UNIVERSITY OF CALIFORNIA
RIVERSIDE

Toward Resilience and Data Reduction in Exascale Scientific Computing

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Xin Liang

December 2019

The Dissertation of Xin Liang is approved:

Committee Co-Chairperson

Committee Co-Chairperson

University of California, Riverside

Acknowledgments

First and foremost, I am extremely grateful to my Ph.D. advisors, Dr. Zizhong Chen and Dr. Franck Cappello, for their guidance, patience, and support during my Ph.D. study. I would like to thank Dr. Zizhong Chen for leading me into the research in the field of high-performance computing and giving me suggestions on both research and career. Without him, I would not have been here to purse my Ph.D. degree in Computer Science. Without him, I would not decide to continue my career in academia. Dr. Zizhong Chen supported me to explore whatever interesting research directions that I wanted to pursue, in the meantime, contributing valuable comments, advice, and encouragements. I would like to thank Dr. Franck Cappello for supervising my research work when I was taking the long-term internship at Argonne National Laboratory. I have learned a lot from his rigorous altitude and critical thinking toward research, and have been deeply motivated by his research enthusiam. Dr. Franck Cappello taught me to seriously treat every detail discovered during the study and encouraged me to exchange ideas with a wide range of people across various domains. I appreciate both Dr. Zizhong Chen and Dr. Franck Cappello for engaging me in new ideas and demanding high-quality work.

I would like to thank Dr. Rajiv Gupta, Dr. Zhijia Zhao, Dr. Daniel Wong, and Dr. Sheldon Tan for serving on my Ph.D. dissertation committee. Their insightful comments and valuable suggestions helped me finish a well-structured and high-quality thesis.

I am very lucky to have the opportunities to collaborate with many excellent researchers and scientists during my internships in national laboratories. Particularly, I am especially grateful to Dr. Sheng Di for his guidance and support during my long-term internship at the resilience group in Argonne National Laboratory. Also, I sincerely appreciate the guidance and help from Dr. Hanqi Guo and Dr. Thomas Peterka for my internship at the Parallel Extreme-Scale Data Analytics team in Argonne National Laboratory. I would like to thank Dr. Abhinav Vishnu for providing me the internship at Pacific Northwest National Laboratory, and Dr. Qiang Guan and Dr. James Ahrens for providing me the internship at Los Alamos National Laboratory. Moreover, I would like to thank my collaborators, Dr. Kristopher Keipert, Dr. Ali Murat Gok, Dr. Mukund Raj, Dr. Bogdan Nicolae, Dr. Jeff Daily, Ollie Lo, Xinyu Chen, and Robert Underwood. I cannot finish the thesis without their help.

I am indebted to all members of the Supercomputing Laboratory at UCR, Dr. Li Tan, Dr. Panruo Wu, Dr. Dingwen Tao, Dr. Hongbo Li, Dr. Jieyang Chen, Sihuan Li, Kaiming Ouyang, Yuanlai Liu, Kai Zhao, Yujia Zhai, Quan Fan, and Elisabeth Giem. I would like to thank everyone for their good advice, collaboration, and assistance.

Funding Acknowledgement I am grateful to the funding supports from the National Science Foundation under Grant No. 1305624, No. 1305622, No. 1513201, and No. 1619253, the Exascale Computing Project 17-SC-20-SC (a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration), the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357, the SZSTI basic research program JCYJ20150630114942313, and the MOST key project 2017YFB0202100.

Publication Acknowledgement I acknowledge that part of this thesis was published previously in the following conferences.

- Chapter 2 was previously published [62] in the Proceedings of the 29th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, USA, Nov 12 - 17, 2017.

- Chapter 3 was previously published [63, 65] in the Proceedings of the 2018 IEEE International Conference on Big Data, Seattle, WA, USA, Dec 10-13, 2018 and the Proceedings of the 31st ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, USA, Nov 17 - 22, 2019.

- Chapter 4 was previously published [64,66] in the Proceedings of the 2018 IEEE International Conference on Cluster Computing, Belfast, UK, September 10 - 13, 2018 and the Proceedings of the 2019 IEEE International Conference on Cluster Computing, Albuquerque, New Mexico, USA, September 23 - 26, 2019.

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Toward Resilience and Data Reduction in Exascale Scientific Computing

by

Xin Liang

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2019
Dr. Zizhong Chen, Co-Chairperson
Dr. Franck Cappello, Co-Chairperson

Because of the ever-increasing execution scale, reliability and data management are becoming more and more important for scientific applications. On the one hand, exascale systems are anticipated to be more susceptible to soft errors ,e.g. silent data corruptions, due to the reduction in the size of transistors and the increase of the number of components. These errors will lead to corrupted results without warning, making the output of the computation untrustable. On the other hand, large volumes of highly variable data are produced by scientific computing with high velocity on exascale systems or advanced instruments, and the I/O time on storing these data is prohibitive due to the I/O bottleneck in parallel file systems. In this work, we leverage algorithm-based fault tolerance (ABFT) and error-bound lossy compression to tackle the two problems, in order to support efficient scientific computing on exascale systems.

We propose an efficient fault tolerant scheme to tolerant soft errors in Fast Fourier Transform (FFT), one of the most important computation kernels widely used in scientific computing. Traditional redundancy approaches will at least double the execution time or resources, limiting the usage in practice because of the large overhead. Previous works on offline ABFT algorithms for FFT mitigate this problem by providing resilient FFT with lower overhead, but these algorithms fail to make progress in vulnerable environments with high error rates because they can only detect and correct errors after the whole computation finishes. We propose an online ABFT scheme for large-scale FFT inspired by the divide-and-conquer nature of the FFT computation. We devise fault tolerant schemes for both computational and memory errors in FFT, with both serial and parallel optimizations. Experimental results demonstrate that the proposed approach provides more timely error detection and recovery as well as better fault coverage with less overhead, compared to the offline ABFT algorithm.

To alleviate the I/O bottleneck in the parallel file systems, we work on a prediction-based error-bounded lossy compressor to significantly reduce the size of scientific datasets while retaining the accuracy of the decompressed data, with adaptive prediction algorithms and compression models. We first propose a regression-based predictor for better prediction accuracy than traditional approaches under large error bounds, followed by an adaptive algorithm that dynamically selects between the traditional Lorenzo predictor and the proposed regression-based predictor, leading to very high compression ratios with little visual distortion. We further unify the prediction-based model and transform-based model by using transform-based compressors as a predictor, with novel optimizations toward efficient coefficient encoding for both the two models. The proposed adaptive multi-algorithm design provides better compression ratios given the same distortion, significantly reducing storage requirements and I/O time.

We further adapt the compression algorithms and compressors to different requirements and/or objectives in realistic scenarios. We leverage a logarithmic transform to precondition the data, which turns a relative-error-bound compression problem into an absolute-error-bound compression problem. This transform aligns two different error requirements while improving the compression quality, efficiently reducing the workload for compressor design. We also correlate the compression algorithm with system information to achieve better I/O performance compared to traditional single compressor deployment. These studies further improve the efficiency of lossy compression from the perspective of efficient I/O in the context of scientific simulation, making scientific applications running on exascale systems more efficient.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Resilience and data reduction are two of the most important problems for scientific applications in the comming exascale era. Due to the increasing complexity of the systems as well as renewed emphasis on limiting power and energy consumption, the failure rate on exascale systems is expected to keep growing [17, 39], resulting in soft errors in both logic circuits and memory subsystems. These soft errors do not lead to noticeable system crashes, but to silent data corruption (SDC) that corrupts the output without warning. This phenomenon has already been observed on several real-world leadership-class supercomputers, which severely affects the reliability of the output from large-scale scientific applications. How to make these applications resilient to such errors is still an open question. On the other hand, extreme-scale scientific simulations and experiments on scientific instruments are already generating more data than that can be stored, transmitted and analyzed. The up-comping exascale systems and higher-resolution scientific instruments are going to exacerbate this problem, due to the slow increase of storage capacity and transmission bandwidth compared with the fast increase of data generation. Data compression is a direct way to cope with such difficulty, because it can drastically reduce the storage requirement and accelerate data transmission. Currently, scientists choose to perform the analysis on decimated data, which leads to suspect results that cannot be trusted.

Although many techniques have been proposed to detect and/or correct soft errors in fault tolerance literature, the system-level approaches can be very expensive on HPC systems. For example, the best-known general technique to detect soft errors is the double modular redundancy (DMR) approach, which either uses two different hardware units to perform the same computation at the same time or performs the same computation on the same hardware twice, then compares the two results to detect whether errors occur or not, leading to at least 200% overhead to detect errors. The most well-known general technique to correct single soft errors is the triple modular redundancy (TMR) approach, which either performs the same computation on three different hardware units or uses the same hardware to perform the same computation for three times, then compares and votes the majority results as the correct result, resulting in 200% overhead for error correction.

In order to mitigate the high overhead from the system-level approaches, algorithm-based fault tolerance (ABFT) is proposed to leverage the semantics and structure of a specific application for resilience. It is dedicated to the detection and correction of errors in a certain algorithm, sacrificing part of the generality for performance. ABFT is first proposed by Abraham and Huang [44] for matrix multiplication where massive computations are necessary and the overhead of DMR/TMR is not acceptable, and applied to various aspects [6, 10–14, 18–20, 22, 31, 34, 38, 48, 57, 61, 76, 80, 83, 84, 86, 88, 97, 112, 114]. Later, the

ABFT has also been extended by [29, 111, 113] to detect and correct errors online so that lower latency detection and faster recovery can be achieved.

Fast Fourier transform (FFT) is one of the most important kernels widely used in scientific applications. Because of its importance and impact in wide areas, fault tolerant FFT schemes have been extensively studied over the last three decades. In [5], Antola et al. proposed a time-redundant scheme for 2-dimensional FFT. In [24], Choi and Malek introduced a fault tolerance scheme that is based on recomputing through an alternate path. But their throughput is only 50%. In [50], Jou and Abraham proposed an ABFT scheme for the FFT networks. They can achieve 100% fault coverage and throughput theoretically at the cost of $O(\frac{2}{\log_2 N})$ hardware overhead. But either fault coverage or throughput may not be very satisfactory due to round-off errors. Later, Tao and Hartmann [98] came up with a novel encoding scheme for FFT networks which has higher fault coverage by adding 5% hardware. They also gave some formulas to estimate the upper and lower bound of fault coverage that influenced by round-off error. After that, Wang and Jha [105] presented a new concurrent error detection (CED) scheme that achieves same or better result with less or equal hardware redundancy. Also, Oh [75] showed a similar CED scheme using a different checksum aimed at increasing fault coverage. However, these approaches suffer from either high overhead or low fault coverage. They also have high detection latency such that they can only detect and correct the error after the whole computation. To address this problem, we propose an online resilient scheme for FFT by leveraging its divide-and-conquer nature, which is able to detect and correct errors with low overhead, high fault coverage and timely fault detection and correction.

Another problem arising from exascale scientific computing is storage and transmission of the extremely large volume of data produced scientific applications. These big data are generally stored in a parallel file system (PFS), with limited storage space and limited I/O bandwidth to access. Some climate studies, for example, need to run large ensembles of 1 km$\times$1 km simulations, with each instance simulating 15 years of climate in 24 h of computing time. Every 16 seconds, 260 TB of data will be generated across the ensemble, when estimating even one ensemble member per simulated day [35]. For another instance, the Hardware/Hybrid Accelerated Cosmology Code (HACC) [43] can simulate 1$\sim$10 trillion particles in one simulation [103], producing up to 220 TB of data for each snapshot, for a total of 22 PB of data if there are 100 snapshots during the simulation. Even considering a sustained bandwidth of 500 GB/s, the I/O time will still exceed 10 hours, which is prohibitive. As a result, the researchers has to output the data by decimation, in other words, storing one snapshot every K time steps in the simulation. This process definitely degrades the temporal constructiveness of the simulation and also loses valuable information for postanalysis.

Compared to the traditional decimation approach, designing efficient data compressors is an alternative way to address the big data problem. Error-controlled lossy compression techniques have been proposed as an option, becase lossless compressors such as [4,15,25,30,45,71,79,115,116] cannot achieve high compression ratios [68,87] whereas general lossy compressor compressors [99,104] cannot guarantee the fidelity of the decompressed data. In the past decade, several error-controlled lossy data compressors (including [2,3,9,32,55,59,67,69,81,91,92,94]) have been developed to significantly reduce the scientific data size for different purposes [16]. These lossy compressors can be classified into two groups, based on how they decorrelate the original data. The first group of compressors [9, 58, 67, 81, 85] use a transform-based model that leverages invertible transforms for the decorrelation. The second group of compressors [32, 55, 60, 64, 69, 91, 92] use a prediction-based model that leverages various prediction methods for the decorrelation. Generally speaking, no compression model can always outperform the others. Even for the same dataset, the best-fit model may differ depending on distortions. Hence, the method must be carefully selected at runtime. Inspired by this nature, we propose adapive compression algorithms with combined prediction methods and compression models to improve the quality of error-bounded lossy compression.

The adaptive compression algorithms may not be sufficient for scientific applications due to the variaties in the requirements and/or objectives. For example, scientific application users may prefer relative error bound to traditional absolute error bound, and favor I/O performance over storage reduction. In comparison with the absolute error bound that has been widely used to control the data distortion by existing state-of-the-art lossy compressors [67,92], pointwise relative error bound is significant for many scientific applications but much tougher to deal with than absolute error bound according to the principles of lossy compressions. Under such requirement, the smaller the data value is, the lower the absolute error bound is applied on the data point. Some application users demand pointwise relative error bound based on the physical meaning of the simulation. According to cosmologists (such as the users and developers of HACC and NYX), for instance, the higher a particle's velocity is, the larger the compression error it can tolerate. Similarly, I/O performance may be more important to users when minimal storage requirements are met. To deal with these variaties, we use an efficient logarithmic transform to solve the relative error bound requirement and propose a general compression framework for I/O performance.

1.1 Problem Statement

This thesis mainly tackle the two important problem arising from exascale scientific computing, namely resilience and data reduction. The algorithms and implementations are designed for scientific applications on high-performance computing facilities, especially the up-coming exascale systems. The targets are large-scale scientific applications including, but not limited to, cosmological simluations such as HACC [43] and NYX [73], climate simulation such as Hurricane ISABEL [46] and CESM [53], molecular dynamic simulations such as EXAALT [77], quantum simulations such as QMCpack [54], scientific instruments from facilities such as Argonne Advanced Phonton Source (APS) and Linear Coherent Light Source (LCLS) [78], and so on. For resilience, soft errors including single and multiple bit-flips in on-chip and off-chip memory systems and computation error in logic circuits are both considered.

1.2 Thesis Statement

The proposed fault tolerant scheme for FFT is able to detect and correct errors with low overhead, high fault coverage and timely fault detection and correction in FFT computations widely used in scientific applications. The proposed compression algorithms can significantly reduce the storage and I/O burden for scientific applications while guaranteeing the fidelity of the decompressed data. The adaption to different requirements and objectives can further improve the efficiency of the compression algorithms for scientific applications.

1.3 Contribution

1.3.1 For Soft Errors in FFT

To achieve low overhead, high fault coverage and timely fault detection and correction in FFT computations, we take advantage the divide-and-conquer nature of the algorithm to protect the divided FFTs, with multiple serial and parallel optimizations. Specifically, the contributions of this work include

- **The first online ABFT scheme for FFT:** Existing ABFT schemes for FFT [5,24, 50,74,75,98,105] detect soft errors offline after the FFT computation finishes. Even if an error occurs at the beginning of the FFT, existing ABFT schemes can not detect it in a timely manner, hence, have to allow the corrupted computation to continue until it finishes, then verify the correctness. After an error is detected, the whole FFT computation has to be restarted. This paper designs an online ABFT scheme that is able to detect errors online soon after the error occurs so that the corrupted computation can be terminated in a timely manner. After the corrupted computation is terminated, instead of repeating the whole computation from the beginning, the proposed online ABFT scheme only need to repeat a small fraction the computation, which greatly improves the computation efficiency when errors occur.

- **The first soft-error-resilient FFT software implementation - FT-FFTW:** Existing FFT ABFT schemes are mostly designed under the context of hardware implementation. This paper develops soft-error-resilient FFT software for the first time. We develop FT-FFTW, which incorporates both the existing offline ABFT and the proposed online ABFT into one of the today's fastest FFT software libraries - FFTW, and validate the implementations on TIANHE-2 supercomputer. Experimental results demonstrate that the proposed online ABFT is able to detect soft errors in a timely manner and improve the computation efficiency by a factor of two when errors occur.

- **Multiple optimizations for online ABFT FFT:** It is very challenging to add fault tolerance capability to the highly optimized FFTW library without introducing significant performance penalty. Simply applying existing ABFT to each small FFTs within a large FFT introduces too much overhead. This paper develops several optimization strategies to reduce the overhead. The optimized online ABFT FFT introduces lower overhead than the existing offline scheme even if no error occurs.

- **The first online ABFT scheme for parallel in-place FFT:** Different from the out-of-place sequential FFT, the parallel FFT tends to use in-place FFT with no auxiliary space. We develop an online ABFT scheme for in-place FFT and extend our FFT ABFT scheme from sequential to parallel.

- **Parallel optimization strategy to minimize the overhead:** We develop a communication-computation overlap strategy to hide half of the fault tolerance cost for our parallel FT-FFTW. With the re-designed plan, the parallel FT-FFTW is able to achieve comparable performance to the original FFTW library.

- **Significant improvement in numerical stability and fault coverage:** Round-off errors for floating point calculations affect the numerical stability and fault coverage. This paper analyzes the impact of round-off errors for our online ABFT scheme in detail and shows that our online ABFT scheme has higher numerical stability and better fault coverage than the existing schemes.

1.3.2 For Error-bounded Lossy Compression Algorithms

Inspired by the fact that the best-fit model differ in distortions, we propose adapive compression algorithms with mutiple prediction methods and hybrid compression models, as well as selection method to automatically choose the best-fit methods/models to improve the quality of error-bounded lossy compression. The contributions are

- We propose an adaptive lossy compression framework that is more effective in compressing the scientific datasets with relatively large error bounds. In particular, we split the whole dataset into multiple non-overlapped blocks, and select the best-fit prediction method based on their data features.

- We develop new prediction methods that are particularly effective for the lossy compression with relatively large error bounds. On the one hand, we develop a hybrid Lorenzo prediction method by combining the classic Lorenzo predictor [47] and the densest mean-value based data approximation method (also called mean-integrated Lorenzo prediction). On the other hand, we develop a linear regression method that can obtain much higher prediction accuracy in this case since the design is beyond the limitation that the decompressed values have to be used in the prediction.

- We explore how to select adaptively and efficiently the best-fit prediction method based on the data features across blocks during the compression.

- We improve the coefficient-encoding efficiency for the data-fitting predictor such that the compression ratio is improved significantly for relatively high-compression cases.

- We design a transform-based predictor by adopting the transform techniques of transform-based models in the prediction stage of prediction-based models. This is the first such design, to the best of our knowledge.

- We optimize the encoding strategy for the transform-based predictor, significantly improving the compression ratio for cases requiring relatively high compression ratios.

- We develop a best-fit predictor selection algorithm that can automatically select the best predictor between the data-fitting one and transform-based one during the compression.

1.3.3 For Error-bounded Lossy Compression With Different Requirements/Objectives

With respect to different requirements and objectives from scientific application, we propose an efficient logarithmic transform, which serves as a predictioner for scientific data, to deal with relative error bound as well as a general compression framework for I/O performance. The contributions are

- We formalize the transformation problem between absolute error bound and relative error bound in the context of lossy data compression mathematically. We also solve this problem and find the unique mapping function, which is consistent with using logarithmic change for error measurement.

- We investigate the impact of the selection of different logarithmic bases on the compression quality for SZ and ZFP, respectively. We prove that various bases lead to the similar compression results theoretically.

