Data Encoding in Lossless Prediction-Based Compression Algorithms

Ugur Cayoglu[†*], Frank Tristram[‡], Jörg Meyer[*], Jennifer Schröter[†],
Tobias Kerzenmacher[†], Peter Braesicke[†], and Achim Streit[*]
[*]Steinbuch Centre for Computing
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen
Email: {Ugur.Cayoglu, Joerg.Meyer2, Achim.Streit}@kit.edu
[†]Institute of Meteorology and Climate Research - Atmospheric Trace Gases and Remote Sensing
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen
Email: {Jennifer.Schroeter, Peter.Braesicke, Tobias.Kerzenmacher}@kit.edu
[‡]Institute of Applied Physics (APH) - 3D Matter Made to Order (3DMM2O)
Karlsruhe Institute of Technology, Kaiserstraße 12, 76131 Karlsruhe
Email: Frank.Tristram@kit.edu

***Abstract*—The increase in compute power and development of sophisticated simulation models with higher resolution output triggers a need for compression algorithms for scientific data. Several compression algorithms are currently under development. Most of these algorithms are using prediction-based compression algorithms, where each value is predicted and the residual between the prediction and true value is saved on disk. Currently there are two established forms of residual calculation: Exclusive-or and numerical difference. In this paper we will summarize both techniques and show their strengths and weaknesses. We will show that shifting the prediction and true value to a binary number with certain properties results in a better compression factor with minimal additional computational costs. This gain in compression factor allows for the usage of less sophisticated prediction algorithms to achieve a higher throughput during compression and decompression. In addition, we will introduce a new encoding scheme to achieve an 9% increase in compression factor on average compared to the current state-of-the-art.**

***Index Terms*—data compression, xor, lossless, lossy, climate data**

I. Introduction

With the advent of next-generation models and ever-increasing compute power on HPC clusters, climate scientists have achieved a breakthrough in high-resolution simulations that calculate global simulations with a horizontal resolution of 13 kilometres (e.g. ICON-ART [1]). These models produce an unprecedented volume of data, so that future studies are limited by storage capacity rather than numerical calculations. This constraint often forces the scientists to reduce the temporal resolution of their simulation output and use interpolation techniques for the missing time steps. As a result, the generated output is just an inferior representation of the actual model used for the simulation, which means, that the quality assessment of the model can only be performed with an inferior representation of limited output. Additionally, simulations might need to run more than once, when data is required for model assessments that have not been saved in the first instance. One method to tackle this problem is to remove redundant information in the data by compression and thus allow for a higher temporal resolution of the simulation output.

In this paper we will give an overview of currently applied encoding schemes in compression algorithms for floating-point data. We will introduce a novel encoding scheme to further improve either compression rate or throughput. We will show that shifting the prediction and true value to a binary number with certain properties results in a better compression factor with minimal additional computational costs. This gain in compression factor allows the usage of less sophisticated prediction algorithms to achieve a higher throughput.

The remainder of this paper is divided into five sections: An overview of current encoding schemes is presented in Section II. Related Work is described in Section III. Section IV describes our proposed method and metrics. Finally in Section V an evaluation of our proposed method is presented and discussed. In the concluding section we give a short summary and an outlook regarding future work.

II. Preliminaries

There are lossless and lossy compression algorithms. Lossless algorithms are used if the reconstructed data needs to be exactly the same as the input data. Lossy algorithms are preferred if size reduction is the biggest concern and a small information loss is acceptable. Both types of algorithms work by first decorrelating and then encoding the data. The decorrelation step eliminates redundancy in the data being it autocorrelated or cross correlated information. The encoding step is where the actual compression happens and the data is written on disk. A lossy compression algorithm has additionally an approximation step in between to further align the data for encoding. The approximation step reduces the complexity of the data for example by using methods of quantized representation for data points.

Since most of the following algorithms use a prediction-based compression algorithm, it might be helpful to remind the

reader about the steps involved. A prediction-based algorithm traverses the data in a predefined path. During this traversal each data point is visited exactly once. For each of these data points the most probable value is predicted (further on referred to as the prediction value p). After p has been calculated it is being compared with the true value t for that data point. The residual $r = \text{diff}(p, t)$ is then encoded and saved on disk. Since the traversal path and calculation of p is predefined by the compression algorithm, the value t can be reproduced by saving r. Although the calculation of a good predictive value p can be decisive for a good compression algorithm, this assessment is beyond the scope of this paper. Please refer to [2] for a comparison of prediction models. In this paper we study the influence of different encoding steps and assume that the same prediction model has been used by all algorithms.

The difficulty with floating-point data is the potentially infinite candidate space for p. While compression of textual data uses a 26 letter alphabet, the number of words of length l occurring in an English text is rather limited. This is not the case for numerical, especially floating-point data. Further, it is important to notice that the actual calculation is not being done using floating-point operations. Before the residual is calculated, both values are mapped to unsigned integers to guarantee reproducibility across hardware architectures[1]. Two different methods for residual calculations are currently established.

$$\text{diff}_{xor}(p, t) = p \oplus t \tag{1}$$

$$\text{diff}_{abs}(p, t) = \mid p - t \mid \tag{2}$$

Both methods have their strengths and weaknesses. The first approach (Eq. 1) uses a bitwise Exclusive-Or (XOR) for residual calculation. The advantage of this approach is that it is a very fast operation on modern hardware. Another advantage is that the reverse operation applied during decompression is literally the same operation with $t = p \oplus r$. No further information is needed to be transferred between encoder and decoder. The disadvantage of XOR is that two numbers representing very close values can still produce a very large residual. This is due to the two's-complement binary representation of floating-point numbers. This pitfall can be observed if p and t are close and on the opposite side of a power of two e.g. $p = 256.321$ and $t = 255.931$. While the difference is 0.39 the residual calculated using Eq. 1 is $r = 16762689$ and using Eq. 2 is $r = 15041$. These residuals need to be saved on disk. The former residual needs 24 bits to be represented and the latter only 14.

