Eurographics Symposium on Parallel Graphics and Visualization (2014)
M. Amor and M. Hadwiger (Editors)

A Study of Parallel Data Compression Using Proper Orthogonal Decomposition on the K Computer

Chongke Bi[1] and Kenji Ono[1] and Kwan-Liu Ma[2] and Haiyuan Wu[3] and Toshiyuki Imamura[1]

[1]Advanced Institute for Computational Science, RIKEN, Japan
[2]University of California, Davis, USA
[3]Wakayama University, Japan

Abstract

The growing power of supercomputers continues to improve scientists' ability to model larger, more sophisticated problems in science with higher accuracy. An equally important ability is to make full use of the data output from the simulations to help clarify the modeled phenomena and facilitate the discovery of new phenomena. However, along with the scale of computation, the size of the resulting data has exploded; it becomes infeasible to output most of the data, which defeats the purpose of conducting large-scale simulations. In order to address this issue so that more data may be archived and studied, we have developed a scalable parallel data compression solution to reduce the size of large-scale data with low computational cost and minimal error. We use the proper orthogonal decomposition (POD) method to compress data because this method can effectively extract the main features from the data, and the resulting compressed data can be decompressed in linear time. Our implementation achieves high parallel efficiency with a binary load-distributed approach, which is similar to the binary-swap image composition method. This approach allows us to effectively use all of the processors and to reduce the interprocessor communication cost throughout the parallel compression calculations. The results of tests using the K computer indicate the superior performance of our design and implementation.

Categories and Subject Descriptors (according to ACM CCS): Compression [I.4.2]: Approximate methods—; Modes of Computation [F.1.2]: Parallelism and concurrency—

1. Introduction

One major challenge presented by extreme-scale scientific computing is the huge amount of data that the simulation is capable of generating. Since each run of a state-of-the-art simulation can output data at the petascale, storing all of the data is no longer an option. Aside from simply dropping selected time steps, as has been the practice, additional data reduction methods must be considered in order to meet data movement and storage requirements, while maintaining the accuracy and integrity of the data. This is particularly important as scientific supercomputing moves toward exascale.

In order to address the pressing need for further data reduction, we have chosen to develop a scalable data compression solution that will be usable for in situ simulation. Our parallel compression method achieves high compression ratios and is at least as scalable as the simulation on the supercomputer. Furthermore, we have chosen lossy compression in order to benefit scientists who require compression ratios of smaller than the 50 to 70% achieved by state-of-the-art lossless methods [FM12]. Note that the following function is used to define the *compression ratio* in this paper:

$$c = size_{compressed}/size_{original} \times 100\%, \quad (1)$$

where $size_{compressed}$ and $size_{original}$ are the sizes of the compressed dataset and the original dataset, respectively.

In employing lossy compression methods, the following two issues must be considered:

- We must find a balance between the compression ratio and the errors introduced in the context of the data analysis and visualization tasks. Under the premise that important features can be fully retained, large-scale datasets should be compressed as much as possible. Lossy compression methods that can preserve the main features of the data are the most desirable for our purposes.
- In order to address interactive visualization of time-varying datasets, we should be able to decompress the

DOI: 10.2312/pgv.20141078

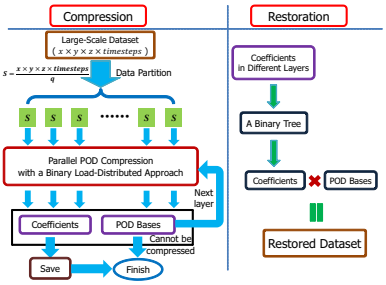


Figure 1: *Flow chart of the parallel data compression and decompression system.*

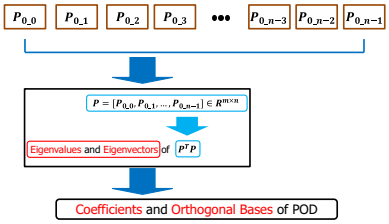


Figure 2: *The POD data compression in serial. P_{0_j} represents the dataset in the j-th time step. m is the size of the dataset in one time step, and n is the number of time steps.*

data quickly. For this requirement, decompression methods in linear time are the best choice.

The proper orthogonal decomposition (POD) method addresses the two issues noted above. The POD method can greatly reduce the size of large-scale datasets [CWJ10] with a small error. The error is analyzed in [HUT06]. However, the high computational cost and the large memory requirements for resolving eigenvalues and eigenvectors of huge matrices arising from large-scale datasets limit the use of traditional POD methods.

In this paper, a novel POD-based parallel compression approach is presented. Our approach fully utilizes the merits of the POD method and also resolves the drawbacks noted above. An overview of our approach is shown in Figure 1. On the left side of the illustration, our parallel compression approach is shown. The linear decompression method is shown on the right side of Figure 1.