- We propose an efficient pointwise relative-error-bounded lossy compression algorithm by combining the logarithmic data transform scheme and state-of-the-art absolute-error-bounded compressors. Specifically, we integrate the transformation scheme into both SZ and ZFP.

- We propose an optimized data dumping performance model that can effectively represent the writing performance with different execution scales and data sizes.

- We analyze our proposed performance model in terms of the lossy compression technique, which is a fundamental guideline to develop an efficient algorithm for optimizing the data dumping performance.

- We develop an adaptive lossy compression framework with a series of optimization strategies to improve the dumping performance for the scientific simulations with error-bounded lossy compressors. The optimized framework has two critical steps: compression quality estimation and online optimization of compression settings.

Chapter 2

Correcting Soft Errors Online in Fast Fourier Transform

2.1 Introduction

Fast Fourier Transform (FFT) is widely used to compute the discrete Fourier transform (DFT). DFT plays a very important role in engineering, science, and mathematics. Therefore, reliable and fast computing of DFT will benefit not only a large number of people but also a wide range of fields.

As the size of transistors continues to reduce and the number of components continues to increase, large-scale FFTs are also susceptible to soft errors in supercomputers. As mentioned before, system-level approaches such as DMR and TMR introduces at least 100% overhead for error deteation and 200% overhead for error correction, significantly impacting its practical use. While many ABFT schemes have been proposed for FFT over the past thirty years, a careful review of the existing ABFT literature indicates that no previous ABFT schemes can detect and correct soft errors online before an FFT computation finishes. To address this issue, this paper proposes an online ABFT scheme to efficiently tolerant computational and memory errors in FFT by taking advantages of its divide-and-conquer nature. Both serial and parallel optimizations have been adopted to reduce the fault tolerant overhead. Specifically, the contributions of this paper can be summarized as follows:

- **The first online ABFT scheme for FFT:** Existing ABFT schemes for FFT [5,24, 50,74,75,98,105] detect soft errors offline after the FFT computation finishes. Even if an error occurs at the beginning of the FFT, existing ABFT schemes can not detect it in a timely manner, hence, have to allow the corrupted computation to continue until it finishes, then verify the correctness. After an error is detected, the whole FFT computation has to be restarted. This paper designs an online ABFT scheme that is able to detect errors online soon after the error occurs so that the corrupted computation can be terminated in a timely manner. After the corrupted computation is terminated, instead of repeating the whole computation from the beginning, the proposed online ABFT scheme only need to repeat a small fraction the computation, which greatly improves the computation efficiency when errors occur.

- **The first soft-error-resilient FFT software implementation - FT-FFTW:** Existing FFT ABFT schemes are either designed for hard errors or designed under the context of hardware implementation. This paper develops soft-error-resilient FFT software for the first time. We develop FT-FFTW, incorporate both the existing offline ABFT and the newly proposed online ABFT into one of the today's fastest FFT software libraries - FFTW, and validate the implementations on TIANHE-2 supercomputer. Experimental results demonstrate that the proposed online ABFT is able to detect soft errors in a timely manner before the computation finishes and improve the computation efficiency by a factor of two when errors occur.

- **Innovative optimizations for online ABFT FFT:** It is very challenging to add fault tolerance capability to the highly optimized FFTW library without introducing significant performance penalty. Simply applying existing ABFT to each small FFTs within a large FFT introduces too much overhead. This paper develops several optimization strategies to reduce the overhead. The optimized online ABFT FFT introduces lower overhead than the existing offline scheme even if no error occurs.

- **The first online ABFT scheme for parallel in-place FFT:** Different from the out-of-place sequential FFT, the parallel FFT tends to use in-place FFT with no auxiliary space. We develop an online ABFT scheme for in-place FFT and extend our FFT ABFT scheme from sequential to parallel.

- **Parallel optimization strategy to minimize the overhead:** We develop a communication-computation overlap strategy to hide half of the fault tolerance cost for our parallel FT-FFTW. With the re-designed plan, the parallel FT-FFTW is able to achieve comparable performance to the original FFTW library.

- **Significant improvement in numerical stability and fault coverage:** Round-off errors for floating point calculations affect the numerical stability and fault coverage.

This paper analyzes the impact of round-off errors for our online ABFT scheme in detail and shows that our online ABFT scheme has higher numerical stability and better fault coverage than the existing schemes.

2.2 Background

2.2.1 DFT and FFT

The DFT for a complex sequence can be calculated as follows:

$$X_j = \sum_{n=0}^{N-1} x_n \omega_N^{jn}, j = 0, 1, \ldots, N-1$$

where $\omega_N = \exp^{-i\frac{2\pi}{N}}$ and $i = \sqrt{-1}$ is the unit imaginary root. Correspondingly, the inverse discrete Fourier transform (IDFT) can be calculated as:

$$X_j = \frac{1}{N} \sum_{n=0}^{N-1} x_n \omega_N^{-jn}, j = 0, 1, \ldots, N-1$$

If DFT or IDFT is calculated directly, it is obvious that $O(N^2)$ operations are needed as each element costs $O(N)$ operations. To save more time, the fast Fourier transform (FFT) has been proposed to reduce the number of operations to $O(N \log N)$. The most popular Cooley-Tukey algorithm for FFT can be derived as follows. If the size N can be factorized into two smaller integers as $N = N_1 N_2$, (1) can be rewritten by letting $j = j_1 N_2 + j_2$ and $n = n_2 N_1 + n_1$:

$$X_{j_1 N_2 + j_2} = \sum_{n_1=0}^{N_1-1} (\sum_{n_2=0}^{N_2-1} (x_{n_2 N_1 + n_1} \omega_{N_2}^{n_2 j_2}) \omega_N^{n_1 j_2}) \omega_{N_1}^{n_1 j_1}$$

$\sum_{n_2=0}^{N_2-1} x_{n_2 N_1 + n_1} \omega_{N_2}^{n_2 j_2}$ is an N_2-point DFT and $\sum_{n_1=0}^{N_1-1} (\ldots) \omega_{N_1}^{n_1 j_1}$ is an N_1-point DFT. Thus the original N-point DFT is decomposed to N_1 inner DFTs of size N_2 and N_2 outer DFTs of size N_1. These N_1-point DFTs and N_2-point DFTs can also be decomposed into DFTs of smaller sizes recursively. By this means, the total operations of DFT is reduced to $O(N \log N)$.

2.2.2 Previous Fault Tolerant Work for FFT

Many ABFT schemes have been designed to detect and correct soft errors in FFT. These schemes typically use concurrent error detection scheme with encoding and decoding system. To illustrate how these ABFT schemes work, we take Wang's approach in [105] as an example. As a special case of matrix-vector multiplication, a DFT can be written into matrix form according to equation (1):

$$\begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} \omega_N^0 & \omega_N^0 & \dots & \omega_N^0 \\ \omega_N^0 & \omega_N^1 & \dots & \omega_N^{n-1} \\ \omega_N^0 & \omega_N^2 & \dots & \omega_N^{2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^0 & \omega_N^{n-1} & \dots & \omega_N^{(n-1)^2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}$$

Let A denote the coefficient matrix where $A_{ij} = \omega_N^{ij}$, X denotes the output vector, x denotes the input vector, the matrix form can be simply written as $X = Ax$. The equation maintains by multiplying X and Ax with a selected checksum vector r:

$$\begin{bmatrix} X \\ rX \end{bmatrix} = \begin{bmatrix} Ax \\ rAx \end{bmatrix}$$

r is called the weighted checksum for this matrix operation. The last row of the matrix can be expanded as:

$$\sum_{j=0}^{N-1} r_j X_j = \sum_{j=0}^{N-1} (rA)_j x_j$$

Then by comparing the results of the two checksums, any computational error can be detected.

However, not all checksum schemes are suitable for ABFT FFT. It has been proved in [105] that the following checksum scheme works well for ABFT FFT:

$$r = (\omega_3^0, \omega_3^1, \ldots, \omega_3^{N-1})$$

where $\omega_3 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$ is the first cube root of 1.

As for error correction, time redundancy methods are preferred in almost all the approaches. Re-calculation is necessary to produce the correct result.

Algorithm 1 Offline ABFT FFT Algorithm

```
1: Set the calculation flag $calcFlag = true$
2: Calculate input checksum vector $c = rA$
3: **while** $calcFlag$ **do**
4:     Calculate the FFT: $X = Ax$
5:     $calcFlag = (|rX - cx| > \eta)$
6: **end while**
```

All of these ABFT schemes mentioned above are proposed for hardware implementation with an assumption that the size of input is fixed for a specific FFT implementation. Under this assumption, the input checksum vector rA can be pre-calculated when output checksum vector r is given. Then they detect errors by comparing the difference of rX and rAx. However, software FFT implementations usually accept varying sizes of input, and thus extra overhead will be introduced to calculate rA. The software-level implementation of this approach is shown in Algorithm 1.

2.3 Online ABFT FFT Schemes

To correct errors in a more timely manner, two online schemes are proposed in this section. As faults are categorized into two types in this work, the first subsection introduces an online scheme aiming at computational faults while the second subsection proposes an online scheme that can deal with both computational faults and memory faults. The computational fault tolerant scheme in the first subsection is also complementary to ECC memory. It can detect and correct computational errors that ECC may not be able to handle.

2.3.1 Computational Fault Tolerance

A basic idea to detect errors in FFT online is to use a hybrid scheme of offline ABFT and DMR. For example, an N-point can be decomposed into 2 $\frac{N}{2}$-point FFTs, twiddle multiplication and then $\frac{N}{2}$ 2-point FFTs. The first 2 $\frac{N}{2}$-point FFTs can be executed and checked one by one. If any error is detected in any one of the 2 $\frac{N}{2}$-point FFTs, the corrupted $\frac{N}{2}$-point FFT will be re-executed. It would be faster than the offline when an error occurs since the offline one can only detect the error at the end of the computation and restart the whole computation.

Although this approach is able to detect errors online, it suffers from certain performance loss when the chosen radix is small. Experiments show that the performance loss varies from 5% to 20% for tested initial radices and FFT sizes according to Table 2.1.

To keep comparable performance with the original FFT computation, the default radix of $\sqrt{N}$ should be adopted. However, the DMR part will introduce a lot of overhead

Table 2.1: FFTW Execution Time (Seconds) of Different Radices on TIANHE-2

Initial Radix	$N = 2^{25}$	$N = 2^{26}$	$N = 2^{27}$	$N = 2^{28}$
$k = 4$	4.27	8.79	18.64	38.21
$k = 8$	4.19	8.63	17.79	37.43
$k = 16$	4.27	8.87	17.34	37.64
$k \approx \sqrt{N}$(default)	3.67	7.91	16.79	35.1

in this situation. For example, if the default initial radix is equal to $\sqrt{N}$, the DMR part will incur about 50% overhead as the re-calculated part will take up around half time of the total computation. Because of the divide-and-conquer nature of FFT computation, it is noticeable that the residue computation contain a twiddle multiplication with another FFT (2-point FFT in the above example). Therefore, we leverage this algorithmic characteristic and offline ABFT FFT scheme to propose an online ABFT scheme for FFT computation. The key idea is to protect each decomposed FFT by offline ABFT. Taking the tradeoff of fault tolerant ability and overhead into consideration, we opt to a two-layer approach that leverages the highest level of decomposition of a Cooley-Tukey FFT to protect the first part and second part by two separate ABFT schemes.

The structure of online ABFT scheme is shown in Fig. 2.1. From the view of the highest level of decomposition, an N-point FFT is calculated by computing k m-point FFTs, twiddle multiplications and m k-point FFTs when $N = m * k$. The k m-point FFTs can be protected separately by the ABFT approach. So can the m k-point FFTs. On the other hand, the twiddle multiplication that is left over can be protected by DMR with low

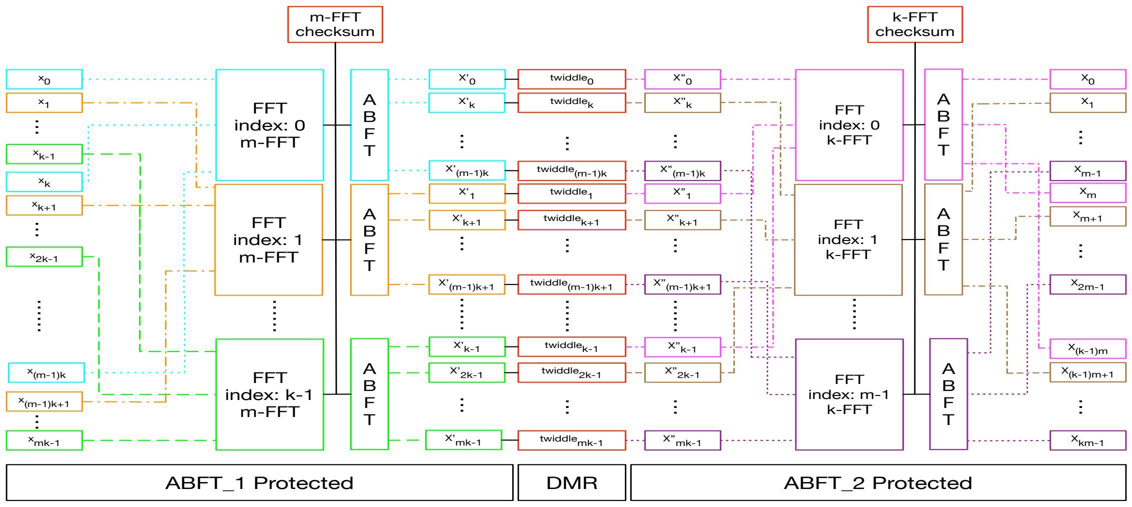


Figure 2.1: The Two-Layer ABFT FFT Scheme (When $N = m * k$)

overhead because it is memory-intensive. The colored parts in this figure are protected by their own FFTs while the red parts, including the twiddle multiplication and input checksum vector generation, are protected by DMR.

Algorithm 2 Online ABFT FFT Algorithm

```
1: Get initial radix $k$ and corresponding $m = \frac{N}{k}$
2: Calculate checksum vector $c_m = r_m A_m$ with DMR
3: for $i$ from 0 to $k-1$ do
4:     Set the calculation flag $calcFlag = true$
5:     while $calcFlag$ do
6:         Calculate the $i$-th FFT: $X'_i = A_m x_i$
7:         $calcFlag = (|r_m X'_i - c_m x_i| > \eta_1)$
8:     end while
9: end for
10: Calculate checksum vector $c_k = r_k A_k$ with DMR
11: for $i$ from 0 to $m-1$ do
12:     Multiply twiddle factor $X''_i = twd_i .* X'_i$ with DMR
13:     Set the calculation flag $calcFlag = true$
14:     while $calcFlag$ do
15:         Calculate the $i$-th FFT: $X_i = A_k X''_i$
16:         $calcFlag = (|r_k X''_i - c_k X_i| > \eta_2)$
17:     end while
18: end for
```

The corresponding algorithm of this approach is shown in Algorithm 2. At first, the input checksum $c_m = r_m A_m$ is calculated, and the m-point FFTs are executed and verified one by one. If there is error in the i-th FFT, it can be detected by comparing the checksum $c_m x_i$ and $r_m X_i$ and corrected by an immediate re-execution of current FFT. Note that the re-execution overhead is negligible because the arithmetic operations in the decomposed FFT is only $\frac{1}{2\sqrt{N}}$ of the total ones. After the output is verified to be correct, each element in the intermediate output will multiply itself with the corresponding twiddle factor $(\omega_N^{n_1 j_2})$ to generate the input for the latter k-point FFTs. Then the k-point FFTs are executed and verified in a similar way. However, if an error strikes the twiddle multiplication, the ABFT scheme cannot detect the error since the input has already been corrupted. Therefore, online DMR is equipped for the twiddle multiplication. Each multiplication is executed twice and verified immediately to ensure correctness. If an error is detected here, a third execution is performed and the final result would be the majority of the three executions. Since computation would only happen in one of the three parts or in the checksum calculation, any single computational error can be revealed. Besides these parts, the other parts are protected by one and only one ABFT FFT so that no computation is wasted. This ensures no masked error and no repeated protection on the same data.

The two-layer online scheme only needs to compute two input checksum vectors of size m and k while the offline one needs to compute one input checksum vector of size N. As this computation is one of major overhead, the online scheme should have better performance. Furthermore, since each small FFT is equipped with separate protection, the online scheme is expected to achieve timely recovery when errors occur.

2.3.2 Memory Fault Tolerance

Besides the logic units, faults may also strike memory to cause memory errors. This may be even more common than computational errors. If memory fault strikes some intermediate result during computation in some decomposed FFT, this error would behave like a computational error and can be detected and recovered by the ABFT schemes above. However, if it strikes the input before the calculation or the output after the calculation, the error cannot be detected by this scheme alone. Thus, more strict mechanisms are needed to tolerate memory faults.

As usual, two checksums $r_1 = (1, 1, \dots, 1)$ and $r_2 = (1, 2, \dots, n)$ are used to detect and recover from a memory error. If any error occurs and changes the input x_j into x'_j, the difference will be:

$$r_1x - r_1x' = x_j - x'_j$$

$$r_2x - r_2x' = j(x_j - x'_j)$$

Then the error can be located by $(r_2x - r_2x')/(r_1x - r_1x')$ and corrected by adding $r_1x - r_1x'$ to the corrupted value.

In our fault model for the memory faults, we assume that memory faults would not occur when the checksums are being generated, otherwise, the error cannot be detected by ABFT approaches. This is reasonable because the checksum generation would only take very little time (the time complexity is $O(N)$ and its coefficient is very small). Our basic idea to detect memory error is to verify data before use. Denote CCG as computational checksum generation, MCG as memory checksum generation, CCV as computational check-

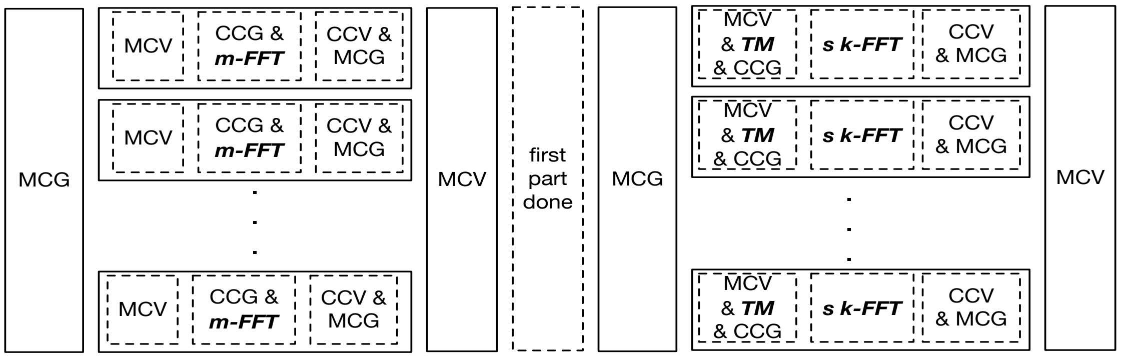


Figure 2.2: Hierarchy of Memory Protection

sum verification, MCV as memory checksum verification, TM as twiddle multiplication, s as the number of FFTs to be computed together, then the hierarchy of memory protection is shown in Fig. 2. Bold italic operations are original operations in FFTW. To ensure the correctness of the input, memory checksums of each m-point FFT are generated before any of the m-point FFT calculations. Then the k m-point FFT calculations would start one by one and verifications are invoked at the beginning of these calculations. If an error occurs, the corrupted input will be located and recovered by the 2 checksums and a restart will be performed immediately. Otherwise, the computation is thought as fault-free and memory checksums for the intermediate output are generated. These checksums will be used for verification before twiddle multiplication to make sure there is no memory error in the output between the end of this m-point FFT and the end of all the k m-point FFTs.

