DOI: 10.14529/jsfi170402

Towards Decoupling the Selection of Compression Algorithms from Quality Constraints – An Investigation of Lossy Compression Efficiency

Julian M. Kunkel[1], *Anastasiia Novikova*[2], *Eugen Betke*[1]

Data intense scientific domains use data compression to reduce the storage space needed. Lossless data compression preserves information accurately but lossy data compression can achieve much higher compression rates depending on the tolerable error margins. There are many ways of defining precision and to exploit this knowledge, therefore, the field of lossy compression is subject to active research. From the perspective of a scientist, the qualitative definition about the implied loss of data precision should only matter.

With the Scientific Compression Library (SCIL), we are developing a meta-compressor that allows users to define various quantities for acceptable error and expected performance behavior. The library then picks a suitable chain of algorithms yielding the user's requirements, the ongoing work is a preliminary stage for the design of an adaptive selector. This approach is a crucial step towards a scientifically safe use of much-needed lossy data compression, because it disentangles the tasks of determining scientific characteristics of tolerable noise, from the task of determining an optimal compression strategy. Future algorithms can be used without changing application code.

In this paper, we evaluate various lossy compression algorithms for compressing different scientific datasets (Isabel, ECHAM6), and focus on the analysis of synthetically created data that serves as blueprint for many observed datasets. We also briefly describe the available quantities of SCIL to define data precision and introduce two efficient compression algorithms for individual data points. This shows that the best algorithm depends on user settings and data properties.

Keywords: data reduction, compression, lossy, climate data.

Introduction

Climate science is data intense. For this reason, the German Climate Computing Center spends a higher percentage of money on storage compared to computation. While providing a peak compute performance of 3.6 PFLOPs, a shared file system of 54 Petabytes and an archive complex consisting of 70,000 tape slots is provided. Compression offers a chance to increase the provided storage space or to provide virtually the same storage space but with less costs. Analysis has shown that with the proper preconditioning an algorithm can achieve a compression factor of roughly 2.5:1 with lossless compression, i.e., without loss of information [7]. However, the throughput of compressing data with the best available option is rather low (2 MiB/s per core). By using the statistical method in [9] to estimate the actual compression factor that can be achieved on our system, we saw that LZ4fast yield a compression ratio (we define compression ratio as $r = \frac{\text{size compressed}}{\text{size original}}$; inverse is the compression factor) of 0.68 but with a throughput of more than 2 GiB/s on a single core. Therefore, on our system it even outperforms algorithms for optimizing memory utilization such as BLOSC.

Lossy compression can yield a much lower ratio but at expense of information accuracy and precision. Therefore, users have to carefully define the acceptable loss of precision and properties of the remaining data properties. There are several lossy algorithms around that target scientific applications.

[1]Deutsches Klimarechenzentrum, Hamburg, Germany
[2]Universität Hamburg, Hamburg, Germany

However, their definition of the retained information differs: some allow users to define a fixed ratio useful for bandwidth limited networks and visualization; most offer an absolute tolerance and some even relative quantities. The characteristics of the algorithm differs also on input data. For some data, one algorithm yields a better compression ratio than another. Scientists struggle to define the appropriate properties for these algorithms and must change their definition depending on the algorithm decreasing code portability.

In the AIMES project we develop libraries and methods to utilize lossy compression. The SCIL library[3] provides a rich set of user quantities to define from, e.g., HDF5. Once set, the library shall ensure that the defined data quality meets all criteria. Its plugin architecture utilizes existing algorithms and aims to select the best algorithm depending on the user qualities and the data properties.

In the paper [10], we introduced the architecture and idea of the compression library SCIL, together with two new lossy algorithms and analyzed the results for a single data set of a climate model. This paper repeats key concepts from the previous paper but its **key contribution** is providing new experiments and a different perspective by investigating understandable synthetic data patterns in depth. Understanding the general properties (ratio, speed) when compressing different types of data enables us to approximate behavior for similar types of data.

This paper is structured as follows: We give a review over related work in Section 1. The design is described in Section 2. An evaluation of the compression ratios is given in Section 3. Final section provides a summary.

1. Related Work

The related work can be structured into: 1) algorithms for the lossless data compression; 2) algorithms designed for scientific data and the HPC environment; 3) methods to identify necessary data precision and for large-scale evaluation.

Lossless algorithms: The LZ77 [17] algorithm is dictionary-based and uses a “sliding window”. The concept behind this algorithm is simple: It scans uncompressed data for two largest windows containing the same data and replaces the second occurrence with a pointer to the first window. DEFLATE is a variation of LZ77 and uses Huffman coding [5]. GZIP is a popular lossless algorithm based on DEFLATE.

Lossy algorithms for floating point data: FPZIP [15] was primarily designed for lossless compression of floating point data. It also supports lossy compression and allows the user to specify the bit precision. The error-bounded compression of ZFP [15] for up to 3 dimensional data is accurate within machine epsilon in lossless mode. The dimensionality is insufficient for the climate scientific data. SZ [4] is a newer and effective HPC data compression method, it uses a predictor and the lossless compression algorithm GZIP. Its compression ratio is at least 2x better than the second-best solution of ZFP. In [7], compression results for the analysis of typical climate data was presented. Within that work, the lossless compression scheme MAFISC with preconditioners was introduced; its compression ratio was compared to that of standard compression tools reducing data 10% more than the second best algorithm. In [6], two lossy compression algorithms (GRIB2, APAX) were evaluated regarding to loss of data precision,

[3]The current version of the library is publicly available under LGPL license:
`https://github.com/JulianKunkel/scil`

compression ratio, and processing time on synthetic and climate dataset. These two algorithms have equivalent compression ratios and depending on the dataset APAX signal quality exceeds GRIB2 and vice versa.

Methods: Application of lossy techniques on scientific datasets was already discussed in [2, 3, 8, 12–14]. The first efforts for determination of appropriate levels of precision for lossy compression method were presented in [1]. By doing statistics across ensembles of runs with full precision or compressed data, it could be determined if the scientific conclusions drawn from these ensembles are similar.

In [9], a statistical method is introduced to predict characteristics (such as proportions of file types and compression ratio) of stored data based on representative samples. It allows file types to be estimated and, e.g., compression ratio by scanning a fraction of the data, thus reducing costs. This method has recently been converted to a tool[4] that can be used to investigate large data sets.

