

Toward Decoupling the Selection of Compression Algorithms from Quality Constraints

Julian Kunkel¹(✉), Anastasiia Novikova², Eugen Betke¹,
and Armin Schaare²

¹ Deutsches Klimarechenzentrum, Hamburg, Germany
kunkel@dkrz.de

² Universität Hamburg, Hamburg, Germany

Abstract. Data intense scientific domains use data compression to reduce the storage space needed. Lossless data compression preserves the original information accurately but on the domain of climate data usually yields a compression factor of only 2:1. Lossy data compression can achieve much higher compression rates depending on the tolerable error/precision needed. Therefore, the field of lossy compression is still subject to active research. From the perspective of a scientist, the compression algorithm does not matter but the qualitative information about the implied loss of precision of data is a concern.

With the Scientific Compression Library (SCIL), we are developing a meta-compressor that allows users to set various quantities that define the acceptable error and the expected performance behavior. The ongoing work a preliminary stage for the design of an automatic compression algorithm selector. The task of this missing key component is the construction of appropriate chains of algorithms to yield the users requirements. 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 ground characteristics of tolerable noise, from the task of determining an optimal compression strategy given target noise levels and constraints. Future algorithms are used without change in the application code, once they are integrated into SCIL.

In this paper, we describe the user interfaces and quantities, two compression algorithms and evaluate SCIL's ability for compressing climate data. This will show that the novel algorithms are competitive with state-of-the-art compressors ZFP and SZ and illustrate that the best algorithm depends on user settings and data properties.

1 Introduction

Climate science is data intense. For this reason, the German Climate Computing Center spends a higher percentage of money on storage compared to compute. 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 proper preconditioning and algorithm, a compression factor of roughly 2.5:1 can be achieved with lossless compression, i.e., without loss of information/precision [1]. 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 [2] to estimate the actual compression factor that can be achieved on our system, we saw that LZ4fast yield a compression ratio¹ 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 factors 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.

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 library² 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.

Contributions of this paper are: (1) Introduction of user defined quantities data precision and performance; (2) Description of two new lossy compression algorithms; (3) The analysis of lossy compression for climate data.

This paper is structured as follows: We give a review over related work in Sect. 2. The design is described in Sect. 3. An evaluation of the compression ratios is given in Sect. 4. Finally, in Sect. 5 a summary is provided.

2 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 [3] 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 [4] is a variation of

¹ We define compression ratio as $r = \frac{\text{size compressed}}{\text{size original}}$; inverse is the compr. factor.

² The current version of the library is publicly available under LGPL license: <https://github.com/JulianKunkel/scil>.

LZ77 and uses Huffman coding [5]. GZIP [6] is a popular lossless algorithm based on DEFLATE.

Lossy algorithms for floating point data: FPZIP [7] 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 [7] for up to 3 dimensional data is accurate within machine epsilon in lossless mode. The dimensionality is insufficient for the climate scientific data. SZ [8] is a newer and effective HPC data compression method. Its compression ratio is at least 2x better than the second-best solution of ZFP. In [1], 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 [9], two lossy compression algorithms (GRIB2, APAX) were evaluated regarding to loss of data precision, 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 [10–15]. The first efforts for determination of appropriate levels of precision for lossy compression method were presented in [16]. 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 [2], 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 tool³ that can be used to investigate large data sets.

3 Design

The main goals 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 chosen⁴. SCIL also comes with a pattern library to generate various relevant synthetic test patterns. Further tools are provided to plot, add noise or to compress CSV and NetCDF3 files. Internally, support functions simplify the development of new algorithms and the testing.

³ <https://github.com/JulianKunkel/statistical-file-scanner>.

⁴ The implementation for the automatic algorithm selection is ongoing effort and not the focus of this paper. SCIL will utilize a model for performance and compression ratio for the different algorithms, data properties and user settings.