The same goes for the second part. Each k-point FFT needs memory checksum verification before computation, computational checksum verification, and output memory checksum generation after computation. At last, the final output is verified to ensure correctness of the result.

Besides the protection of the input, output and intermediate result, the input checksum vector rA for the m-point FFT and k-point FFT should also be checked. These verifications can be done in time intervals related to the error rate, which is quite feasible across the whole computation. As there is only $O(\sqrt{n})$ time consumed in each verification, it introduces little overhead.

2.4 Sequential Optimizations

This section introduces sequential/serial optimizations that we apply to minimize overhead.

2.4.1 Memory Checksum Modification

Though the traditional checksums $r_1 = (1, 1, \ldots, 1)$ and $r_2 = (1, 2, \ldots, n)$ work well for correcting memory error, they may involve redundant computation because they do not make use of the computational checksum $r = (\omega_3^0, \omega_3^1, \ldots, \omega_3^{N-1})$. Since rAx will be calculated under any circumstance to detect computational error, r_1 can be replaced by $r_1' = r$ directly to save the computation time of $r_1 x$. Correspondingly the j-th element in the second checksum r_2 can be replaced by $(r_2')_j = j * (rA)_j$. Similar to the original checksum r_1 and r_2, the difference the new checksums would be:

$$r_1'x - r_1'x' = (rA)_j(x_j - x_j')$$

$$r_2'x - r_2'x' = j * (rA)_j(x_j - x_j')$$

Then the error can be located by $(r_2'x - r_2'x')/(r_1'x - r_1'x')$ and correction can be done by adding $(r_1'x - r_1'x')/(rA)_j$ to the corrupted element. As the generation time for r_1' and r_2'

is $O(\sqrt{N})$, the extra overhead on input checksum vector generation is negligible. On the other hand, it saves the checksum generation time since it only costs $10N$ operations ($8N$ for $r_1'x$, $2N$ for $r_2'x$) while the original one costs $14N$ ($8N$ for rx, $2N$ for r_1x, $4N$ for r_2x).

2.4.2 Verification & Correction Postponing

According to Fig. 2, there is input memory checksum generation when FFT starts, followed immediately by memory checksum verification and m-point FFTs. Inspired by the fact that the errors, both computational errors and memory errors, would propagate to the end of each decomposed FFT, MCVs before each m-point FFT can be postponed to the CCVs after this m-point FFT. Since CCV can detect the error, the postponed MCV is eliminated.

Similarly, the MCVs after k-point FFTs can be postponed to the final MCV and these MCVs as well as the MCGs after k-point FFTs can be eliminated for lower overhead. Unfortunately, this cannot be done directly since the second part is always done in-place where the input will be overwritten by the output. If the output verification is postponed, the error can still be detected since the checksums will not match. However, it cannot be corrected since the input is overwritten. Thus, another copy of the intermediate output is needed. It can be copied to the original input array for no extra memory. Though the copy operation also involves N elements, it would be much faster than the original redundant MCVs and MCGs.

Besides, the correction operations $r_2'x$ can be postponed to the time when an error is detected at the cost of slower recovery. It trades higher overhead in error-occurring runs for lower overhead in error-free runs.

2.4.3 Incremental Checksum Generation

From Section 3.2, the MCG before twiddle multiplication is necessary because there is a rearrangement of data between the two ABFT parts. However, the verification mechanism still seems inefficient since each element is verified twice. In this optimization, we use incremental generation for the checksums to reorganize the memory checksums efficiently.

After the input checksums are generated in the beginning, extra space is allocated to store the information of the output. Unlike the previous approach, these output checksums directly store the checksums for the k-point FFTs in the second part. At first, these checksums are initialized to 0. At the end of each m-point FFT, the k outputs increase their corresponding slots by their own value, i. e. the first element X'_0 would increase the first slot in the checksum by X'_0 while the second element X'_1 would increase the second slot by X'_1. By this means, the j-th slot in the checksum would be the checksum for elements in the j-th k-point FFT. Thus only one verification is needed before the second part.

2.4.4 Non-contiguous Memory Access

When a big FFT is broken down into smaller ones, the inputs of each smaller FFT would be non-contiguous as the first k m-point FFTs in Fig. 2. The stride (distance between adjacent inputs) of each m-point FFT would be $2k$. It is usually $O(\sqrt{N})$ and will result in low spatial locality in the cache. Besides basic use in FFT to compute the result, the inputs are also needed in CCGs and MCVs. Another read would be relatively expensive since there would be cache misses all the time, which leads to large overhead. This happens

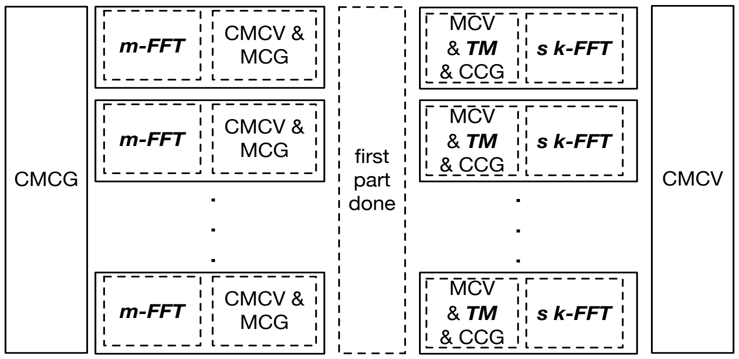


Figure 2.3: Optimized Hierarchy of Memory Protection

to MCG in the first part. To resolve this, the corresponding MCGs are brought forward to the beginning of all the m-point FFTs and the new MCGs are computed via the incremental checksum generation approach above. It actually accesses each element twice. But each access has little low overhead due to cache reuse.

Denote CMCG as the modified checksum generation and CMCV as the modified checksum verification in Section 4.1, the hierarchy of memory protection can be simplified to Fig. 2.3 with all the optimizations above. Compared to the original memory fault tolerance scheme (Fig. 2.2), the optimized one is much simpler and faster.

2.5 Online ABFT FFT on Parallel Systems

FFT of large sizes becomes very common nowadays [28]. Therefore, FFT may need to be performed in parallel to avoid the limited memory and low computational efficiency on single processor when FFT size becomes large. Although the idea of sequential ABFT FFT can be borrowed, challenge comes that parallel FFTs are always done in-place for better utilization of memory. In-place and out-of-place are property of an algorithm. In-place means that the algorithm will be done without auxiliary data structure. To make it simple for FFT, the in-place algorithm will store the output in the original input memory and does not bother to allocate a new memory space of size N. The out-of-place algorithm will allocate the memory space to store output in the beginning of the algorithm.

To compute parallel FFT, $FFTW$ tends to choose a plan which computes $\frac{N}{p}$ p-point FFTs at first and then p $\frac{N}{p}$-point FFTs. Unfortunately, the data needed for each FFT is not always on the same processor. Thus communication among processors is needed during the computation. Assume FFT size is N and the number of processors is p. Data on each processor is divided into p blocks of size $\frac{N}{p^2}$. Then a six-step algorithm that involves 3 transpositions is adopted for 1D parallel FFTs. A transposition is a communication that exchanges the i-th block of data in processor i with the j-th block of data in processor j for all i and j from 0 to $p-1$. Denote the $\frac{N}{p^2}$ p-point FFTs on a processor as FFT_1 and the latter $\frac{N}{p}$-point FFT as FFT_2. The first transposition is performed at first to deliver data needed for FFT_1 to the same processor. Then FFT_1 is done on each processor in parallel. After that, the second transposition occurs to exchange data for FFT_2. FFT_2 is performed as the next step. When FFT_2 is done, the third transposition is executed to deliver data to its belonging processor. At last, there is some local adjustment to place the final output in a correct order.

Because original input will be overwritten by output, the restart would not work for in-place FFTs. Fig. 4 shows the flowchart of adding fault tolerance to in-place FFTs. Compared to the out-of-place protection in the sequential scheme, input in each in-place

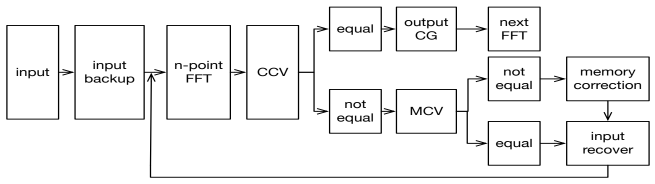


Figure 2.4: Flowchart of Protected In-Place FFT.

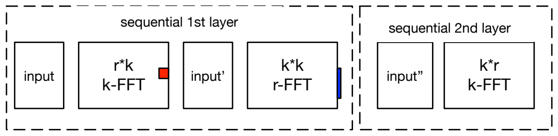


Figure 2.5: Sequential ABFT scheme no longer works: if an error occurs in the red part, it will be detected in the blue part. At this time, the procedure has to fail since original input is overwritten. Twiddle multiplication is omitted.

FFT should have a backup in case an error occurs. Also, checksum verifications should be done immediately after the output is generated. When a memory error is detected, it should be corrected right away. After that, the input will be recovered by the backup and a restart will be performed.

FFT_1 can be protected by the mechanism above because each p-point FFT only asks for $2p$ space. However, FFT_2 cannot be protected in this way because space will be doubled. Fortunately, the idea of the online sequential ABFT scheme can be applied here for timely detection, faster recovery and less space overhead because FFT_2 will be decomposed to smaller FFTs. Nevertheless, the sequential ABFT scheme cannot be leveraged directly because in-place FFTs tend to select a different execution plan from out-of-place FFTs for efficiency. For example, if $\frac{N}{p}$ is a square number, $FFTW$ may choose a plan similar to the out-of-place one to employ a two-layer decomposition; if it is not, i. e. $\frac{N}{p} = r * k^2$, $FFTW$ would prefer a more complicated plan. It may perform $r * k$ k-point FFTs at first, then do twiddle multiplications and k^2 r-point FFTs, finally another twiddle multiplications and $r*k$ k-point FFTs. In this situation, the original two-layer online ABFT can no longer work as shown in Fig. 5. Because the FFT is done in-place, the initial input is overwritten after the $k*r$ k-point FFTs so any restart after the $r*k$ k-point FFTs cannot be performed. A checkpoint for input would definitely work here. However, it will have 100% space overhead and longer correction time.

The solution to this kind of plan is to add one flexible verification layer between the original two layers. The added layer would be protected by DMR since r is usually small (2 or 8 for $\frac{N}{p}$ is a power of 2), making the k m-point FFTs an ABFT-DMR scheme. As the execution time of the DMR part is very small (the same magnitude of the time for checksum generation and verification), we can assume there is no memory error in this part. Then the input verification can be brought forward to all the DMR computations and the output checksum generation can be postponed to end of this part.

Besides the modifications on fault-tolerant mechanisms, there are some modifications on communication as well. In order to detect and correct errors that occur in communication, checksums for communicated data should be generated and sent. As there are only 2 checksums for each block of communicated data, the communication overhead would be negligible.

This scheme can be optimized by some of the optimizations mentioned in previous part. After these optimizations, it is good from the sequential point of view because there are no redundant checksum generations and verifications.

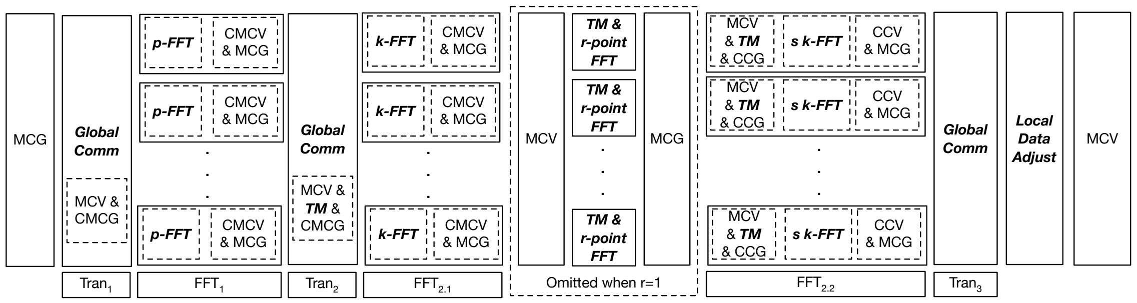


Figure 2.6: Online ABFT Scheme for Parallel In-Place FFT After Communication-Computation Overlap.

2.6 Parallel Optimizations

Besides sequential optimizations that mentioned in previous part, we also adopt several optimizations specifically for parallel FFTs. In this section, we introduce the parallel optimizations that are incorporated into our implementation for lower overhead.

2.6.1 Computation-Communication Overlap

In FFTW, blocking communication is used for transpositions. It is good because the following step usually needs data from all processors so the non-blocking method would have little benefit. However, the checksum generation and verification in the ABFT FFT scheme are totally uncorrelated with FFT computation, showing great potential for computation-communication overlap.

The adopted communication-computation overlap algorithm is shown in Algorithm 3. It is very similar to the idea of pipeline. It doubles the number of send buffer and receive buffer. When $Isend()$ is used to send data in send buffer sb_1 and $Irecv()$ is used to receive data in receive buffer rb_1, data received in another receive buffer rb_2 can be processed and data to be sent in another send buffer sb_2 can be generated. When these operations are done, $Wait()$ can be used to wait for communication. After that, data in rb_1 can be processed and data to be sent to next processor can be generated in sb_1 while sending data in sb_2 and receiving data in rb_2.

With this technique, MCV and CCG before the p-point FFTs can be overlapped with $transpose_1$. MCV, TM and CMCG before the k-point FFTs can be overlapped with $transpose_2$. Besides, the send buffer initialization and receive buffer data transfer in each communication can also be overlapped.

The online ABFT scheme for parallel in-place FFT after overlap is shown in Fig. 2.6. Bold italic operations are original operations in FFTW. This overlap is optimal since all the other operations are either in the critical path or dependent on the communication. Also, this optimization can be applied to $FFTW$ to overlap the twiddle multiplication in $FFT_{2.1}$ with communication.

2.6.2 Re-design Plan

Since the input and output are both non-contiguous with a large stride in FFT_1, there is high latency in accessing these elements due to cache misses. Fault-free FFTs do not suffer much from this because the input and output are read and written once during the whole computation. However, with the fault tolerant operations, the input and output are at least accessed twice, which may lead to high overhead. Inspired by the implementation of sequential FFT, we use a similar idea to adjust the execution plan of FFT_1.

In order to mitigate the overhead in multiple accesses, one buffer is allocated

Algorithm 3 Communication-Computation Overlap

```
1: sched[0 to p-1]: schedule for communication
2: alloc send buffers $sb_1$,$sb_2$ and receive buffers $rb_1$,$rb_2$
3: generate data for processor sched[0] in $sb_1$
4: Isend($sb_1$) to and Irecv($rb_1$) from processor sched[0]
5: generate data for processor sched[1] in $sb_2$
6: Wait() for processor sched[0]
7: for i from 1 to p-3 do
8:     Isend($sb_2$), Irecv($rb_2$) with processor sched[i]
9:     process data from processor sched[i-1] in $rb_1$
10:    generate data for processor sched[i+1] in $sb_1$
11:    Wait() for processor sched[i]
12:    Isend($sb_1$), Irecv($rb_1$) with processor sched[i+1]
13:    process data from processor sched[i] in $rb_2$
14:    generate data for processor sched[i+2] in $sb_2$
15:    Wait() for processor sched[i+1]
16:    increase i by 2
17: end for
18: Isend($sb_2$), Irecv($rb_2$) with processor sched[p-1]
19: process data from processor sched[p-2] in $rb_1$
20: Wait() for processor sched[p-1]
21: process data from processor sched[p-1] in $rb_2$
```

to store the input contiguously. The input is read into the buffer and computed in the buffer. The result is then verified in the buffer and copied to the output location when the computation is correct. To maximum reuse for data in the cache, the buffer can be made c times the size of data in the p-point FFT, where c is the number of data in the cache line. Each time one element is read into the buffer, the latter c elements are also read and stored in the buffer as well. In this way, cache can be better utilized.

This change may have more operations because there are data assignments between input, output and the buffer. However, it may perform quite well when the p-point FFT barely fits in the cache. In this case, the original implementation would suffer a lot since there is no reuse of cached data. On the other hand, this optimization can make use of cache because data are moved into the buffer. It would be more scalable compared to the original plan.

2.7 Overhead Analysis

This section analyzes the theoretical overhead for the various ABFT schemes mentioned above. In the following subsections, c_1, c_2, r_1, r_2 will be used to denote one operation of complex number multiplication, complex number addition, real number multiplication, real number addition. Assume one real number addition or one real number multiplication is the unit of operation, $c_1 = 6$, $c_2 = 2$ and $8r_1 + 3r_2 = 11$ for the complex number division can be derived. This part only discusses the number of operations needed to add fault tolerance. The true overhead may differ since it heavily depends on the implementation. As a comparison, the total number of computational operations in the original FFT would roughly be $5N \log_2 N$.

2.7.1 Overhead in the Sequential Scheme

Computational FT in the Offline Scheme

The overhead in the offline scheme comes from input checksum vector generation, CCG and CCV. In the offline scheme, rA can be calculated according to characteristics of arithmetic arrays to reduce overhead:

$$(rA)_j = \omega_3^0 \omega_n^{0j} + \omega_3^1 \omega_n^{1j} + ... + \omega_3^{n-1} \omega_n^{(n-1)j} = \frac{1 - \omega_3^n}{1 - \omega_3^1 \omega_n^j}$$

Then it can be optimized by replacing trigonometric functions with 2 complex number multiplications. Then the overhead would be:

$$T_{rAGen} = (c_1 + c_1 + 2c_2 + 8r_1 + 3r_2) * N = 27N$$

CCG involves 1 complex number multiplication and 1 complex number addition for each element. Its overhead would be:

$$T_{CCG} = N * c_1 + N * c_2 = 8N$$

As for CCV, the total number of complex multiplications can be reduced to 2 by merging elements of same factors. So the overhead turns out to be:

$$T_{CCV} = 2 * c_1 + N * c_2 \approx 2N$$

Therefore, the total overhead for the offline scheme would be $37N$. If an error occurs, the correction would be another run of the whole FFT and final verification. So the correction time would be $39N + 5N \log_2 N$.

Computational FT in the Online ABFT Scheme

The overhead for the online scheme comes from checksum operations in the two ABFT parts and DMR for input checksum vector generation and twiddle multiplication. DMR for input checksum vector generation is negligible since the checksum sizes are $O(\sqrt{N})$. DMR for twiddle multiplication would cost $12N$ because it needs 2 complex number multiplications.

Overhead for ABFT comes from CCG and CCV. They cost $8N$ and $2N$ respectively. The two ABFT parts have the same overhead. Thus the total overhead for the two-layer ABFT scheme would be:

$$T_{ABFT} = 12N + 2 * (8N + 2N) = 32N$$

If an error occurs in DMR, it will be detected and corrected in no time. If an error strikes the ABFT parts, it will be detected by the ABFT scheme and an FFT of size k or m will be performed. As k and m are usually $\theta(\sqrt{N})$, the recalculation will always be an FFT of size $\theta(\sqrt{N})$, which is negligible. Therefore, the overhead for the online scheme would still be $32N$ even if an error occurs.

Total Overhead in the Offline Scheme

The extra operations in the offline scheme would be the computation of $r_2'x$ when the corresponding optimizations are applied. This computation will cost $4N$ operations. Therefore, the total overhead for the offline scheme would be:

$$T_{offline_m} = 37N + 4N = 41N$$

If there is error, whole computation after checksums generation will be restarted, including the verification operations. The overhead would be $5N\log_2 N + 43N$.

Total Overhead in the Online Scheme

In CMCG, there are $4N$ extra operations for the new checksum r_2' calculation. Besides, there is one more MCG and MCV, which corresponds to $6N$ operations. Also, there is one more CMCV of $2N$ operations in the end. Then the total overhead will be:

$$T_{ABFT_m} = T_{ABFT} + 4N + 6N + 2N + 2N = 46N$$

As the recovery time for both computational error and memory error is negligible, the overhead would still be $46N$ when an error occurs during the execution.