The second approach (Eq. 2) uses the absolute difference of the two numbers, or in other words, it represents the amount of binary numbers between the two numbers using two's-complement binary representation. Since the accuracy of two's-complement binary representation of a value is limited, the value range of the compared values plays a central role in residual calculation using this approach. Given $p = 847,390.837$ and $t = 847,794.417$ the difference is 403.58, but the residual r is 6458 (Eq. 1). Despite the fact that the prediction $p = 847,390.837$ (compared with $t = 847,794.417$) seems to be worse than $p = 256.321$ (compared with $t = 255.931$) at first glance, it is still considered a better prediction using Eq. 1 given two's-complement binary representation of floating-point values.

The advantage of this approach is that the resulting residual is always smaller or at worst the same size as the residual calculated by Eq. 1. The disadvantage is that one needs to store additional information about the prediction p. Without the information if p is above or below t a successful decompression can not happen. Another disadvantage is that to avoid an overflow or underflow one cannot calculate r in a single computation step like an XOR. This must be split into two steps by first calculating $\max(p, t)$ and then subtracting the smaller one from the bigger. This is especially important if the values are close to the smallest or biggest representable numbers using single or double precision floating-point numbers.

In the following section we will describe related work and current state-of-the-art approaches for residual calculation and data encoding.

III. Related work

In the last couple of years the development of compression algorithms for floating-point data experienced a renaissance. Several new methods were introduced [2]–[14] focusing on structural dependencies in gridded floating-point data. Most of these compression algorithms use a prediction-based compression.

For brevity, we will only describe the most relevant methods in this section based on citation count and effectiveness of method. Please refer to the appropriate papers for more details. Most of the methods use XOR residual calculation. They encode the Leading Zero Count (LZC) of the residual and save the remainder of the residual verbatim on disk. If the residual is small, it will have a high LZC. This LZC is then encoded using Huffman [15] or Arithmetic Encoding [16] which further reduces the file size.

Residuals calculated using Eq. 2 are similarly encoded as the ones using XOR. Here, too, the LZC is calculated and encoded. Additionally, a single bit is needed to represent if p is below or above t. This information is saved verbatim on disk like the remainder of the residual.

One exception of these approaches is given in Lindstrom et al. [11]. Here the authors merge both information (LZC and position of p relative to t) into a single value by adding or subtracting the LZC to 32 (in case of single precision floating-point data) or 64 (in case of double precision floating-point data). While this approach elegantly circumvents the additional bit needed for the position of p relative to t, it also increases the alphabet of the Range Encoding [16] used by the authors. But nonetheless this approach is highly effective and proofs as state-of-the-art in lossless compression of floating-point data

[1]While there are several methods to do this, these methods are beyond the scope of this paper. For the following we assume a simple mapping of the raw binary representation from floating-point values to unsigned integers.

regarding the balance between compression factor and data throughput.

In the following section we will introduce a novel algorithm for calculating and encoding the residual using XOR residual calculation.

IV. Proposed Method

We compare two different residual calculation and encoding schemes in this paper. Both methods are depicted in Figure 1. As we highlighted in the previous section, the current state-of-the-art lossless compression algorithm for floating-point data is `fpzip` [11]. Its residual calculation and encoding scheme is highlighted with a dark grey background and from now on referred to as `fpzip`. Our proposed algorithm is highlighted using a light grey background and further referred to as `pzip`.

The first step in our proposed algorithm is to shift p and t to another value range which is more suitable for the difference calculation. Then the actual difference calculation is applied using the XOR method (Eq. 1). For our investigation, the residual is then split into three streams. Two of these are first transformed using the Burrow-Wheeler-Transform (BWT) and then written on disk using Range Encoding (RE). The third and final stream is saved verbatim on disk.

In the following section we will describe each of these steps in more detail.

A. Shifted XOR

In Section II we mentioned that the disadvantage of using XOR in residual calculation is that small differences between p and t might result with a large residual. An example was given with $p = 256.321$ and $t = 255.931$. The reason for this disadvantage can be seen if we look at the two's-complement binary representation of both values:

The XOR is defined as follows:

$$p \oplus t = (p \vee \neg t) \wedge \neg(p \wedge t) \tag{3}$$
$$= (p \wedge \neg t) \vee (\neg p \wedge t) \tag{4}$$

The bit at index i of $p \oplus t$ is set to 1, if the bit at index i of p is different than the bit of t at index i. In the example given in Figure 2 the XOR calculation has a small LZC of eight and a rather long following one count (FOC) of ten. Although the example is deliberately chosen so that the FOC is large, these instances occur very frequently when XOR is used. One advantage, however, is that extreme cases with very large FOC are well predictable. Due to the high number of zeros at positions 9-18, we can estimate that a possible bit flip is imminent and can act accordingly.

Our proposed residual calculation method adds two additional steps to the common method described in Section II. The first is to calculate a shift value s to be added to the prediction p so that $s = g(p) - p$ is satisfied, where $g(p)$ is defined by the following equation:

$$g(p) = \begin{cases} \sum_{k=1}^{15} 2^{2k-1} & \text{if } p < 2^{30} \\ \sum_{k=0}^{14} 2^{2k} & \text{if } 2^{30} \leq p < 2^{31} \\ \sum_{k=1}^{16} 2^{2k-1} & \text{if } 2^{31} \leq p < 2^{32} \\ \sum_{k=0}^{15} 2^{2k} & \text{if } 2^{32} \leq p \end{cases} \tag{5}$$

The binary representation of each of the goals is a fluctuation of zeros and ones. An example for the goal is the following:

$$g_1 = \sum_{k=1}^{15} 2^{2k-1} \text{ and } g_2 = \sum_{k=0}^{15} 2^{2k}$$
$$\text{bin}(g_1) = 00101010101010101010101010101010$$
$$\text{bin}(g_2) = 01010101010101010101010101010101$$

After the goal g has been identified and the necessary shift s calculated the same shift value will be added to the true value t. Following, the residual calculation proceeds as usual with applying the XOR operation to the shifted prediction $\hat{p}$ and shifted true value $\hat{t}$ to calculate the new residual $\hat{r}$:

$$\hat{r} = \hat{p} \oplus \hat{t} \tag{6}$$

The shift value can be recalculated without any information transfer between the encoder and decoder, since the decoder can recalculate the shift s with the available information.