2. Design

The main goal of the compression library SCIL is to provide a framework to compress structured and unstructured data using the best available (lossy) compression algorithms. SCIL offers a user interface for defining the tolerable loss of accuracy and expected performance as various quantities. It supports various data types. In Fig. 1, the data path is illustrated. An application can either use the NetCDF4, HDF5 or the SCIL C interface, directly. SCIL acts as a meta-compressor providing various backends such as the existing algorithms: LZ4, ZFP, FPZIP, and SZ. Based on the defined quantities, their values and the characteristics of the data to compress, the appropriate compression algorithm is chosen[5]. SCIL also comes with a pattern library to generate various relevant synthetic test patterns. Further tools are provided to plot, to add noise or to compress CSV and NetCDF3 files. Internally, support functions simplify the development of new algorithms and the testing.

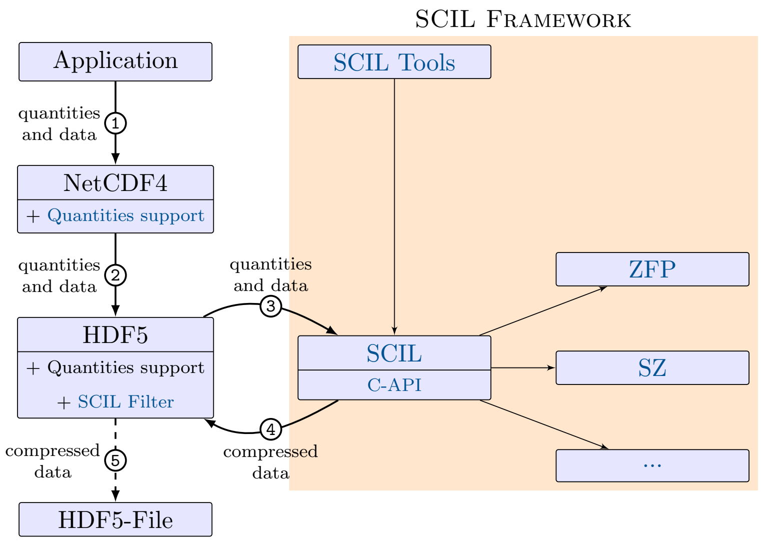


Figure 1. SCIL compression path and components

[4]`https://github.com/JulianKunkel/statistical-file-scanner`

[5]The implementation for the automatic algorithm selection is ongoing and not the focus of this paper. We will model performance and compression ratio for the different algorithms, data properties and user settings.

2.1. Supported Quantities

There are three types of quantities supported:

Accuracy quantities define the tolerable error on lossy compression. When compressing the value v to $\hat{v}$, it bounds the residual error ($r = v - \hat{v}$):

- **absolute tolerance**: $v - \text{abstol} \leq \hat{v} \leq v + \text{abstol}$;
- **relative tolerance**: $v/(1 + reltol) \leq \hat{v} \leq v \cdot (1 + reltol)$;
- **relative error finest tolerance**: used together with rel tolerance; absolute tolerable error for small v's. If relfinest $> |v \cdot (1 \pm reltol)|$, then $v - \text{relfinest} \leq \hat{v} \leq v + \text{relfinest}$;
- **significant digits**: number of significant decimal digits; and
- **significant bits**: number of significant digits in bits.

SCIL must ensure that all the set accuracy quantities are honored, meaning that one can set, e.g., absolute and relative tolerance and the value that is most strict quantity is chosen.

Performance quantities define the expected performance behavior for both compression and decompression (on the same system). The value can be defined according to: 1) absolute throughput in MiB or GiB; or 2) relative to network or storage speed. It is considered to be the expected performance for SCIL but it may not be as strictly handled as the qualities – there may be some cases in which performance is lower. Thus, SCIL must estimate the compression rates for the data, this value is not yet covered by the algorithm selection but will be in the future. The system's performance must be trained for each system using machine learning.

Supplementary quantities: An orthogonal quantity that can be set is the so called *fill value*, a value that scientists use to mark special data points. This value must be preserved accurately and usually is an specific high or low value that may disturb a smooth compression algorithm.

An example for using the low-level C-API is illustrated in Listing 1.

```c
#include <scil.h>
int main(){
  double data[10][20]; // our raw data, we assume it contains sth. useful

  // define the quantities as hints, all specified conditions must hold
  scil_user_hints_t hints;
  hints.relative_tolerance_percent = 10;
  hints.absolute_tolerance = 0.5;
  hints.significant_digits = 2;
  // define performance expectation on decompression speed
  hints.decomp_speed.unit = SCIL_PERFORMANCE_GIB;
  hints.decomp_speed.multiplier = 3.5;
  // ... add more quality constaints if desired
  // create a compression context for a given datatype
  scil_context_t* ctx;
  scil_create_context(&ctx, SCIL_TYPE_DOUBLE, 0, NULL, &hints);

  // the multi-dimensional size of the data, here 10x20
  scil_dims_t dims;
  scil_initialize_dims_2d(& dims, 10, 20);

  // the user is responsible to allocate memory for the output/tmp buffers
  size_t buffer_size = scil_get_compressed_data_size_limit(& dims, SCIL_TYPE_DOUBLE);
  byte * compressed_data = malloc(buffer_size);

  size c_size; // will hold the number of bytes of the compressed buffer
  scil_compress(compressed_data, buffer_size, data, &dims, &c_size, ctx);
  // now do something with the data in compressed_data
```

Listing 1. Usage of the low-level API

2.2. Algorithms

The development of the two algorithms sigbits and abstol has been guided by the definition of the user quantities. Both algorithms aim to pack the number of required bits as tightly as possible into the data buffer. We also consider these algorithms useful baselines when comparing any other algorithm.

2.2.1. Abstol

This algorithm guarantees the defined absolute tolerance. Pseudocode for the Abstol algorithm is provided in Listing 2.

```
compress(data, abstol, outData){
  (min,max) = computeMinMax(data)
  // quantize the data converting it to integer, according to abstol
  tmp[i] = round((data[i] - min) * abstol)
  // compute numbers of mantissa bits needed to store the data
  bits = ceil(log2(1.0 + (max - min) / abstol))
  // now pack the neccessary bits from the integers tightly
  outData = packData(tmp, bits)
}
```

Listing 2. Pseudocode for the Abstol algorithm

2.2.2. Sigbits

This algorithm preserves the user-defined number of precision bits from the floating point data but also can honor the relative tolerance. One precision bit means we preserve the floating point's exponent and sign bit as floating point implicitly adds one point of precision. All other precision bits are taken from the mantissa of the floating point data. Notice, that for denormalized number (the leading "hidden" bit is always 0), for example, mantissa 0.0000001 with 5 precision bits will be cutted to 0.00000. That means that the significand part is missed. Note that the sign bit must only be preserved, if it is not constant in the data. Pseudocode for the Sigbits algorithm is illustrated in Listing 3. When a relative tolerance is given it is converted to the number of precision bits.

```
compress(data, precisionBits, outData){
  // preserve the exponent always
  (sign, min, max) = computeExponentMinMax(data)
  // compute numbers of bits needed to preserve the data
  bits = sign + bits for the exponent + precisionBits - 1
  // convert preserved bits into an integer using bitshift operators
  tmp[i] = sign | exponent range used | precision Bits
  // now pack the bits tightly
  outData = packData(tmp, bits)
}
```

Listing 3. Pseudocode for the Sigbits algorithm

Since the values around the 0 contain a huge range of exponents in IEEE floating point representation, e.g., 0 ± 10^{-324} which is usually not needed, the *relative error finest tolerance* limits the precision around 0 to remove exponents and preserve space.