As a first step, we address memory size issues by dividing the large-scale datasets into spatial and temporal components. The optimal partition size is determined by analyzing the computational cost of the POD algorithm. Our POD data compression process is carried out in parallel on all of these small blocks. POD bases can be obtained in all small partitions. However, the size of these POD bases remains too large, and they must be compressed again.

Next, we set all of the POD bases as new datasets that are to be compressed again by reapplying the parallel POD scheme. The compressed POD bases have the same properties as the corresponding original datasets. Therefore, if the original datasets can be compressed, the resulting POD bases can also be compressed. For example, we compress a dataset with eight time steps using the POD method. If only one processor is used serially for this compression, the dataset can be compressed into one POD basis. On the other hand, if we use two processors in parallel to compress this dataset, two POD bases will be obtained (one for each processor). In this case, according to the results of serial compression, these two bases can be compressed again by using the POD algorithm one more time. Therefore, this compression process is carried out recursively until the dataset cannot be compressed further. For this process of recursive compression, our implementation achieves high parallel efficiency with a *binary load-distributed approach*. This is similar to the binary-swap image composition method used in parallel volume rendering [MPHK94, YWM08]. This approach allows us to effectively use all of the processors and reduce the interprocessor communication cost throughout the parallel compression calculations. Furthermore, a minimization function is set in order to further reduce errors. Finally, a binary-tree is constructed in order to calculate the coefficients for decompression.

We have tested our design and implementation on the K computer and achieved both good compression ratios and scalable performance. In particular, we show that the interprocessor communication cost decreases as more processors are used, which is key to scalable parallel computing.

2. Related Work

Compression is an effective tool for analyzing and visualize massive scale datasets with small throughput on current hardware devices.

Lossless compression methods are frequently used for datasets that do not allow degradation after compression, such as [LI06, FM12]. Lossy compression methods have been applied to situations where throughput limitations demand high compression ratios and some introduced error is allowable. Abrardo [Abr10] presented a low-complexity lossy compression scheme based on prediction, quantization, and rate-distortion optimization. Lukin et al. [LZPK10] introduced a lossy compression method in which distortions in the compressed data were invisible and provided potential applications of their method. In general, the most significant advantage of lossy compression methods involves their compression speed. Lakshminarasimhan et al. [LSE*11] presented an effective method for in-situ sort-and-B-spline error-bounded lossy abatement (ISABELA) of scientific data that is widely regarded as being effectively incompressible in terms of runtime. Iverson et al. [IKK12] addressed this advantage with a fast and effective lossy compression

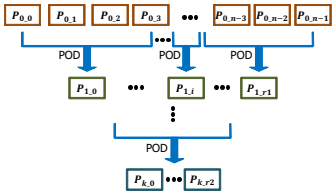


Figure 3: *The POD compression in parallel. The obtained POD bases are recursively compressed until the users' requirements are satisfied. r_i is the number of POD bases.*

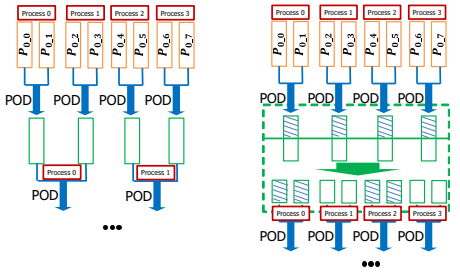


Figure 4: *Utilization rate of processors in parallel, where the binary load-distributed approach is (b) used and (a) not used. In the case of (a), Processor 2 and 3 are idle in the second compression layer.*

method that can be used for scientific simulation data computed on structured and unstructured grids. In addition, error control is of key importance in lossy compression methods. García-Vílchez et al. [GVMMZ*11] developed a method for evaluating the error impact for different compression ratios.

Following the prior research, we have developed a lossy compression method using the POD algorithm in parallel with high parallel efficiency, low computational cost, small compression ratio, and minimized error. Readers can find more information about POD in [Rao11].

3. Parallel Data Compression Using the POD Algorithm

In this section, we describe the details of our parallel POD data compression method. First, a basic POD algorithm in serial will be explained with the reasons that it is unsuitable for use in large-scale dataset compression. We then parallelize the POD algorithm to achieve low computational cost and high parallel efficiency. This is achieved by using an optimal partition method and a binary load-distributed approach. The partition method will be introduced in Section 3.2, and the binary load-distributed approach will be described in Section 4. Finally, an error minimization scheme is used to minimize the error of the parallel POD algorithm. This error minimization function can further reduce the size of compressed data by replacing the mean value vector of the original POD algorithm.