B. Splitting of the Residual

In the next step, we split each residual into three components: LZC, FOC and the remainder of the residual. As mentioned before, the LZC specifies how many of the Most Significant Bits (MSB) are set to zero. Due to its definition, this block of zeros is always followed by a block of ones with size ≥ 1. The FOC determines the size of this block. The sum of LZC and FOC for a residual is restricted to 32 (64) for single (double) precision floating-point values. The remaining residual is then finally captured as the third component. Since we know that the bit following FOC will be zero, this bit will not be saved on disk.

Given the example in Section IV-A with $p = 256.321$ and $t = 255.931$ the following components are obtained for $p \oplus t$:

$$p \oplus t \ = 00000000111111111100011101000001$$
$$\text{LZC}(p \oplus t) = 8$$
$$\text{FOC}(p \oplus t) = 10$$
$$\text{RES}(p \oplus t) = 0011101000001$$

where RES represents the remaining residual as binary representation.

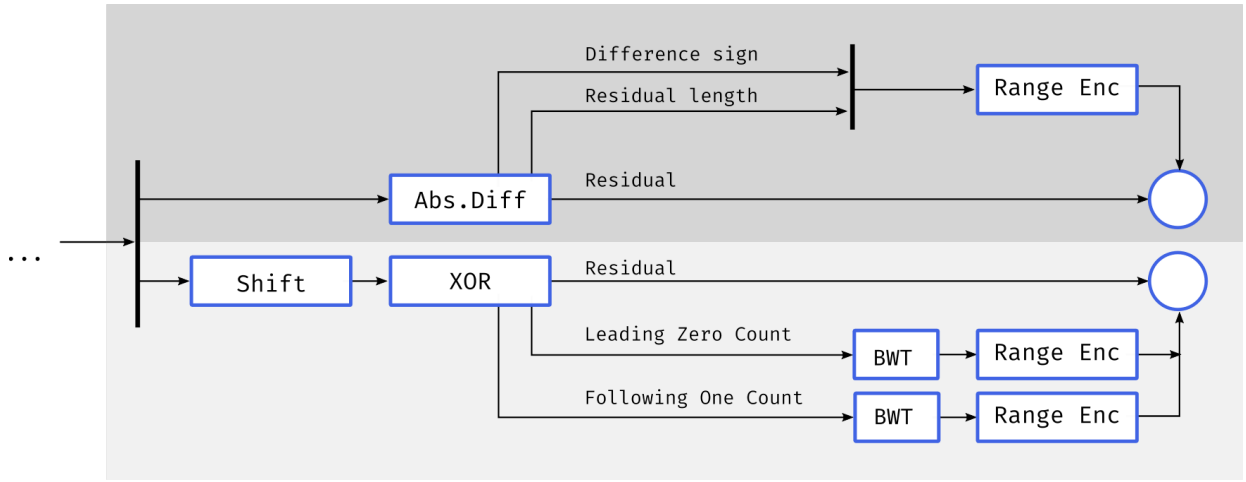

Fig. 1. Flowchart of the residual calculation and encoding phase of the current state of the art lossless compression algorithm for floating-point data (dark grey) and our proposed method (light grey). Since it is assumed that both methods use the same decorrelation steps (including the prediction model used for the calculation of p) the computation steps until the encoding step are not depicted.

```
bin(p=256.321) = 01000011100000000010100100010111
bin(t=255.931) = 01000011011111111111011001010110
         p ⊕ t = 00000000111111111100011101000001
                 index 1  3  5  7  9  11 13 15 17 19 21 23 25 27 29 31
```

Fig. 2. An example for the XOR residual calculation method.

C. Encoding of LZC/FOC

The LZCs and FOCs of the data is then reordered using BWT [17]. The BWT algorithm rearranges a given set of values in such a way, that same values are more likely to appear one after another than in the input. Finally the newly transformed set is encoded using RE. Since the LZC and FOC are independent from each other the BWT and RE can be performed concurrently for both sets.

D. Metrics

For evaluating the encoding algorithms we are using two metrics: compression factor (CF) and throughput. CF puts the file size before the compression and after the compression into relation:

$$\text{CF} = \frac{\text{Filesize before compression}}{\text{Filesize after compression}} \tag{7}$$

The higher the CF, the better the encoding algorithm.

The second quality measure for the encoding algorithms is the throughput, which indicates the amount of data processed per time unit. In our case we use either Bytes/s or MiB/s.

$$\text{throughput} = \frac{\text{Filesize [Bytes or MiB]}}{\text{Total Processing Time [sec]}} \tag{8}$$

E. Data

For the experiments described in Section V we are using two different datasets. The first dataset is a synthetic dataset generated using a Gaussian distribution with several mean and standard deviations. These data cover a broad spectrum of possible datasets, which might be compressed with both algorithms. The synthetic data was mainly used for the analysis of the XOR residual calculation when the data is close to a power of two (see Fig. 4 and 5).

The second dataset is obtained from a climate simulation performed using the ICON model [1]. The simulation was performed for an analysis of the POLar STRAtosphere in a Changing Climate (POLSTRACC) [18], [19] campaign, which performed flight measurements between December 2015 and March 2016. It consists of data with two different vertical resolutions. The datasets consists of a 901x351 (longitude, latitude) structured grid with 47 (respectively 90) vertical levels and four time steps (every six hours). The following variables were available as single precision floating-point values: geopotential, vertical velocity, potential vorticity, cloud water, cloud ice content, specific humidity, temperature, virtual temperature, zonal wind, vorticity and meridional wind.