2.3. Compression chain

Internally, SCIL creates a compression chain which can involve several compression algorithms as illustrated in Fig. 2. Based on the basic datatype that is supplied, the initial stage of the chain is entered. Algorithms may be preconditioners to optimize data layout for subsequent

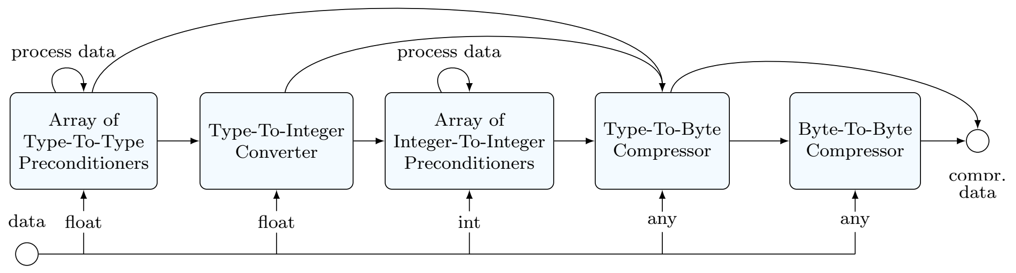

process data
Array of Type-To-Type Preconditioners
Type-To-Integer Converter
process data
Array of Integer-To-Integer Preconditioners
Type-To-Byte Compressor
Byte-To-Byte Compressor
compr. data
data
float
float
int
any
any

Figure 2. SCIL compression chain. The resulting data path depends on the input data type.

Table 1. Schemas for Pattern Names

Pattern name	Example
random[mutator][repeat]-[min-max]	randomRep10-100
random[min]-[max]	random0-1
steps[number]	steps2
sin[frequency][iterations]	sin35
poly4-[random]-[degree]	poly4-65432-14
simplex[frequency][iterations]	simplex102

If not specified otherwise, min=0 and max=100.

compression algorithms, converters from one data format to another, or, on the final stage, a lossless compressor. Floating point data can be first mapped to integer data and then into a byte stream. Intermediate steps can be skipped.

2.4. Tools

SCIL comes with tools useful for evaluation and analysis: 1) To create synthetic data for compression studies, i.e., well-defined multi-dimensional data patterns of any size; 2) To modify existing data adding a random noise based on the hint set; 3) To compress existing CSV and NetCDF data files.

Synthetic data covers patterns such as constant, random, linear steps (creates a hyper-plane in the diagonal), polynomial, sinusoidal or by the OpenSimplex algorithm. OpenSimplex implements a procedural noise generation [11]. An example for the Simplex data is given in Fig. 3; original data and the compressed data for the Sigbits algorithm preserving 3 bits from the mantissa. Each pattern can be parameterized by the min/max value, random seed and two pattern-specific arguments:

- sin: Base frequency (1 means to have one sine wave spanning all dimensions) and number of recursive iterations by which the frequency is doubled each time and the amplitude is halved;
- poly4: Initial number for random number generator and degree of the polynomial;
- steps: Number of steps between min/max; and
- simplex: Initial frequency and number of iterations, in each iteration the frequency is doubled and the amplitude halved.

Additionally, mutators can be applied to these patterns such as step (creating a linear N-dimensional interpolation for the given number of data points) and repeat which repeats a number N-times. Explanation of file names used in this paper are listed in Section 2.4.

3. Evaluation

In the evaluation, we utilize SCIL to compress the data with various algorithms. In all cases, we manually select the algorithm. The test system is an Intel i7-6700 CPU (Skylake) with 4 cores @ 3.40GHz; one core is used for the testing and turbo boost is disabled.

3.1. Test Data

All data uses single precision floating point (32 bit) representation. A pool of data is created 10 times with different random seed numbers from several synthetic patterns generated by SCIL's pattern library and kept in CSV-files. Synthetic data has the dimensionality of (300 x 300 x 100 = 36 MB).

Additionally, we utilize the output of the ECHAM atmospheric model [16] which stored 123 different scientific variables for a single timestep as NetCDF and the output of the hurricane Isabel model which stored 633 variables for a single timestep as binary[6]. The scientific data varies in terms of properties and in particular, the expected data locality. For example, in the Isabel data many variables are between 0 and 0.02 many between -80 and +80 and some are between -5000 and 3000.

3.2. Experiments

For each of the test files, the following setups are run[7]:

- Lossy compression preserving T significant bits
  - Tolerance (T): 3, 6, 9, 15, 20 bits;
  - Algorithms: zfp, sigbits, sigbits+lz4[8];
- Lossy compression with a fixed absolute tolerance
  - Tolerance: 10%, 2%, 1%, 0.2%, 0.1% of the data maximum value[9];
  - Algorithms: zfp, sz, abstol, abstol+lz4.

Each configuration is run 3 times measuring compression and decompression time.

3.3. Understanding Synthetic Patterns

The compression ratios for various synthetic patterns are shown in: Fig. 4 and Fig. 5, the figures show a variable absolute tolerance and the number of precision bits, respectively. Note that for random patterns and simplex patterns, 10 different seeds have been applied and each measurement results in one data point. In most cases, they do not differ notably for a given precision and algorithm.

Absolute tolerance: First, we look at the random patterns in Fig. 4 and the absolute tolerance (left column). The x-axes contains the absolute tolerance between 0.001 and 0.1. In this experiment, the actual tolerance is computed based on the maximum value to enable comparison between different datasets. A value of 0.1 means that 10% of the maximum is used as absolute tolerance. Thus, data can be quantized and encoded in 5 values representing the mean of (0-20%,

[6]`http://vis.computer.org/vis2004contest/data.html`

[7]The versions used are SZ from Aug 14 2017 (git hash 29e3ca1), zfp 0.5.0, LZ4 (Aug 12 2017, 930a692).

[8]This applies first the Sigbits algorithm and then the lossless LZ4 compression.

[9]This is done to allow comparison across variables regardless of their min/max. In practice, a scientist would set the reltol or define the abstol depending on the variable.