3.1. The POD Algorithm in Serial

The POD method is usually used to analyze the main components of scalar data and cannot be used directly for large-scale scientific datasets, such as simulation data, which are often four-dimensional time-varying datasets. In order to use the POD method for compression of such types of multidimensional simulation datasets, we should first transform the dataset in each time step into a vector $\boldsymbol{P}_{i\_j}$. Note that $\boldsymbol{P}_{i\_j}$ represents the dataset of *layer* i and *timestep* j, where $i = 0$ represents the original dataset (Figure 2), and $i > 0$ represents the corresponding POD bases in *layer* i, as shown in Figure 3. Then, the data for all of the time steps are transformed into two-dimensional datasets, which can be represented as a matrix $\boldsymbol{P} \in \boldsymbol{R}^{m \times n}$, where $m = x \times y \times z$ is the number of data points in one time step, and n is the number of time steps of the dataset.

Then, the orthogonal bases and corresponding coefficients of POD are obtained. This can be achieved by calculating the eigenvalues λ_j and eigenvectors $\boldsymbol{e}_j$ of matrix $\boldsymbol{P}^T\boldsymbol{P}$, where $j \in (0, 1, ..., n-1)$. For example, the orthogonal bases of POD in *layer* 1 $\boldsymbol{P}_{1\_j}$ can be obtained using Eq. (2).

$$\boldsymbol{P}_{1\_j} = \boldsymbol{P}\boldsymbol{e}_j / \sqrt{\lambda_j}. \tag{2}$$

However, matrix $\boldsymbol{P}$ is usually much larger than the memory size. Furthermore, the computational cost of large-scale matrix calculation is significantly high. Therefore, the serial POD method cannot be directly used for large-scale data compression. We introduce a parallel POD method to overcome these drawbacks.

Note that, in general, not all of the eigenvalues of the POD method are save. These eigenvalues are usually ordered from large to small initially, and then several larger ones will be selected as the final result. In our parallel POD compression method, only the largest one is saved.

3.2. Efficient Parallel POD Data Compression

In order to parallelize the POD algorithm, two issues are very important. The first one is that the compressed data should be small. That is, the number of POD bases from the parallel algorithm is larger than that from the serial algorithm. For example, we compress a dataset with eight time steps using the POD algorithm. If only one processor is used for this compression in serial mode, then the dataset can be compressed into one POD basis. On the other hand, if two processors are used to compress the dataset, at least two POD bases will be obtained (one for each processor). The second issue is to decide an optimal partition size to reduce computational cost.

In order to address the first issue, we use a recursive algorithm in our approach. As shown in Figure 3, the original datasets $\boldsymbol{P}_{0\_j}$ is first compressed using POD in parallel. Then we compress the obtained POD bases again, applying the same parallel POD algorithm. The POD bases have the same properties as the corresponding original datasets. Therefore, if the original datasets can be compressed, the resulting POD bases can also be compressed. This process is performed recursively until the users' requirements are satisfied. Here, the users' requirements include a better compression ratio and minimal error. The balance of these two factors is very important. If the number of layers of recursive compression is increased by one, then the compression ratio is decreased but the error is increased. Actually, this is a useful mechanism by which users can easily control the balance between compression ratio and error.

With respect to the second issue, the total computational cost of the parallel POD algorithm depends on the product of the matrix computational cost and the recursive layers, as shown in Eq. (3). Temporal partitioning can greatly reduce the matrix computational cost, while increasing the recursive layers. For example, a dataset with eight time steps is compressed using POD in parallel. If we divide the eight time steps into four groups, the number of layers of recursive compression becomes three. If we divide the eight time steps into two groups, the number of layers of recursive compression becomes two. In order to obtain the optimal computational efficiency, we analyze the computational cost of Eq. (3) to decide the optimal partition size. Note that we define the matrix of the divided small group of the original dataset as $\boldsymbol{P}_s \in \boldsymbol{R}^{m_s \times n_s}$, where m_s and n_s are the space size and time steps of small group datasets.

$$cost = O((m_s n_s^2 + n_s^3 + m_s n_s r) \times log_{n_s}^{n}). \tag{3}$$

Here, $log_{n_s}^{n}$ denotes the recursive layers, and $m_s n_s^2 + n_s^3 + m_s n_s r$ is the computational cost of one small group, as described in Section 3.1. In other words, $m_s n_s^2$, n_s^3, and $m_s n_s r$ correspond to the computational costs of $\boldsymbol{P}_s^T \boldsymbol{P}_s$, the eigenvalues and eigenvectors of $\boldsymbol{P}_s^T \boldsymbol{P}_s$, and the bases of POD in Eq. (2), respectively. Here, r is the numbers of bases. In order to minimize the computational cost shown in Eq. (3), the parameters should be determined as follows: 1) m_s should be as small as possible based on the memory size and 2) m_s should also be as small as possible, therefore n_s is set as 2. As shown in Figure 4(a).