2.7.2 Sequential Space Overhead

When FFT is calculated on single processor, the space overhead only comes from the checksums of each small FFTs and protection for the buffered intermediate output. As these sizes are at most $4k$ or $4m$, the whole scheme only requires $O(\sqrt{N})$ extra space.

2.7.3 Overhead in the Parallel Scheme

Overhead Before Communication-Computation Overlap

Before overlap, the fault tolerant operations for parallel online scheme include MCG before $transpose_1$, MCV, CMCG after $transpose_1$, CMCV and MCG before $transpose_2$, MCV, and CMCG after $transpose_2$, CMCV and MCG in $FFT_{2.1}$, 2 MCVs, CCG, MCG in $FFT_{2.2}$ and MCV after $transpose_3$. So there are 2 CMCGs, 2 CMCVs, 4 MCGs, 4 MCVs, 1 CCG and 1 CCV when $r = 1$. The overhead for this situation would be:

$$T_{ABFT_{p1}} = 2 * (12n + 2n + 8n + 2n) + 4 * (6n + 2n) = 96n$$

When $r \neq 1$, there is 1 more MCV and 1 more MCV as well as DMR for TM and r-point FFTs, thus the overhead in this situation is:

$$T_{ABFT_{p2}} = 96n + 6n + 2n + 12n + 5n\log_2 r = 116n + 5n\log_2 r$$

Overhead After Communication-Computation Overlap

The overlapped communication includes 2 MCVs, 2 CMCGs and 1 TM, thus the new overhead when $r = 1$ would be:

$$T'_{ABFT_{p1}} = 96n - (2 * (12n + 2n) + 12n) = 56n$$

The overhead when r is not equal to 1 can be computed in a similar way, which will lead to:

$$T'_{ABFT_{p2}} = 116n + 5n\log_2 r - (2*(12n+2n)+12n) = 76n + 5n\log_2 r$$

The correction time for the parallel online scheme would also be negligible since correction in each part would cost negligible time.

2.7.4 Parallel Space Overhead

Assume the size of used space is $n = \frac{N}{p}$ on each processor, the largest allocated extra memory would be the checksum arrays in FFT_1, which totally take up $\frac{2n}{p}$ space. Besides, there are buffers for communication. Our communication-computation overlap operations allocate four buffers, each of which takes up $\frac{n}{p}$ space. Thus total space overhead would be $\frac{6n}{p}$. The other extra memory are all $O(m)$ or $O(k)$, which is $\theta(\sqrt{n})$. Also, the operations in communication can reuse the space freed from the send and receive buffers in the communication, which requires no extra memory. Therefore, the required extra space would be $\frac{6n}{p}$, then the relative space overhead would be $\frac{6}{p}$.

2.7.5 Parallel Communication Overhead

The communication overhead of the ABFT scheme comes from the increased message size in the communication. During each communication, the proposed scheme needs to send and receive two checksums for each block of data, which corresponds to an overhead of $\frac{2p^2}{N}$. As there is no extra overhead in the number of messages, the communication overhead would be at most $\frac{2p^2}{N} = \frac{2p}{n}$.

2.8 Impact of Round-off Errors

Due to the finite word length in floating number arithmetic, round-off errors are unavoidable in software level implementations. Therefore, the two checksums in the ABFT scheme may not be equal even though the whole FFT system is fault free. To avoid the situation above to be diagnosed as faulty, a small difference η between the result is allowed as in previous work. The selection of η is essential because it is a tradeoff between throughput (true negative, fault-free while diagnosed as faulty) and fault coverage (false positive, faulty while diagnosed as fault-free). This section analyzes the estimation of round-off errors and how to choose suitable η.

2.8.1 Round-off Errors in Computational FT

In existing work, the hardware implementations always employ the fixed-point round off strategy, which is quite different from the floating point arithmetic in the software level. Fortunately, Liu [52], Weinstein [109] and Gentleman [40] have already conducted some research on this topic. Assuming the N real numbers and N imaginary numbers in the input are mutually uncorrelated random variables with zero means. According to [109], the noise-to-signal ratio in an N-point FFT computation would be:

$$\frac{\sigma_E^2}{\sigma_X^2} = 2\sigma_\epsilon^2 \log_2 N$$

Where σ_E^2 is the variance of round-off error, σ_X^2 is the variance of the output, σ_ϵ is the error due to rounding floating point multiplication or addition. σ_ϵ can be assumed uniformly distributed in $(-2^{-t}, 2^{-t})$ or experimentally measured as ${\sigma_\epsilon}^2 = (0.21)2^{-2t}$ in [40], where t is the number of bits in the mantissa part of a floating point.

Assume the input x of an m-point FFT has zero means and variance σ_0. Its output X will have zero means and variance $\sigma_1 = \sqrt{m}\sigma_0$. According to equation (3), the variance of round-off would be $\sigma_e = \sqrt{2m\sigma_0^2\sigma_\epsilon^2 \log_2 m}$. We use $m * \sigma_e$ to estimate the round-off error after the summation. As the input precision loss would be much smaller than the output precision loss, the variance of the final difference would be $\sigma_{roe} = m * \sigma_e = m\sqrt{2m\sigma_0^2\sigma_\epsilon^2 \log_2 m}$. In the k-point FFTs, the input has variance $\sqrt{m}\sigma_0$ and output has variance $\sqrt{km}\sigma_0$. Similarly, we can derive $\sigma_{roe2} = k\sqrt{2km\sigma_0^2\sigma_\epsilon^2 \log_2 k}$.

After that, an approach similar to [105] can be employed to set the coefficient η. According to central limit theory, the throughput of an N-point FFT can be estimated as:

$$throughput(\eta, N, \sigma_i) = \frac{1}{1 + P(\frac{|F|}{\sqrt{N}\sigma} > \frac{\eta}{\sqrt{N}\sigma})} = \frac{1}{3 - 2\Phi(\frac{\eta}{\sqrt{N}\sigma})}$$

When $\eta = 3\sqrt{N}\sigma$, the theoretical throughput is 0.997. According to this formula, different η can be set to different parts of the online ABFT scheme. I.e., $\eta_1 = 3\sqrt{m}\sigma_{roe}$, $\eta_2 = 3\sqrt{k}\sigma_{roe2}$ can be chosen respectively for m-point FFTs and k-point FFTs in sequential FFT. In parallel FFT, things are similar. The only difference is that there are three ηs to be set respectively for FFT_1, $FFT_{2.1}$ and $FFT_{2.2}$.

2.8.2 Round-off Errors in Memory FT

Memory round-off errors would be much smaller since it only involves simple summation. According to the analysis above, the summation of m elements in the array x will result in a variance $m * \sqrt{var(x)}\sigma_\epsilon$ in the precision loss in the result for data with high precision. Then threshold can be set by the approach above.

2.9 Experimental Evaluations

We implement the proposed ABFT scheme into the widely used FFTW library [36, 37] - one of the fastest software implementations of FFT and reports the experimental results in this section.

2.9.1 Experiment Setup

We evaluated our implementation on TIANHE-2, the current *2nd* fastest supercomputer in the world. Each node of TIANHE-2 has 2 E5-2692 processors (with *24 cores* in all) and 64GB memory.

2.9.2 Overhead in Sequential Scheme

This section evaluates the sequential schemes for out-of-place FFT on single processor. FFT sizes from 2^{25} to 2^{28} are tested. Each experiment is run 9 times and the average number is recorded.

Experiments without Fault

Four schemes are evaluated at this part and the results are shown in Fig. 7. Fig. 7(a) shows the evaluations for computational FT schemes. The first bar shows the overhead of the naive offline scheme. The second bar is the evaluation of the optimized offline scheme. A naive online scheme is displayed as the third bar and an optimized online scheme is shown as the last bar. Fig. 7(b) shows the evaluations for computational and memory FT schemes. The only difference is that the third bar displays the online scheme with computational FT optimizations.

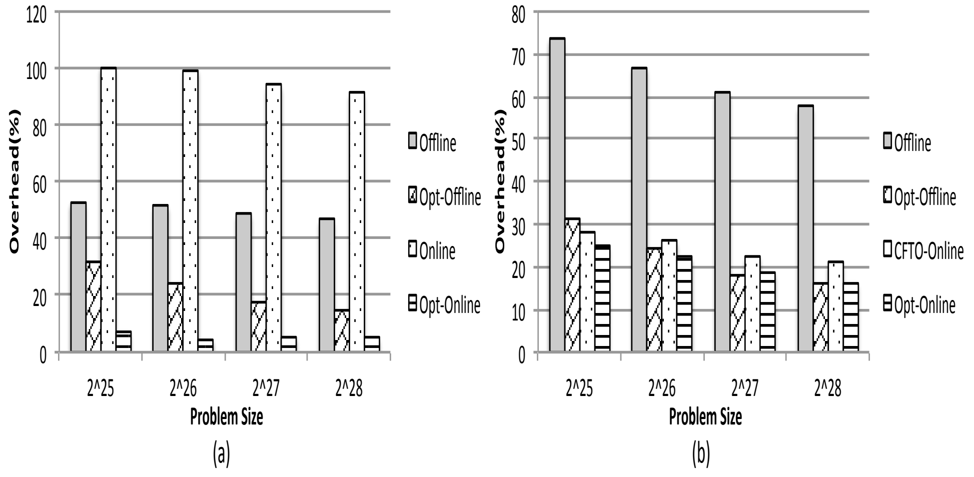


Figure 2.7: Overhead of ABFT-FFT Schemes on TIANHE-2 When There Is No Fault: (a) Computational FT (b) Computational & Memory FT

From the figure, we can see that the optimization techniques play an important role in the FT-FFT schemes. The optimized offline scheme is much better than the naive offline scheme due to the number of calls to the trigonometric functions. The optimized online scheme outperforms the offline one a lot when only computational errors are considered. Also, it has comparable performance to the optimized offline scheme even when memory errors are considered.

Experiments with Faults

This part shows the timely recovery of the online scheme. As the offline scheme only guarantees to detect one error, only one memory fault is injected in the optimized offline scheme. Three fault injections are performed on the online scheme: one computational fault ($1c$); one computational fault and a memory fault ($1m + 1c$); two computational faults and one memory fault ($1m + 2c$). (0) indicates fault-free executions as comparison.

Table 2.2: Execution Time (Seconds) Comparison of FT-FFT on TIANHE-2 When There Are Faults

Problem Size	$N = 2^{25}$	$N = 2^{26}$	$N = 2^{27}$	$N = 2^{28}$
$FFTW(0)$	3.71	8.04	16.79	34.97
$Opt - Offline(0)$	4.88	10.01	19.86	40.52
$Opt - Offline(1m)$	9.63	20.21	42.89	87.65
$Opt - Online(0)$	4.64	9.83	19.94	40.64
$Opt - Online(1c)$	4.78	9.92	20.17	40.92
$Opt - Online(1m + 1c)$	4.83	9.98	20.44	41.28
$Opt - Online(1m + 2c)$	4.86	10.17	20.77	41.68

Computational fault is simulated as adding some constant to an element while memory fault is simulated by changing one element to another constant. Table 1 shows the execution time of the optimized schemes with different number of errors.

According to the table, the online scheme does have strong fault tolerant ability. The offline scheme suffers from the re-execution when an error occurs thus it costs about twice the time the online scheme does. On the other hand, because one error only leads to a recalculation of a m-point FFT or s k-point FFTs which costs $O(\sqrt{N}\log\sqrt{N})$ time, the execution time of the online scheme can almost maintain the same when the number of errors increases. In fact, as long as no two errors strike the same m-point FFT or s k-point FFTs at the same time, the online scheme is able to detect and correct all of them quickly. Therefore, the online scheme is able to perform well even when the error rate is relatively high, showing great advantage over the offline scheme.

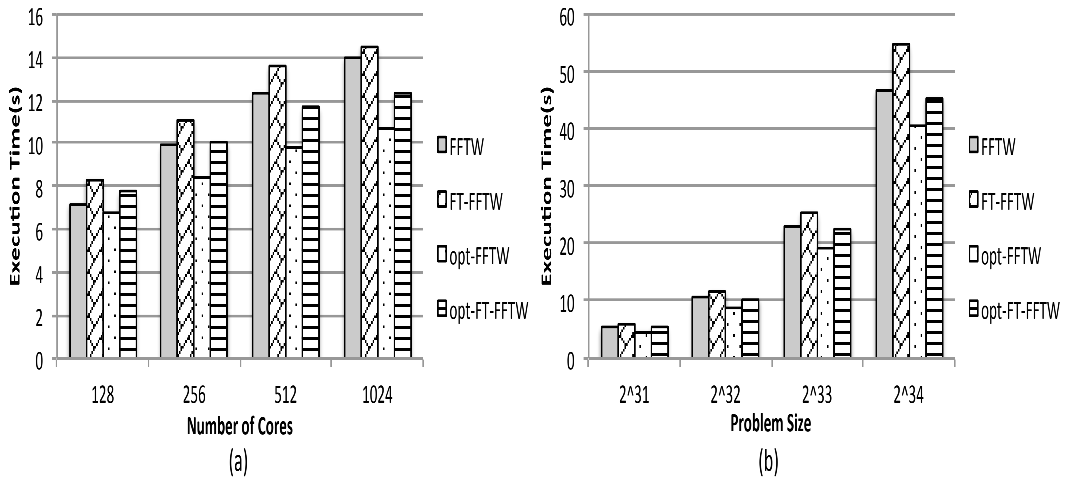


Figure 2.8: Execution Time (Seconds) of Parallel FT-FFT Schemes on TIANHE-2 When There Is No Fault: (a) Fixed size on each processor: $n = 2^{26}$ (b) Fixed number of processors: $p = 256$

2.9.3 Performance in Parallel Scheme

This section evaluates the parallel online scheme for in-place FFT in large scale. Because of fluctuations, each experiment is run 20 times and the average number is recorded.

Experiments without Fault

Three implementations together with original $FFTW$ are evaluated at this part. The results are shown in Fig. 8. The first bar shows the execution time of original $FFTW$. The sequentially optimized fault tolerant scheme $FT - FFTW$ is displayed as the second bar. The third bar $opt-FFTW$ is $FFTW$ with parallel optimizations in Section 6. The last bar $opt-FT-FFTW$ is the parallel fault tolerant scheme with both sequential and parallel optimizations. According to the figure, the sequentially optimized ABFT scheme has some overhead over the original $FFTW$. The overhead comes from the checksum operations. On the other hand, the online scheme with parallel optimizations beats the original $FFTW$ in

Table 2.3: Execution Time (Seconds) Comparison of FT-FFTW (Fixed $n = 2^{26}$) on TIANHE-2 When There Are Faults

Number of Cores	$p = 128$	$p = 256$	$p = 512$	$p = 1024$
$Opt - FT - FFTW(0)$	7.83	10.24	11.34	12.47
$Opt - FT - FFTW(2m)$	7.85	10.23	11.39	12.57
$Opt - FT - FFTW(2c)$	7.85	10.28	11.33	12.59
$Opt - FT - FFTW(2m + 2c)$	7.86	10.23	11.34	12.56

Table 2.4: Execution Time (Seconds) Comparison of FT-FFTW (Fixed $p = 256$) on TIANHE-2 When There Are Faults

Problem Size	$N = 2^{31}$	$N = 2^{32}$	$N = 2^{33}$	$N = 2^{34}$
$Opt - FT - FFTW(0)$	5.45	10.35	22.45	45.63
$Opt - FT - FFTW(2m)$	5.42	10.35	22.55	45.31
$Opt - FT - FFTW(2c)$	5.43	10.36	22.47	45.46
$Opt - FT - FFTW(2m + 2c)$	5.45	10.31	22.55	45.47

error-free runs because the parallel optimizations work very well. However, it still has some overhead over $opt - FFTW$ due to checksum operations.

Experiments with Faults

This part shows the fault tolerant ability of the parallel online scheme. Fault injection mechanimsm is similar to the one used in Section 9.2.2 except that faults are injected in each processor. Experiments of no faults (0), 2 memory faults (2m), 2 computational faults (2c), 2 memory faults and 2 computational faults (2m+2c) are shown in Table 2 and Table 3, respectively.

According to the tables, this scheme does have strong fault tolerant ability. It only takes very little time to recover from multiple faults because each fault only revokes a restart of one or several p-point FFTs or $\sqrt{\frac{n}{p}}$-point FFTs. Note that sometimes the error free run may have longer execution time. This is caused by fluctuation.

2.9.4 Round-off Errors

As parallel FFTs have similar round-off error impact to sequential scheme, only experiments on sequential schemes of 2^{25}-point FFT are tested. These results can be generalized to the parallel scheme.

Round-off Error Approximation

In this part, the accuracy of round-off analysis in Section 8 is evaluated. Input from uniform distribution $U(-1,1)$ and normal distribution $N(0,1)$ is tested respectively. 1000 runs are performed thus there are 8192000 m-point FFTs and 1024000 s k-point FFTs. The result is shown in Table 4.

In Table 4, the column Max_1 shows the max round-off error in the m-point FFTs. Est_1 shows the estimated η for this part. $Thput_1$ shows the throughput of the scheme. The latter three columns show the same property of the k-point FFTs. The selected η provides nearly 100% throughput while keeping close to the round-off error bound. It promises good coverage.

Table 2.5: Approximation of Round-off Error

$Input$	Max_1	Est_1	$Thput_1$	Max_2	Est_2	$Thput_2$
$U(-1,1)$	$0.92 * 10^{-8}$	$1.45 * 10^{-8}$	100%	$0.61 * 10^{-6}$	$3.86 * 10^{-6}$	100%
$N(0,1)$	$3.8 * 10^{-8}$	$2.51 * 10^{-8}$	99.96%	$1.11 * 10^{-6}$	$6.69 * 10^{-6}$	100%

Detection Ability Comparison

This section compares the detection ability of the online scheme and the offline scheme. Same fault is injected into the same position of the different schemes. Three fault injection positions are tested in this part. e_1 is injected in the input after checksum verification; e_2 is injected in the input of the second FFT; e_3 is injected in the final output. In the fault injection, the selected element will increase itself by the given error magnitude. I. e., if the magnitude of error is 10^{-3}, 10^{-3} is added to the selected element and whether the error is detected is observed. η of the offline scheme is set as the round-off error bound of error-free runs to allow for 100% throughput. From Table 5, the online scheme can detect a much smaller magnitude of errors than the offline scheme. Thus, when throughput is similar, the online scheme should have much larger fault coverage.

Fault Coverage Tests

This section shows the relative errors of FFT output after an error occurs in a 2^{25}-point sequential FFT with input drawn from $U(-1,1)$. As random computational errors are hard to simulate and some of them can be simulated as memory errors, only memory error of single bit flip is tested here.

Table 2.6: Minimal Magnitude of Error That Can Be Detected

$Schemes$	e_1	e_2	e_3
$Offline$	10^{-2}	10^{-2}	10^{-2}
$Online$	10^{-7}	10^{-6}	10^{-6}

Table 2.7: Distribution of Relative Errors of FFT Output in 1000 Runs When One Random Fault Is Injected in Each Run

$\frac{\|\|x'-x\|\|_\infty}{\|\|x\|\|_\infty}$	$Uncorrected$	$> 10^{-6}$	$> 10^{-8}$	$> 10^{-10}$	$> 10^{-12}$
$No\_Correction$	–	73.4%	82.4%	84.0%	84.2%
$Offline$	4.4%	5.2%	20.8%	33.4%	35.7%
$Online$	2.5%	2.5%	2.5%	2.5%	3.9%

Some fault-free runs of 2^{25}-point FFT are performed at first to get a rough upper bound of the round-off errors of the offline schemes. After that, η is set as this rough upper-bound to allow for nearly 100% throughput and relative errors are evaluated after randomly flipping one higher bit (flipping lower bit is usually masked) in the input or output array. Define the relative error as $\frac{||x'-x||_\infty}{||x||_\infty}$, where x is the correct output, x' is the output with fault injection and $|| \bullet ||_\infty$ is the infinity norm of vector. 1000 independent runs are performed and the distribution of relative errors is shown in Table 6. The first row shows the relative error of runs without correction. It indicates the impact of errors on output as a comparison. The second column $Uncorrected$ shows the percentage of uncorrected errors due to wrong indexing caused by round-off errors. It can be improved by changing the indexing checksum r_2. For these situations, the relative error is set as infinite.