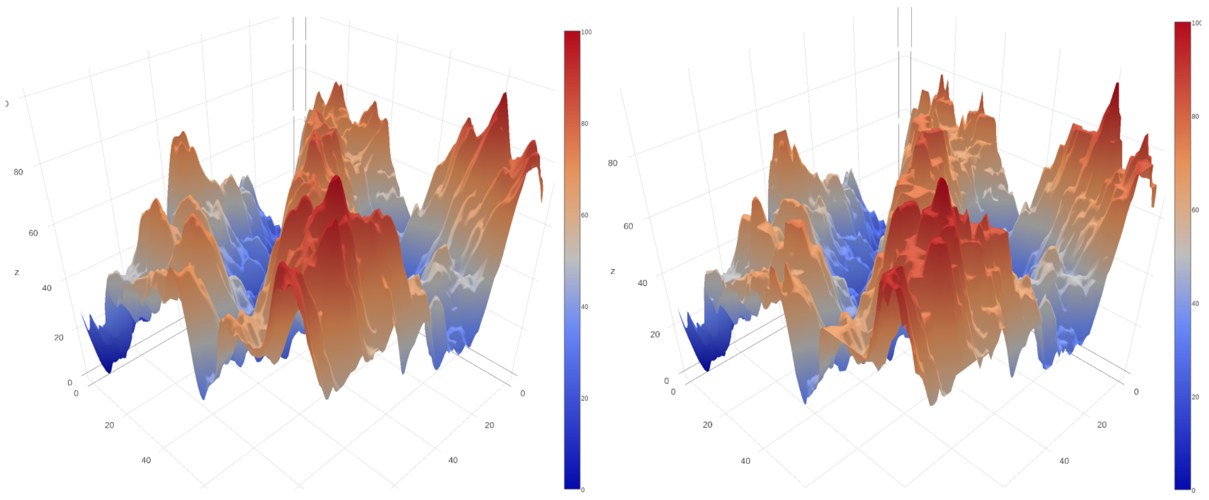


Figure 3. Example of synthetic pattern: Simplex 206 in 2D

20-40%, ...), i.e., 3 bits are needed out of 32 bits ⇒ a compression ratio of 9.4% can be achieved. Therefore, as abstol performs this quantization it is expected to reach that level regardless of the redundancy in the data which it does. ZFP and SZ predict the next data point based on the experience, therefore, as expected the achievable ratio for random data is worse than for abstol by a constant of 0.05. Applying a lossless compressor on pure random data does not help, too. When interpolating between 10x10x10 neighboring cubes (randomIpol10), then the prediction of SZ yields a similar performance to abstol. Abstol+LZ4 improves the ratio slightly for interpolated data as it may reuse some prefix in data. The ratio is a bit better when compressing random data between 1-100 vs. -1 and +1, the reason is that twice the number of intervals are used when data spans a negative and positive range. The randomRep data repeats a value in a block of 2x2x2 or 10x10x10, therefore, SZ and ZFP reduce their ratio but only to 80% and 66% for the small and large block, respectively. This is surprising as 8 and 1000 neighboring points contain the identical value, respectively. For SZ, the interpolation (10x cube) and replication (2x cube) leads to comparable results. Abstol+LZ4 exploits the redundancy better, even better than the GZIP lossy compressor as part of SZ. Note that in most cases the achievable ratio is independent of the random seed for creating the data (ZFP sometimes has some variance).

The polynomial of degree 3 is compressed well by all algorithms, SZ benefits from the locality. Realistic terrain data is created by the simplex algorithm. In Fig. 5, it can be seen that SZ and ZFP can predict this pattern well. Abstol+LZ4 cannot exploit the jitter in the data much but with increasing absolute tolerance, jitter is rounded and repetition can be exploited better. The more refinements are made by the simplex algorithm, the more fine structures are visible, increasing the difficulty for ZFP and Abstol+LZ4 to exploit similarities. The prediction steps of SZ work well in all cases. A similar pattern can be observed with the more simple sinusoidal patterns, they are a bit more uniform and easier to exploit for all algorithms.

Precision bits: Sigbits uses a number of bits needed for encoding the sign bit and exponent range (max - min) and the precision bits as defined by the user. For the random data between 0 to 1, 5 bits are used to represent the exponent[10] leading to a ratio of 22% for 3 precision bits (= 2 mantissa bits). With negative and positive values an extra bit is needed, using interpolation on a range between -1 and +1 may cause additional exponents to emerge, reducing the ratio. Since the remainder of the mantissa looks random, the additional LZ4 stage cannot improve the ratio for

[10]Since the function `rand()` is used to create the test data.