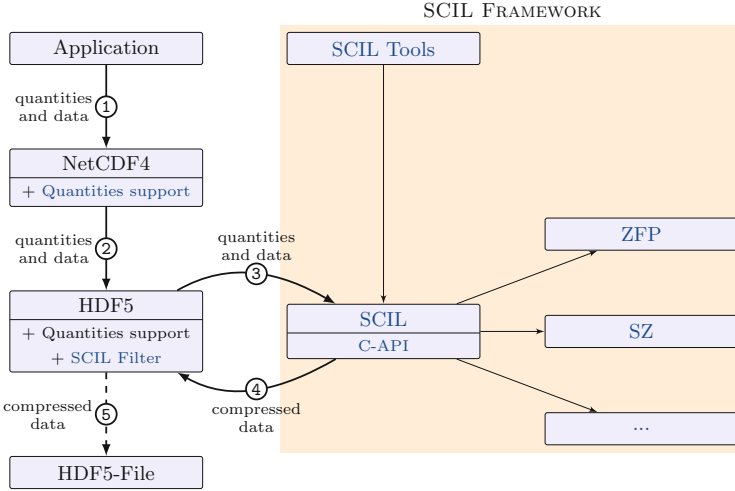


Fig. 1. SCIL compression path and components

3.1 Supported Quantities

The tolerable error on lossy compression and the expected performance behavior can be defined. Quantities define the properties of the residual error ($r = v - \hat{v}$):

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

Additional, the performance behavior can be defined 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. Thus, SCIL must estimate the compression rates for the data. The system's performance must be trained for each system using machine learning. An example for using the low-level C-API:

```

1  #include <scil.h>
2  int main(){
3      double data[10][20]; // our raw data, we assume it contains sth. useful
4
5      // define the quantities as hints, all specified conditions will hold
6      scil_user_hints_t hints;
7      hints.relative_tolerance_percent = 10;
8      hints.absolute_tolerance = 0.5;
9      hints.significant_digits = 2;
10     // define permformance limit on decompression speed
11     hints.decomp_speed.unit = SCIL_PERFORMANCE_GIB;
12     hints.decomp_speed.multiplier = 3.5;
13     // ... add more limitations if desired
14     // create a compression context for a given datatype

```

```

15  scil_context_t* ctx;
16  scil_create_context(&ctx, SCIL_TYPE_DOUBLE, 0, NULL, &hints);
17
18  // the multi-dimensional size of the data, here 10x20
19  scil_dims_t dims; scil_initialize_dims_2d(& dims, 10, 20);
20
21  // the user is responsible to allocate memory for the output/tmp
   buffers
22  size_t buffer_size = scil_get_compressed_data_size_limit(& dims,
   SCIL_TYPE_DOUBLE);
23  byte * compressed_data = malloc(buffer_size);
24
25  size c_size; // will hold the number of bytes of the compressed buffer
26  scil_compress(compressed_data, buffer_size, data, &dims, &c_size, ctx);
27  // now do something with the data in compressed_data

```

3.2 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.

Abstol. This algorithm guarantees the defined absolute tolerance. Pseudocode for the Abstol algorithm:

```

1  compress(data, abstol, outData){
2  (min,max) = computeMinMax(data)
3  // quantize the data converting it to integer, according to abstol
4  tmp[i] = round((data[i] - min) * abstol)
5  // compute numbers of mantissa bits needed to store the data
6  bits = ceil(log2(1.0 + (max - min) / abstol))
7  // now pack the necessary bits from the integers tightly
8  outData = packData(tmp, bits)
9  }

```

Sigbits. This algorithm preserves the user-defined number of precision bits from the floating point data. 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. Note that the sign bit must only be preserved, if it is not constant in the data. Pseudocode for the Sigbits algorithm:

```

1  compress(data, precisionBits, outData){
2  // preserve the exponent always
3  (sign, min, max) = computeExponentMinMax(data)
4  // compute numbers of bits needed to preserve the data
5  bits = sign + bits for the exponent + precisionBits - 1
6  // convert preserved bits into an integer using bitshift operators
7  tmp[i] = sign | exponent range used | precision Bits
8  // now pack the bits tightly
9  outData = packData(tmp, bits)
10 }

```

3.3 Compression Chain

Internally, SCIL creates a process which can involve several compression algorithms. Algorithms may be preconditioners to optimize data layout for subsequent compression algorithms, converters from one data format to another, or, on the final stage, a lossless compressor. Floating point data can be first converted into integer data and then into a byte stream. Intermediate steps can be skipped. Based on the basic datatype that is supplied, the initial stage of the chain is entered. Figure 2 illustrates the chain.