According to the table, the online scheme outperforms the offline scheme a lot in fault coverage because the relative errors it introduces are of much smaller magnitude. For example, if the error bound is set as 10^{-12}, the fault coverage in the online scheme would be 96.1% compared to 64.3% in the offline scheme. It shows great potential in practical use.

2.10 Summary

We present an online ABFT scheme to correct soft errors online in the widely used FFT computations. The proposed scheme only needs to repeat a small fraction of the computation after errors occur. Experimental results demonstrate that the proposed scheme improves the computing efficiency by 2X over existing schemes when errors occur.

Chapter 3

Optimizing Error-controlled Lossy Compression for High Performance I/O in Scientific Simulations

3.1 Introduction

An efficient data compressor is increasingly critical to today's scientific research because of the extremely large volume of data produced by high performance computing (HPC) simulations and experimental instruments. Based on our communication with researchers working on extreme-scale HPC simulations, they expect to see a compression ratio up to dozens of times or even 100:1, meaning that the bit-rate (i.e., the number of bits used to represent one data point on average after compression) should be no greater than 2 and best less than 1.

Although considerably reducing the data size can definitely improve I/O performance significantly as well as the post-analysis efficiency, the decompressed data may suffer from significant distortion compared with the original dataset. If the compression ratio reaches 64:1 (that is, the bit-rate is about 0.5 bit per 32 bit data point), the precision of the decompressed data will degrade significantly in general, even using the best existing lossy compressors such as SZ [92], ZFP [67], and FPZIP [69]. This will cause a huge distortion in the visualization (shown in Section 4.4.1). The question addressed in this paper is, can we significantly improve the precision of the decompressed data for the lossy compression with a fairly low bit-rate compared to state-of-the-art lossy compressors?

In the past decade, several error-controlled lossy data compressors (including [2, 3, 7, 9, 21, 27, 32, 41, 42, 55, 67, 69, 72, 81, 91, 91, 92]) have been developed to significantly reduce the scientific data size for different purposes [16, 49, 95]. These lossy compressors can be classified into two groups, based on how they decorrelate the original data. The first group of compressors [9, 27, 67, 81, 85] use a transform-based model that leverages invertible transforms for the decorrelation. The second group of compressors [32, 55, 69, 91, 92] use a prediction-based model that leverages various prediction methods for the decorrelation. Generally speaking, no compression model can always outperform the others. Even for the same dataset, the best-fit model may differ depending on distortions. Hence, the method must be carefully selected at runtime.

Designing an efficient error-bounded lossy compressor that can significantly reduce the data size with a relatively high resolution of decompressed data is very challenging. We explain this point based on the two most effective lossy compressors [70]: SZ and ZFP. SZ and ZFP adopt largely different compression models: respectively, a data prediction model and an orthogonal transform model. In the data prediction model [92], each data value needs to be predicted by using its adjacent data points in multidimensional space according to the order of scanning data points. Moreover, the data used in the prediction during the compression have to be the decompressed values, in order to guarantee that the error bound is respected during the decompression, which is a strict limitation to the design of SZ [92]. Such a limitation may significantly degrade the prediction accuracy, especially for the lossy compression with a relatively large error bound, leading to limited compression quality. On the other hand, in the orthogonal transform-based compressor such as ZFP [67], the entire dataset has to be split into many small blocks (e.g., 4x4x4 for 3D data), each of which will be transformed to another decorrelated domain individually based on a fixed coefficient matrix. Relatively large error bound setting will introduce significant loss to the coefficients, leading to the over-distortion of decompressed data in turn.

In this part, we propose an adaptive lossy compression framework in terms of the data prediction compression model that can obtain a stable, high compression quality when the error bound is set to a large value for reaching a high compression ratio. We also focus on the significant improvement of compression quality over the existing state-of-the-art lossy compression techniques. This task raises the following challenges. (1) Scientific simulations may produce vast volumes of data with largely different characteristics, such that none of the existing lossy compression techniques can work well stably on all datasets. (2) Designing a lightweight, adaptive framework that can always choose the best compressor is nontrivial because many lossy compressors exist each with distinct design principles. Moreover, comparing the compression quality between any two lossy compressors is nontrivial because, given the same error bound, one compressor may have a higher compression ratio with higher overall precision—such as the peak signal-to-noise ratio (PSNR)—than the other compressor has, leading to a dilemma in making a choice. (3) Further improving any of these compressors is nontrivial because each has an elaborate design and optimized implementation. ZFP, for instance, adopts an optimized transformation method by extensive explorations on various transforms and couples it with embedded encoding to provide fast and efficient lossy compression. Specifically, our contributions are as follows:

- We propose an adaptive lossy compression framework that is more effective in compressing the scientific datasets with relatively large error bounds. In particular, we split the whole dataset into multiple non-overlapped blocks, and select the best-fit prediction method based on their data features.

- We develop new prediction methods that are particularly effective for the lossy compression with relatively large error bounds. On the one hand, we develop a hybrid Lorenzo prediction method by combining the classic Lorenzo predictor [47] and the densest mean-value based data approximation method (also called mean-integrated Lorenzo prediction). On the other hand, we develop a linear regression method that can obtain much higher prediction accuracy in this case since the design is beyond the limitation that the decompressed values have to be used in the prediction.

- We explore how to select adaptively and efficiently the best-fit prediction method based on the data features across blocks during the compression, with optimizations ito improve prediction accuracy.

- We improve the coefficient-encoding efficiency for the data-fitting predictor such that the compression ratio is improved significantly for relatively high-compression cases.
- We design a transform-based predictor by adopting the transform techniques of transform-based models in the prediction stage of prediction-based models. This is the first such design, to the best of our knowledge.
- We optimize the encoding strategy for the transform-based predictor, significantly improving the compression ratio for cases requiring relatively high compression ratios.
- We develop a best-fit predictor selection algorithm that can automatically select the best predictor between the data-fitting one and transform-based one in the compression.

3.2 Background

In this paper, we focus on how to improve the compression quality under the restrictions of error-bounded lossy compression. Here we mainly use PSNR instead of the error bounds to assess the compression quality, because domain scientists often care more about the overall statistical errors, especially for visualization purposes. Specifically, our objective is to ensure that the decompressed data follow the error-bounding requirements and to optimize the rate distortion metric (i.e., statistical errors), while incurring little degradation on the compression speed.

Rate distortion is one of the most important metrics to assess lossy compression quality. The *rate* here is short for bit rate (denoted τ), which refers to the mean number of bits used to represent one data point after compression. We define the compression ratio (denoted R) to be the ratio of the original raw data size to the compressed data size. For a floating-point dataset, we have following equations: $\tau = \frac{32}{R}$, based on single precision, and $\tau = \frac{64}{R}$, based on double precision. Obviously, the lower the bit rate, the higher the compression ratio.

To assess the *data distortion*, we use PSNR as follows:

$$PSNR = 20 \cdot \log_{10}(value\_range) - 10 \cdot \log_{10}(MSE), \tag{3.1}$$

where MSE stands for mean-squared error. This formula has been widely used in the community [67, 69, 92, 93], because a higher PSNR generally indicates better visual quality or higher overall precision. In this case, given a dataset with N floating-point data values, the research problem can be formulated as follows,

$$\min_{\forall PSNR}(\tau) \quad s.t. \quad |d_i - d'_i| \leq e, \forall i = 1, 2, \cdots, N \tag{3.2}$$

where $\{d_1,\ d_2,\ \cdots,\ d_N\}$ and $\{d'_1,\ d'_2,\ \cdots,\ d'_N\}$ refer to the original raw data values and decompressed data values, respectively.

I/O performance on parallel file systems is also an important evaluation metric when lossy compressors are used. Specifically, we also target significantly reducing the overall data-dumping time (dumping data to parallel file systems) and the data-loading time (loading data from parallel file systems) for large-scale parallel executions. In addition to the optimization of the rate distortion within an error-bounded setting, the compression/decompression time are also essential for I/O performance, because the data-dumping time is the compression time plus the compressed data-writing time and the data-loading time is the compressed data-reading time plus the decompression time for lossy/lossless compressors.

3.3 Multi-algorithm Predictor

3.3.1 Design Overview

In the following, we first present an overview of our adaptive lossy compression framework and then describe the compression techniques in detail.

The key idea of our adaptive solution is splitting the entire dataset into multiple non-overlapped equal-sized blocks in multidimensional space and selecting the best-fit data prediction method dynamically for each block based on its data feature. Algorithm 4 presents the pseudo-code of the entire design. The basic idea is to select in each block the best-fit prediction method, from among the three data prediction approaches: *classic Lorenzo predictor* [47], *mean-integrated Lorenzo predictor*, and *linear regression-based predictor*. The first one was already adopted by some existing compressors such as SZ and FPZIP, while the other two are proposed as a critical contribution in this paper, because they can improve the lossy compression quality significantly.

We describe our algorithm in the following text. At the beginning (line 1), the algorithm searches for the densest interval based on the error-bound ε and calculates its data frequency (denoted by p_1), in order to estimate the prediction ability of the *mean-integrated Lorenzo predictor*. This part involves three steps: (1) $\sqrt{N}$ data points will be sampled uniformly in space; (2) the mean value of the sampled data points is calculated

Algorithm 4 Adaptive Error-bounded Compressor

```
Input: user-specified error bound $\varepsilon$
Output: compressed data stream in form of bytes
 1: Estimate the densest position (denoted as $v_0$) and calculate the frequency (denoted $p_1$) of the densest
    error-bound-based interval surrounding it;
 2: Calculate the densest frequency of classic Lorenzo predictor (denoted $p_2$);
 3: if ($p_1 > p_2$) then
 4:     $\mu \leftarrow \frac{\sum_{|d_i - v_0| \leq \varepsilon} d_i}{\|\{d_i \| d_i - v_0| \leq \varepsilon\}\|}$; /*Compute mean value of densest interval*/
 5:     $\ell$-Predictor ← mean-integrated Lorenzo predictor;
 6: else
 7:     $\ell$-Predictor ← classic Lorenzo predictor;
 8: end if
 9: for (each block in the multi-dimensional space) do
10:     Calculate regression coefficients; /*4 coefficients/block in 3d dataset*/
11: end for
12: Calculate statistics of all coefficients for compression of coefficients later;
13: for (each block in the multi-dimensional space) do
14:     Create a sampling set (denoted $S_M$) with $M$ sampled data points;
15:     Compute cost values $E_{reg\text{-}predictor}$ and $E_{\ell\text{-Predictor}}$ based on $S_M$;
16:     if ($E_{reg\text{-}predictor} < E_{\ell\text{-Predictor}}$) then
17:         Execute regression-based prediction and quantization;
18:     else
19:         Execute $\ell$-Predictor and quantization;
20:     end if
21: end for
22: Construct Huffman tree according to the quantization array;
23: Encode/compress quantization array by Huffman tree;
24: Compress regression coefficients;
```

and a set of consecutive intervals (each with 2ε in length) will be constructed surrounding the mean value; (3) we then calculate the number of data points in the consecutive intervals and select the one with the highest frequency of data points as the densest interval, whose center is called the densest position (denoted as v_0). Here $\sqrt{N}$ is chosen by heuristics because it already exhibits good accuracy. On line 2, the algorithm checks the prediction ability of the *classic Lorenzo predictor*, by calculating the data frequency (denoted p_2) of its error-bound based prediction interval (i.e., [pred_value$-\varepsilon$, pred_value$+\varepsilon$], where pred_value refers to the predicted value), based on 1% of uniformly sampled data points. The number of sampled data points (i.e., 1%) is a heuristic setting, which is similar to the configuration of SZ. Based on the sampled data points, we select the best-fit Lorenzo predictor (denoted as ℓ-PREDICTOR) according to our estimated prediction ability of the two predictors (line 3-8). If the best-fit predictor is the mean-integrated Lorenzo predictor, we need to calculate the mean value of the densest interval (line 4), which will be used later.

After determining the best-fit Lorenzo predictor, the algorithm calculates the linear regression coefficients (lines 9-11) as well as the statistics (such as the value range of the coefficients), which will be used in the compression of the coefficients later (Section 3.3.3).

The most critical stage is scanning the entire dataset and performing the best-fit prediction and linear-scaling quantization in each block (lines 13-21). In each block, M data points (1/8 data points for 2D dataset and 1/9 for 3D dataset) are sampled uniformly in space, in order to select the best-fit prediction method as accurately as possible. The sampling method will be further detailed in Section 3.3.3. Then, the algorithm determines which prediction method (either regression-based predictor or ℓ-PREDICTOR) should be used in the current block in terms of their estimated overall prediction errors (denoted by $E_{\textit{reg-predictor}}$ and $E_{\ell\text{-PREDICTOR}}$ respectively). How to estimate the prediction errors for the two predictors will be detailed in Section 3.3.4.

Our algorithm then compresses the quantization array constructed in the prediction stage by Huffman encoding (lines 22-23). It also compresses the regression coefficients for the blocks selecting the regression-based prediction methods, by IEEE 754 binary analysis (detailed in Section 3.3.3).

The time complexity of the algorithm is $O(N)$, because the algorithm is composed of three parts, whose time complexities are no greater than $O(N)$. Specifically, the first part estimates densest frequency (lines 1-2) with a time complexity of $O(\sqrt{N})$; the second part calculates regression coefficients (lines 9-11) that costs $O(N)$ in total (for details, see Lemma 1 to be presented later); and the last part performs prediction+quantization (lines 13-21), which also costs $O(N)$ since each data point will be scanned only once.

3.3.2 Mean-integrated Lorenzo Predictor

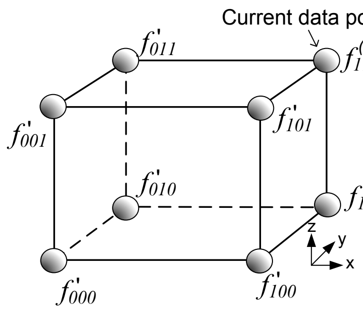
In this section, we develop a new predictor based on the Lorenzo predictor [47]. Fig. 3.1(a) illustrates the classic Lorenzo prediction method in a 3D dataset. Specifically, it predicts the current data point based on the following formula for a 3D dataset:

$$f_{111}^{(L)} = f'_{000} + f'_{011} + f'_{101} + f'_{110} - f'_{001} - f'_{010} - f'_{100} \tag{3.3}$$

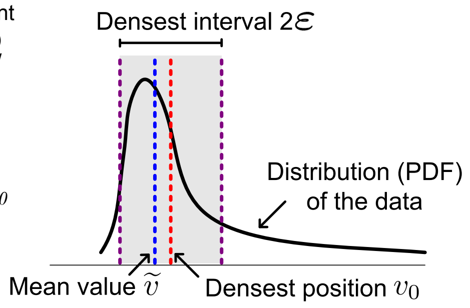
where $f^{(L)}$ and f' refer to the predicted value and decompressed value, respectively. $\{111\}$ is the current data point to deal with, and the other seven data points are adjacent to it on a unit cube which have been processed. $f_{111}^{(L)}$ is the predicted value for the data point $\{111\}$ and the decompressed value f'_{111} can be obtained by applying the linear-scaling quantization on the difference between $f_{111}^{(L)}$ and the origin data value at $\{111\}$. The compressor will continue this procedure data point by data point until all the data are processed.

The classic Lorenzo predictor has a significant defect: many predicted values would be uniformly skewed from the original values if the error bound is relatively large in the lossy compression, leading to an unexpected artifact issue. Our developed mean-integrated Lorenzo predictor can solve this issue well.

The fundamental idea is approximating those data points whose values are clustered intensively by a fixed value, if majority of data values are clustered to a small interval



(a) Classic Lorenzo predictor



Densest interval 2ε
Distribution (PDF) of the data
Mean value $\widetilde{v}$
Densest position v_0

(b) Data approximation by mean value

Figure 3.1: Illustration of mean-integrated Lorenzo predictor

with pretty high density (called *densest interval*). This situation appears in about one-third of the fields of the NYX cosmology simulation [73] and about half the fields of the Hurricane simulation [46]. In the dark matter density field of NYX, for instance, 84+% of the data values are in the range of [0,1], while the remaining data are in the range of [1,1.34$\times$10^4].

Suppose we are given a dataset $D = \{d_i | i = 1, \ldots, N\}$ such that the interval [$v_0-\varepsilon$,$v_0+\varepsilon$] can cover a large percentage of data points, where ε is the compression error bound and v_0 is the densest position (as illustrated in Fig. 3.1(b)). If this percentage is greater than some threshold, we will select the mean-integrated Lorenzo predicator. In practice, we set the threshold to the sampled prediction accuracy of classic Lorenzo predictor (line 3 in Algorithm 4). However, the classic Lorenzo predictor would be highly over-estimated when error bound is relatively large because the sampling stage does not take into account the impact of decompressed data, leading to a serious artifact issue. In order to mitigate this issue, we let the algorithm opt to select the mean-integrated Lorenzo predictor directly when [$v_0-\varepsilon$,$v_0+\varepsilon$] can cover more than half of the data. Now, we need to derive an optimal value to approximate the data in this interval.

Lemma 1. *The optimal value used to approximate the majority of data points should be the mean value of data in the densest interval* $[v_0{-}\varepsilon, v_0{+}\varepsilon]$.

Proof. If a fixed value v is used to approximate all the data points in this interval, the corresponding MSE can be represented as follows:

$$MSE = \int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)(x-v)^2 dx$$

where $p_d(x)$ is the probability density function. Then, the problem becomes an optimization problem aiming to minimize MSE. Letting the partial derivative $\frac{\partial MSE}{\partial v} = 0$, we have $-2\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)xdx + 2v\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)dx = 0$. Solving this equation will obtain the optimal value $\widetilde{v}{=}\frac{\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)xdx}{\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)dx}$. It is the mean value of all data points in this interval (as shown in Fig. 3.1(b)). This lemma also holds in discrete cases, where the optimal approximation value $\widetilde{v}$ can be calculated by $\frac{\sum_{x\in[v_0-\varepsilon,v_0+\varepsilon]} x}{\|\{x\,||x-v_0|\leq\varepsilon\}\|}$ in practice, where $\|\{x\,||x-v_0|\leq\varepsilon\}\|$ is the number of data points in the densest interval. □

3.3.3 Regression-based Prediction

Although the proposed mean-integrated Lorenzo predictor alleviates the artifact and reduces prediction errors in predicting the intensively-clustered data, it does not work well on the data following a rather uniform distribution. To address this issue, we propose a regression-based prediction model, which can deal with generic datasets. We adopt a linear-regression model instead of a higher-order regression model considering the overhead. Quadratic regression model, for example, requires 2.5X the number of coefficients the linear-regression model needs, with ≥3X the computation workload.