F. Experiments

Several experiments were carried out to test the individual steps of the algorithm. First, the residual generated using XOR was analysed. We focused on the difficult case where the target value ranges around powers of two. The distribution of set and unset bits in the residual were analysed. This should give us clues as to whether the residual still contains redundant information or whether it is noise. Further we analysed the effects of the shifted XOR calculation regarding throughput and compression factor.

Another experiment was conducted to test different encoding schemes. For data transformation we used the Burrow-Wheeler-Transform [17] and Move-to-front [20]. Data encoding was performed using Range Encoding, Huffman Encoding and Run-length Encoding. These transformations and encoding schemes were applied on the original data as well as the difference (i.e. Δx) of the data.

Further, we analysed the throughput and timing of each step in our proposed compression algorithm. The final analysis, the

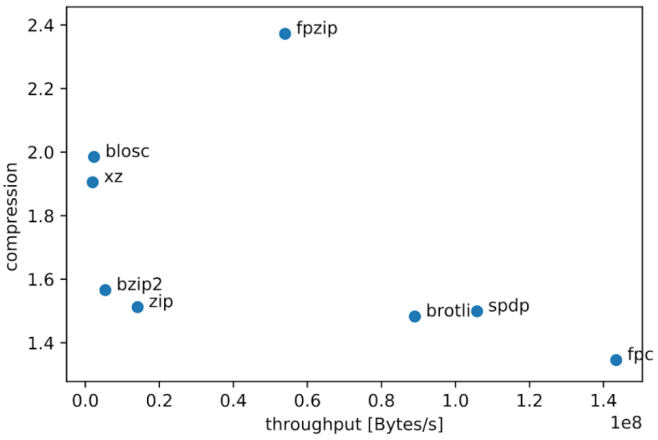

Fig. 3. Average compression factor and throughput of the climate simulation dataset with currently available compression algorithms.

compression rate and throughput as well as the complexity of the proposed compression algorithm were tested.

G. Implementation

All experiments were conducted on an Intel i5-7200U with 2.5 GHz running GNU/Linux 4.19.28 Debian with 16 GiB RAM. A native C implementation of `fpzip` was used. Our proposed algorithm was implemented in Rust 1.33.0-nightly.

V. Evaluation

This chapter has been divided into five sections: Each section describes an experiment performed to gain insights for each step of the compression process. The first experiment was conducted to assess currently available lossless compression algorithms using climate data. The next two experiments analyse more in depth the behaviour of XOR residual calculation and helps optimize its performance. The fourth experiment helps us assess available transformation and encoding steps. Finally we compare our proposed method with the state-of-the-art lossless compression algorithm `fpzip`.

A. State-of-the-art Compression Algorithms

In order to determine the currently best compression algorithm for floating-point data, we have run the climate simulation output with various current compression applications. The results are shown in Figure 3. The applied compression applications are: `blosc` [21], `fpzip` [11], `xz`, `bzip2`, `zip`, `brotli` [22], `spdp` [23], and `fpc` [24]. All algorithms were set in such a way to maximize compression factor.

Fig. 3 indicates that `fpzip` performed best with a compression factor close to 2.4. The runner-up was `blosc` with a compression factor of approximately 2.0. The throughput of `fpzip` was not as good as that of `brotli`, `spdp` or `fpc`, but taking into account the mediocre performance of these algorithms regarding compression factor, it can be seen that `fpzip` is currently the best performing algorithm regarding lossless compression of floating-point data.

TABLE I
Effects of using a shifted XOR with a more sophisticated prediction algorithm (Lorenz) and inferior one (Last Value) for selected climate simulation variables. Depicted are the Throughput and average LZC with and without (in parenthesis) shift operation.

Climate variable	Prediction	Throughput [MiB/s]	Avg. LZC
Temperature	Lorenz	19.86 (22.32)	24.01 (23.75)
	Last Value	31.33 (36.15)	22.89 (22.65)
Zonal wind	Lorenz	18.23 (18.68)	19.88 (19.61)
	Last Value	30.27 (37.49)	18.12 (17.83)
Geopotential	Lorenz	18.86 (19.92)	29.05 (28.85)
	Last Value	32.10 (37.91)	28.92 (28.83)

Observation 1: While there might be usage scenarios where `fpzip` is not the most successful compression algorithm, the results indicate that `fpzip` is on average the best performing algorithm for lossless compression of floating-point data regarding compression factor. Every future algorithm should measure itself against these results.

B. Shifted XOR

First we analyse the average LZC of a residual build with XOR for different Gaussian distributions (i.e. synthetic dataset). The results are plotted in Figure 4 and 5. The LZC breaks most severely when the average value is a power of two (marked by dotted lines). The intensity of these LZC dips is larger and stronger the smaller the standard deviation in the data. The closer the data points are to each other, the more important it is to get out of this range and perform the residual calculation in a different value range. The shift operation introduced by us is characterized by the solid vertical line. The shift achieves a higher average LZC and thus reduces the data to be compressed.

The shift can also help to increase throughput by using a simpler prediction model for compression and compensating for the weaknesses of the simpler prediction model with the shift. Table I shows that using a simple prediction model (Last Value) we can achieve a 50% higher throughput while still coming close to the LZC of the more sophisticated prediction method without shifted residual calculation[2].

Observation 2: The compression performance is dependent on the value ranges covered by the data and its distribution. A shifted residual calculation can improve the LZC and help reduce the data to be compressed.

Observation 3: The shifted XOR calculation improves the average LZC and therefore the compression factor. It allows the use of simpler prediction models for higher throughput with comparable average LZCs.