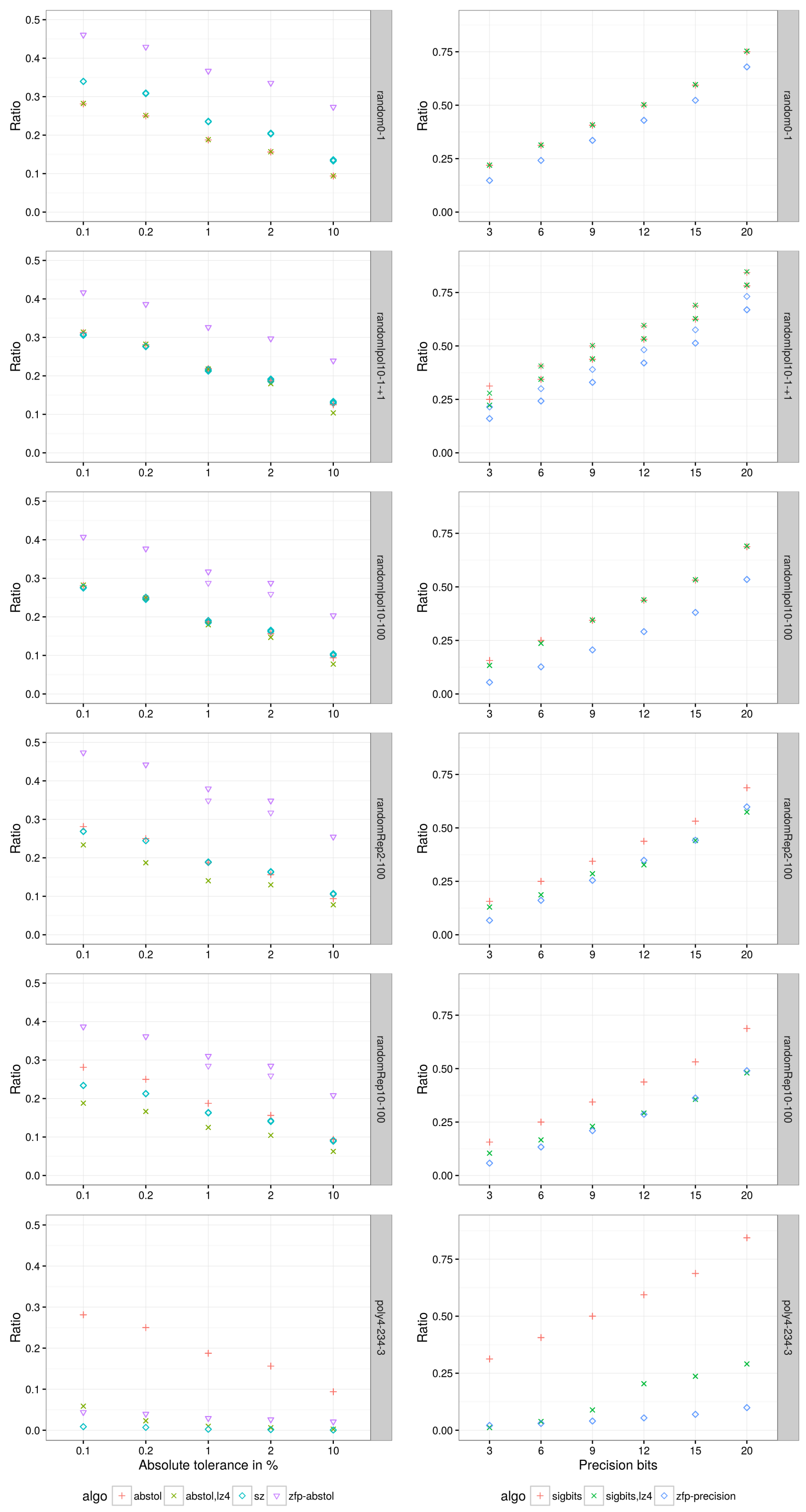


Figure 4. Mean harmonic compression factor for synthetic data based on user settings

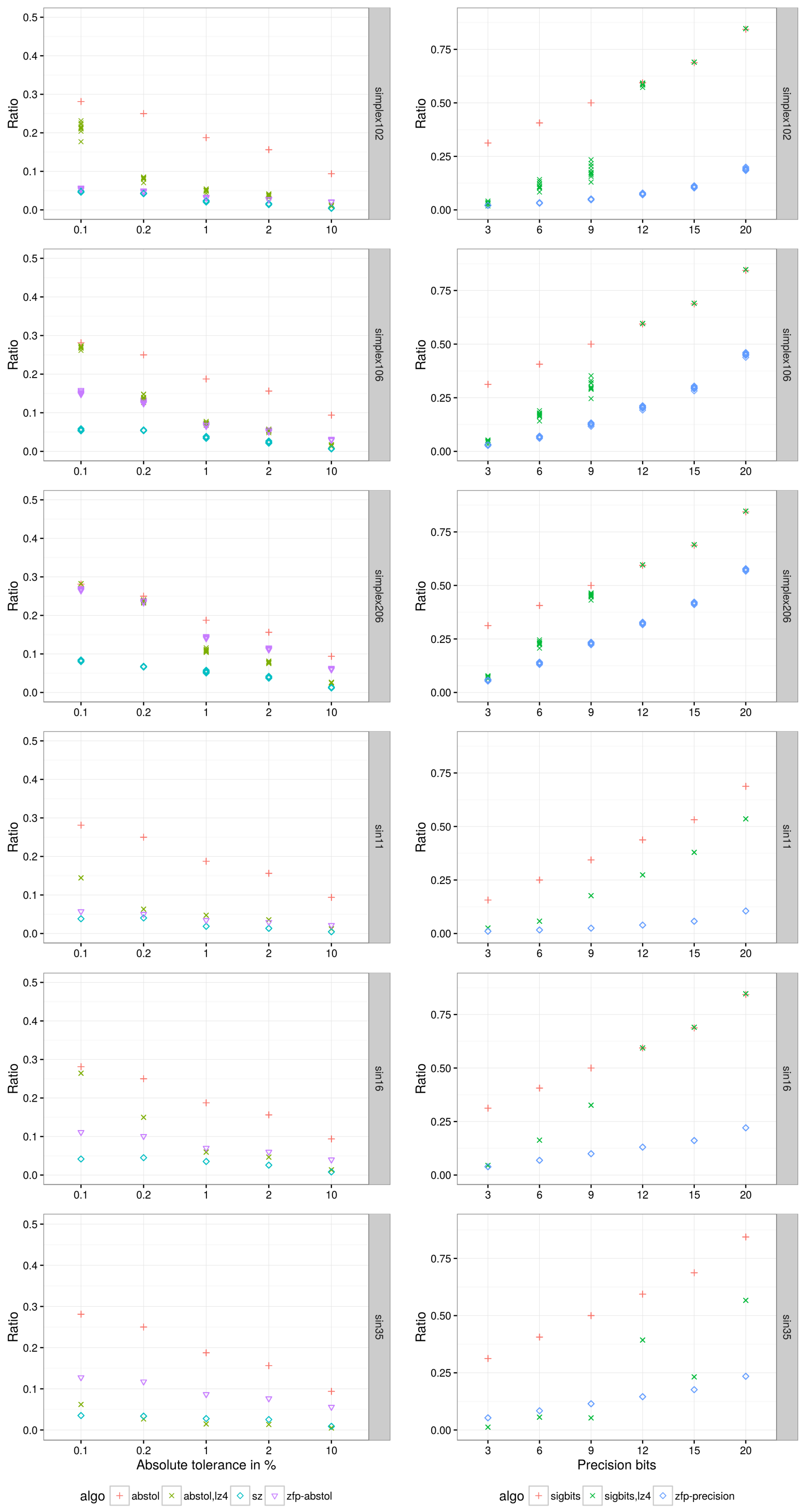


Figure 5. Mean harmonic compression factor for synthetic data based on user settings

interpolated data but only when repeated data blocks emerge. The regular polynomial benefits for a low precision from the repeatability but behaves non-linearly. Similarly, this appears for the variants of the simplex pattern and sin16, the higher the frequency the quicker it degrades. Sin35 behaves non-monotonic for Sigbits+LZ4, with 15 precision bits the ratio is lower than for 12 bits. The reason presumably is that in this case the resulting number of bits is 20 that allows LZ4 to find patterns more effectively (6 bits are needed for the exponent covering the range 1-100, 14 bits for the mantissa).