Derivation of Regression Coefficients

In what follows, we derive the regression coefficients from a generic perspective in terms of a dataset with m dimensions ($n_1{\times}n_2{\times}\cdots{\times}n_m$), which can be easily extended to block-wise situations. The value at position $\boldsymbol{x} = (i_1, \ldots, i_m)$ is denoted as $f(\boldsymbol{x}) = f_{i_1 \ldots i_m}$, where i_j is its index along each dimension. We also denote $f^{(r)}$ as the linear regression prediction. Then the linear regression model can be built as:

$$f^{(r)}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\beta} + \alpha$$

where $\boldsymbol{\beta} = (\beta_1,\ldots,\beta_m)$ denotes the slope coefficient vector and the constant α is the intercept coefficient. By redefining $\boldsymbol{x'} = (1, i_1, \ldots, i_m)$ and $\boldsymbol{\beta'} = (\beta_0{=}\alpha,\beta_1,\ldots,\beta_m)$, the formula above can be rewritten as:

$$f^{(r)}(\boldsymbol{x'}) = \boldsymbol{x'}^T \boldsymbol{\beta'}$$

Since each data point corresponds to a position $\boldsymbol{x}$ and its value $f(\boldsymbol{x})$, the objective of the regression model is to minimize the squared error (SE) between predicated and original values:

$$SE = \sum_{\boldsymbol{x'} \in \{(1,i_1,\ldots,i_m) | 0 \le i_j < n_j\}} (\boldsymbol{x'}^T \boldsymbol{\beta'} - f_{i_1 \ldots i_m})^2$$

This is a convex function, and its optimal can be obtained by derivation. The derivation over each element in $\boldsymbol{\beta'}$ will result in a linear system of m unknowns. By solving the linear system, the optimal solution can be achieved as:

$$\boldsymbol{\beta'} = (X^T X)^{-1} X^T y \tag{3.4}$$

X is the full permutation of $\{i_1,i_2,\cdots,i_m\}$, where $i_j{\in}\{0,1,\cdots,n_j{-}1\}$, and y is the sequence of the corresponding data values $(f_{00\ldots0},f_{00\ldots1},\cdots,f_{(n_1-1)(n_2-1)\ldots(n_m-1)})$.

Since X is a $\{\prod_{1\le i\le m} n_i\}\times(m+1)$ matrix, computing the closed-form solution (3.4) is very expensive. However, we can derive the solution to a simple form, significantly reducing the computation cost. Let us take a 3D dataset as an example to describe our method, which can be extended to datasets with higher dimensions easily without loss of generality.

Lemma 2. *The regression coefficients of a 3D dataset with dimensions n_1, n_2, n_3 can be calculated as:*

$$\begin{cases} \beta_1 = \frac{6}{n_1 n_2 n_3 (n_1+1)}(\frac{2V_x}{n_1-1} - V_0) \\ \beta_2 = \frac{6}{n_1 n_2 n_3 (n_2+1)}(\frac{2V_y}{n_2-1} - V_0) \\ \beta_3 = \frac{6}{n_1 n_2 n_3 (n_3+1)}(\frac{2V_z}{n_3-1} - V_0) \\ \beta_0 = \frac{V_0}{n_1 n_2 n_3} - (\frac{n_1-1}{2}\beta_1 + \frac{n_2-1}{2}\beta_2 + \frac{n_3-1}{2}\beta_3) \end{cases} \tag{3.5}$$

$$\textit{where } V_0 = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} f_{ijk}, \quad V_x = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} i * f_{ijk},$$

$$V_y = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} j * f_{ijk}, \quad V_z = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} k * f_{ijk}.$$

Proof. Substitute the index with the coordinate values. The SE expression will turn out to be:

$$SE = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} (\beta_0 + \beta_1 i + \beta_2 j + \beta_3 k - f_{ijk})^2$$

The following linear system can be derived by getting all its partial derivatives over the coefficients and setting them to 0:

$$\sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} \begin{bmatrix} 1 & i & j & k \\ i & i^2 & ij & ik \\ j & ij & j^2 & jk \\ k & ik & jk & k^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} V_0 \\ V_x \\ V_y \\ V_z \end{bmatrix}$$

Since $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ and $\sum_{i=0}^{n-1} i^2 = \frac{(2n-1)n(n-1)}{6}$, the above linear system can be simplified to:

$$\begin{bmatrix} 1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2} \\ 1 & \frac{(2n_1-1)}{3} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2} \\ 1 & \frac{(n_1-1)}{2} & \frac{(2n_2-1)}{3} & \frac{(n_3-1)}{2} \\ 1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(2n_3-1)}{3} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \frac{V_0}{n_1n_2n_3} \\ \frac{2V_x}{n_1n_2n_3(n_1-1)} \\ \frac{2V_y}{n_1n_2n_3(n_2-1)} \\ \frac{2V_z}{n_1n_2n_3(n_3-1)} \end{bmatrix}$$

After that, Gaussian elimination can be leveraged to transfer the linear system to the following:

$$\begin{bmatrix} 1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \frac{V_0}{n_1n_2n_3} \\ \frac{6}{n_1n_2n_3(n_1+1)}(\frac{2V_x}{n_1-1} - V_0) \\ \frac{6}{n_1n_2n_3(n_2+1)}(\frac{2V_y}{n_2-1} - V_0) \\ \frac{6}{n_1n_2n_3(n_3+1)}(\frac{2V_z}{n_3-1} - V_0) \end{bmatrix}$$

Then Equation (3.5) can be derived accordingly. □

These coefficients will be used in the regression model for predicting the data accurately. Each data point with index (i, j, k) will be predicted as $f_{ijk}^{(r)} = \beta_0 + i\beta_1 + j\beta_2 + k\beta_3$. Then, we will compute the difference between each predicted value $f_{ijk}^{(r)}$ and its original value f_{ijk}, and perform the linear-scaling quantization [92] to convert the floating-point values to integer codes. The data size will be significantly reduced after conducting Huffman encoding on the quantization codes.

Compressing Regression Coefficients

We adopt the block size 6×6×6 for 3D data and 12×12 data for 2D data in our implementation, since such settings already lead to satisfying compression quality. For each block that adopts the linear regression model, four coefficients have to be kept in the compressed bytes together with the encoding of regression-based predicted values in that block. If each block is a 6×6×6 cube, the overhead of saving the four coefficients (single-precision floating-point values) would be $\frac{4}{6\times6\times6}=\frac{1}{54}$ of the original data size (i.e., bit-rate=0.6). Note that we are targeting high-compression ratios (such as 100:1), so we have to further compress the coefficients significantly by a lossy compression technique with controlled impact of lossy coefficients on the prediction accuracy. We reorganize all the coefficients into four groups and compress each group of coefficients in a similar way by the unpredictable data compression method used in SZ [32].

3.3.4 Adaptive Selection of Best-fit Predictor

In this section, we analyze the nature of the linear regression-based predictor (proposed in Section 3.3.3) and Lorenzo predictor (introduced in Section 3.3.2). We also propose a cost function (or a metric), based on which we can accurately select the best-fit predictor via a sampling approach.

Analysis of Linear Regression versus Lorenzo Predictor

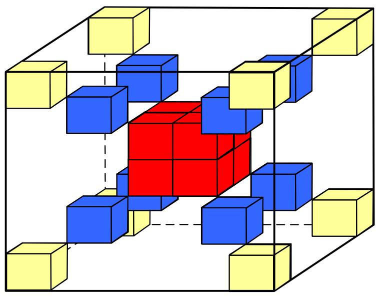
The two predictors are particularly suitable for various data blocks with different data features. For each data point to predict, the Lorenzo predictor [47] constructs a fixed quadratic hyperplane based on its 7 adjacent data points in a $2 \times 2 \times 2$ cube. No coefficients need to be saved for reconstructing the hyperplane during the decompression. However, Lorenzo predictor must conform to a strict condition when being used in lossy compression [92]. The prediction performed during the compression must use the decompressed values instead of original values; otherwise, unexpected data loss would be accumulated during the decompression, introducing significant distortion of data eventually. Using the decompressed values to perform the Lorenzo predictor would degrade the prediction accuracy significantly especially when the error bound is relatively large [92], leading to limited compression quality.

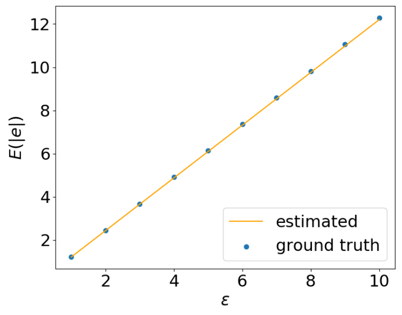
Unlike Lorenzo predictor, the linear regression-based predictor constructs an MSE-minimized linear hyperplane ($f^{(r)}_{111}$=$\beta_0 + \beta_1 x + \beta_2 y + \beta_3 z$) with four coefficients to store for each data block. The advantage of this predictor, however, is that it does not depend on the decompressed values during the prediction stage because the hyperplane will be reconstructed by the coefficients. In contrast, linear-regression may not be as accurate as Lorenzo predictor if the data exhibit spiky changes in a pretty small region or the compression error bound is relatively small, because the Lorenzo predictor corresponds to a quadratic hyperplane. This issue inspires us to seek an adaptive solution between the two predictors.

Estimate Prediction Errors and Select Best-fit Predictor

We adopt an effective, lightweight sampling method to select the best of the two predictors. In the following, we use the 3D dataset to explain our idea, cases with other dimensions could be extended similarly.

In a 3D block ($6\times6\times6$), we sample 24 points that are distributed along the diagonal lines. These points can also be regarded as on the 8 corners of the innermost $2\times2\times2$ cube, $4\times4\times4$ cube and $6\times6\times6$ cube, respectively, as shown in Fig. 3.2(a). We denote the set of the 24 sample points by $\{S_{24}\}$. Then, the cost function for both regression model and the Lorenzo predictor is defined as follows:


(a) Points sampled in a block



(b) Decompressed noise estimation

Figure 3.2: Sample points and decompressed noise estimation

$$E'_{predictor} = \sum_{(i,j,k)\in\{S_{24}\}} |f^{(p)}_{ijk} - f_{ijk}| \tag{3.6}$$

where $f^{(p)}_{ijk}$ refers to the predicted values ($f^{(r)}_{ijk}$, $f^{(L)}_{ijk}$). If one predictor is better than the other, it tends to have a lower cost (i.e., summed error). The reason we adopt absolute error instead of squared error in the cost function is that the squared errors may easily over amplify the cost value if there is one outlier, leading to skewed cost values. Formula (3.6) can be used directly as the cost function (denoted $E_{reg\text{-}predictor}$ in Algorithm 4) for the regression-based predictor.

The Lorenzo predictor, however, would be overestimated because the formula does not take into account the influence of decompressed data, such that we have to further adjust the cost function for the Lorenzo predictor. Without loss of generality, we assume that the compression errors are independent random variables following a uniform distribution in $[-\varepsilon, \varepsilon]$, according to the recent study on the distribution of compression errors [68]. Then, the perturbation in the final prediction will be a random variable e following a shifted and scaled Irwin-Hall distribution with parameter $n = 7$, which is a piecewise function. However, the expectation of the absolute value is hard to derive. Fortunately, since the

Lorenzo predictor always adopts 7 points to predict the data, the corresponding value can be approximated offline. We take $1M$ samples from this distribution and compute the expectation of their absolute values. The fitting curve is supposed to be a linear curve, because the error bound is only a scale factor for this distribution. Then, we achieved a very good fitting curve ($E(|e|) = 1.22\varepsilon$) as illustrated in Fig. 3.2(b). The cost (i.e., aggregated error) of the classic Lorenzo predictor can be adjusted as follows:

$$\begin{aligned} & E'_{LP} = \textstyle\sum_{(i,j,k)\in\{S_{24}\}} |f^{(L)}_{ijk} - f_{ijk} + e| \\ & \leq \textstyle\sum_{(i,j,k)\in\{S_{24}\}} |f^{(L)}_{ijk} - f_{ijk}| + \sum_{(i,j,k)\in\{S_{24}\}} |e| \\ & \approx \textstyle\sum_{(i,j,k)\in\{S_{24}\}} |f^{(L)}_{ijk} - f_{ijk}| + 24 * 1.22\varepsilon \end{aligned} \tag{3.7}$$

We estimate the prediction cost by this inequality because we opt to select the regression model considering the artifact issue that may occur to the Lorenzo predictor in the situation with relatively large error bounds. When the error bound is small, this extra adjustment has little impact on the final selection.

The cost value of the mean-integrated Lorenzo predictor(denoted E'_{mLP}) is estimated as the minimum value between the classic Lorenzo prediction error (i.e., Formula (3.7)) and mean prediction error, as shown below:

$$E'_{mLP} \approx \sum_{(i,j,k)\in\{S_{24}\}} min(|f^{(L)}_{ijk} - f_{ijk}| + 1.22\varepsilon\,,\, |\mu - f_{ijk}|) \tag{3.8}$$

where μ is calculated in line 4 of Algorithm 4.

In summary, as for the ℓ-PREDICTOR proposed in Algorithm 4, we estimate the error cost for the classic Lorenzo predictor and mean-integrated Lorenzo predictor by Formula (3.7) and Formula (3.8), respectively. In each data block, we select the final predictor with lowest cost based on their cost functions.

3.3.5 Optimizing Block-wise Prediction Accuracy

In this section, we introduce an optimization strategy that can further enhance the prediction accuracy in the block-wise design, such that the compression quality could be further improved.

A straightforward block-wise implementation idea is splitting the entire 3D dataset into multiple blocks/cubes and performing the prediction inside each block individually. ZFP, for example, adopts such a way to organize all the blocks in the compression, in that the blocks in ZFP are independent with each other. Such a straightforward block-wise implementation, however, may significantly degrade the overall data prediction accuracy in that many data points are located at boundaries of cubes such that they cannot be predicted by leveraging 3D Lorenzo predictor. For instance, if the block is a 6×6×6 cube, there will be $\frac{6^3-5^3}{6^3}$=41% of data points that can be predicted only by using 2D or 1D Lorenzo prediction method, significantly limiting the prediction accuracy. By comparison, if we perform as many 3D Lorenzo predictions as possible (similar to SZ [92] and FPZIP [69]) without block-wise limitation, only $\frac{512^3-511^3}{512^3}$=0.6% of the data points will suffer from degraded prediction accuracy for a 512×512×512 dataset.

We try to keep 3D Lorenzo predictions for the boundary points in our implementation. In the compression phase, we use ghost elements outside the prediction buffer. Suppose the global dataset is $N_1 \times N_2 \times N_3$ in three dimensions, and each block is a $b_1 \times b_2 \times b_3$ cube. Then, we allocate two $(b_1 + 1) \times (N_2 + 1) \times (N_3 + 1)$ prediction buffers and fill the bottom, front and left surface with 0s. Then, we apply the 3D prediction for each data point in the current prediction buffer and write predicted data into current data buffer for later prediction. Similarly, the data points on the top of the block (highest index is n_1-1) will be written to the bottom of the other buffer to eliminate unnecessary copy for prediction in next layer. Based on this method, only the boundary points on the entire $N_1 \times N_2 \times N_3$ dataset have to be predicted by degraded lower dimension prediction.

As for the decompression, ghost elements are no longer used because no buffers should be used in this situation. We unroll the loops in the prediction to eliminate conditions and branches for the boundary data points.

3.4 Hybrid Prediction Model

In this section, we propose a novel, adaptive lossy compression method based on the prediction-based model, which can improve the lossy compression quality significantly over that of existing state-of-the-art compressors. The fundamental idea is to improve the prediction accuracy by leveraging both data-fitting predictors (such as the Lorenzo predictor [47]) from prediction-based models and transform-based predictors from transform-based models. The design motivation comes from the fact that these two prediction models are particularly effective on different datasets or different error bounds, as we observed in numerous experiments based on large-scale simulation datasets.

We present a workflow of our adaptive error-controlled lossy compressor in Fig. 3.3. Highlighted boxes indicate the key modules to be described in the following text. The arrows in this figure indicate the execution order of each operation. We note that only one of the dashed boxes in step 4 is executed at runtime because we select the better solution of the two. In the first step we use a lightweight blockwise data sampling technique that samples the data along the diagonal of the datasets (with a granularity of block size 8, e.g., 8x8x8 for 3D datasets). The most critical parts are steps 2∼4, which are also the key novelty of our new design. Specifically, we estimate the overall rate distortion for the two types of predictors based on the data sampled in the previous step, respectively. Our estimation method covers the whole compression procedure for each type of predictor, including the efficiency of data prediction, the effectiveness of the corresponding coefficient compression, and the error-bounded quantization and encoding algorithms. We also optimize the strategies of encoding/compressing coefficients for both predictors, which will be detailed later. Step 5 involves error-bounded quantization and encoding, as a result, the overall compression method can strictly respect the user-specified error bound; details can be found in [92].

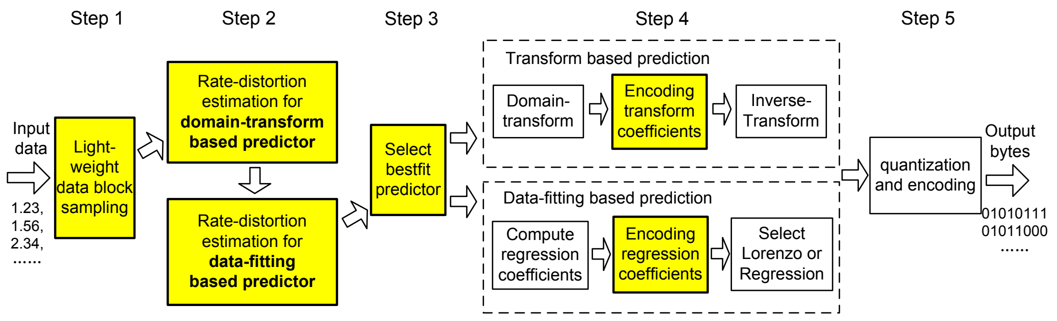

Step 1
Step 2
Step 3
Step 4
Step 5
Input data
1.23, 1.56, 2.34, ...
Light-weight data block sampling
Rate-distortion estimation for domain-transform based predictor
Rate-distortion estimation for data-fitting based predictor
Select bestfit predictor
Transform based prediction
Domain-transform
Encoding transform coefficients
Inverse-Transform
Data-fitting based prediction
Compute regression coefficients
Encoding regression coefficients
Select Lorenzo or Regression
quantization and encoding
Output bytes
01010111 01011000 ......

Figure 3.3: Steps of Our Adaptive Error-Bounded Lossy Compressor

3.4.1 Optimization of Lossy Compression with Data-Fitting Predictor

For the data-fitting predictor, we adopt the existing hybrid predictor used by SZ2.0 [65], and we propose an improved coefficient-compression strategy that can increase the compression ratio for high-compression cases. In what follows, we introduce the hybrid prediction model first and then describe our improvement strategy.

Hybrid Design in SZ2.0

In SZ2.0, the whole dataset is split into non-overlapped blocks (e.g., multiple 6x6x6 cubes for the 3D dataset), and the prediction method—either a Lorenzo predictor [47] or a linear regression method—is selected in each block by a sampling method. The former method is the same as the prediction method proposed in [92]. In the latter method, a linear hyperplane ($f(i,j,k) = \beta_0 + \beta_1 i + \beta_2 j + \beta_3 k$, where i,j,k are the indices of the data points), is constructed to fit the data in each block by using a least-squares linear regression method. Specifically, the four coefficients β_0, β_1, β_2, and β_3 (as per block) are calculated by letting the partial derivatives of the squared error loss function $\sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1}(\beta_0 + \beta_1 i + \beta_2 j + \beta_3 k - f_{ijk})^2$ be equal to 0. The constructed linear hyperplane is then used to predict the data values in the data block. In our implementation, we set the block size to 8x8x8, since it is neither too small for SZ to avoid high coefficient overhead (usually requiring 6+) nor indivisible by the block size required by ZFP (4) so that the efficiency of the two types of predictors on the same block is comparable.

