

Czip: A Fast Lossless Compression Algorithm for Climate Data

Xiaomeng Huang^{1,2} · Yufang Ni^{1,2} ·
Dexun Chen^{1,2} · Songbin Liu^{1,2} · Haohuan Fu^{1,2} ·
Guangwen Yang^{1,2}

Received: 1 December 2014 / Accepted: 2 March 2016 / Published online: 9 March 2016
© Springer Science+Business Media New York 2016

Abstract Climate data have been dramatically increasing in volume in recent years. This huge volume of climate data poses considerable challenges for data storage, archiving and sharing. In this paper, we propose a lossless compression algorithm for climate data, named czip. We efficiently eliminate data redundancy through several new methods, including adaptive prediction, eXclusive OR differencing, multiway compression and static regions. To utilize the multiple cores available on modern computers, czip is implemented in parallel. Experimental results show that czip can achieve outstanding compression ratios as well as deflating and inflating throughputs; czip can achieve 800 MB/s deflating throughputs and over 2600 MB/s inflating throughputs on a server with 16 cores.

✉ Xiaomeng Huang
hxm@tsinghua.edu.cn

Yufang Ni
nyf13@mails.tsinghua.edu.cn

Dexun Chen
adch@263.net

Songbin Liu
liusb08@mails.tsinghua.edu.cn

Haohuan Fu
haohuan@tsinghua.edu.cn

Guangwen Yang
ygw@tsinghua.edu.cn

¹ Ministry of Education Key Laboratory for Earth System Modeling, Center for Earth System Science, Tsinghua University, Beijing 100084, China

² Joint Center for Global Change Studies, Beijing 100875, China

Keywords Lossless · Learning · Prediction · Multiway compression

1 Introduction

The volumes of global climate data collected from climate models, satellites, radar and other observational instruments have been increasing at an unprecedented speed and scale in recent years. These data are crucial for understanding the past and predicting the future of our planet. Overpeck et al. [1] project that the volume of global climate data holdings for climate models will increase to 100 petabytes (PB) by 2018. This huge volume of climate data poses considerable challenges for physical data storage and archiving as well as for the ease of data sharing among the climate research community.

Data compression is useful for mitigating these challenges by reducing the required data storage space and transmission capacity. Data compression can be either lossless or lossy. Lossless schemes reduce data volume by eliminating statistical redundancy, and no information is lost. Lossy schemes reduce data volume by identifying unnecessary information and removing it. For the climate research community, lossless compression is generally preferred for the pre-processing or post-processing of climate data.

Climate data are typically defined over space-time dimensions and presented as multidimensional arrays of high-precision floating-point numbers. Thus, there are fundamental differences between climate data and common text, image and video data. A standard form for these multidimensional arrays is $(X, Y, Z, T) \rightarrow (V_1, V_2, \dots, V_n)$, where X, Y, Z and T represent longitude, latitude, altitude and time. $V_i (i = 1, 2, \dots, n)$ represents a physical attribute, such as wind, pressure, temperature, or humidity. A significant feature of climate data is their temporal and spatial correlations, i.e., the fact that values in neighboring ranges tend to be numerically close to each other.

Several universal compression algorithms [2–6] operating on byte granularity are excellent for byte-oriented formats. However, they are inefficient for the floating-point format because they split the floating-point numbers into bytes and ignore the physical meaning of the floating-point format. Most existing compression algorithms [7–9] for floating-point numbers treat continuous floating-point numbers as a stream so that they can compress the exponential part of the floating-point format. These stream-oriented algorithms do not make full use of the temporal and spatial locality of climate data. Certain multidimensional compressors, such as fpzip [10] and ndims [11], do not distinguish the time dimension from the spatial dimensions, and they cannot adaptively select prediction schemes to decouple the correlations for different climate variables. In addition, most current compression algorithms are serial, and thus, they cannot comply with the needs of the epoch of big data.

In this paper, we propose a fast lossless compression algorithm for climate data, called czip (Climate ZIP). The underlying concept of czip is to attempt to eliminate data redundancy by exploiting the correlations among multidimensional climate data. The compression process can be divided into three phases. First, we sample several time slices of an array and learn the features of the array by evaluating sev-

eral different prediction schemes along the time dimension; thus, we can select the prediction scheme that can best express the correlations among the values in the array. In the case that none of the prediction schemes can express the correlation sufficiently precisely, no prediction scheme will be chosen. For climate data, it is generally true that the values of certain regions do not change along the time dimension. For instance, the numerical results for land regions do not change when the temperatures of ocean regions are computed. These regions are identified during the learning step, and the constant values of these regions are compressed using a byte-oriented compressor (e.g., zlib). Second, if one of the prediction schemes is chosen, XOR differencing will be applied for each value based on its predicted value obtained from the previously selected prediction scheme. After this phase, the correlations within the multidimensional arrays are decoupled. As the final step, we send each byte of each residual from the XOR differencing operation, or the original value if no prediction scheme was selected in the learning phase, to another byte-oriented compressor. We implement the czip algorithm in parallel to efficiently utilize the multiple cores available on modern computers. The experimental results show that czip exhibits superior performance in most testing scenarios in terms of compression ratios, deflating throughputs and inflating throughputs. It can achieve 800 MB/s deflating throughputs and over 2600 MB/s inflating throughputs on a server with 16 cores.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces the czip algorithm in detail. Section 4 describes the implementation of czip in parallel. Section 5 evaluates the performance of czip and compares it with that of state-of-the-art compressors. Section 6 concludes the paper.

2 Related Work

According to the level on which each compression method understands the structure of the data to be compressed, current compression methods can be broadly classified into three categories.

The first category is byte stream methods. Compressors on this level treat the input as an unstructured byte stream and attempt to identify data redundancy within the 1D linear byte stream. Many traditional general-purpose compression methods [2–4, 12, 13] operate at this level. Compression algorithms on this level have been extensively studied. Whereas some of them emphasize high throughputs [5, 13], others attempt to strike a balance between compression ratio and throughput [2–4]. However, because most climate modeling output data are floating-point numbers, compressors at this level generally cannot find long repetitive byte strings and thus cannot achieve high compression ratios for climate modeling output data.