However, Figure 4(a) reveals another problem. In the second recursive layer of POD calculation, only half of the processors are used, whereas the remaining processors are idle. In order to resolve this problem, we have developed a binary load-distributed approach, which is introduced in Section 4.

3.3. Error Minimization

The POD method is used for analyzing the principle component of the dataset. The POD method allows the most important bases analyzed from the dataset to be kept, while the remaining unimportant bases are ignored. These ignored bases introduce error into the results, although the error is usually very small. In this section, we minimize the error of the POD method without increasing the compressed data size.

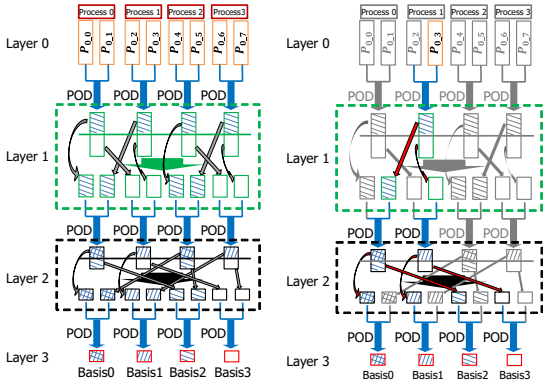


Figure 5: *Example of a binary load-distributed approach. (a) The green and black regions divide the dataset for the binary load-distributed approach. The black arrows in the green and black dashed boxes indicate the directions of the distribution of the divided datasets for full use of all processors. (b) Decompression of the dataset of time step* 3 P_{0_3}. *The blue arrow indicates the process of decompression.*

Assume that the original dataset of timestep i is $\boldsymbol{P}_{0\_i} = [v_0, v_1, ..., v_{m_s}]^T$, while the corresponding compressed data is $\boldsymbol{P}_{0\_i}{}' = [v_0', v_1', ..., v_{m_s}']^T$. Now, we introduce x to the compressed data $\boldsymbol{P}_{0\_i}{}' = [v_0' + x, v_1' + x, ..., v_{m_s}' + x]^T$ to minimize the error. This can be achieved by minimizing the function $f(x)$ in Eq. (4). The optimal value of x can be obtained as Eq. (5).

$$f(x) = \sum_{j=0}^{m_s} \frac{((v_j' + x) - v_j)^2}{v_j^2}. \tag{4}$$

$$x = (\sum_{j=0}^{m_s} \frac{v_j' - v_j}{v_j^2}) / (\sum_{j=0}^{m_s} \frac{1}{v_j^2}). \tag{5}$$

4. Binary Load-Distributed Approach

In this section, we describe a binary load-distributed approach, which enables us to achieve high parallel efficiency.

4.1. Improvement of Computing Efficiency

Figure 4(a) shows parallel data compression without the binary load-distributed approach. In the second recursive process, only *Processor* 0 and *Processor* 1 are working, and the

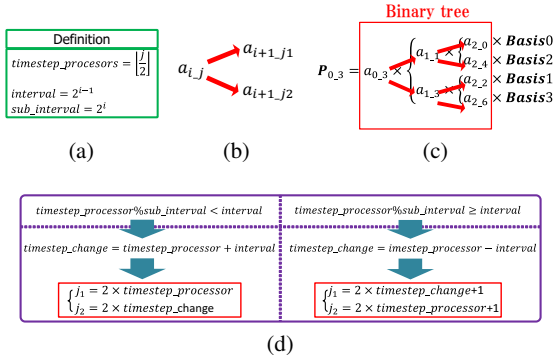


Figure 6: *Construction of a binary-tree for obtaining all coefficients to decompress compressed datasets. (a) Several definitions. (b) Relationship between a father processor and two son processors. (c) Binary tree constructed for the coefficients for decompressing the third timestep in Figure 5 (b). (d) Algorithm for computing two son processors. Note that a_{i_j} is the coefficient of the i-th layer and the j-th time step.*

other two processors are idle. As the number of recursive layers increases by one, half of the remaining processors become idle. A binary load-distributed approach can resolve this problem. In the green dashed box of Figure 4(b), we halve the POD bases in one time step and exchange half of the data of each processor with half of the data of another processor. Therefore, the computational cost can be greatly reduced because the size of the dataset matrix is halved. As the number of recursive layers increases, the burden of each processor is reduced by half again. If the cost of compressing the datasets for two time steps is s, then the total cost of compressing the eight time steps without using the binary load-distributed approach in Figure 4(a) is $s+s+s=3s$. On the other hand, the computing cost of Figure 4(b) is only approximately $s+s/2+s/4=7s/4$.