3.4 Tools

SCIL comes with tools useful for evaluation and analysis: (1) To create 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.

4 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 @ 3.40 GHz.

4.1 Test Data

A pool of (single precision floating point) data is created from several synthetic patterns generated by SCIL’s pattern library such as constant, random, linear steps, polynomial, sinusoidal or by the OpenSimplex [17] algorithm. An example is given for the Simplex data in Fig. 3; original data and the compressed data for the Sigbits algorithm preserving 3 bits from the mantissa.

Additionally, utilize the output of the ECHAM atmospheric model [18] which stored 123 different scientific variables for a single timestep as NetCDF. This scientific data varies in terms of properties and in particular, the expected data locality. Synthetic data are kept in CSV-files.

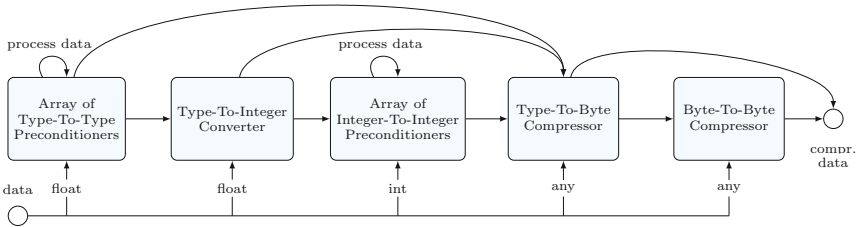


Fig. 2. SCIL compression chain. The choice of blocks and the resulting data path depend on input data.

4.2 Experiments

For each of the test files, the following setups are run⁵:

- Lossy compression preserving T significant bits
 - Tolerance: 3, 6, 9, 15, 20 bits
 - Algorithms: zfp, sigbits, sigbits+lz4⁶
- Lossy compression with a fixed absolute tolerance
 - Tolerance: 10%, 2%, 1%, 0.2%, 0.1% of the data maximum value⁷
 - Algorithms: zfp, sz, abstol, abstol+lz4

In each test, only one thread of the system is used for the compression/decompression. Each configuration is run 10 times measuring compression and decompression time and compression ratio.

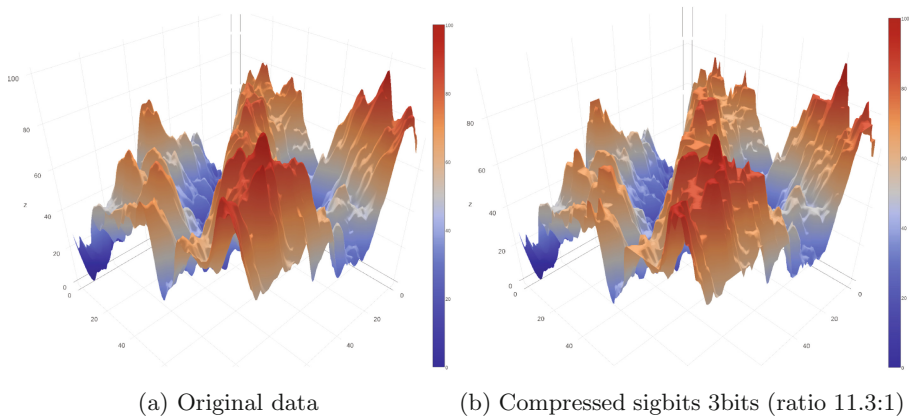


Fig. 3. Example synthetic pattern: Simplex 206 in 2D

4.3 Compression Ratio Depending on Tolerance

Firstly, we investigate the compression factor depending on the tolerance level. The graphs in Fig. 4 show the mean compression factor for all scientific 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 the compressed files occupy only 2% of the original size.

With 0.2% absolute tolerance, the compression ratio of abstol+lz4 is better than our target of 10:1; on average 3.2 bits are needed to store a single float. The SZ algorithm yields similar results than abstol+LZ4. The LZ4 stage boosts the factor for Abstol and Sigbits significantly.