Improved Regression Coefficient Compression

To improve the compression ratio significantly for high-compression cases in the data-fitting predictor, we propose a more efficient coefficient compression strategy for the linear regression prediction. In SZ 2.0 [65], all the regression coefficients were collected in terms of their types and then compressed by an IEEE-754-binary analysis. Compressing each group of coefficients is similar to the unpredictable data compression method used in SZ [32]. Four steps are involved: (1) calculate the value range and median value based on the coefficients in this group, (2) normalize all coefficients by taking away the median value from each data value, (3) encode the XOR-leading zero bytes using a 2-bit code for the data values that have the same leading bits with their preceding adjacent values, and (4) truncate the insignificant mantissa bits based on the error bound. Although this approach alleviates the required storage for the regression coefficients, it cannot help much when the error bound is large, because of the limitation of the XOR encoding.

In our new solution, we compress these coefficients based on the relatively high smoothness of the coefficients across the adjacent blocks selecting the linear-regression method. Specifically, we predict the current coefficient according to its adjacent coefficients of the same type, and we perform the linear quantization on the difference, followed by the Huffman encoding algorithm. That is, we apply a separate compression pipeline (except the lossless compression) to these coefficients besides the data points, at a cost of $\frac{4}{n_1 n_2 n_3}$ overhead since 4 coefficients out of $n_1 n_2 n_3$ data points in the block need to be compressed. Since the blocks selecting the linear regression may not be consecutive, the multidimensional prediction mechanism in the Lorenzo predictor cannot be used anymore to discard the unused regression coefficients. To overcome this issue, we first linearize the regression coefficients in the blocks selecting linear regression and perform the 1D prediction. If many blocks use regression, this method will lead to a high compression ratio because of the high smoothness in adjacent blocks. If only a small fraction of blocks use regression, the performance will degrade because the prediction cannot be accurate in this case. At this time, however, the data would be hard to compress, and the number of coefficients to be stored is small. Thus, the regression coefficients would take only a small percentage of storage, making little difference to the final rate distortion. A sample visualization of the regression coefficients (Hurricane Uf48, value-range-based error bound $5e-3$) is displayed in Fig. 3.4(a), when 86% of the blocks are choosing linear regression. The black pixels indicate the blocks selecting the Lorenzo predictor. From this figure, we can observe that the regression coefficients are smooth, from which the prediction mechanism would definitely benefit.

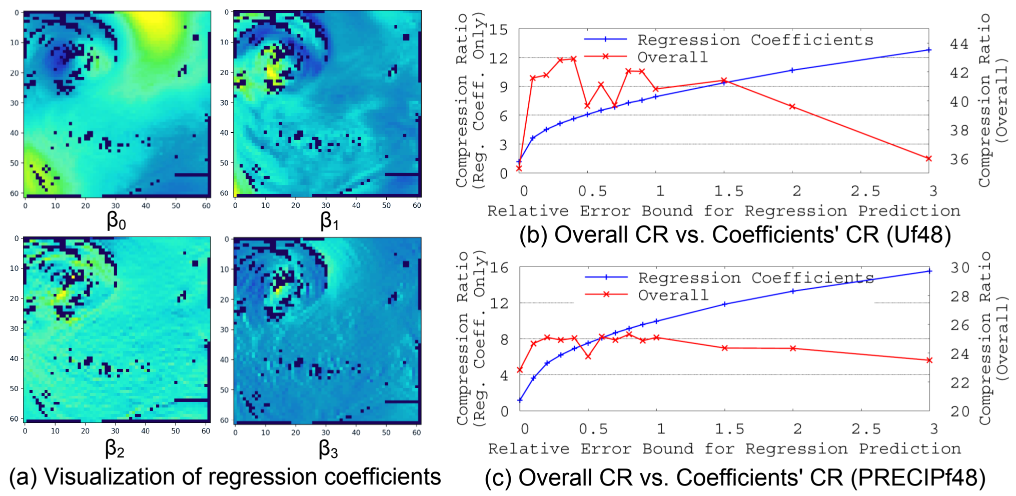
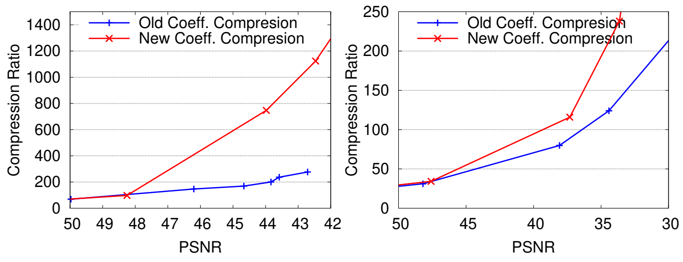


Figure 3.4: Analysis of Regression Coefficients on the Hurricane Simulation

A critical parameter for this approach is the error bound for the regression coefficients. We choose it so that the deviation caused by the decompressed regression coefficients (compared with the computed regression coefficients) will be bounded by an error bound ϵ. Let ϵ_0, ϵ_1, ϵ_2, and ϵ_3 stand for the absolute error bound for β_0, β_1, β_2, and β_3, respectively. A sufficient condition is $\epsilon_0 = \frac{\epsilon}{4}$, $\epsilon_1 = \frac{\epsilon}{4(n_1-1)}$, $\epsilon_2 = \frac{\epsilon}{4(n_2-1)}$, and $\epsilon_3 = \frac{\epsilon}{4(n_3-1)}$ because it guarantees $|\beta_0'+\beta_1'i+\beta_2'j+\beta_3'k-(\beta_0+\beta_1 i+\beta_2 j+\beta_3 k)| \leq |\epsilon_0|+|\epsilon_1(n_1-1)|+|\epsilon_2(n_2-1)|+|\epsilon_3(n_3-1)| \leq \epsilon$, where β_0', β_1', β_2', and β_3' denote the decompressed regression coefficients.

This method provides a solution to bound the prediction error, but how to set the error bound ϵ is still problematic. One must make a tradeoff between the efficiency of regression coefficients compression and the accuracy of regression prediction. A large error bound will lead to higher compression ratios for the regression coefficients, while degrading the overall compression ratio and/or PSNR since the prediction is not as accurate as before. On the contrary, a small error bound could guarantee the deviation of regression prediction at the cost of a lower compression ratio for the coefficients. We explore the error setting relative to the user-set error bound via an empirical way and the results of two representative fields of the hurricane datasets are shown in Fig. 3.4(b) and Fig. 3.4(c) when the value-range-based error bound is set to $5e-3$. In the figure, 86% and 66% of the blocks select regression for the given settings, respectively. Both figures indicate that the compression ratio of the regression coefficients keeps increasing while the overall compression ratio goes up and then down as the error bound gets looser, a result consistent with our analysis. We also note that the PSNR also decreases slowly with the looser error bound, from 51.85 to 50.94 (Uf48) and 52.62 to 51.11 (PRECIPf48). Therefore, we choose $\epsilon = 0.1$ as our default setting, because it has a reasonable overall compression ratio and PSNR while guaranteeing that the regression prediction is relatively accurate.

Figure 3.5 shows the improvement of the compression ratio based on our new coefficient compression method compared with the XOR encoding used by SZ 2.0 (denoted by old coeff. compression) for two example fields from the hurricane simulation. We can clearly observe that our new solution improves the compression ratio by up to 200∼300% in relatively high-compression cases.



(a) Uf48 (b) PRECIPf48

Figure 3.5: Ratio Distortion of Coefficients Compression on Hurricane Simulation

3.4.2 Optimization of Lossy Compression with Transform-Based Predictor

Another critical contribution in this work is that we develop and optimize a transform-based predictor based on transform-based compression models. This contribution can significantly improve the compression quality in many cases.

Transform-based models usually leverage a multidimensional transform for the purpose of value decorrelation [67], followed by some encoding algorithm. Embedded encoding is one of the most efficient encoding strategies. The key idea of our optimization is to improve the coefficient encoding in the embedded encoding stage for transform-based predictors. We adopt ZFP's nonorthogonal transform [67] as our transform-based predictor because its decorrelation efficiency has been confirmed to be more efficient than that of other transforms such as a discrete cosine transform or wavelet transform. Our approach is a generic solution that can also be applied to other types of transforms.

We apply the transform-based predictor on each data block with block size of 4 (e.g., 4x4x4 for 3D data), following three steps: (1) perform nonorthogonal transform to convert the 64 data points to 64 coefficients, (2) perform our improved coding strategy to compress the coefficients, and (3) predict each data value by the inverse nonorthogonal transform based on the reconstructed coefficients. Step 2 is the most critical in our improvement, and it has two key issues to resolve: developing an efficient coefficient compression method and estimating the optimal error setting for the transform-based predictor.

Efficient Transform Coefficient Compression Method

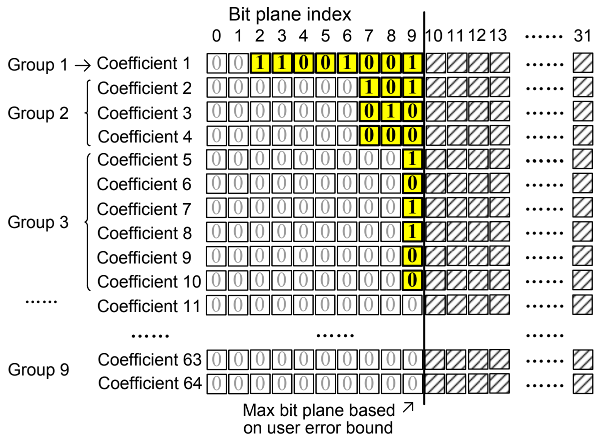


Figure 3.6: Embedded Coding (shaded bits would be dropped)

In what follows, we illustrate in Fig. 3.6 the classic embedded encoding proposed in [67] and then describe our improvement method. This classic coding strategy first reorders the transform coefficients from the upper left corner (highest energy in the transformed domain) to the bottom right corner (lowest energy in the transformed domain) such that these coefficients will be roughly sorted in a descending order automatically. Then, it divides the 64 coefficients into nine groups, according to the group sizes displayed in Table 3.1 (see second row). Next, it computes the max bit plane (in terms of the user-defined error bound and max value in current block), after which the bit planes are negligible and can be discarded. The bit planes located before the max bit plane (called *significant bit planes*, bits 0–9 in Fig. 3.6) will be compressed one by one losslessly. Since there are many zero bits due to the previous reordering of the coefficients, only a few bits (called *encoding bits*) that are organized in groups need to be stored (as highlighted in Fig. 3.6). In addition, three more types of metadata need to be stored: (1) *sign bits,* used to indicate the sign for the nonzero coefficients; (2) *group bits,* used to mark the grouping outlines; and (3) *skip bits,* used to indicate the ending bit of the current bit plane. As shown in Fig. 3.6, for example, group 1 has 8 encoding bits, 1 sign bit, 1 group bit, and 5 skip bits; group 2 has 9 (=3$\times$3) encoding bits, 2 sign bits (because one data is 0), 1 group bit, and 2 skip bits; and group 3 has 6 (=1$\times$6) encoding bits, 3 sign bits, 1 group bit, and 1 skip bit. In this example, other groups are not recorded because they are all zeros. Therefore, the total number of bits used to encode this block of data is 15+14+11=40. Note that from among the 40 bits, the number of encoding bits is 9+8+6=23, which means that 17 bits belong to metadata. They cannot be overlooked, as is validated by Table 3.1, because of the large difference between the average overall bit rate (see row 4) and the average encoding bit rate (see row 3).

A critical drawback of this embedded encoding method is the significantly uneven compression quality for different coefficients. As demonstrated in Table 3.1, the first and the second groups of coefficients are compressed poorly—only 2x for coefficients in the first group and 7x for those in the second group, far less than desired given the error bound. Nevertheless, these two groups of coefficients take up more than 75% of the storage space in the compressed data, significantly affecting the final compression ratio.

Table 3.1: Storage Decomposition of the Transform Coefficients on NYX Velocity_x with Value-Range-Based Error Bound 6e-3

Group ID	1	2	3	4	5	6	7	8	9
# coeffcients	1	3	6	10	12	12	10	6	4
bit rate (encoding)	8.29	2.938	0.4837	0.1835	0.09343	0.05405	0.03231	0.01808	0.009857
bit rate (overall)	15.64	4.847	0.6865	0.2324	0.1148	0.06632	0.04093	0.02534	0.01462
weighted percent	39.678	36.8899	10.4497	5.8959	3.4949	2.019	1.0384	0.3857	0.1484

We propose a novel encoding algorithm that can significantly improve the coding efficiency. The key idea comes from our observation that the coefficients are still consecutive across blocks such that we can improve the compression efficiency by taking advantage of this feature. As an example, we demonstrate in Fig. 3.7 the visualization of the first four sets of transform coefficients (128x128) aggregated across all blocks for the slice 300 (a 512x512 image) of the NYX velocity_x field. In particular, the first coefficients extracted from all the blocks construct an image that looks much like the original raw data image (the left image in Fig. 3.7). The second, third, and fourth transform coefficients across all blocks also have a relatively high spatial continuity, based on which we can further improve the coefficients' compression ratios.

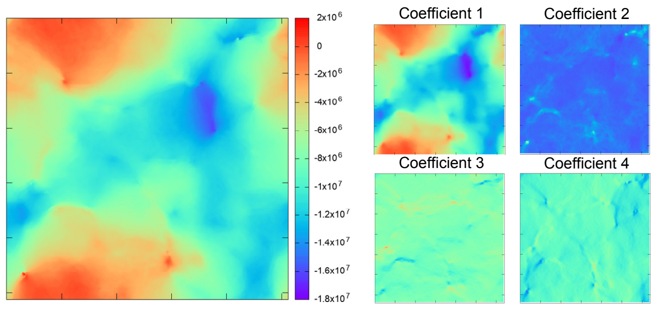


Figure 3.7: Visualizing Raw Data and Transform Coefficients (NYX:Velocity x)

In our solution, we adopt the data-fitting compression method (see Section 3.4.1) to compress/encode these coefficients with the same error bound as the traditional embedded encoding. We call it *SZ encoding* in this section. The groupwise average bit rates of these coefficients are displayed and compared with those of embedded encoding in Fig. 3.8(a). This figure illustrates an interesting trend of the groupwise bit rates on the two encoding strategies: the SZ encoding outperforms the embedded encoding on the first two groups, while the embedded encoding is better on the other groups. These results inspire us to seek a hybrid approach selecting SZ encoding for the first few groups and switching to embedded encoding for the remaining groups.

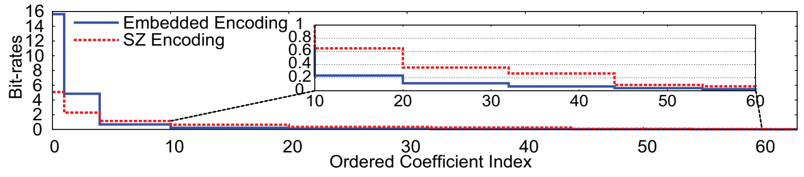
We use a sampling approach to determine the appropriate number of coefficients to be compressed by SZ encoding. We sample the data blocks on the diagonal of the dataset to involve data with different positions with low sampling rate. Specifically, we divide the whole dataset into $\frac{N_1}{N_m}\times\frac{N_2}{N_m}\times\frac{N_3}{N_m}$ subblocks (each being a N_mxN_mxN_m cube), where $N_m = \min(N_1, N_2, N_3)$, and N_i refer to the dimension sizes (i.e., number of elements along the dimension i). Then, we sample the four diagonals in each subblock by the granularity of 8x8x8 blocks,[1] which will glean $4 * \frac{N_m}{8}\frac{N_1N_2N_3}{N_m^3} * 8^3 = \frac{256N_1N_2N_3}{N_m^2}$ data points, introducing small overhead to the dataset with the comparable dimension sizes. In fact, other sampling methods such as uniform sampling could work equally well as long as they are done in block granularity. But this diagonal sampling method could lead to reasonable result with quite small sampling rate when the dimension sizes are close.

The sampled data will be transformed to the coefficients, which will be used to estimate the bit rates of the two encoding approaches by our designed estimation method. For

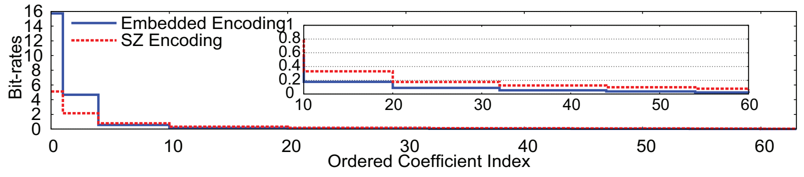
[1]We choose the granularity of 8x8x8 blocks in order to be consistent with the sampling granularity.

the embedded encoding approach, its groupwise number of encoded bits can be computed precisely. The encoding bits of a data point can be computed exactly by subtracting the max bit plane by the position of the first nonzero element. The sign bits of a group can be approximated as the number of coefficients whose encoding bits have at least one nonzero bit. The group bit of one group is incremented by 1 as long as there exhibits one coefficient with non-all-zero encoding bits. The skip bits, on the other hand, can be computed by taking away the length of the encoding bits of the next group from the length of the encoding bits of the current group. Therefore, we can estimate the overall bit rate of each group for the embedded encoding accurately. For the estimation of SZ encoding's bit rate, we estimate the Lorenzo prediction errors by computing the differences between the four top 4x4x4 subblocks and the four bottom 4x4x4 subblocks in each sampled 8x8x8 block. This is accurate enough for the bit rate estimation because it simulates the 1D Lorenzo predictor. We then compute quantization indices and use the entropy formula $br = \Sigma_i p_i \log p_i$ to estimate the final bit rate, since SZ adopts a Huffman encoding in the last step.

Figure 3.8 presents the estimated bit rates based on the sampled data versus the real bit rates based on all data when compressing the transform coefficients over the Velocity_x field in the NYX dataset. By comparing the blue curves and red curves between the two subfigures, respectively, we can see that our estimation is accurate for both embedded encoding and SZ encoding for the first ten coefficients, which take the majority of the storage space (over 85% according to Table 3.1). The estimation for the remaining coefficients of the SZ encoding is not that accurate because the bit rate estimated by the entropy formula is a lower bound. We argue that it would not degrade the overall performance prominently



(a) Real Bit Rates of Coefficient Compression



(b) Estimated Bit Rates of Coefficients Compression on Sampled Data

Figure 3.8: Efficiency of Compressing Coefficients (NYX: Velocity_x with value-range based error bound 6e-3)

because the difference of SZ encoding and embedded encoding is relatively small and the weighted percentages of the remaining coefficients are low for the target bit rate. If we compress the example coefficients in Fig. 3.6 using our hybrid approach, the average number of bits after the coefficient compression would be 5.1+2.3×3+11=23, obtaining 70+% improvement over the original embedded encoding (40 bits).

Estimation of the Optimal Error Setting

In the transform-based prediction model, we still need to set an error bound to compress the transformed coefficients. Doing so involves tradeoffs. Too small an error bound may degrade the prediction model to the domain-transform kernel without quantization, overpreserving the errors unexpectedly. Too large an error bound, on the other hand, will cause a large loss after the inverse-transform and thus result in low quantization efficiency.

To obtain the best tradeoff, we develop an efficient strategy by a sampling method. We reuse the same sampled data as well as the transformed coefficients to reduce the overhead. Our objective is to search for the optimal error bound ratio (i.e., the ratio of the error bound used by the transform-based predictor to the user's target error bound) in terms of the sampled data. Specifically, we set the initial error bound ratio to 1, which means that the prediction model is degraded to the domain-transform kernel because the reconstructed data must be within the user's error bound. Then, we estimate the optimal numbers of the transform coefficients for SZ encoding and embedded encoding. Next, we reconstruct the transform coefficients, based on which the inverse transform will be performed. After that, the bit rate can be estimated by adopting the remaining steps of the prediction-based compression model (i.e., error-bounded quantization and lossless compression), and the mean squared error (MSE) can be calculated. This procedure is repeated with doubled error bound ratios 8 times (we set it to 8 times because it can always cover the best tradeoff based on our experiential observations).