Compressors in the second category have prior knowledge of the data type in the input stream, and compressors at this level are often specially designed for a certain type of data. For example, szip [14] is designed to compress scientific data. fpc [8, 15] can compress double-precision floating-point data with high throughputs. lcfp [7] breaks the bytes of floating-point data into sign, exponent and mantissa and compresses these three types of information separately via context-based arithmetic

coding. Because floating-point compressors at this level can recognize the floating-point format and process the input stream in 4-byte granularity (or 8 bytes for double-precision floating-point data), they can typically achieve better compression ratios than general-purpose compressors when compressing climate modeling output data. However, compressors at this level still treat the input data as a linear stream; thus, they cannot exploit the multidimensional features of climate modeling output data to remove redundancy.

Methods in the third category have even more priori knowledge of the input stream than do the compressors at the previous two levels. Compressors at this level not only know the data type, but also have some knowledge of the relationships among the values in higher dimensions. Therefore, these compressors can compress the multidimensional data more efficiently [10, 11, 16–20]. However, some of them treat multidimensional data simply as geometric structures without distinguishing the time dimension from the spatial dimensions [10, 11, 20]. Although other methods can achieve even better compression ratios by exploiting the temporal redundancy between spatial data splices [16–19], the throughputs of these compressors are not high enough to handle the huge volumes typical of climate data.

3 Design of Czip

Before we present the technical details of czip, we must define two basic concepts: static regions and dynamic regions. In a global climate model simulation, there are certain regions of the global grid that are not involved in the simulation, and thus, the values of these regions do not change. For example, an earth system model includes at least an atmospheric component, an oceanic component, a land component and a sea ice component. When the oceanic component is running, the sea surface temperature is varying in time within the ocean regions. However, the values for the continental regions remain constant at their default settings. Figure 1 shows that the sea surface temperature (SST) varies with the dimensions of both space (vertical layers, in units of layer) and time (in units of simulated days) in an ocean simulation. There are obvious similarities evident in this data set. In other words, the SST values in neighboring ranges in the space-time dimensions tend to be numerically similar to each other. We also observe that the SST values in continental regions do not vary in the space-time dimensions. Therefore, we define the static regions as the grid points whose values do not change, and the remaining grid points in the climate data set are defined as dynamic regions. In our oceanic case, the continental regions are static regions and the ocean regions are dynamic regions.

As shown in the flow chart of czip presented in Fig. 2, we learn the features of a multidimensional array by sampling several time slices of the array, and then we individually test several different prediction schemes along the time dimension. Based on the results, we can select the prediction scheme that can best express the correlations among the values in the array. If none of the prediction schemes can express the correlation sufficiently precisely, we do not use any prediction scheme. If one of the prediction schemes is suitable, then XOR differencing will be applied to each value based on its predicted value. Thereby, the correlations between the multidimensional

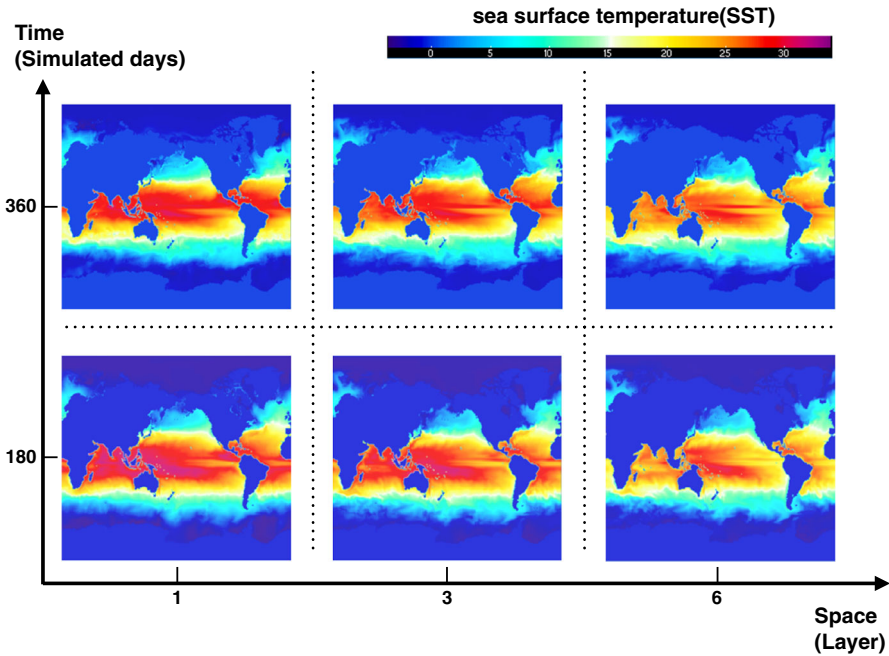


Fig. 1 The values for ocean regions in the results from an ocean simulation are time-varying (*dynamic regions*). However, the values of the land regions (*blue regions*) do not change along the time dimension. These regions are called static regions (Color figure online)

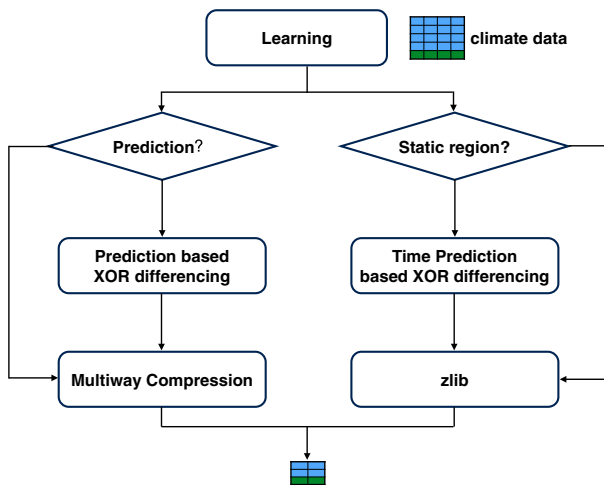


Fig. 2 The flow chart of czip

arrays will be decoupled. As the final step, we send each byte of each residual from the XOR differencing operation, or the original value if no prediction scheme was selected in the learning phase, to a byte-oriented compressor. We have also designed a simple

method to determine the static regions, and we apply several existing professional compression methods to address the values of those static regions.