C. Splitting of the Residual

In the next experiment we will take a closer look at the distribution of set and unset bits in the residual. If there is

[2]For details about the Lorenz and Last Value prediction methods please refer to [2], [11]

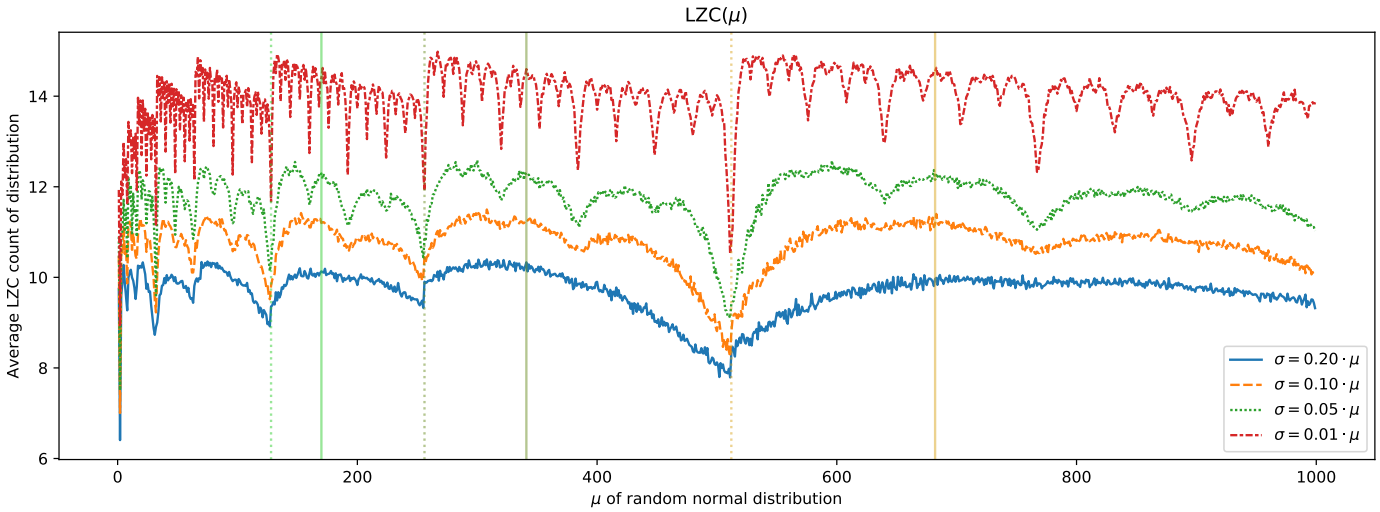

Fig. 4. Average LZC for datasets with Gaussian distribution using a σ dependent on μ. The coloured vertical lines represent the resulting LZC for powers of two with (solid line) and without (dotted line) a shifted XOR calculation as described in Section IV-A.

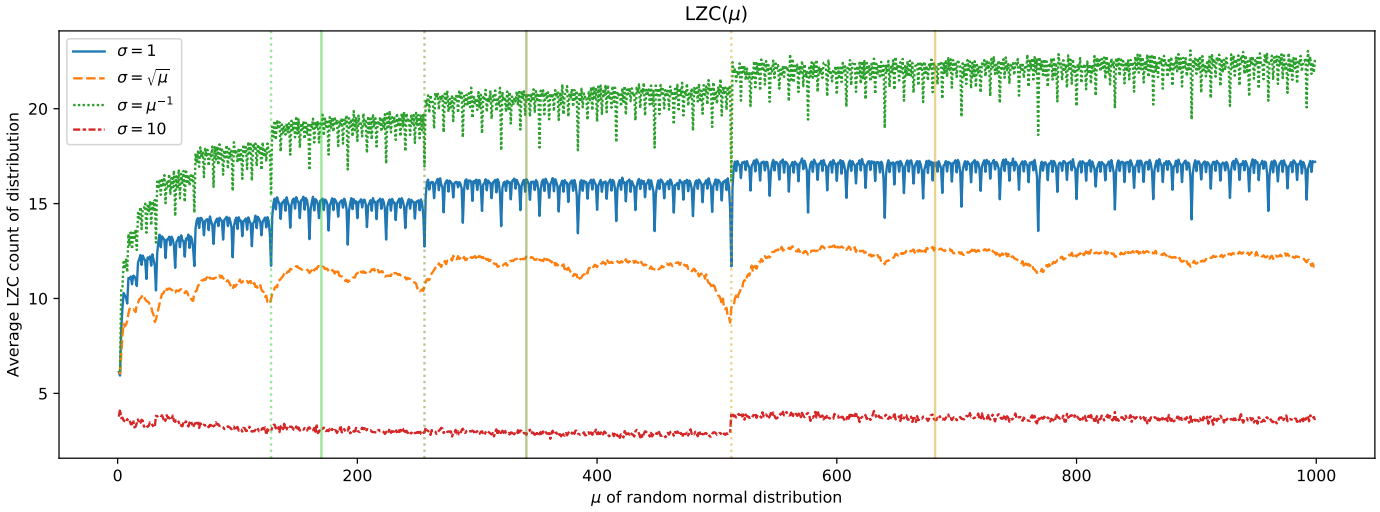

Fig. 5. Average LZC for datasets with Gaussian distribution using an independent σ. The coloured vertical lines represent the resulting LZC for powers of two with (solid line) and without (dotted line) a shifted XOR calculation as described in Section IV-A.

no information in the residual, we expect an even distribution of set and unset bits (i.e. white noise). Like in the previous experiment, we used the Gaussian distributed artificial dataset.