ZFP cannot be compared easily as it does not support the precision bit quantity but a fixed ratio and compresses data blockwise – this leads to validation errors. For comparison reasons, we use `zfp_stream_set_precision()` with the same number of bits as utilized by Sigbits. Still with a fixed setting, ZFP typically yields a better ratio at the cost of precision.

3.4. Scientific Data

Ratio Depending On Tolerance Next, we investigate the compression factor depending on the tolerance level for the scientific data. The graphs in Fig. 6 and 7 show the mean compression factor for two types of climate data files varying the precision for the algorithms ZFP, SZ, Sigbits and Abstol. The mean is computed on the pool of data, i.e., after compression, a factor of 50:1 means all compressed files occupy only 2% of the original size.

We will discuss the absolute tolerance first: With 1% of tolerance, a compression factor of more than 15 can be achieved on both data sets. For the ECHAM dataset, Abstol+LZ4 outperforms SZ, for the Isabel data SZ outperforms Abstol+LZ4 initially clearly. In both cases, the ratio of Abstol+LZ4 improves with the absolute tolerance.

When using precision bits, ZFP yields a better factor on the Isabel data than Sigbits. However, note that since this dataset uses high fill values, the result cannot be trusted. With 3 precision bits (relative error about 12.5%), a ratio of around 10 is achievable. It can be concluded that the ECHAM dataset is more random compared to the Isabel data which stores a finer continuous grid.

Fixed Absolute Tolerance To analyze throughput and compression ratio across variables, we selected an absolute tolerance of 1% of the maximum value.

Mean values are shown in Tbl. 2a. The synthetic random patterns serve as baseline to understand the benefit of the lossy compression; we provide the means for the different random patterns. Abstol+LZ4 yields a 20% reduction compared to SZ for ECHAM and the random data while for Isabel data it needs about 50% more space. Compression and decompression

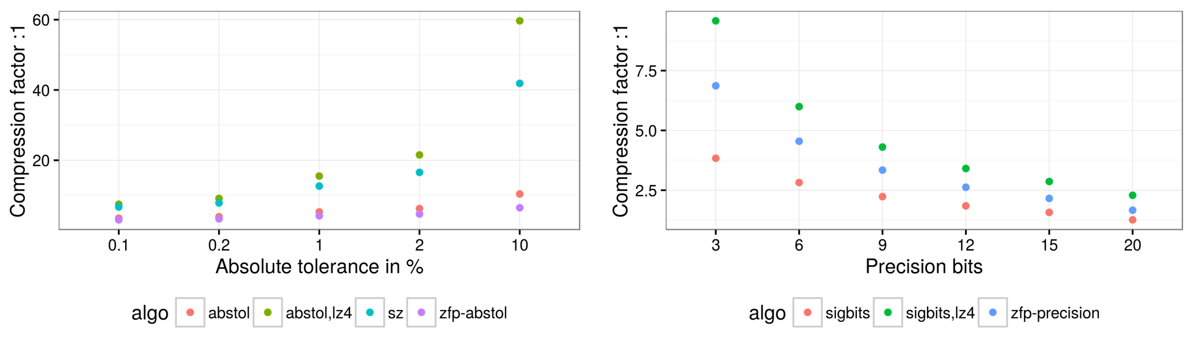


Figure 6. Mean harmonic compression factor for ECHAM data based on user settings

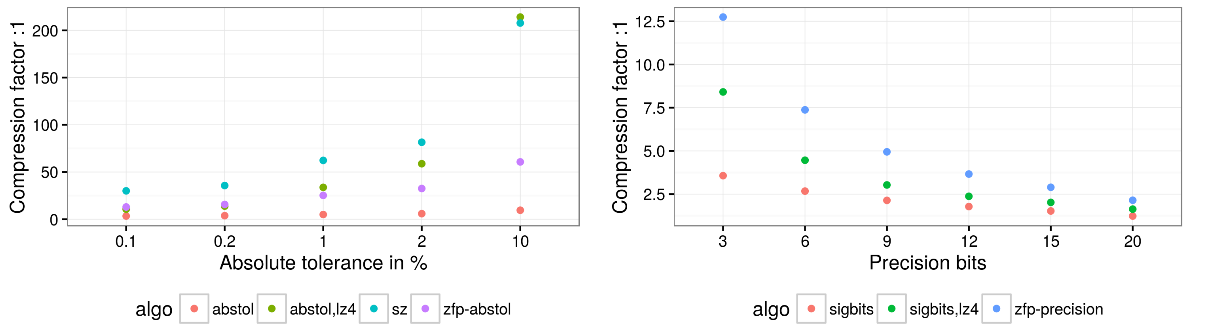


Figure 7. Mean harmonic compression factor for Isabel data based on user settings

speed of Abstol+LZ4 is at least 2x the speed of SZ. LZ4 supports the detection of random data and avoids compression in that case increasing performance for random data significantly. ZFP achieves worse compression ratios but with a better compression speed than SZ. For Isabel data the decompression is even a bit higher than when using Abstol+LZ4.

The results for the individual climate variables are shown in Fig. 8 for ECHAM and Isabel data; on the x-axis are the different variables sorted on compression ratio to ease identification of patterns, e.g., to see the impact of higher compression ratio to performance. It can be observed that for the ECHAM data Abstol+LZ4 yields in most cases the best compression ratio and the best compression and decompression speeds. For some variables (on the right), SZ compresses better. When dealing with Isabel data, in most cases SZ outperforms the ratio of Abstol+LZ4, but for a few Abstol+LZ4 is better. SZ performance is quite robust and so is Abstol but the LZ4 step depends on the compressability on the data. Both Abstol+LZ4 and SZ reveal some steps in the data, where the complexity to compress data increases.