4.2. Improvement of Communication Efficiency

Using the binary load-distributed approach for the POD data compression method requires data exchange among different processors. There are two important rules for data exchange, and these rules are also used in the process of decompressing the compressed datasets in Section 5. We will explain in detail the data exchange rules using a simple example.

First, we preset the two rules for data exchange.

- The spatial region of the datasets to be compressed in each processor must be the same. In Figure 5 (a), only the space with the same filler can be compressed together.
- In the *layer i*, the data exchange should be conducted between *processor k* and *processor* $k+2^{i-1}$, where $k \in \{0, 1\times(2^{i-1}+1), 2\times(2^{i-1}+1)...\}$.

As shown in Figure 5 (a), in *layer* 0, the original eight-time-step dataset has been compressed into four POD bases of *layer* 1 (indicated by the green dashed box). The green line divides the four POD bases in half, where the upper half is indicated by the *slash* marks. According to the first data exchange rule, the regions with the *slash* marks should be exchanged within the same processors. According to the second rule, the data should be exchanged as $\{processor\,0 \leftrightarrow processor\,1\}$, $\{processor\,2 \leftrightarrow processor\,3\}$. In the same manner, in *layer* 2 of the black dashed box, the datasets with the same filler should be exchanged with each other. The data should be exchanged as $\{processor\,0 \leftrightarrow processor\,2\}$, $\{processor\,1 \leftrightarrow processor\,3\}$. Finally, for the deepest layer (the *layer* 3 of Figure 5 (a)), it is not necessary to divide the datasets again. Note that the length of the four bases in the deepest layer of Figure 5 (a) is $m_s/4$. In the process of decompression, they are used to decompress four parts in the space of one time step.

As described above, the data transfer cost is also greatly reduced. In Figure 5 (a), the transfer costs with and without binary load-distributed approach of are $m_s/2+m_s/4=3m_s/4$ and $m_s+m_s=2m_s$, respectively.

The binary load-distributed approach can reduce both the computational cost and the communication cost.

5. Decompressing the Compressed Datasets

In this section, a linear decompression method is introduced. This method allows users to interactively analyze and visualize the compressed dataset. The basic idea is to linearly decompress the different parts of the datasets in one time step. These parts constitute the entire time step. In this process, it is not necessary to decompress the POD bases in all layers, and only the corresponding coefficients need to be obtained. This is achieved through constructing a binary tree.

5.1. Linear Decompression

Figure 5 (b) shows an example how to decompress the dataset *time step* 3 P_{0_3} using the linear method. In this process, the dataset will be decompressed as four parts from top to bottom. This is determined by the compression process. Figure 5 (b) indicates the four parts. In the green dashed box, the compressed green POD bases are divided into two small green POD bases for the binary load-distributed approach. In the same manner, two divided green POD bases are compressed into two black POD bases, which are divided into four POD bases again for further compression. Finally, four red POD bases are generated. The size of each POD basis is just $1/4$ that of the original dataset. Therefore, the four parts of the original dataset can be decompressed by using the four red POD bases with the corresponding coefficients.

5.2. Constructing the Coefficient Binary Tree

Figure 6(c) shows the constructed binary tree for decompressing the dataset of *time step* 3, which is represented by $\boldsymbol{P}_{0\_3}$ in the figure. As described above, the subscript indicates the *layer* 0 and *time step* 3. Two parameters must be determined in order to construct such a binary tree. The first parameter is *time step* j in coefficients a_{i_j}. The second one is the subscript k of the POD basis $\boldsymbol{Basis}k$.

First, we will introduce the second parameter, because this parameter can be easily defined. Suppose that the coefficient of the deepest layer is a_{l_t}. Then, the subscript should be $k = t/2$. For example, in Figure 6(c), $a_{2\_4} \times \boldsymbol{Basis}2$.