⁵ The versions used are SZ from Mar 5 2017 (git hash e1bf8b), zfp 0.5.0, LZ4 (May 1 2017, a8dd86).

⁶ This applies first the Sigbits algorithm and then the lossless LZ4 compression.

⁷ 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.

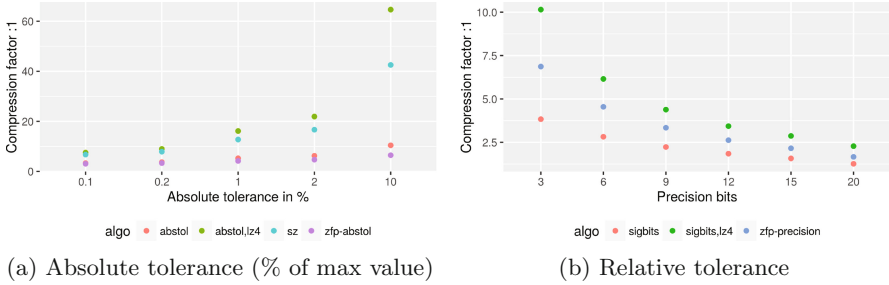


Fig. 4. Mean harmonic compression factor based on user settings

For the precision bits, when preserving three mantissa bits, roughly 9:1 could be achieved with sigbits+LZ4. Note that in roughly half the cases, ZFP could not hold the required precision, as it defines the number of bits for the output and not in terms of guaranteed precision⁸.

4.4 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 Table 1. Synthetic random patterns serve as baseline to understand the benefit of the lossy compression; we provide the means for 5 different random patterns. For abstol, a random pattern yields a ratio of 0.229 (factor of 4.4:1) and for climate data the ratio is slightly better. But when comparing SZ and Abstol+LZ4, we can observe a decrease of the compression ratio to 1/3rd of the random data. Compression speed is similar for random and climate data but decompression improves as there is less memory to read.

The results for the individual climate variables are shown in Fig. 5; the graph is sorted on compression ratio to ease identification of patterns. The x-axis represents the different data files, each point in the synthetic data represents one pattern of the given class created with different parameters. It can be observed that Abstol+LZ4 yields mostly the best compression ratio and the best compression and decompression speeds. For some variables, SZ compresses better, this is exactly the reason why SCIL should be able to automatically pick the best fitting algorithm below a common interface.

4.5 Fixed Precision Bits

Similarly to our previous experiment, we now aim to preserve 9 precision bits for the mantissa. The mean values are shown in Table 2. Figure 6 shows the ratio and performance across climate variables. The synthetic random patterns yield a compression factor of 2.6:1. It can be seen that Sigbits+LZ4 outperforms ZFP mostly, although ZFP does typically not hold the defined tolerance.

⁸ Even when we added the number of bits necessary for encoding the mantissa to ZFP.

Table 1. Harmonic mean compressing with an absolute tolerance of 1% max

Algorithm	Ratio	Compr. Decomp.		Algorithm	Ratio	Compr. Decomp.	
		MiB/s	MiB/s			MiB/s	MiB/s
abstol	0.19	260	456	abstol	0.194	265	482
abstol,lz4	0.062	196	400	abstol,lz4	0.151	226	456
sz	0.078	81	169	sz	0.165	74	147
zfp-abstol	0.239	185	301	zfp-abstol	0.295	161	266