Table 2. Harmonic Mean Compression of Scientific Data

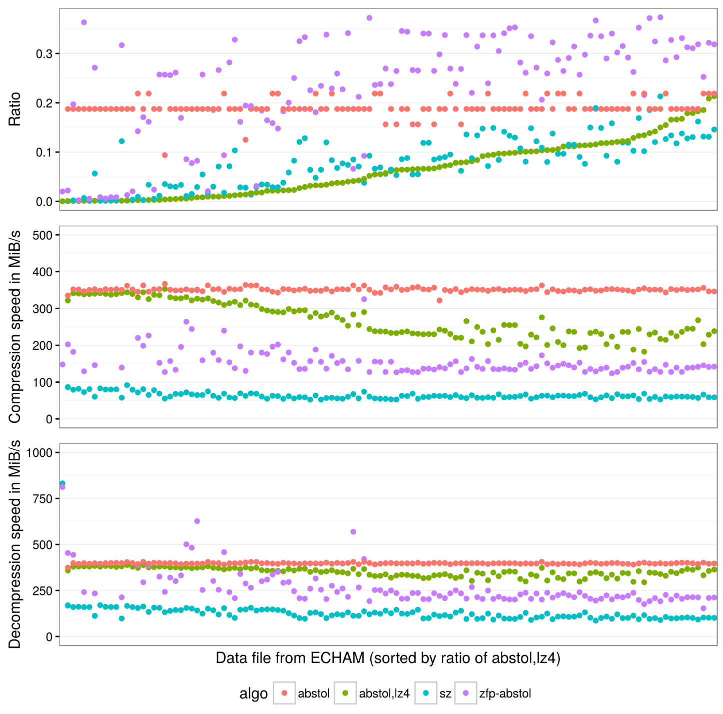
	Algorithm	Ratio	Compr.	Decomp.
			MiB/s	MiB/s
ECHAM	abstol	0.190	260	456
	abstol,lz4	0.062	196	400
	sz	0.078	81	169
	zfp-abstol	0.239	185	301
Isabel	abstol	0.190	352	403
	abstol,lz4	0.029	279	356
	sz	0.016	70	187
	zfp-abstol	0.039	239	428
Random	abstol	0.190	365	382
	abstol,lz4	0.194	356	382
	sz	0.242	54	125
	zfp-abstol	0.355	145	241

a) 1% absolute tolerance

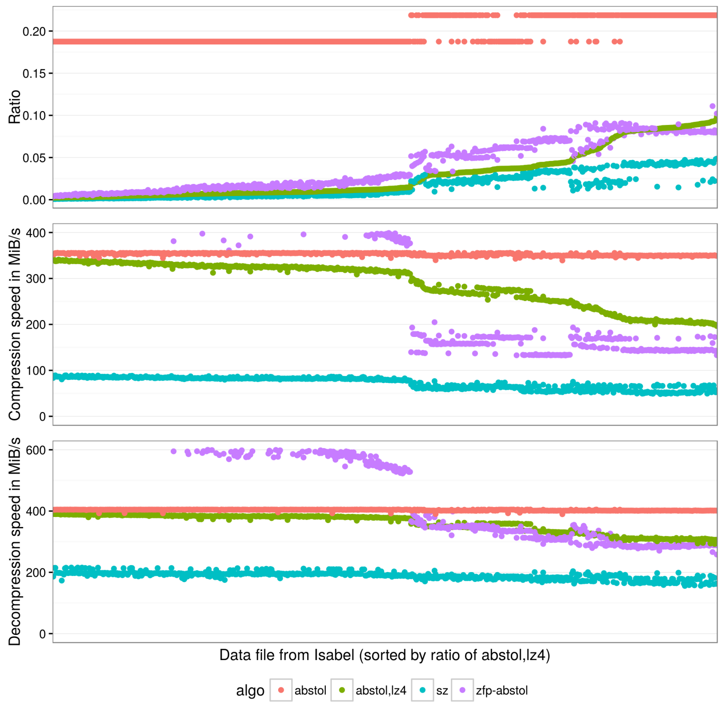
	Algorithm	Ratio	Compr.	Decomp.
			MiB/s	MiB/s
ECHAM	sigbits	0.448	462	615
	sigbits,lz4	0.228	227	479
	zfp-precision	0.299	155	252
Isabel	sigbits	0.467	301	506
	sigbits,lz4	0.329	197	366
	zfp-precision	0.202	133	281
Random	sigbits	0.346	358	511
	sigbits,lz4	0.348	346	459
	zfp-precision	0.252	151	251

b) 9 bits precision

Fixed Precision Bits Similarly to our previous experiment, we now aim to preserve 9 precision bits. mean values are given in Tbl. 2b and Fig. 9 shows the ratio and performance across



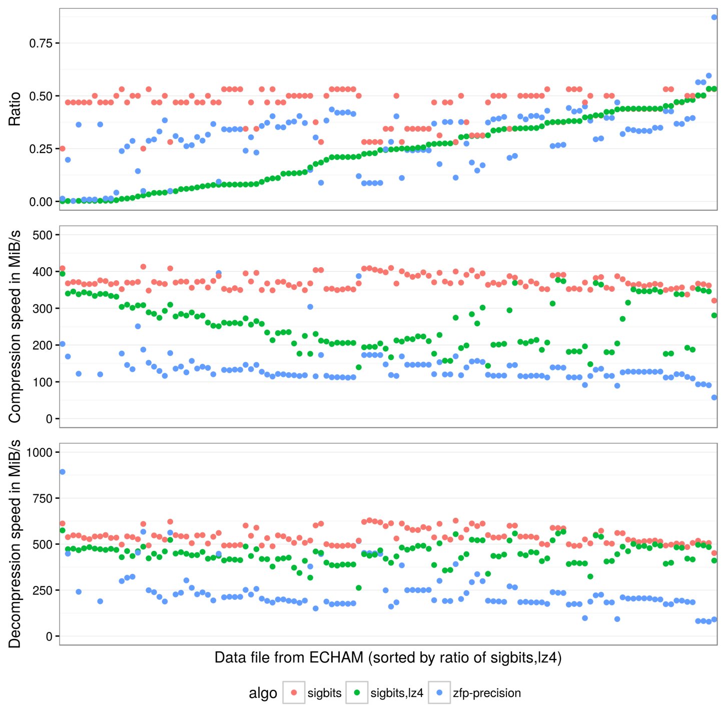
a) ECHAM



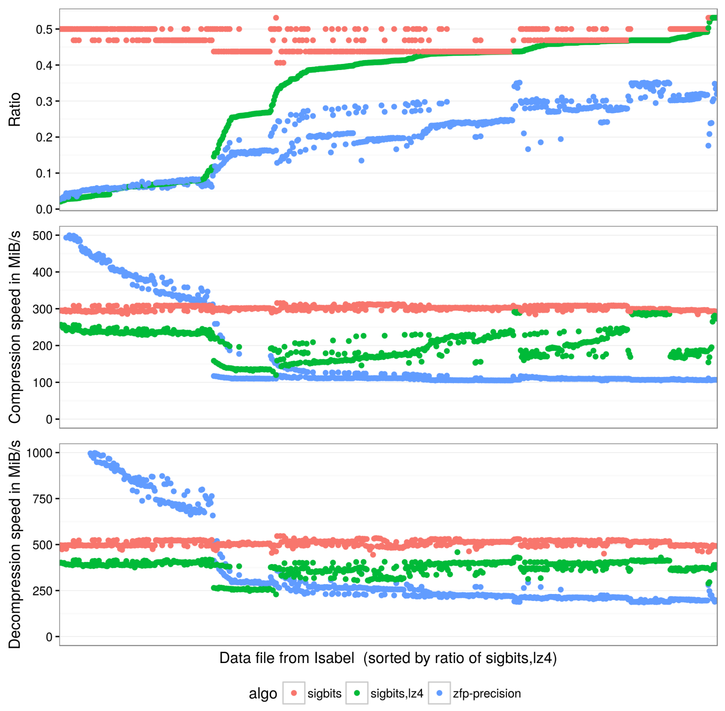
b) Isabel

Figure 8. Compressing climate data variables with an absolute tolerance of 1% max

climate variables. The Sigbits algorithm is generally a bit faster than Abstol. It can be seen that Sigbits+LZ4 outperforms ZFP mostly for ECHAM data, although ZFP does typically not hold the defined tolerance. For Isabel data ZFP is typically better than Sigbits+LZ4 (but again does not hold the precision bits). In this case, the LZ4 compression step is beneficial for a small fraction of files, for 50% it does not bring any benefit.