3.1 Learning

Prediction is a useful technique for compressing data streams. Many of the existing compression algorithms for floating-point numbers adopt prediction-based differencing to achieve favorable compression ratios, including lcfp [7], ndims [11], fpzip [10] and etc. [18,19,21]. In an ideal case, the predicted value is exactly equal to the real value and the differencing residues are all zero bits.

A major challenge facing prediction techniques is that there is no “master key” for all types of climate data. Different climate variables in a climate data set are associated with different physical rules and processes. Thus, different prediction schemes are needed to decouple the correlations of numerous climate variables. Our solution is to first learn the features of the multidimensional data and then adaptively select the best prediction scheme from among a pre-defined set of prediction schemes based on the learning results.

The primary purpose of the learning phase is to determine whether prediction can be successfully applied to the data and if so, which predictor is most suitable. The learning phase proceeds as follows. For each multidimensional (usually 3D or 4D) array, we distinguish the time dimension from the spatial dimensions (usually 2D or 3D) and treat the multidimensional array as values in the spatial dimensions that are evolving along the time dimension. From among a pre-defined set of prediction schemes, the accuracy of each predictor is evaluated over several consecutive time steps, and the predictor that offers the most accurate predictions for the array is selected.

3.2 Prediction

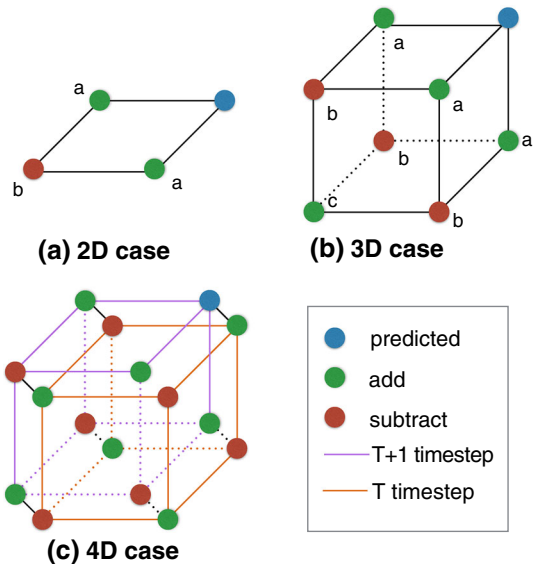
Predictor design and selection are key to czip. An efficient predictor for climate data must achieve a higher compression ratio than common predictors. We developed a set of predictor candidates for climate data. This predictor set exploits the correlations among values to predict the next value to be encoded based on a set of previously encoded values. The optimal predictor is selected based primarily on accuracy and speed. The differencing residual between the real value and its predicted result will be smaller if the predicted result is more accurate. Moreover, users cannot tolerate slow speeds for the compression of data.

Table 1 lists a set of predictor candidates designed based on our known knowledge of physical climate processes and existing work [11]. There are three categories of predictors available in czip. The first category addresses spatial correlations, and the second category addresses time correlations. For each dimension, we use the simple yet effective first-order and second-order polynomial predictors provided by [20,22,23]. Furthermore, to exploit the potential correlations of nearby values in higher dimensions, we include a third category of predictors named Lorenzo predictors [11], which are general-purpose predictors for multidimensional arrays [10,20]. In an n -dimensional cube, the Lorenzo predictor computes the predicted value of a corner from

Table 1 Predictor candidates: each predictor attempts to exploit the correlations along one dimension or among several dimensions (by means of Lorenzo predictors)

Correlation	Predictors
Latitude	$P_n = X_{n-1}; P_n = 2X_{n-1} - X_{n-2}$
Longitude	$P_n = Y_{n-1}; P_n = 2Y_{n-1} - Y_{n-2}$
Altitude	$P_n = Z_{n-1}; P_n = 2Z_{n-1} - Z_{n-2}$
Time	$P_n = T_{n-1}$
Combo	$P_n = \text{Lorenzo}(X, Y);$ $P_n = \text{Lorenzo}(X, Y, Z);$ $P_n = \text{Lorenzo}(X, Y, Z, T)$

Fig. 3 In an n-dimensional cube, the Lorenzo predictor computes the predicted value at a corner from the scalar values at the other $(2^n - 1)$ corners



the scalar values of the other $(2^n - 1)$ corners. As shown in Fig. 3, in the 2D case, i.e., $\text{Lorenzo}(X, Y)$, we can add the scalar values of the two green “a” corners and subtract the value of the red the “b” corners; in the 3D case, i.e., $\text{Lorenzo}(X, Y, Z)$, we can add the values of the “a” corners, subtract the values of the “b” corners, and add the values of the “c” corner; in the 4D case, i.e., $\text{Lorenzo}(X, Y, Z, T)$, we can add the values at the first and third degree neighbors and subtract the values of the second and fourth degree neighbors [11].

Because the number of values accessed by the Lorenzo predictors increases exponentially with the number of dimensions, the computational cost of the Lorenzo predictors is much higher than that of the other predictors. In czip, low-cost predictors are preferred if they can achieve comparable accuracy to that of the Lorenzo predictors. We implement the predictor selection phase of czip in the following three steps.

First, we assign a predictor to each of the grid points in each sampled time step. We measure the absolute error, whose value is the differencing residual between the

predicted result and the real value. A smaller absolute error indicates a more accurate predictor.

Second, we decide whether to use the predictor. Our principle is that if the value predicted by the candidate predictor is sufficiently close to the real value of the grid point, we assign the predictor to *czip*. Otherwise, we abandon the attempt to use a predictor for this grid point and mark it as an unpredictable point. The key is to determine how to evaluate the meaning of “sufficiently close”. Our method is to compute the relative error of the result from the candidate predictor. If the relative error is smaller than a given threshold, we consider the predicted result to be “sufficiently close” to the real value. Based on a number of experiments, we set this threshold to 2^{-8} for 32-bit single-precision floating-point values in *czip*.

Finally, we count how many times each predictor has been assigned by counting its corresponding grid points among the samples. The predictor that has been chosen the largest number of times is selected as the final predictor for the array. The number of grid points marked as unpredictable also must be counted. If the number of unpredictable points is too large, it means that the data cannot be predicted effectively by any of the given predictors. The data will then be compressed without predication-based differencing encoding. Furthermore, if several predictors have similar numbers of selections, the predictor with the lowest computational cost will be chosen for the purpose of speed.