The results shown in Figure 6 indicate that the distribution of set and unset values is not uniform. The number of set bits at the most significant positions of the residual occur at a much higher rate than the number of unset bits. Even at a length of six bits there is a clear difference to the expected even distribution (i.e. horizontal line in figure). This unequal distribution is caused by the bitflips occurring due to the application of the XOR residual. Due to this observation we decided to extract this information using the FOC described in Section IV

Observation 4: Using XOR residual calculation there is a skewed distribution of the set and unset bits. This unequal distribution shows that there still remains some information in the residual and that the compression factor can be increased. A method to extract this information is to encode the number of FOCs separately, as proposed by us.

D. Performance of Encoding Methods

In this section we will take a closer look at different encoding scheme. these methods are: Burrow-Wheeler-Transform (bwt), Delta difference (diff), Huffman encoding (huff), Range encoding (range), Move-To-Front (mtf), and Run-length Encoding (rle). These encoding schemes are applied to the LZC and FOC. This experiment was conducted using the climate simulation output. The results[3] are shown in Table II.

The BWT transformation coupled with Range Encoding performs best regarding LZC. Regarding FOC it performs

[3]For reasons of brevity, we only show the results for temperature. While the actual values are different for each climate variable, the performance is similar for each climate variable

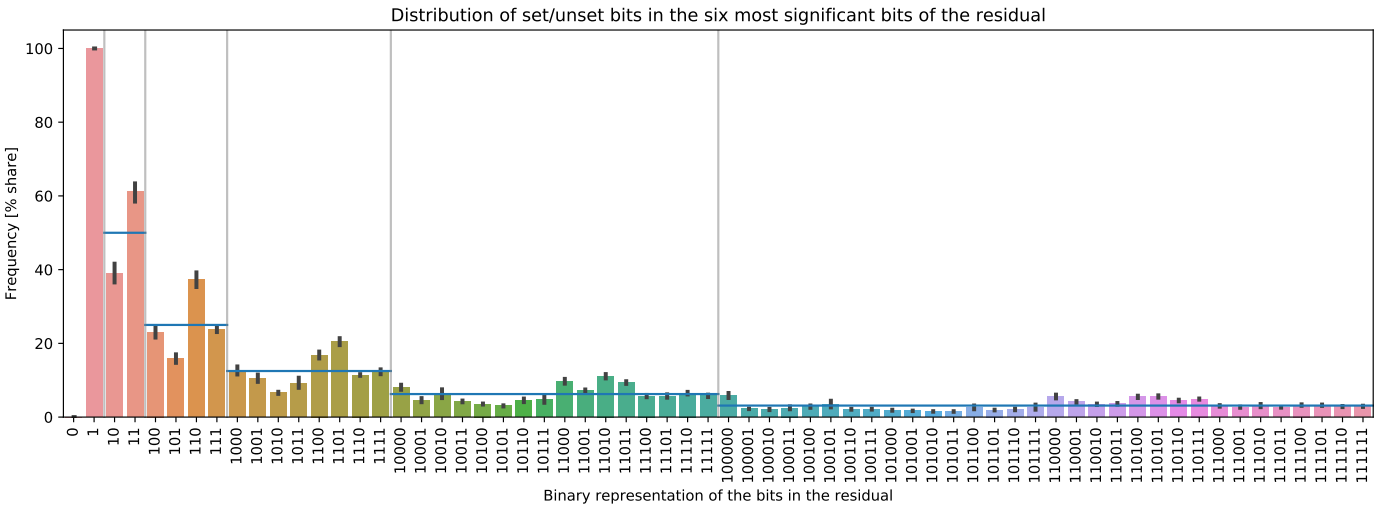

Fig. 6. Distribution of set/unset bits in the six most significant bits of the residual. Horizontal lines show expected distribution if values would have been equally distributed (i.e. white noise). The vertical lines separate the individual bit lengths. The distribution was calculated using the ICON model simulation output.

only third behind BWT+Huffman Encoding and Huffman Encoding, alone.

Huffman Encoding (without any transformation steps) performs best regarding FOC. This is due to the small value range of FOC. The downside of Huffman Encoding is that the codelist used for encoding the data must be transferred to the decompresser. There is no such a necessity for Range Encoding.

Observation 5: The BWT algorithm coupled with Range Encoding outperforms all other combinations of encoding methods. While BWT+Range Encoding is not optimal regarding FOC encoding, it still reaches third place. Regarding the added value of using a single encoding process for both components FOC and LZC, choosing BWT+Range Encoding is the most reasonable choice.

E. Comparison of `pzip` *and* `fpzip`

In the following section we will compare our proposed algorithm with `fpzip`. The results are depicted in Table III. This experiment was conducted using the climate simulation output.

As can be seen from the table, our proposed algorithm outperforms `fpzip` in almost every case. The only exception is Cloud water, where `pzip` achieves a CF of 67.59 and `fpzip` 73.28. While there could be many reasons for this, we observed that the Cloud water data have a lot of fill values. Due to the encoding scheme used by `fpzip`, it can compress exact predictions better than `pzip`. Future research might try to determine whether an alternative scheme should be used in such a case. In every other case `pzip` outperforms `fpzip` by eight percent on average. The performance for cloud ice content should also be emphasized. Here, our proposed algorithm achieves an improvement of 36.9%.

While the compression factor is better, the throughput of `fpzip` can not be achieved with the current implementation.

TABLE II
FILESIZE OF TEMPERATURE DATA AFTER TRANSFORMATION AND ENCODING SCHEMES. THE VALUES ARE IN BYTES. THE ABBREVIATIONS ARE: BURROW-WHEELER-TRANSFORM (BWT), DELTA DIFFERENCE (DIFF), HUFFMAN ENCODING (HUFF), RANGE ENCODING (RANGE), MOVE-TO-FRONT (MTF), RUN-LENGTH ENCODING (RLE). THE BEST PERFORMING ENCODING METHOD FOR EACH CATEGORY IS HIGHLIGHTED.

