of dimensionality or source (simulation or observation). These higher compression rates come at a cost, however, as ACE is also the slowest of the methods, although objectively it is not slow. Our other approach, APE, achieves the second best rate on all datasets except for the Supernova dataset, in which the Lorenzo method is second best. Of particular interest are the remarkable rates that APE/ACE is able to achieve on the 3D medical datasets. By examining the predictor selection histogram we are able to attribute this phenomenon to the success of the 4th order Z-oriented polynomial predictor, which is an excellent model for these particular datasets.

By examining the compression rates in conjunction with the predictor selection histograms, we can make some inferences about the characteristics of the datasets themselves. We can think of each predictor as a model for the data, so that if a particular prediction method achieves a low rate then that model is valid. From this we can infer that when the predictor selection pmf shows significant contributions from multiple predictors then there are multiple regions within the data. This heterogeneity can be exploited by the selection mechanism of ACE, and so we expect the ACE rate to be significantly better than the best single method. On the other hand, if the predictor selection pmf is dominated by one predictor, then we can infer that the dataset is homogenous, in which case the ACE method will only do marginally better than the best individual predictor. In general these observa-

Dataset	*Lorenzo*	*FCM/DFCM*	*APE*	*ACE*
Sci_Comb_Dr	30.4 (9.7)	51.5 (16.5)	25.3 (8.1)	**23.3 (7.5)**
Sci_Comb_Mx	54.5 (17.4)	71.4 (22.8)	49.0 (15.7)	**47.8 (15.3)**
Sci_Vortex	60.9 (19.5)	80.9 (25.9)	53.9 (17.2)	**53.7 (17.2)**
Sci_Ohm	64.3 (20.6)	81.7 (26.1)	57.6 (18.4)	**55.2 (17.7)**
Sci_Super_D	31.2 (10.0)	51.6 (16.5)	32.1 (10.3)	**29.2 (9.4)**
Sci_Super_P	40.5 (13.0)	58.1 (18.6)	40.8 (13.1)	**37.9 (12.1)**
Med_MRI_T1	74.2 (23.8)	90.1 (28.8)	46.3 (14.8)	**42.0 (13.4)**
Med_MRI_T2	68.9 (22.1)	86.4 (27.7)	49.8 (16.0)	**45.9 (14.7)**
Med_MRI_PD	68.9 (22.0)	87.3 (27.9)	52.3 (16.7)	**49.1 (15.7)**
Med_DTMRI_T	45.6 (14.6)	63.4 (20.3)	41.9 (13.4)	**41.6 (13.3)**
Med_DTMRI_A	40.9 (13.1)	55.5 (17.8)	33.4 (10.7)	**30.7 (9.8)**

Table 3. Results for lossless compression of single-precision floating-point datasets, reported as compression rate in percent with mean bits per float in parenthesis. Results using four compression methods are given: Lorenzo and FCM/DFCM refer to previously published methods, with APE and ACE being the proposed methods. For each dataset the best compression rate is highlighted in bold. As indicated in the table, ACE achieves the best rates for all datasets.

tions are borne out in the data. In terms of dataset modeling, we see that APE models the Sci_Comb and Med_MRI datasets well, whereas Lorenzo works well for Sci_Super. Results for the other datasets are fairly mixed. The conclusion we make is that no one model works well for all types of data, thereby justifying our switched prediction framework (in particular ACE).

In regards to compression speed (shown in Table 4), FCM/DFCM is easily the fastest method on all datasets. The Lorenzo and APE methods lie between FCM/DFCM and ACE in terms of rates and speed. Lorenzo is faster than APE but with lower compression rates in general. The slower speed of ACE compared to APE is primarily due to the computational cost of the Lorenzo predictor and to the additional overhead related to FCM/DFCM caching. Results for decompression speed (shown in Table 5) mirror those of compression, with FCM/DFCM being the fastest decoder. Again, Lorenzo and APE offer similar speeds. The decoding times demonstrate the same trend as encoding times, but are shorter than encoding times, mostly due to fast entropy decoding.

Dataset	*Lorenzo*	*DFCM/FCM*	*APE*	*ACE*
Sci_Comb_Dr	31	**24**	37	50
Sci_Comb_Mx	36	**26**	39	52
Sci_Vortex	0.3	**0.3**	0.4	0.5
Sci_Ohm	1.2	**1.1**	1.7	2.3
Sci_Super_D	110	**105**	148	174
Sci_Super_P	113	**102**	145	172
Med_MRI_T1	2.0	**1.7**	2.6	3.3
Med_MRI_T2	2.0	**1.7**	2.6	3.2
Med_MRI_PD	2.1	**1.7**	2.6	3.2
Med_DTMRI_T	16	**10**	24	33
Med_DTMRI_A	1.7	**1.1**	2.5	3.4

Table 4. Compression timing results for encoding of single-precision datasets by previously published methods (Lorenzo and FCM/DFCM) and the proposed methods (APE and ACE), reported in seconds. The best time for each dataset is highlighted in bold. FCM/DFCM is shown to be the fastest method on all datasets, with Lorenzo and APE marginally slower.