The process of predictor selection incurs some overhead. Fortunately, this process is invoked only once for the entire array. For arrays with several hundreds or thousands of time steps, the samples will consist of only 100 time steps. The overhead for this process is a very small portion of the entire compression process. The key advance regarding predictor selection in our algorithm is its flexibility, and this feature lays a strong foundation for *czip* to achieve high compression ratios for different types of climate data.

3.3 Prediction Based XOR Differencing

Differencing is performed between the predicted value and the real value, and then the residual of the differencing operation is sent to the multiway compressor. Because floating-point operations may cause underflows, with irreversible loss of information, the computation of the residual between two floating-point numbers is typically performed through integer operations instead of floating-point operations. One common method is integer subtraction operation [10,20]. This method maps the two floating-point numbers to integers and computes the difference between the two integers. To make the absolute value of the residual smaller, absolute integer subtraction is applied, and the sign of the residual is tracked explicitly.

Another simple XOR operation method is differencing between the two numbers after they are mapped to integers [8]. Because of the standard format used to represent floating-point numbers, floating-point numbers in close proximity will often have identical bit patterns in their most significant bits. Therefore, the XOR residual will be close to zero, with leading zero bits.

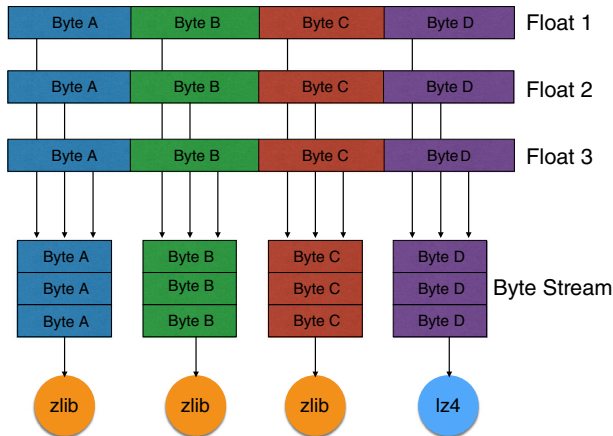


Fig. 4 The multiway compressor splits the bytes of each floating-point number or residual and compress them separately

Although the subtraction-based approach can produce more leading zero bits in certain cases, it is more complex and slower than the simple XOR-based approach because the sign must be tracked and compressed separately. Our *czip* method does not rely on leading-zero-count-based encoding [8, 10], so we chose XOR differencing for residual computation because of its superior performance.

3.4 Multiway Compressing

As shown in Fig. 4, the multiway compressor operates on the basic concept of rearranging the four bytes of a series of IEEE 32-bit single-precision floating-point numbers or their XOR differencing residuals. The rearranged byte stream can make better use of the advantages of byte-stream-oriented compressors, such as *zlib* [4] and *lz4* [5]. The compressor first reads in a chunk (6 MB in *czip*) of 32-bit floating-point numbers and splits each number into four bytes. Thereafter, it appends each byte to its corresponding byte stream. Finally, it inputs the rearranged data chunk into byte-stream-oriented compressors, including *zlib* and *lz4* in current *czip*.

The multiway compressor can operate efficiently for both residual streams and original floating-point streams. In the case that an accurate predictor can be selected for XOR differencing, the higher bytes of each residual will contain more zero bits, whereas the lower bytes will contain more random bits. The multiway compressor can separate the easily compressible bytes from the more difficult-to-compress bytes. Thus, better compression ratios can be expected.

For original floating-point streams (without XOR differencing), the multiway compressor can still function because of the spatial-temporal similarity of climate data. Floating-point values of the same variable in the same array tend to be of the same order of magnitude. Thus, the sign and exponent bits should be similar.

In addition, the multiway compressor can achieve a good trade-off between compression ratio and throughput. For easily compressible bytes streams, *zlib* is adopted

to achieve good compression ratios, whereas for difficult-to-compress byte streams, lz4, which offers high throughputs, is used to achieve better performance without a significant sacrifice in compression ratio.

3.5 Static Regions

For the detection of static regions, we compare the value of each grid point in the current time step with its value from the previous time step in the learning process. If the values are the same, this grid point is marked as belonging to a static region. False positives may occur when the values of a grid point happen to remain identical for only two time steps. To reduce the occurrence of false positives, consecutive time steps are used to verify that the values of the regions marked as static do not change. If the value of any of the identified static regions changes, then that static region is once again flagged as a dynamic region. A table structure is used to record the grid points that are marked as static regions. Because the table is stored in bitmap format, the storage overhead of the flag table is negligible, especially when there are hundreds of time steps in the multidimensional array.

We simply use time-predictor-based XOR differencing to compute the residuals of the static regions in czip. The residuals of the static regions will all be zeros, which can be easily compressed. However, if the residuals of these regions were to be interleaved with the residuals of the dynamic regions, it would be more difficult for the compressors to find and remove the redundancy because of the interleaving of the zero residuals of the static regions with the non-zero residuals of the dynamic regions. Therefore, to achieve better compression ratios, we separate the residuals of the static regions from the residuals of the dynamic regions and then use zlib to compress the residuals of the static regions. For the purpose of speed, the static regions are processed separately only when the proportion of static regions is higher than a certain threshold (10 % in current czip).

4 Parallelization

The compression and decompression processes in czip run in parallel. They can be run on multi-core computers at high speed. We split each array into several chunks along the time dimension, such that each chunk contains a certain number of time steps. After the initial learning phase, czip assign all threads in a thread pool to the chunks in the front of the queue. Once a thread has compressed a chunk completely, it will be collected by the thread pool and wait for its next assignment. There is a process called a manager that is designed to check the thread pool. Once a thread returns, the manager assigns it to the top chunk in the waiting queue. Additionally, there is a collector that collects the compressed chunks in the order in which they were placed in the waiting queue. The collector combines the collected chunks into a stream to be written to the disk. The decompression process is similar to the compression process. After the data are read, we decompress the chunks in parallel and merge the results by means of a collector.

Table 2 Overview of the two data sets (size in GB)

Data set	Atmosphere	Ocean	Land	Sea-ice	Land-ice	Total
LASG	70.3	53.0	18.6	12.0	2.2	156.1
BNU	66.2	46.0	15.1	10.0	10.0	147.3

The chunk size should be carefully chosen. If the data chunks are too large, the concurrency may be affected. If the data chunks are too small, the compression ratio may decrease. Based on a large number of experiments, we set the default chunk size to 6 MB.