Transformation and Encoding Methods	**FOC**	**LZC**
bwt_diff_huff	29 715 223	41 347 759
bwt_diff_range	30 187 482	35 727 381
bwt_huff	21 625 804	50 204 194
bwt_mtf_diff_huff	32 046 132	45 767 747
bwt_mtf_diff_range	32 531 946	40 367 196
bwt_mtf_huff	25 382 523	37 223 861
bwt_mtf_range	25 958 407	32 657 611
bwt_mtf_rle_diff_huff	44 054 552	50 393 489
bwt_mtf_rle_diff_range	44 836 681	50 698 553
bwt_mtf_rle_huff	32 794 335	38 277 196
bwt_mtf_rle_range	33 716 265	38 402 906
bwt_range	22 043 891	28 862 090
diff_huff	29 451 797	37 695 831
diff_range	29 720 133	36 274 895
huff	21 625 389	50 203 689
mtf_diff_huff	31 908 557	46 008 528
mtf_diff_range	32 408 277	44 602 439
mtf_huff	25 374 374	35 935 034
mtf_range	25 782 694	34 990 704
mtf_rle_diff_huff	43 577 596	51 026 248
mtf_rle_diff_range	44 320 799	51 588 757
mtf_rle_huff	32 380 683	38 119 328
mtf_rle_range	33 313 109	38 667 389
range	22 489 931	42 918 321

TABLE III
COMPRESSION FACTOR (CF) AND THROUGHPUT [MIB/S] OF CLIMATE SIMULATION OUTPUT USING THE PROPOSED ALGORITHM `PZIP` AND `FPZIP` (HIGHER IS BETTER).

	Compression factor			Throughput [MiB/s]		
Climate variable	**`fpzip`**	**`pzip`**	**change [%]**	**`fpzip`**	**`pzip`**	**change [factor]**
Geopotential	9.96	11.36	+14.0	125.68 ± 2.28	35.13 ± 0.20	3.58
Vertical velocity	2.04	2.22	+8.8	60.31 ± 0.63	9.25 ± 0.15	6.52
Potential vorticity	2.24	2.42	+8.0	67.12 ± 1.40	9.94 ± 0.11	6.75
Cloud water	73.28	67.59	−7.8	211.53 ± 7.97	57.40 ± 0.52	3.69
Cloud ice content	2.87	3.93	+36.9	84.84 ± 0.46	16.88 ± 0.19	5.03
Specific humidity	2.60	2.78	+6.9	67.62 ± 1.79	10.88 ± 0.12	6.21
Temperature	3.23	3.48	+7.7	75.14 ± 1.09	12.35 ± 0.16	5.87
Virtual Temperature	3.23	3.48	+7.7	71.42 ± 0.96	12.16 ± 0.19	5.87
Zonal wind	2.23	2.37	+6.3	61.73 ± 0.99	9.50 ± 0.12	6.50
Vorticity	2.02	2.17	+7.4	60.33 ± 1.13	9.30 ± 0.10	6.48
Meridional wind	1.99	2.18	+9.5	59.88 ± 1.75	9.13 ± 0.14	6.56
∅ Average			+9.6			5.73

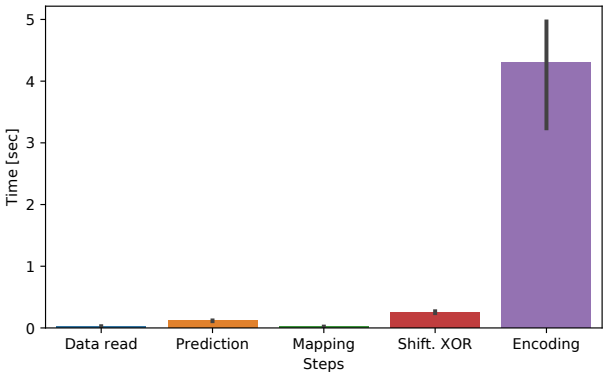

Fig. 7. Time spend in each step of the compression algorithm.

On average the `pzip` implementation is about six times slower than `fpzip`. To analyse this behaviour we timed each step of `pzip`. The result is depicted in Fig. 7. The majority of the time is spend in the encoding step. This is due to the memory space needed by BWT. While the time complexity of both algorithms are the same, the memory consumption is different. The currently known best implementation of BWT has a memory and time complexity of $\mathcal{O}(n)$. Because of this the implementation can not use the L1, L2 and L3 caches of the CPU as effectively as `fpzip`. Due to cache-misses the throughput decreases.

Observation 6: The compression factor of `pzip` is in most cases around 9% better than `fpzip` and other state-of-the-art lossless compression algorithms for real-world climate data. The BWT transformation is the most time consuming task. More research is needed to optimize this step of the algorithm.

VI. SUMMARY AND OUTLOOK

In this work we analysed different schemes in lossless compression algorithms for scientific data. We showed that shifting the prediction and true value before calculating the residual results in a better compression factor with minimal additional computational costs. This gain allows the use of less sophisticated prediction algorithms for higher throughput.

Our results show that the compression performance is dependent on the value range covered by the data and its distribution. The shifted XOR calculation bypasses the disadvantages of the XOR calculation by moving the data to a different value range. Using XOR for residual calculation, results into a skewed distribution of set and unset bits. This non-uniform distribution suggests, that there is still information contained in the residual. By splitting the residual into LZC, FOC and the remaining residual, a decorrelation of this information is achieved.

Our proposed encoding scheme outperforms the current state-of-the-art scheme by on average 9% regarding the compression factor. The time complexity of `fpzip` and `pzip` are the same, while the memory consumption is higher for `pzip`. Therefore further research is necessary to optimize the memory usage of the algorithm.

Further research is needed regarding special cases such as the cloud water data, where the data are mostly fill values. It may be advantageous if such cases are detected early and a custom solution is applied.

CODE AVAILABILITY

The code of our proposed compression algorithm described above will be made available under GNU GPLv3 license at [5].