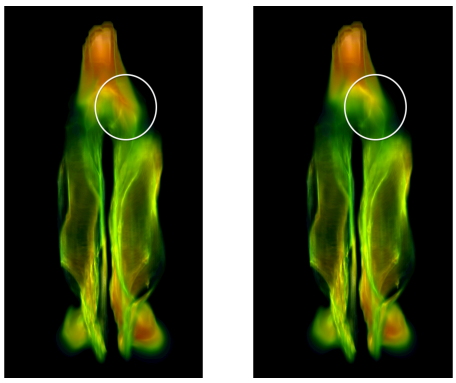
We then describe how to construct the coefficient binary tree. The basic concept is to define the two child coefficients a_{i+1_j1} and a_{i+1_j2} of coefficient a_{i_j} recursively, as shown in Figure 6(b). As in the second exchange rule defined in Section 4.2, we need only recover the two exchanged processors for constructing the binary tree. Suppose that the current processor is *timestep_processor*, which should be half of time step j in coefficient a_{i_j}. Therefore, as shown in the definition of Figure 6(a), $timstep\_processor = \lfloor j/2 \rfloor$. Suppose the exchanged processor is *timestep_change*, which may be defined in two ways depending on whether the *timstep_processor* is that on the left ($timstep\_processor \Leftrightarrow timestep\_change$) or that on the right ($timestep\_change \Leftrightarrow timstep\_processor$). The second exchange rule of Section 4.2 specifies, as shown in the definition of Figure 6(a), the interval between them is $interval = 2^{i-1}$. According to the two possible positions of *timstep_processor*, there are two definitions of the subscript, i.e., j_1 and j_2, as shown in the left- and right-hand sides of Figure 6(d). Now, we can use the coefficients and POD bases to easily and efficiently decompress the compressed dataset.

6. Results and Discussions

This section demonstrates the effectiveness of our approach by compressing three sets of dataset of practical simulation results on the K computer. The results will be evaluated based on computational cost, communication cost, compression ratio, error, and standard deviation of the error.

The prototype system is implemented on the *K* computer at RIKEN, Japan. The K computer has 88,128 nodes and a six-dimensional mesh torus interconnect *TOFU*network, each node has eight cores, 16 GB of memory, and achieves 128 GFLOPS. The theoretical peak performance of the entire system is 11.28 PFLOPS. The source code has been written in C++. In addition, we used the LAPACK library for the eigenvalue and eigenvector calculation.

The first experiment compares the visualization results between the original dataset and the compressed dataset, as shown in Figure 7. The data is a sneeze flow simulation by volume rendering of the magnitude of the velocity. Figure 7 (a) and (b) show the velocity fields of the original and compressed datasets, respectively. The error of the time step is 1.1%. It can be seen that the two visualisation results are almost the same.



(a) (b)

Figure 7: *Result of compressing the time-varying dataset of a sneeze flow simulation by volume rendering of the magnitude of velocity. The size of the dataset is* $121 \times 361 \times 181 \times 11$. *(a) Original dataset. (b) Compressed dataset.*

The second experiment is to compress a large-scale time-varying dataset of the flow simulation in the air jet mixer of a machinery. The size of the dataset is $300 \times 200 \times 200 \times 128$ $[x_{size} \times y_{size} \times z_{size} \times timesteps]$. Figure 8 (a) shows the visualization result for one time step in the dataset. Figure 8 (b) shows the computational cost and the interprocessor communication cost when the datasets are compressed with the largest compression ratio, 0.78%. The results fully validate the high efficiency of our parallel compression algorithm. As the number of processors increased, the computational cost and the communication cost decrease. This demonstrates the effectiveness of the binary load-distributed approach. Figure 8 (c) shows the tested compression ratio using different numbers of POD bases. Here, the compression ratio is calculated according to Eq. (1) and is also shown in Figure 8 (c). The compression ratio is 0.78% with one POD basis. In other words, we achieve a size reduction of approximately 99.22%. Moreover, the compression ratios with two POD bases and four POD bases are 1.56% and 3.12%, respectively. Figure 8 (d), (e), and (f) demonstrate the corresponding errors, while the standard deviations of the errors are shown in Figure 8 (g), (h), and (i). As the compression ratio increases, the error and the standard deviation of error become smaller. Here, Figure 8 (d) and (g) show the error and the standard deviation of error using 1,600 processors. Figure 8 (e) and (h) show the error and the standard deviation of error using 4,800 processors, and Figure 8 (f) and (i) show the error and the standard deviation of error using 9,600 processors. Note that, several *waveforms* are observed in the results, proving the validity of our implementation. In other

words, one waveform indicates that only one POD basis is retained. Moreover, the error is larger than that for the results with two and four waveforms, whereas the compression ratio is lower. Here, the number of compression layers for one waveform is one greater than that for two waveforms and two greater than that for four waveforms. Therefore, users can control the balance of error and compression ratio by changing the deepest compression layer.