5 Experiments and Evaluations

In this section, we will evaluate the performance of *czip* and compare it with other lossless compression methods.

5.1 Data Sets

We will use two climate model data sets submitted to the fifth phase of the Climate Model Intercomparison Project (CMIP5) [24]. CMIP5 provides a freely available state-of-the-art multi-model data set for examining climate change and improving the ability of models to predict future climate states. At present, 19 institutes have contributed 41 model data sets to CMIP5. CMIP5 requires that each modeling group submit at minimum 2300 years of model output [24]. Therefore, most of the output data include many time steps and thus are suitable for *czip*.

For this paper, two CMIP5 data sets from China were chosen to evaluate the effectiveness of our proposed compression method. One data set is from the output of the BNU-ESM model created by Beijing Normal University (BNU), and the other is from the output of the LASG-CESS model developed by the Chinese Academy of Sciences and Tsinghua University. These data sets are in NetCDF [25] format, following the CMOR standard [26], and can be accessed through the Earth System Grid Federation (ESGF) [27].

As shown in Table 2, both models contain five different components: atmosphere, ocean, land, sea ice and land ice. Because of the different physical processes driving them, the data generated for these different components exhibit different compressibilities [28]. These data sets include different variables, such as relative humidity, wind, sea surface temperature, and so on. These variables are recorded with different frequencies, e.g., monthly, daily, every 6 h, and every 3 h. We will show in the following experiments that *czip* functions well for these diverse data sets.

5.2 Experiment Setup

We will first compare *czip* with other popular compressors, including general-purpose compressors at the first level (zlib [4], lz4 [5], lz4hc [5]), level (zlib [4]), floating-

Table 3 The percentages of the prediction schemes that are selected for the LASG data set, as well as the proportions of the static regions in the data set

Components	Prediction schemes (%)				Static regions (%)	
	Spatial	Time	Lorenzo	None	Bytes	fnum
Atmosphere	10.34	25.93	0.0	63.73	0.12	0.16
Ocean	0.0	0.0	82.64	17.36	35.12	100.0
Land	0.0	96.88	2.68	0.44	68.63	100.0
Land-ice	14.29	0.0	0.0	85.71	74.42	100.0
Sea-ice	2.63	0.0	0.53	96.84	84.63	100.0

point-oriented compressors at the second level (szip [29], lcfp [7]), and a third-level compressor (fpzip [10]). We will also compare czip with another parallel compressor, pigz [30].

The version of the zlib compressor used is 1.2.7. The lcfp compressor was downloaded from Isenburg’s source code [31]. The szip compressor is szip-2.1 [32], with BITS_PER_PIXEL set to 32, PIXELS_PER_BLOCK set to 32, and SCANLINE set to 128. The compression level for zlib is set to 6, and MEMORY_USAGE for lz4 and lz4hc is set to 14. The experimental platform is a Red Hat 4.4.5-6 X86-64 server, with two Intel Xeon E5-2650 2.0 GHz CPUs with 8 cores and 32 GB of main memory. Therefore, we had 16 cores in total.

5.3 Benefits of Adaptive Prediction and Static Regions

The four prediction columns of Table 3 show the selection percentages for each prediction scheme for the different components of the LASG data set. The spatial scheme includes six predictors along each spatial dimension, X, Y, and Z, as listed in Table 1. The time scheme is the Time predictor, and the Lorenzo scheme refers to the three Lorenzo predictors listed in Table 1. The “none” scheme indicates that none of the predictors is selected and that the array is compressed without prediction-based differencing.

According to the prediction columns in Table 3, we find that the data associated with different components exhibit different behaviors. The Lorenzo scheme is the best choice for most of the ocean data (82.64%). The time prediction scheme is the best choice for land data in most cases (96.88%). However, neither of these two prediction schemes can provide sufficiently precise predictions for the data associated with the sea-ice, land-ice and atmosphere components. For these data, it is better to compress the data without any prediction.

We also notice that even for data corresponding to the same component, different prediction schemes may be selected. For example, of the arrays associated with the atmosphere component, the time prediction scheme is preferred for approximately 25.93%, and the spatial scheme is chosen for 10.34% of the arrays, whereas for the remainder, no prediction scheme is preferred. We can conclude that it is necessary

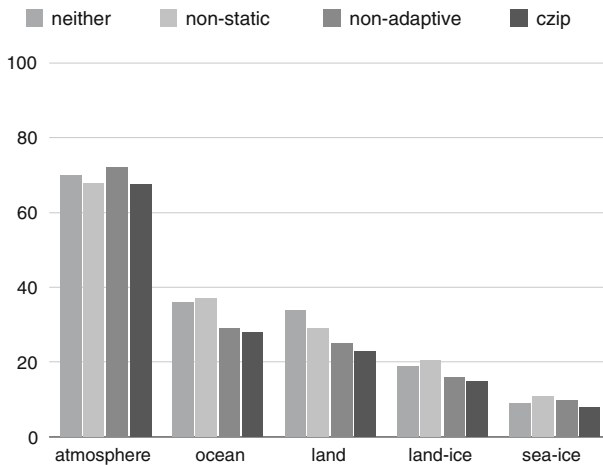


Fig. 5 The impacts on the compression ratio of the adaptive prediction and static regions options

to apply adaptive prediction individually to each array to achieve better compression ratios.

The two static regions columns of Table 3 present the proportions of static regions contained in the data set. The bytes column shows how many bytes belong to these static regions, and the fnum column shows how many files contain static regions. From the fnum column of Table 3, we can see that with the exception of those for the atmosphere data, all of the files contain some static regions. Moreover, according to the bytes column, significant proportions of the data belong to static regions for the ocean, land, land-ice, and sea-ice components. In fact, arrays associated with the same component typically contain similar proportions of static regions. Because the values in the static regions remain constant along the time dimension, we can simply apply XOR differencing based on their values in the previous time step, yielding zero bytes. The compression ratios can be improved by identifying the static regions in each array.