ACKNOWLEDGEMENT

This work is supported by the Helmholtz Association Initiative and Networking Fund under project number ZT-I-0003 (Helmholtz Analytics Framework) and the Helmholtz "Advanced Earth System Modelling Capacity" (ESM) project.

REFERENCES

[1] J. Schröter, D. Rieger, C. Stassen, H. Vogel, M. Weimer, S. Werchner, J. Förstner, F. Prill, D. Reinert, G. Zängl, M. Giorgetta, R. Ruhnke, B. Vogel, and P. Braesicke, "Icon-art 2.1: a flexible tracer framework and its application for composition studies in numerical weather forecasting and climate simulations," *Geoscientific Model Development*, vol. 11, no. 10, pp. 4043–4068, 2018. [Online]. Available: https://www.geosci-model-dev.net/11/4043/2018/

[2] U. Cayoglu, F. Tristram, J. Meyer, T. Kerzenmacher, P. Braesicke, and A. Streit, "Concept and analysis of information spaces to improve prediction-based compression," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 3392–3401.

[3] X. Huang, Y. Ni, D. Chen, S. Liu, H. Fu, and G. Yang, "Czip: A Fast Lossless Compression Algorithm for Climate Data," *Int. J. Parallel Program.*, vol. 44, no. 6, pp. 1248–1267, dec 2016. [Online]. Available: http://link.springer.com/10.1007/s10766-016-0403-z

[4] U. Cayoglu, J. Schröter, J. Meyer, A. Streit, and P. Braesicke, "A modular software framework for compression of structured climate data," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '18. New York, NY, USA: ACM, 2018, pp. 556–559. [Online]. Available: http://doi.acm.org/10.1145/3274895.3274897

[5] U. Cayoglu, "On the Application of XOR for Residual Calculation in Prediction-based Compression (Data Repository)," http://github.com/ucyo/xor-and-residual-calculation/, 2019, [Online; accessed 11-April-2019].

[6] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP," *IEEE Trans. Parallel Distrib. Syst.*, vol. XX, no. X, pp. 1–1, 2019.

[7] L. A. Gomez and F. Cappello, "Improving floating point compression through binary masks," in *Proc. - 2013 IEEE Int. Conf. Big Data, Big Data 2013*, 2013, pp. 326–331.

[8] S. Di and F. Cappello, "Fast Error-Bounded Lossy HPC Data Compression with SZ," in *Proc. - 2016 IEEE 30th Int. Parallel Distrib. Process. Symp. IPDPS 2016*. IEEE, may 2016, pp. 730–739. [Online]. Available: http://ieeexplore.ieee.org/document/7516069/

[9] D. Tao, S. Di, Z. Chen, and F. Cappello, "Exploration of pattern-matching techniques for lossy compression on cosmology simulation data sets," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10524 LNCS, pp. 43–54, 2017.

[10] S. Liu, X. Huang, Y. Ni, H. Fu, and G. Yang, "A high performance compression method for climate data," in *Proc. - 2014 IEEE Int. Symp. Parallel Distrib. Process. with Appl. ISPA 2014*. IEEE, aug 2014, pp. 68–77. [Online]. Available: http://ieeexplore.ieee.org/document/6924431/

[11] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1245–1250, sep 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4015488

[12] J. Diffenderfer, A. Fox, J. Hittinger, G. Sanders, and P. Lindstrom, "Error Analysis of ZFP Compression for Floating-Point Data," may 2018. [Online]. Available: http://arxiv.org/abs/1805.00546

[13] P. Lindstrom, P. Chen, and E. J. Lee, "Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression," *Comput. Geosci.*, vol. 93, pp. 45–54, aug 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2016.04.009http://linkinghub.elsevier.com/retrieve/pii/S0098300416301091

[14] P. Lindstrom, "Fixed-Rate Compressed Floating-Point Arrays," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2674–2683, dec 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6876024/

[15] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, sep 1952. [Online]. Available: http://ieeexplore.ieee.org/document/4051119/

[16] G. N. N. Martin, "Range encoding: an algorithm for removing redundancy from a digitised message," *Video Data Rec. Conf.*, no. March, pp. 24–27, 1979.

[17] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[18] "POLar STRAtosphere in a Changing Climate (POLSTRACC)," www.polstracc.kit.edu, 2015-2016, [Online; accessed 2019-08-02].

[19] H. Oelhaf, B.-M. Sinnhuber, W. Woiwode, H. Bönisch, H. Bozem, A. Engel, A. Fix, F. Friedl-Vallon, J.-U. Grooß, P. Hoor, S. Johansson, Jurkat-Witschas, T. Kaufmann, M. Stefan Krämer, J. Kraus, E. Kretschmer, D. Lörks, A. Marsing, J. Orphal, K. Pfeilsticker, M. Pitts, L. Poole, P. Preusse, M. Rapp, M. Riese, C. Rolf, J. Ungermann, C. Voigt, C. M. Volk, M. Wirth, A. Zahn, and H. Ziereis, "POLSTRACC: Airborne experiment for studying the Polar Stratosphere in a Changing Climate with the high-altitude long-range research aircraft HALO," *Bulletin of the American Meteorological Society*, [accepted, to be published].

[20] B. Y. Ryabko, "Data compression by means of a "book stack"," *Problemy Peredachi Informatsii*, vol. 16, no. 4, pp. 16–21, 1980.

[21] F. Alted, "Why modern cpus are starving and what can be done about it," *Computing in Science & Engineering*, vol. 12, no. 2, p. 68, 2010.

[22] J. Alakuijala and Z. Szabadka, "Brotli compressed data format," Tech. Rep., 2016. [Online]. Available: http://www.rfc-editor.org/info/rfc7932

[23] S. Claggett, S. Azimi, and M. Burtscher, "Spdp: An automatically synthesized lossless compression algorithm for floating-point data," in *2018 Data Compression Conference*. IEEE, 2018, pp. 335–344.

[24] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2008.