The third experiment involves compressing a large-scale time-varying dataset of the flow simulation around a car. The size of the dataset is $1,000 \times 400 \times 250 \times 128$. Based on the visualization results shown in Figure 9(a), this flow simulation contains much more turbulence. The data size is greater than that of the first dataset for the air jet. As in the first experiment, the computational cost and the interprocessor communication cost shown in Figure 9(b) are for the largest compression ratio, 0.78%. The compression ratios with one, two, and four POD basis are 0.78%, 1.56%, and 3.12%, respectively. The high efficiency of our parallel compression algorithm and the effectiveness of the binary load-distributed approach are thus demonstrated. In this experiment, much higher numbers of processors are used to compress the dataset in order to test the effectiveness and validity of the parallel compression method. Here, 3,200 processors, 8,000 processors, and 16,000 processors are used, respectively. As the number of calculation processors increases, especially at 16,000 processors, the error decreases. Furthermore, the standard deviation of the error fluctuates significantly due to the strong turbulence in the dataset. However, the largest error is still small. Therefore, the result of compressing a dataset with significant turbulence further demonstrates the high efficiency and effectiveness of the parallel data compression method.

7. Conclusion

We have experimentally studied our parallel data compression design and implementation on the K computer and proved it a scalable data reduction solution with low computational cost and minimized error. The scalable performance is achieved by ensuring the communication cost decreases as additional processors are used. In the future, we intend to integrate this technology with several simulation codes in order to achieve and evaluate in situ data reduction. We also intend to investigate what compression ratios are acceptable for subsequent data analysis and visualization tasks and may incorporate domain knowledge when setting the compression level. Furthermore, instead of simply using LAPACK to resolve all the eigenvalues and the eigenvectors, we plan to use the parallel matrix library EigenK for the K Computer [IYM12] to test the performance of our framework.

Acknowledgements

Part of the results were obtained by using the K computer at the RIKEN Advanced Institute for Computational Science. This work has been partially supported by JSPS under KAKENHI(Series of single-year grants) No. 26120534.

References

[Abr10] ABRARDO A.: Low-complexity lossy compression of hyperspectral images via informed quantization. In *Proceedings of IEEE International Conference on Image Processing* (2010), pp. 505–508. 2

[CWJ10] CHEN F., WEN F., JIA H.: Algorithm of data compression based on multiple principal component analysis over the wsn. In *The 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)* (2010), pp. 1–4. 2

[FM12] FOUT N., MA K.-L.: An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2295–2304. 1, 2

[GVMMZ*11] GARCÍA-VÍLCHEZ F., MUÑOZ-MARÍ J., ZORTEA M., BLANES I., GONZÁLEZ-RUIZ V., CAMPS-VALLS G., PLAZA A., SERRA-SAGRISTA J.: On the impact of lossy compression on hyperspectral image classification and unmixing. *Geoscience and Remote Sensing Letters 8*, 2 (2011), 253–257. 3

[HUT06] HUTH R.: The effect of various methodological options on the detection of leading modes of sea level pressure variability. *Tellus A 58*, 1 (2006), 121–130. 2

[IKK12] IVERSON J., KAMATH C., KARYPIS G.: Fast and effective lossy compression algorithms for scientific datasets. In *Euro-Par 2012 Parallel Processing*, vol. 7484 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 843–856. 2

[IYM12] IMAMURA T., YAMADA S., MACHIDA M.: Eigen-k: high performance eigenvalue solver for symmetric matrices developed for K computer. In *The 7th International Workshop on Parallel Matrix Algorithms and Applications* (2012). 7

[LI06] LINDSTROM P., ISENBURG M.: Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1245–1250. 2

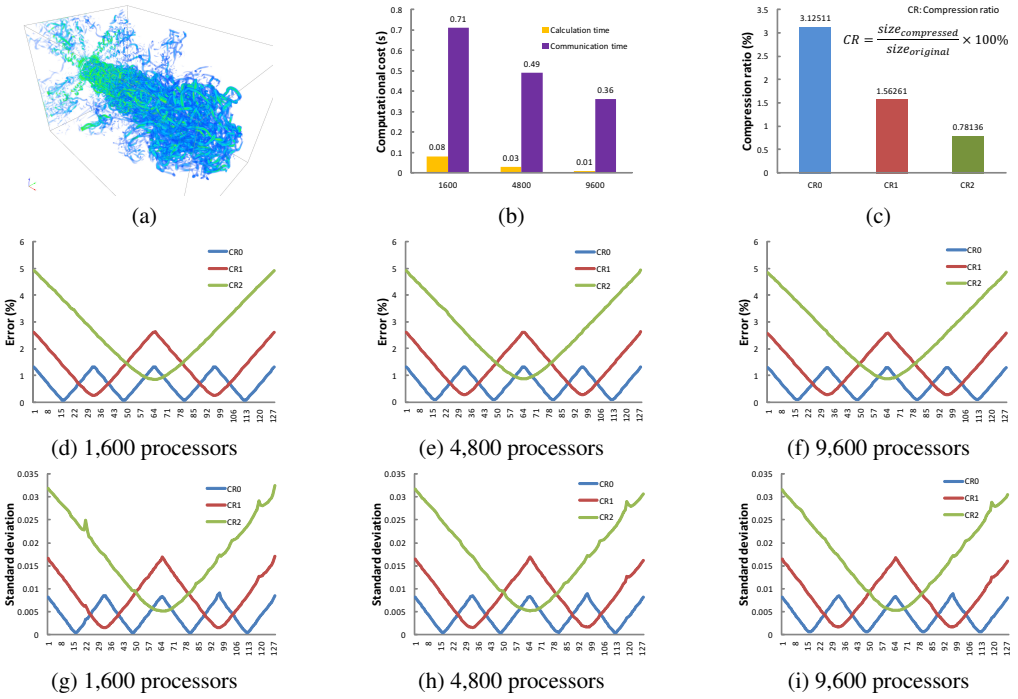
[LSE*11] LAKSHMINARASIMHAN S., SHAH N., ETHIER S., KLASKY S., LATHAM R., ROSS R., SAMATOVA N.: Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *Euro-Par 2011 Parallel Processing*, vol. 6852 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 366–379. 2