To evaluate how much benefit we can gain from exploiting adaptive prediction and static regions, we compare the compression ratios achieved using czip in three scenarios: (1) neither: both the adaptive prediction and static regions options are disabled; (2) non-static version: with the static regions option disabled, the data are compressed without distinguishing static regions from dynamic regions; (3) non-adaptive version: with the adaptive prediction option disabled, all data are compressed using the Lorenzo predictor only; and (4) czip: the data are compressed with both the adaptive prediction and static regions options enabled.

Figure 5 presents the results for the compression ratios achieved using these four approaches. The compression ratios achieved by the adaptive-prediction compressor are superior to those achieved by the non-adaptive compressor for all five data components. The improvement for the atmosphere data is the most significant. This is because the Lorenzo predictor offers the least precise prediction for atmosphere data, and thus, applying Lorenzo prediction can increase the entropy, resulting in worse compression

ratios. Therefore, the atmosphere data can benefit the most from adaptive prediction. The improvement for the ocean data is the least significant because Lorenzo prediction is the best choice for most of the ocean data. The improvement achieved through adaptive prediction for all data components indicates that czip can successfully adapt to the data to be compressed and select appropriate prediction schemes to achieve better compression ratios.

With regard to the consideration of static regions, we can compare the compression ratios achieved by czip with the compression ratios of its non-static version. From Fig. 5, we can see that the compression ratios of the non-static compressor are significantly worse than those of the czip compressor. This remarkable degeneration in the compression ratios can be explained from two perspectives.

One reason is that the inclusion of the static regions may hamper the adaptive prediction ability of the non-static version in many cases. Because static regions account for large proportions of the data for the ocean, land, land-ice and sea-ice components and the best prediction scheme for these static regions is the time scheme, when the static regions are not distinguished from the dynamic regions, the non-static compressor will always choose the time scheme for the prediction of data associated with these four components. However, the time scheme is the worst prediction scheme for the data of the ocean, land-ice, and sea-ice components, according to Table 3. Thus, the false predictions generated by the time scheme can lead to an increase in entropy for the dynamic regions, thereby resulting in worse compression ratios. Another reason for the degeneration of the compression ratios is the interleaving of the zero byte streams of the residuals from the static regions with the non-zero byte streams from the dynamic regions. It would be easier for the entropy compressors to compress these two different types of byte streams separately.

For the atmosphere data, there is little difference between the compression ratios achieved by czip and by the non-static version because most of the atmosphere arrays do not contain static regions. Therefore, the static regions do not affect the prediction scheme selection nor significantly disturb the byte streams.

5.4 Comparisons with the State-of-the-Art Compressors

To gain a better understanding of the effectiveness of our compression method, let us compare czip with other state-of-the-art compressors. The effectiveness of a compression method can be measured from two perspectives: its data compression ratios and its throughputs for deflating and inflating.

5.4.1 Compression Ratios

Figures 6 and 7 present the compression ratios of czip and other state-of-the-art compressors for the two different data sets. From these two figures, we can see that the proposed compressor can achieve compression ratios comparable to the best achieved by the state-of-the-art compressors. For example, for the ocean, land and land-ice components of both data sets and the sea-ice component for the BNU data set, our proposed compressor can achieve the best compression ratios. Even if it is not the best

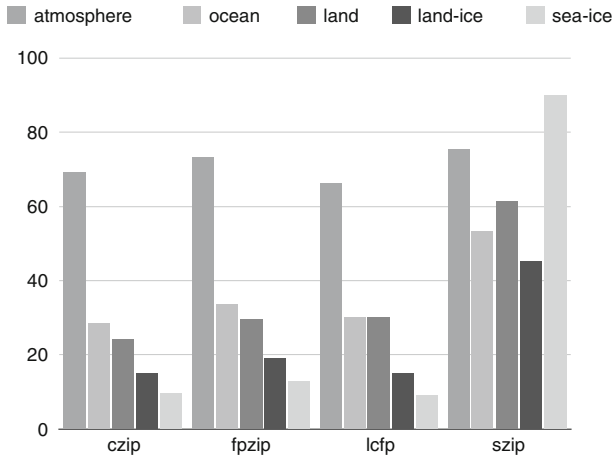


Fig. 6 Comparison of **compression ratios** for the LASG data set. **Smaller values are better**

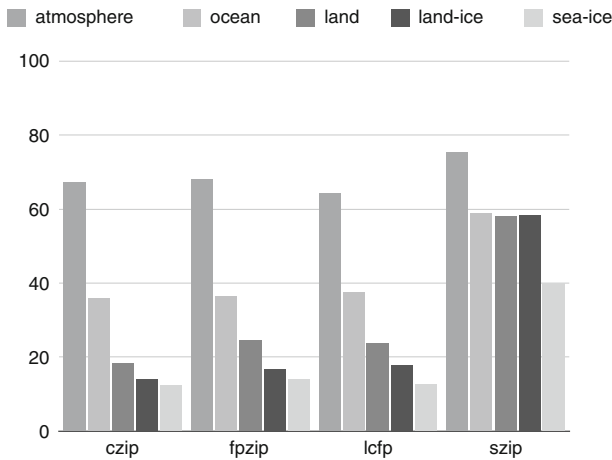


Fig. 7 Comparison of **compression ratios** for the BNU data set. **Smaller values are better**

for the data associated with the other components, czip achieves close performance with lcfp in terming of compression ratios. However, as seen from the throughput evaluations presented below, the throughputs of lcfp are much lower (by at least a factor of 3) than those of czip. For the atmosphere data, czip cannot achieve better compression ratios than those of the lcfp compressor. This is because none of the prediction schemes can provide sufficiently precise predictions for most of the atmosphere data (approximately 64%, according to Table 3) and the proportion of static regions in the atmosphere data is typically small. Therefore, czip cannot effectively leverage these two factors to boost the compression ratios for the atmosphere data.