(a) For ECHAM data files

(b) For 5 different random patterns

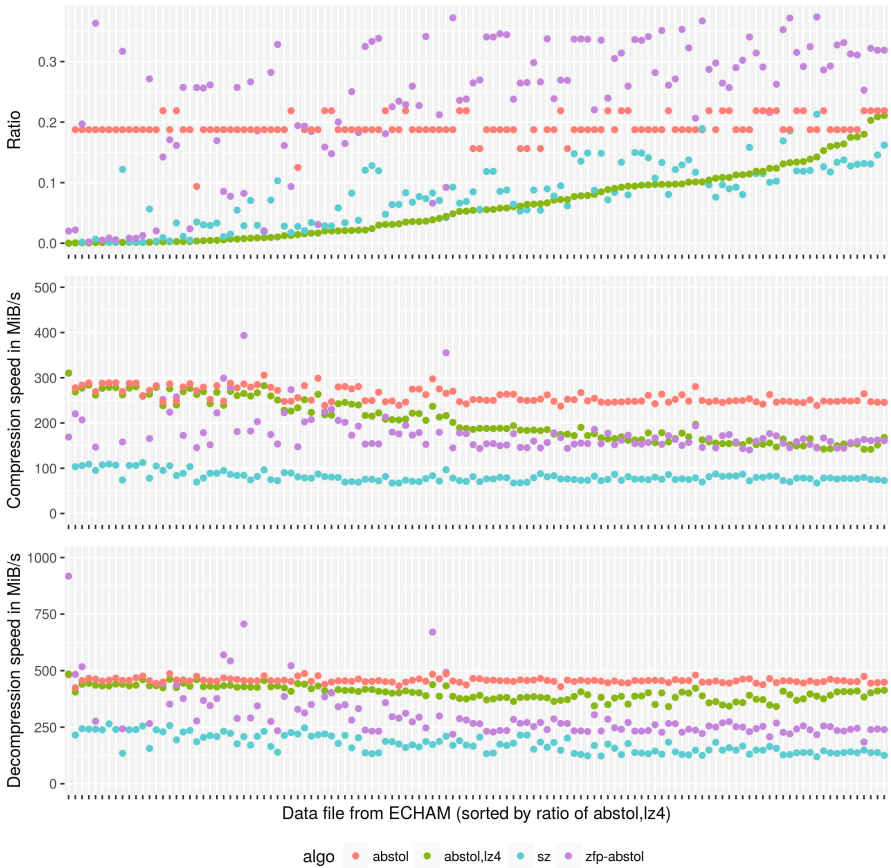


Fig. 5. Compressing various climate data variables with abstol of 1% max

Table 2. Harmonic mean compressing with 9 precision bits

Algorithm	Ratio	Compr. Decomp.		Algorithm	Ratio	Compr. Decomp.	
		MiB/s	MiB/s			MiB/s	MiB/s
sigbits	0.448	462	615	sigbits	0.369	528	672
sigbits,lz4	0.228	227	479	sigbits,lz4	0.304	466	599
zfp-precision	0.299	155	252	zfp-precision	0.232	175	314