8 Conclusion

In this work, we introduced a switched prediction lossless coding scheme for compressing scientific and medical floating-point volume data. Our basic approach, APE, uses a set of interpolating polynomials, whereas ACE, our combined approach, uses the Lorenzo and FCM/DFCM predictors along with polynomials. The end result is an encoding framework that is robust, adapting to the data as it encodes, performs extremely well, and is easily extensible. We demonstrate on a variety volume datasets that our proposed approach achieves significantly better compression rates than existing methods at a modest

Dataset	*Lorenzo*	*DFCM/FCM*	*APE*	*ACE*
Sci_Comb_Dr	22	**19**	24	29
Sci_Comb_Mx	24	**21**	25	32
Sci_Vortex	0.3	**0.2**	0.3	0.4
Sci_Ohm	0.9	**0.8**	1.1	1.7
Sci_Super_D	88	**84**	96	110
Sci_Super_P	89	**81**	94	107
Med_MRI_T1	1.4	**1.3**	1.9	2.3
Med_MRI_T2	1.4	**1.3**	1.9	2.3
Med_MRI_PD	1.4	**1.3**	1.9	2.3
Med_DTMRI_T	12	**8**	18	21
Med_DTMRI_A	1.2	**0.8**	1.9	2.4

Table 5. Decompression timing results for decoding of single-precision datasets by previously published methods (Lorenzo and FCM/DFCM) and the proposed methods (APE and ACE), reported in seconds. The best time for each dataset is highlighted in bold. As with encoding, FCM/DFCM is the fastest method on all datasets, but the differences between the decompression times for the various methods are less than with encoding.

increase in encoding time. We believe this additional cost is justified, since our method is still relatively fast and the data is compressed only once, whereas the cost due to increased storage/transmission is incurred continuously/frequently.

An important future investigation is the use of only integer operations in the prediction calculation. As mentioned previously, this is important when encoding and decoding on multiple platforms, and could be accomplished without loss of accuracy by using an integer-based floating-point implementation. Furthermore, an obvious direction for future work is the development of better prediction methods for volume data. These predictors could be incorporated directly into our ACE encoder, perhaps replacing predictors like FCM/DFCM that aren't often used.

Acknowledgments

This research is sponsored in part by the National Science Foundation through grants OCI 0905008, OCI 0850566, and OCI 0749227, and the Department of Energy through grants DE-FC02-06ER25777 and DE-FC02-12ER26072. The Sci_Ohm dataset was generated at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, and the supernova dataset was obtained from Dr. John Blondin of North Caroline State University. The vortex dataset is from the VIZLAB of CAIP at Rutgers University, and the combustion datasets are from Dr. Jackie Chen of Sandia National Laboratories. The Visible Human Female MRI datasets are from the Visible Human Project of the National Library of Medicine. Floating-point versions of these datasets were obtained from Shigeru Muraki. Adam W. Anderson of Vanderbilt University provided us with the DT-MRI dataset. Finally,

we would like to thank the reviewers for their corrections and helpful recommendations, and to thank M. Burtscher and P. Lindstrom for making their compression code available.

REFERENCES

[1] M. Burtscher and P. Ratanaworabhan. High throughput compression of double-precision floating-point data. In *Data Compression Conf. '07*, DCC '07, pages 293 –302, march 2007.

[2] M. Burtscher and P. Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Trans. on Computers*, 58(1):18 –31, january 2009.

[3] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[4] X. Du and R. Moorhead. Multiresolutional visualization of evolving distributed simulations using wavelets and MPI. In *Scientific Visualization Conference, 1997*, page 54, 1997.

[5] V. Engelson, D. Fritzson, and P. Fritzson. Lossless compression of high-volume numerical data from simulations. In *Conf. on Data Compression '00*, DCC '00, page 574. IEEE Computer Society, 2000.

[6] M. N. Gamito and M. S. Dias. Lossless coding of floating point data with JPEG 2000 part 10. *Applications of Digital Image Processing XXVII*, 5558(1):276–287, 2004.

[7] F. Ghido. An efficient algorithm for lossless compression of IEEE float audio. In *Data Compression Conf. '04*, DCC '04, pages 429 – 438, march 2004.