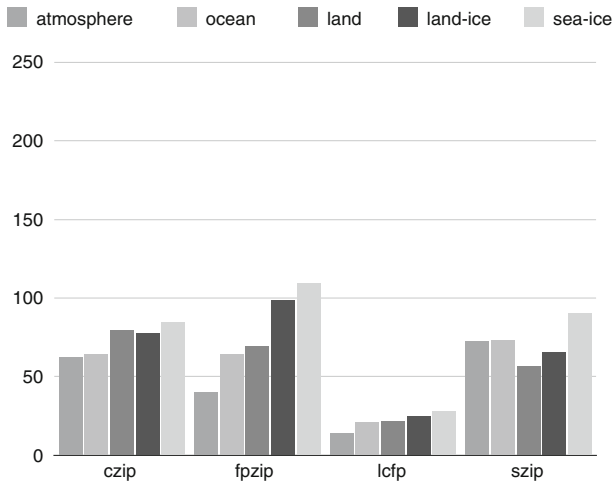


Fig. 8 Deflating throughputs (MB/s) for the LASG data set. Higher values are better

5.4.2 Compression Throughputs

High deflating and inflating throughputs are important features of a good compressor, especially given the rapidly expanding and already extremely large volume of climate data sets. In this section, we will compare the throughputs of czip with those of other compressors for the LASG data set. According to Fig. 6, only the fpzip and lcfp compressors can offer compression ratios comparable to those of czip; therefore, we will mainly compare the throughputs of these three compressors in this section.

To ensure a fair comparison, Fig. 8 presents the single-thread deflating throughputs of all five compressors for the data of the five components. Among fpzip, lcfp and czip, czip can achieve outstanding deflating throughputs for the atmosphere, ocean and land data. For the atmosphere data in particular, which is especially difficult to compress, the deflating throughput of czip is significantly higher than those of the other two compressors.

One phenomenon apparent in Fig. 8 is that the deflating throughputs for most of the compressors tend to increase with increasing compressibility of the data. However, the throughput of czip for the land-ice data is an exception. This is because each array associated with the land-ice component is too small (3.5 MB on average). Therefore, the relative overhead of the learning step is correspondingly increased. Thus, we believe that the throughput of czip for land-ice arrays could be higher if the arrays were larger.

Figure 9 shows the inflating throughputs of all five compressors for the five data components. According to this figure, the inflating throughputs of czip are significantly higher than those offered by lcfp and fpzip. Once again, for the atmosphere data, the throughput of czip is remarkably higher than those of the other two compressors. The adaptive prediction capability is the primary reason that czip can achieve significantly higher throughputs when deflating and inflating atmosphere data than those of fpzip. Because of its adaptive prediction capability, czip either does not apply prediction

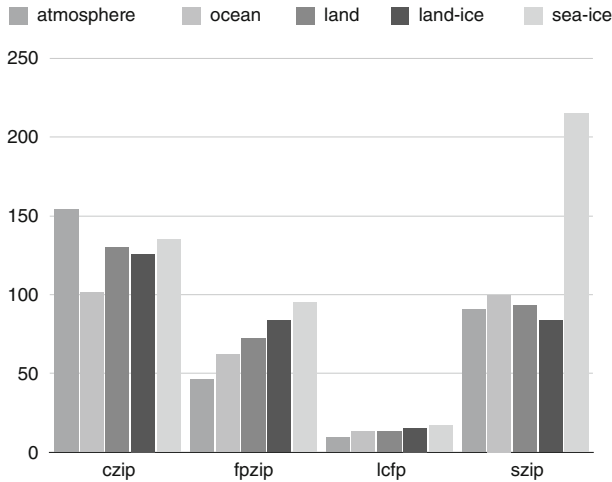


Fig. 9 Inflating throughputs (MB/s) for the LASG data set. Higher values are better

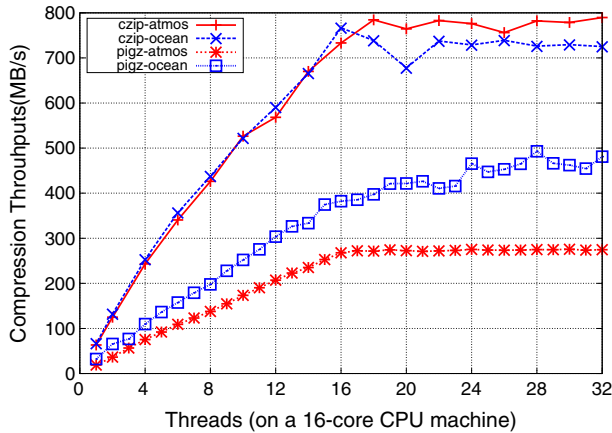


Fig. 10 Parallelized deflating throughputs of czip on a server with 16 cores

for the atmosphere data or performs only simple temporal or spatial prediction. By contrast, fpzip always applies Lorenzo prediction when compressing the atmosphere data. The Lorenzo predictor needs to access more values (7 for 3D arrays), thereby increasing the prediction overhead. Furthermore, the generation of false predictions by the Lorenzo predictor increases the entropy of the data and makes it more difficult for the fpzip entropy encoder to remove redundancy. As a result, fpzip is much slower than czip for the atmosphere data.

5.4.3 Parallelized Throughputs

For comparison with a parallel method, we chose pigz, which is a parallel implementation of gzip using zlib and pthread, to test against czip on a server with 16 cores because

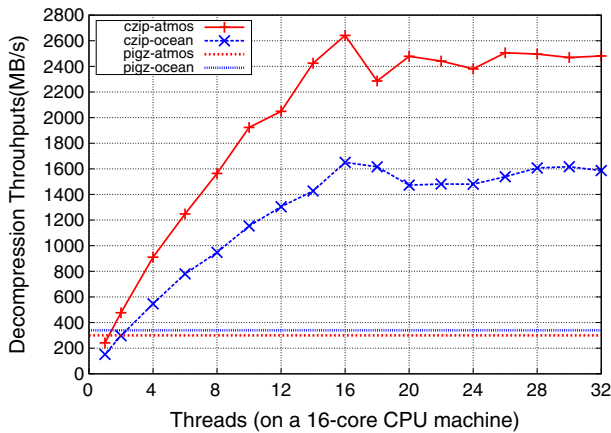


Fig. 11 Parallelized inflating throughputs of czip on a server with 16 cores

we possess two 8-core Intel Xeon E5-2650 CPUs. Figures 10 and 11 show the deflating and inflating throughputs, respectively, of czip and pigz for the atmosphere and ocean data. It is obvious that czip offers better performance and scalability than pigz. The throughputs of czip increase linearly, and it offers the highest overall throughputs (approximately 800 MB/s when deflating and 2600 MB/s when inflating). Because pigz uses only one thread to inflate data, its inflating throughputs do not vary with the number of threads. It should be noted that the deflating and inflating processes are computationally intensive. When we use a number of threads greater than the number of CPU cores, such as 24, 28, or 32 threads, the resource competition among these virtual threads will become severe. Therefore, the deflating and inflating throughputs cannot be further improved with the addition of more threads.