[LZPK10] LUKIN V. V., ZRIAKHOW M., PONOMARENKO N., KRIVENKO S.: Lossy compression of images without visible distortions and its application. In *Proceedings of IEEE 10th International Conference on Single Processing* (2010), pp. 698–701. 2

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl. 14*, 4 (July 1994), 59–68. 2

[Rao11] RAO R. V.: *Advanced Modeling and Optimization of Manufacturing Processes*. Springer Series in Advanced Manufacturing, 2011. 3

[YWM08] YU H., WANG C., MA K.-L.: Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), pp. 1–11. 2

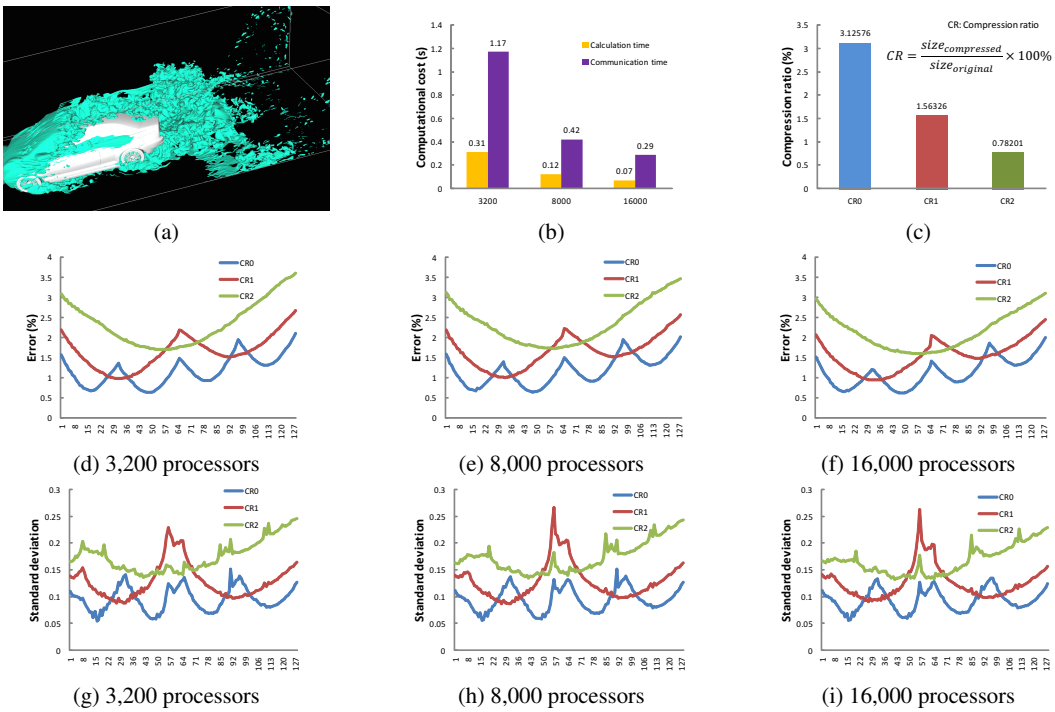


(a) (b) (c)

(d) 1,600 processors (e) 4,800 processors (f) 9,600 processors

(g) 1,600 processors (h) 4,800 processors (i) 9,600 processors

Figure 8: *Results of compressing the data obtained in a flow simulation in an air jet mixer.*



(a) (b) (c)

(d) 3,200 processors (e) 8,000 processors (f) 16,000 processors

(g) 3,200 processors (h) 8,000 processors (i) 16,000 processors

Figure 9: *Results of compressing the data obtained in a flow simulation around a car.*