a) ECHAM



b) Isabel

Figure 9. Compressing climate data variables with 9 precision bits

Summary

This paper describes the concepts for the scientific compression library (SCIL) and compares algorithms with the state-of-the-art compressors ZFP and SZ. We investigate various relevant synthetic test patterns in more detail. As expected the predictors of SZ and ZFP improve the ratio for continuous data, for data containing repeated patterns or with more randomness, Abstol can yield a comparable or better ratio. The structure of the data depends on the scientific meaning of the variable and the scientist setting up the experiment: the output may be a continuous high resolution variable that is highly predictable or a downsampled version, it may contain not differentiable data potentially mixed with extreme values (e.g., fill values that mask points). The difference is relevant since the pattern that repeats a single value within 3D data block, the compression ratios of all algorithms are behind the expectation. Instead of storing 10x10x10 blocks of the same value, a single point could be stored for each block allowing to reduce the data volume to 1/1000 while the best algorithm here only yields a reduction to 1/33.

In any case, the performance of Sigbits and Abstol are better than existing algorithms. Since SCIL aims to choose the best algorithm automatically, it ultimately should be able to take benefit of both algorithms. We believe that decoupling of the interface to define scientific precision from the actual algorithm is a mandatory step for the HPC community to move on. Ongoing work is the development of an algorithm honoring all quantities at the same time and the automatic chooser for the best algorithm.

Acknowledgements

This work was supported in part by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) (GZ: LU 1353/11-1). Thanks for the Hurricane Isabel data produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation (NSF).

References

1. Baker, A.H., Hammerling, D.M., Mickelson, S.A., Xu, H., Stolpe, M.B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Gencarelli, C.N., Dennis, J.M., Kay, J.E., Lindstrom, P.: Evaluating lossy data compression on climate simulation data within a large ensemble. Geoscientific Model Development, 9 pp. 4381–4403 (2016), DOI: 10.5194/gmd-9-4381-2016

2. Bautista-Gomez, L.A., Cappello, F.: Improving floating point compression through binary masks. In: Hu, X., Lin, T.Y., Raghavan, V.V., Wah, B.W., Baeza-Yates, R.A., Fox, G.C., Shahabi, C., Smith, M., Yang, Q., Ghani, R., Fan, W., Lempel, R., Nambiar, R. (eds.) Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA. pp. 326–331. IEEE (2013), DOI: 10.1109/BigData.2013.6691591

3. Bicer, T., Agrawal, G.: A Compression Framework for Multidimensional Scientific Datasets.

Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013 IEEE 27th International pp. 2250–2253 (2013), DOI: 10.1109/IPDPSW.2013.186

4. Di, S., Cappello, F.: Fast Error-bounded Lossy HPC Data Compression with SZ. In: Parallel and Distributed Processing Symposium, 2016 IEEE International. pp. 730–739. IEEE (2016), DOI: 10.1109/IPDPS.2016.11

5. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40(9), 1098–1101 (1952), DOI: 10.1109/JRPROC.1952.273898

6. Hübbe, N., Wegener, A., Kunkel, J., Ling, Y., Ludwig, T.: Evaluating Lossy Compression on Climate Data. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) Supercomputing. pp. 343–356. No. 7905 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2013), DOI: 10.1007/978-3-642-38750-0_26

7. Hübbe, N., Kunkel, J.: Reducing the HPC-Datastorage Footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression. Computer Science - Research and Development pp. 231–239 (2013), DOI: 10.1007/s00450-012-0222-4

8. Iverson, J., Kamath, C., Karypis, G.: Fast and effective lossy compression algorithms for scientific datasets, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7484, pp. 843–856. Springer (2012), DOI: 10.1007/978-3-642-32820-6_83

9. Kunkel, J.: Analyzing Data Properties using Statistical Sampling Techniques – Illustrated on Scientific File Formats and Compression Features. In: Taufer, M., Mohr, B., Kunkel, J. (eds.) High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P3MA, VHPC, WOPSSS. pp. 130–141. No. 9945 2016 in Lecture Notes in Computer Science, Springer (2016), DOI: 10.1007/978-3-319-46079-6_10

10. Kunkel, J., Novikova, A., Betke, E., Schaare, A.: Toward Decoupling the Selection of Compression Algorithms from Quality Constraints. In: High Performance Computing. No. 10524 in Lecture Notes in Computer Science, Springer (2017), DOI: 10.1007/978-3-319-67630-2_1

11. Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D.S., Lewis, J.P., Perlin, K., Zwicker, M.: A survey of procedural noise functions. In: Computer Graphics Forum. vol. 29, pp. 2579–2600. Wiley Online Library (2010), DOI: 10.1111/j.1467-8659.2010.01827.x

12. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.: Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-Temporal Data. European Conference on Parallel and Distributed Computing (Euro-Par), Bordeaux, France (2011), DOI: 10.1007/978-3-642-23400-2_34

13. Laney, D., Langer, S., Weber, C., Lindstrom, P., Wegener, A.: Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics. Super Computing (2013), DOI: 10.3233/SPR-140386

14. Lindstrom, P.: Fixed-Rate Compressed Floating-Point Arrays. IEEE Transactions on Visualization and Computer Graphics 2012 (2014), DOI: 10.1109/BigData.2013.6691591

15. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. IEEE transactions on visualization and computer graphics 12(5), 1245–1250 (2006), DOI: 10.1109/TVCG.2006.143

16. Roeckner, E., Bäuml, G., Bonaventura, L., Brokopf, R., Esch, M., Giorgetta, M., Hagemann, S., Kirchner, I., Kornblueh, L., Manzini, E., et al.: The atmospheric general circulation model ECHAM 5. PART I: Model description (2003)

17. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on information theory 23(3), 337–343 (1977), DOI: 10.1109/TIT.1977.1055714