6 Conclusions

In this paper, we designed and implemented a lossless compression method, named czip, for application to large volumes of climate modeling output data. By distinguishing the time dimension from the spatial dimensions in the climate arrays, czip can adaptively select the most appropriate prediction scheme to detect the correlations among the values in a multidimensional array. Once a suitable prediction scheme is selected, XOR-based differencing is applied between the original and predicted values to reduce data redundancy. The bytes of each residual from the XOR operation are further split into different streams, and a byte-oriented compressor compresses each stream. In climate modeling output data, there are usually several static regions, whose values remain constant along the time dimension. In czip, the values of these static regions are compressed separately to achieve better compression ratios. We used two climate data sets from CMIP5 to test the performance of czip. The experimental results show that czip could achieve outstanding compression ratios as well as deflating and inflating throughputs in most of the test cases.

Acknowledgments The authors would like to thank the editor and the anonymous reviewers for their valuable comments. This study was supported by funding from the National Natural Science Foundation of China (41375102), the National Grand Fundamental Research 973 Program of China (No. 2014CB347800), and the National High Technology Development Program of China (2011AA01A203).

References

1. Overpeck, J.T., Meehl, G.A., Bony, S., Easterling, D.R.: Climate data challenges in the 21 st century. *Science (Washington)* **331**(6018), 700–702 (2011)
2. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **23**(3), 337–343 (1977)
3. Ziv, J., Lempel, A.: Compression of individual sequences via variable rate coding. *IEEE Trans. Inf. Theory* **24**(5), 530–536 (1978)
4. zlib. <http://www.zlib.net> (Online)
5. lz4: Extremely Fast Compression algorithm. <http://code.google.com/p/lz4/> (Online)
6. bzip2. <http://www.bzip.org> (Online)
7. Isenburg, M., Lindstrom, P., Snoeyink, J.: Lossless compression of predicted floating-point geometry. *IEEE Trans. Inf. Theory* **37**(8), 869–877 (2005)
8. Burtcher, M., Ratanaworabhan, P.: FPC: a high-speed compressor for double-precision floating-point data. *IEEE Trans. Comput.* **58**(1), 18–31 (2009)
9. C. 120.0-G-2: Lossless data compression. In: Report Concerning Space Data System Standards. Green Book (Issue 2) (2006)
10. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. *IEEE Trans. Comput.* **12**(5), 1245–1250 (2006)
11. Ibarria, L., Lindstrom, P., Rossignac, J., Szymczak, A.: Out-of-core compression and decompression of large n-dimensional scalar fields. *Comput. Graph. Forum* **22**(3), 343–348 (2003)
12. Wheeler, D., Burrows, M.: A block-sorting lossless data compression algorithm. Digital Systems Research Center Report, vol. 124 (1994)
13. LZO: real-time data compression library. <http://www.oberhumer.com/opensource/lzo/> (Online)
14. Yeh, P.-S., Xia-Serafino, W., Miles, L., Kobler, B., Menasce, D.: Implementation of ccscs lossless data compression in hdf. In: Earth Science Technology Conference (2002)
15. O’Neil, M.A., Burtcher, M.: Floating-point data compression at 75 gb/s on a gpu. In: Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, p. 7. ACM (2011)
16. Sanchez, V., Nasiopoulos, P., Abugharbieh, R.: Lossless compression of 4d medical images using h. 264/avc. In: 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 2. pp. II–II, IEEE (2006)
17. Woodring, J., Mniszewski, S., Brislawn, C., DeMarle, D., Ahrens, J.: Revisiting wavelet compression for large-scale climate data using jpeg, 2000 and ensuring data precision. In: 2011 IEEE Symposium Large Data Analysis and Visualization (LDAV), pp. 31–38 (2011)
18. Ma, K.-L., Shen, H.-W.: Compression and accelerated rendering of time-varying volume data. In: Proceedings of the 2000 International Computer Symposium-Workshop on Computer Graphics and Virtual Reality, pp. 82–89 (2000)
19. Fout, N., Ma, K.-L., Ahrens, J.: Time-varying, multivariate volume data reduction. In: Proceedings of the 2005 ACM Symposium on Applied Computing. ACM, pp. 1224–1230 (2005)
20. Fout, N., Ma, K.-L.: An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Trans. Comput.* **18**(12), 2295–2304 (2012)
21. Engelson, V., Fritzon, D., Fritzon, P.: Lossless compression of high-volume numerical data from simulations. In: Data Compression Conference. Citeseer (2000)
22. Robinson, T.: Simple Lossless and Near-Lossless Waveform Compression. Cambridge University Engineering Department, Cambridge (1995)
23. Hans, M., Schafer, R.W.: Lossless compression of digital audio. *IEEE Trans. Comput.* **18**(4), 21–32 (2001)
24. Taylor, K., Stouffer, R., Meehl, G.: An overview of CMIP5 and the experiment design. *IEEE Trans. Comput.* **93**(4), 485 (2012)
25. Network Common Data Form. <http://www.unidata.ucar.edu/software/netcdf/> (Online)
26. CMIP5 Output Requirements. <http://cmip-pcmdi.llnl.gov/cmip5/output-req.html> (Online)

27. Earth System Grid Federation. <http://pcmdi9.llnl.gov/esgf-web-fe/> (Online)
28. Songbin, L., Xiaomeng, H., Haohuan, F.: Data reduction analysis for climate data sets. In: 10th IFIP International Conference on Network and Parallel Computing (2013)
29. Rice, R.F.: Practical universal noiseless coding. In: 23rd Annual Technical Symposium. International Society for Optics and Photonics, pp. 247–267 (1979)
30. pigz. <http://zlib.net/pigz>. (Online)
31. Homepage of Martin Isenburg. <http://www.cs.unc.edu/~isenburg/> (Online)
32. SZIP 2.1. <http://www.hdfgroup.org/ftp/lib-external/szip/> (Online)