(a) For ECHAM data files

(b) For 5 different random patterns

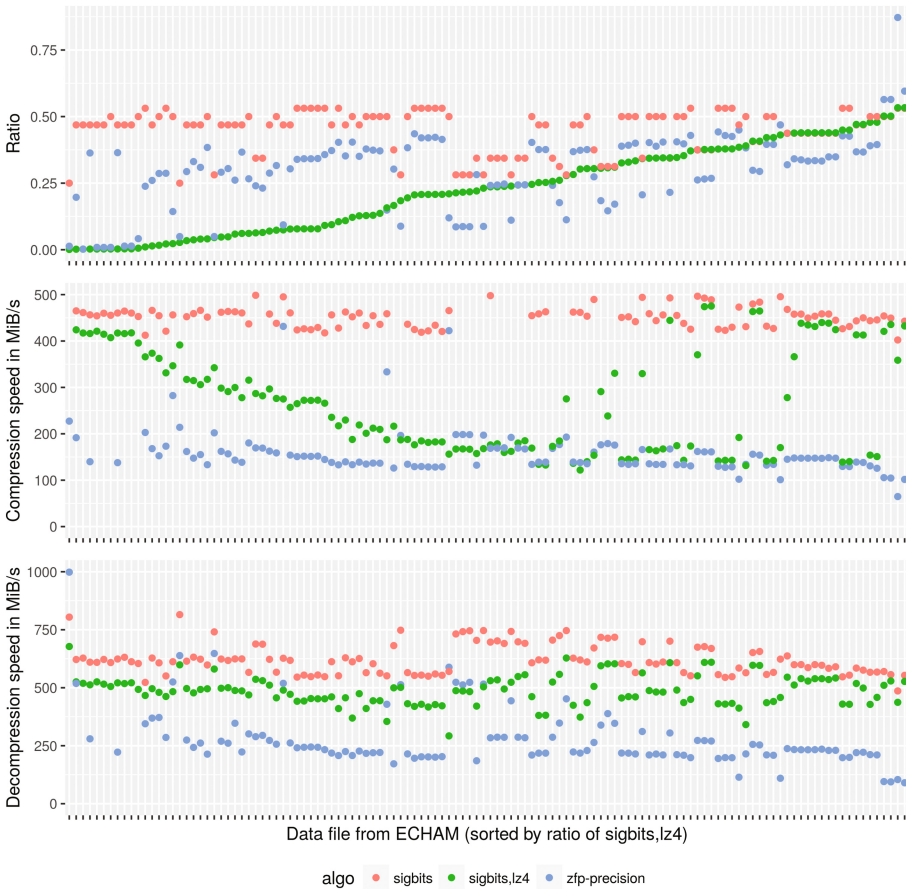


Fig. 6. Compressing various climate data variables with 9 Bits precision

5 Summary

This paper introduces the concepts for the scientific compression library (SCIL) and compares novel algorithms implemented with the state-of-the-art compressors. It shows that these algorithms can compete with ZFP/SZ when setting the absolute tolerance or precision bits. In cases with steady data, SZ compresses better than abstol. Since SCIL aims to choose the best algorithm, it ultimately should be able to take benefit of both algorithms. Ongoing work is the development of a single algorithm honoring all quantities 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).

References

1. Hubbe, 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)
2. 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*, 130–141. Number 9945 2016 in *Lecture Notes in Computer Science*. Springer, Heidelberg (2016)
3. LZ77. <https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/lz77/example.htm>. Accessed 04 Oct 2016
4. DEFLATE algorithm. <https://en.wikipedia.org/wiki/DEFLATE>. Accessed 04 Oct 2016
5. Huffman coding. A Method for the Construction of Minimum-Redundancy Codes. Accessed 04 Oct 2016
6. GZIP algorithm. <http://www.gzip.org/algorithm.txt>. Accessed 04 Oct 2016
7. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. *IEEE Trans. Visual Comput. Graphics* **12**(5), 1245–1250 (2006)
8. Di, S., Cappello, F.: Fast error-bounded lossy HPC data compression with SZ (2015)
9. Hübbe, N., Wegener, A., Kunkel, J.M., Ling, Y., Ludwig, T.: Evaluating lossy compression on climate data. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) *ISC 2013. LNCS*, vol. 7905, pp. 343–356. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38750-0_26](https://doi.org/10.1007/978-3-642-38750-0_26)
10. Bicer, T., Agrawal, G.: A compression framework for multidimensional scientific datasets. In: *2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, pp. 2250–2253 (2013)
11. 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)

12. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: Compressing the incompressible with isabela: in-situ reduction of spatio-temporal data. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011. LNCS, vol. 6852, pp. 366–379. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-23400-2_34](https://doi.org/10.1007/978-3-642-23400-2_34)
13. Iverson, J., Kamath, C., Karypis, G.: Fast and effective lossy compression algorithms for scientific datasets. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 843–856. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-32820-6_83](https://doi.org/10.1007/978-3-642-32820-6_83)
14. Gomez, L.A.B., Cappello, F.: Improving floating point compression through binary masks. In: 2013 IEEE International Conference on Big Data (2013)
15. Lindstrom, P.: Fixed-Rate Compressed Floating-Point Arrays. IEEE Trans. Visualization Comput Graphics 2012 (2014)
16. Baker, A.H., et al.: Evaluating lossy data compression on climate simulation data within a large ensemble. Geosci. Model Dev. **9**, 4381–4403 (2016)
17. OpenSimplex Noise in Java. <https://gist.github.com/KdotJPG/b1270127455a94ac5d19>. Accessed 05 Feb 2017
18. Roeckner, E., Bäuml, G., Bonaventura, L., Brokopf, R., Esch, M., Giorgetta, M., Hagemann, S., Kirchner, I., Kornbluh, L., Manzini, E., et al.: The Atmospheric General Circulation Model ECHAM 5. Model description, PART I (2003)