[8] M. Hans and R. Schafer. Lossless compression of digital audio. *Signal Processing Magazine, IEEE*, 18(4):21 –32, july 2001.

[9] J. Hauser. SoftFloat, 2002. Version 2b, http://www.jhauser.us/arithmetic/SoftFloat.html.

[10] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3):343–348, 2003.

[11] M. Isenburg, P. Lindstrom, and J. Snoeyink. Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869 – 877, 2005. CAD '04 Special Issue: Modelling and Geometry Representations for CAD.

[12] T. Liebchen, T. Moriya, N. Harada, Y. Kamamoto, and Y. Reznik. The MPEG-4 audio lossless coding (ALS) standard - technology and applications. In *119th Convention of Audio Engineering Society*, October 2005.

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

[14] S. Muraki, T. Nakai, Y. Kita, and K. Tsuda. An attempt for coloring multichannel MR imaging data. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):265–274, July 2001.

[15] S. Pigeon. *Lossless Compression Handbook*, chapter Huffman Coding. Number ISBN 0-12-620861-1 in Communications, Networking, and Multimedia. Academic Press, 2003.

[16] P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conf. '06*, pages 133 – 142, march 2006.

[17] T. Robinson. SHORTEN: Simple lossless and near-lossless waveform compression. Technical Report 156, Cambridge Univ. Eng. Dept., Cambridge, UK, 1994.

[18] K. Sayood and K. Anderson. A differential lossless image compression scheme. *IEEE Trans. on Signal Processing*, 40(1):236 –241, jan 1992.

[19] C. R. Schroeder. Adaptive coarsening: simple, effective floating-point compression. In *ACM/IEEE Conference on Supercomputing*, SC '06. ACM, 2006.

[20] T. M. Shafaat and S. B. Baden. A method of adaptive coarsening for compressing scientific datasets. In *Int'l. Conf. on Applied Parallel Computing*, PARA'06, pages 774–780. Springer-Verlag, 2007.

[21] H. Tao and R. Moorhead. Lossless progressive transmission of scientific data using biorthogonal wavelet transform. In *IEEE Image Processing '94*, volume 3, pages 373 –377, nov 1994.

[22] H. Tao and R. J. Moorhead. Progressive transmission of scientific data using biorthogonal wavelet transform. In *Visualization Conf. '94*, VIS '94, pages 93–99. IEEE Computer Society Press, 1994.

[23] H. Tomari, M. Inaba, and K. Hiraki. Compressing floating-point number stream for numerical applications. In *Int'l. Conf. on Networking and Computing '10*, ICNC '10, pages 112 –119, nov. 2010.

[24] A. Trott, R. Moorhead, and J. McGinley. The application of wavelets to lossless compression and progressive transmission of floating point data in 3-d curvilinear grids. In *Proc. of Data Compression Conference '96*, DCC '96, page 458, mar/apr 1996.

[25] 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 *Visualization Conf. '96*, VIS '96, pages 385–388. IEEE Computer Society Press, 1996.

[26] B. Usevitch. JPEG2000 compliant lossless coding of floating point data. In *Data Compression Conf. '05*, DCC '05, pages 484–484. IEEE Computer Society, 2005.

[27] B. Usevitch. JPEG2000 compatible lossless coding of floating-point data. *EURASIP Journal on Image and Video Processing*, 2007, 2007.

[28] B. E. Usevitch. JPEG2000 extensions for bit plane coding of floating point data. In *Data Compression Conf. '03*, DCC '03, page 451. IEEE Computer Society, 2003.

[29] B. Wohlberg and C. Brislawn. JPEG 2000 part 10: Floating point coding. Technical Report 2644, ISO/IEC JTC1/SC29/WG1, 2002.

[30] X. Xie and Q. Qin. Fast lossless compression of seismic floating-point data. In *Int'l. Forum on Information Technology and Applications '09*, volume 1 of *IFITA '09*, pages 235 –238, may 2009.

[31] D. Yang, T. Moriya, and T. Liebchen. A lossless audio compression scheme with random access property. In *IEEE Conf. on Acoustics, Speech, and Signal Processing '04*, volume 3 of *ICASSP '04*, pages 1016–1019, may 2004.

[32] Y. You and M. Y. Sung. Haptic data transmission based on the prediction and compression. In *IEEE Conf. on Communications '08*, ICC '08, pages 1824 –1828, may 2008.

[33] T. Zhou, Y. Liu, Q. Chen, K. Cai, J. Teng, and Z. Chen. An entropy coding method for floating-point texture coordinates of 3D mesh. In *IEEE Int'l. Symp. on Circuits and Systems '10*, ISCAS '10, pages 1835 –1838, june 2010.