Our algorithm selects the best error bound setting based on the 8 rate-distortion results collected from above. Specifically, the rate distortion with an error bound ratio of 1 serves as the baseline (i.e., bit rate br_0 and MSE mse_0) and increases by 2x for each trial. We define the bit-rate decay rate as $\log \frac{br_0}{br_i}$ and the MSE increase rate as $\log \frac{mse_i}{mse_0}$ for all other rate distortion results. The optimal error setting is the one that maximizes $\frac{\log \frac{br_0}{br_i}}{\log \frac{mse_i}{mse_0}}$, because it indicates the highest bit-rate decay over the MSE increase rate.

3.4.3 Selecting the Best-Fit Prediction Method

The basic idea in choosing the best-fit prediction method is to compared the rate distortion estimated from the previous sections between the two types of predictors and selecting the better solution. A good rate distortion means the situation with high PSNR and low bit rate. Based on this, we present the pseudo-code in Algorithm 5, where $R_f(e)$ and $P_f(e)$ refer to the bit rate and PSNR, respectively, using the data-fitting-based prediction method with the error bound e. Similarly, $R_t(e)$ and $P_t(e)$ refer to the bit rate and PSNR using the transform-based prediction method. Given a specific error bound, our algorithm involves two critical stages:

Stage I (line 1): Using the sampled data, estimate the optimal error bound setting (denoted ϵ_t^*) based on Section 3.4.2, such that the selected setting may result in the rate distortion being better than others for the transform-based predictor. We denote the corresponding bit rate and PSNR by R_t^* and P_t^*, respectively.

Stage II (line 4$\sim$22): Using the same sampled data, estimate the rate distortion for the data-fitting-based prediction model, in the vicinity of the rate distortion point (R_t^*, P_t^*) optimized in Stage I.

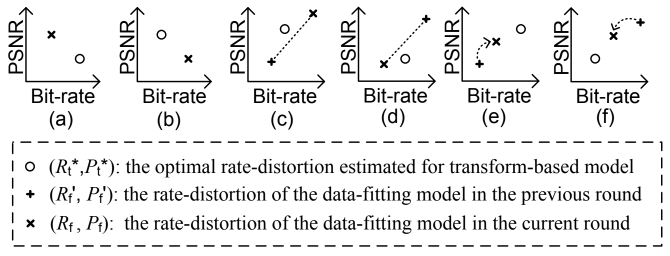

PSNR
Bit-rate
(a)
(b)
(c)
(d)
(e)
(f)
○ (R_t*,P_t*): the optimal rate-distortion estimated for transform-based model
+ (R_f', P_f'): the rate-distortion of the data-fitting model in the previous round
× (R_f, P_f): the rate-distortion of the data-fitting model in the current round

Figure 3.9: Illustration of Best-Fit Predictor Selection Algorithm

Algorithm 5 Best-Fit Predictor Selection Algorithm

Input: user-specified error bound (denoted as ϵ)
Output: bestfit predictor (either data-fitting or domain-transform)

```
1: Estimate the optimal rate-distortion point, denoted by ($R_t^*$, $P_t^*$), for the transform-based prediction
   model (Section 3.4.2);
2: $e \leftarrow \epsilon$; /*Set the initial error setting to the user-specified error bound*/
3: $\mathit{Selection}$←null, $R'_f$←null, $P'_f$←null;/*initialization*/
4: repeat
5:    if ($R_f(e) \leq R_t^*$ and $P_f(e) > P_t^*$) then
6:       Selection ← data-fitting; /*data-fitting is better*/
7:       break;
8:    else if ($R_f(e) \geq R_t^*$ and $P_f(e) < P_t^*$) then
9:       Selection ← domain-transform; /*domain-transform is better*/
10:      break;
11:   else if ((($R'_f < R_t^*$ and $P'_f < P_t^*$) and ($R_f(e) > R_t^*$ and $P_f(e) > P_t^*$)) or (($R'_f > R_t^*$ and $P'_f > P_t^*$) and
      ($R_f(e) < R_t^*$ and $P_f(e) < P_t^*$))) then
12:      Estimate $P_f(\epsilon_t^*)$ by linear interpolation with the two rate-distortion points ($R'_f$,$P'_f$) and
         ($R_f(e)$,$P_f(e)$);
13:      $\mathit{Selection}$← $\arg\max(P_f(\epsilon_t^*), P_t^*)$;/*select with higher PSNR*/
14:      break;
15:   else if ($R_f(e) < R_t^*$ and $P_f(e) < P_t^*$) then
16:      $R'_f \leftarrow R_f(e)$, $P'_f \leftarrow P_f(e)$; /*record the rate-distortion point*/
17:      $e \leftarrow 2e$; /*double the error bound for next checking*/
18:   else if ($R_f(e) > R_t^*$ and $P_f(e) > P_t^*$) then
19:      $R'_f \leftarrow R_f(e)$, $P'_f \leftarrow P_f(e)$; /*record the rate-distortion point*/
20:      $e \leftarrow e/2$; /*half the error bound for next checking*/
21:   end if
22: until ($\mathit{Selection} \neq$ null)
23: Output $\mathit{selection}$;
```

We illustrate in Fig. 3.9 all six cases corresponding to the if-branches of Algorithm 5. The algorithm keeps searching for the error bound setting (with exponential increase/decrease) for the data-fitting-based prediction model (in the cases (e) and (f)) until we can judge which predictor is better (via the cases (a)–(d)). That is, the subfigures (a)–(d) are the termination condition, corresponding to lines 5–7, lines 8–10, and lines 11–14, respectively, in the algorithm, while the subfigures (e) and (f) correspond to lines 15–21, meaning that the error bound needs to be doubled or halved to keep searching toward the target estimated optimal point (R_t^*,P_t^*).

3.5 Experimental Evalutions

In this section, we compare our proposed compression method with other state of the art error-bounded compressors (SZ [92], ZFP [67]), which are currently the best existing generic lossy compressors for scientific data compression [70]. We also compare our method with other lossy compressors that are widely used in the scientific simulations. We also try best to optimize the compression quality of all the six lossy compressors from the perspective of PSNR. For instance, we adopt the absolute error bound mode for both SZ and ZFP in that they lead to better compression quality than other modes in our experiments.

3.5.1 Multi-algorithm Predictor

Experimental Setting

We conduct our experimental evaluations on a supercomputer using 8,192 cores (i.e., 256 nodes, each with two Intel Xeon E5-2695 v4 processors and 128 GB of memory, and each processor with 16 cores). The storage system uses General Parallel File Systems (GPFS). These file systems are located on a raid array and served by multiple file servers. The I/O and storage systems are typical high-end supercomputer facilities. We use the file-per-process mode with POSIX I/O [110] on each process for reading/writing data in parallel [1]. The HPC application data are from multiple domains including CESM-ATM climate simulation [53], Hurricane ISABEL simulation [46], NYX cosmology simulation [73], SCALE-LETKF weather simulation (called *S-L Sim* for short) [108]. All of the data are available online [82] Each application involves many simulation snapshots (or time

[1]POSIX I/O is comparable to MPI-IO [100] on thousands of simultaneous write/read [102].

steps). We assess only meaningful fields with relatively large data sizes (other fields have constant data or too small data sizes). Table 3.2 presents all the 104 fields across these simulations. The data sizes per snapshot are 1.2 GB, 3.2 GB, 3 GB and 2 GB for the above four applications respectively.

Table 3.2: Simulation fields used in the evaluation

Application	**# Fields**	**Dimensions**	**Examples**
CESM-ATM	79	1800×3600	CLDHGH, CLDLOW ⋯
Hurricane	13	100×500×500	CLOUDf48, Uf48 ⋯
NYX	6	512×512×512	dark_matter_density, v_x ⋯
S-L Sim	6	98×1200×1200	U, V, W, QC ⋯

We assess the compression quality based on the four criteria proposed in Section 4.4.1. Since PSNR is the most critical indicator as discussed in Section 4.4.1, we mainly adopt this metric to assess the distortion of data. In addition, we also evaluate the Pearson correlation and structural similarity (SSIM) index for different compressors because they were mentioned in some literatures [8, 106, 107]. Pearson correlation can be computed by $\rho_{X,Y}=\frac{E(X-\mu_X)E(Y-\mu_Y)}{\sigma_X\sigma_Y}$, where μ_c and σ_c ($c\in\{X,Y\}$) are means and deviations of X and Y, respectively. SSIM is considered a critical metric of lossy compression by the climate community [106]. The higher the Pearson correlation or SSIM, the better the compression quality.

Evaluation Results

We first check the maximum compression errors of our compressor using all the 104 fields and confirm that our compressor can respect the error bound (denoted by ε)

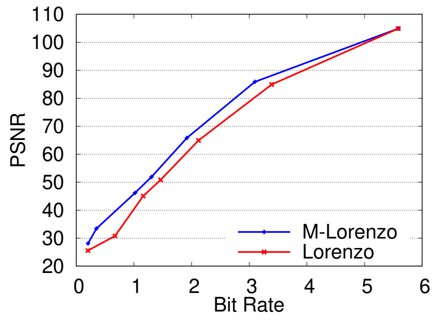
strictly. We present a few examples in Table 3.3.

Table 3.3: Maximum Compression Error vs. Error Bound

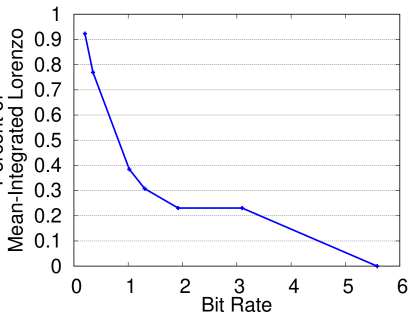
fields	**bound**	**max err**	**bound**	**max err**
Hurricane-CLOUDf48	0.1	0.099999955	0.01	0.00999998
NYX-v_x	0.1	0.0999966	0.01	0.009999995
CESM-CLDHGH	0.1	0.099949	0.01	0.0099999

In what follows, we first compare the three fundamental predictors (Lorenzo, mean-integrated Lorenzo and linear regression) to showcase the importance of the adaptive design in Fig. 3.10 and Fig. 3.11. After that, we compare the overall rate-distortion among all the 6 lossy compressors in Fig. 3.12, which is the most critical evaluation result in terms of compression quality. We also demonstrate the difference of visual quality across various lossy compressors with the same compression ratios. Finally, we investigate the parallel I/O performance gain with different execution scales using our compressor against two other most efficient state-of-the-art compressors.

Fig. 3.10 demonstrates the effectiveness of our mean-integrated Lorenzo predictor (denoted M-Lorenzo) over the original Lorenzo predictor, using Hurricane ISABEL dataset. As shown in Fig. 3.10 (a), the mean-integrated Lorenzo always leads to a smaller bit-rate (or higher compression ratio) than the original Lorenzo predictor does with the same level of rate distortion (i.e., PSNR). Specifically, the mean-integrated Lorenzo predictor obtains the compression ratio about 3X as high as that of the original Lorenzo predictor, when the PSNR is around 30. Fig. 3.10 (b) shows the fraction of the fields adopting the mean-integrated Lorenzo predictor in the Hurricane datasets. We observe that the percentage drops as bit-rate increases, due to the fact that higher bit-rate corresponds to lower coverage of the densest interval, which leads to lower percentage on choosing the mean-integrated Lorenzo predictor in turn. Similar phenomenons can also be observed in other datasets, which we did not show here because of the page limitation.

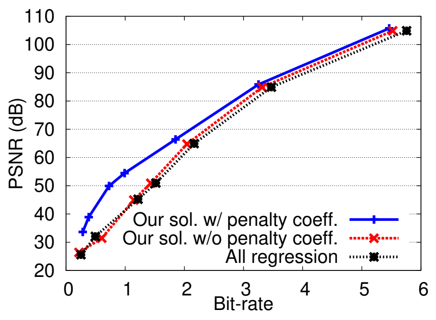


(a) Overall Rate-distortion

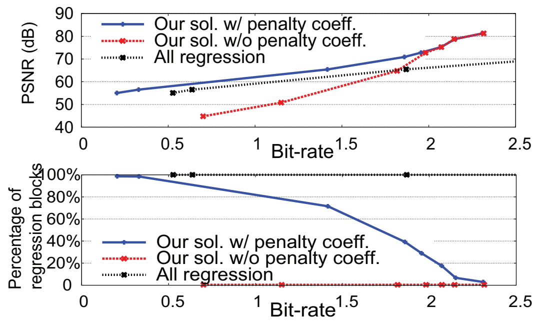


(b) Percentage of M-Lorenzo Predictor

Figure 3.10: Effectiveness of Mean-Integrated Lorenzo Predictor using Hurricane-ISABEL



(a) Overall Rate-distortion

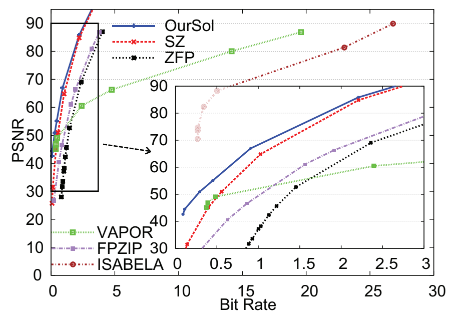


(b) Detailed analysis using TCf48

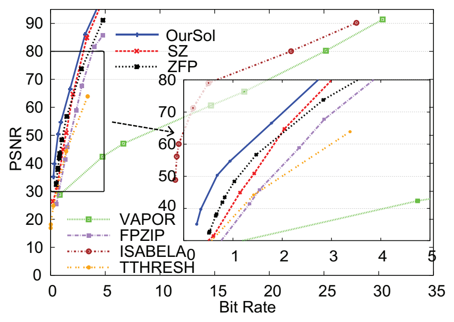
Figure 3.11: Significance Analysis for penalty coefficient using Hurricane-ISABEL

In Section 3.3.4, we derived a *penalty coefficient*, which is critical to determine the best-fit predictor. In Fig. 3.11, we present the significance of the *penalty coefficient*, by comparing three solutions (the solution with/without penalty coefficient and the solution adopting only regression-based predictor). As shown in Fig. 3.11(a), the solution with our derived penalty coefficient outperforms the other two significantly. As illustrated in Fig. 3.11(b), without the penalty coefficient, the compression quality would be degraded significantly, since a large majority (99.6%) of blocks would select the Lorenzo predictor (see red curve in the bottom sub-figure of Fig. 3.11(b)) because of over-estimation of the Lorenzo prediction ability as discussed in Section 3.3.4. We also present the percentage of blocks selecting the regression-based predictors, demonstrating that our adaptive solution does select different predictors for different blocks during the compression. The percent of regression blocks drops from 98% to 3% as the bit-rate increases, which is consistent with the compression quality of the two methods.

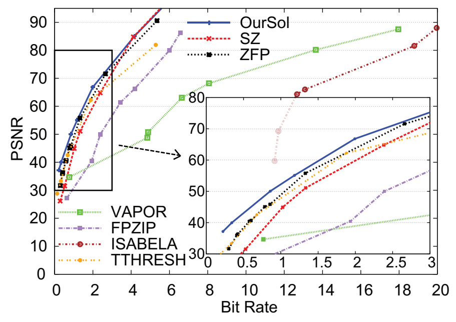
In Fig. 3.12, we present the overall rate-distortion (PSNR versus bit-rate (or compression ratio)) calculated using all the fields for each application (Fig. 3.12 (a) is missing TTHRESH because it cannot work on 2D dataset). It is observed that under the same compression ratio, our solution leads to higher PSNR with than other compressors. Specifically, our solution, SZ and ZFP generally exhibit better rate-distortion than other compressors including FPZIP, VAPOR, and TTHRESH, all of which outperform ISABELA. From among the three best compressors, the PSNR of our compressor is 10%$\sim$100% higher than that of the other two (SZ and ZFP) when the compression ratio is the same. This gap increases with higher compression ratio, as shown in the four sub-figures/BigData18. In particular, as for CESM-ATM data, when the PSNR is about 45, our compressor leads the compression ratio to 200:1, while SZ and ZFP get the compression ratio of 76:1 and 25:1 respectively. Based on the four applications, we observe that the compression ratio of our compressor is 1.5X∼8X as high as that of SZ and ZFP with the same PSNR, which is a significant improvement.



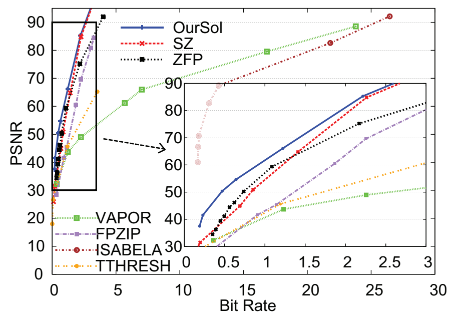
(a) CESM-ATM



(b) Hurricane



(c) NYX



(d) S-L Sim

Figure 3.12: Rate-distortion (PSNR versus Bit-rate or Compression Ratio)

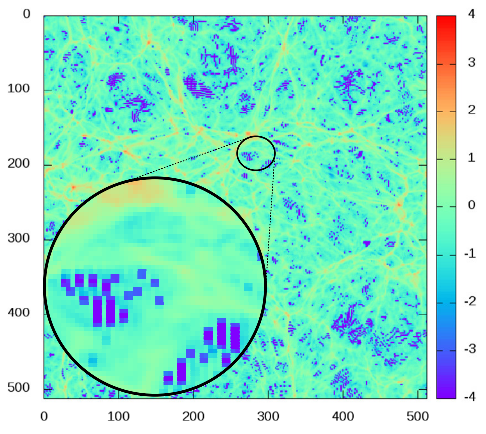
We select three typical examples from the 104 fields across different applications to demonstrate the visual quality of the decompressed data with different compression ratios compared with the original data (also known as raw data).

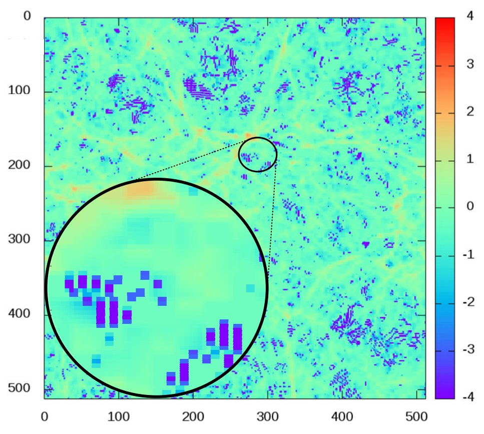
The decompressed data under our solution has a better visual quality than either SZ or ZFP does, based on the NYX (velocity_x) dataset. Correspondingly, the SSIM index of our solution (0.9855) is higher than that of SZ (0.7349) and that of ZFP (0.7004) by 34% and 40%, respectively.

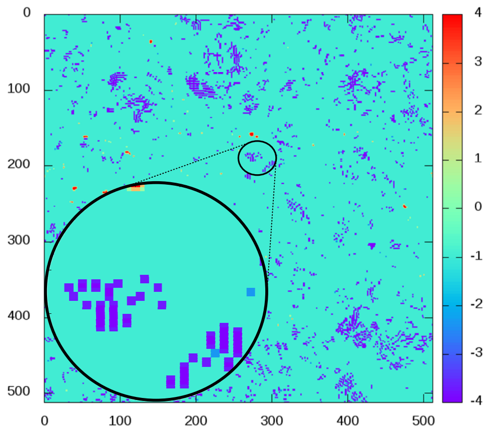
As mentioned previously, we note that some fields have very large value ranges while most of data are clustered in a small close-to-zero range, so the users compute the logarithm of the data before their analysis or visualization. We compress the log data instead of the original data since this would improve the compression quality for all lossy compressors, as suggested by users. As an example, we present the compression result of dark_matter_density field from NYX simulation in Fig. 3.13. By zooming in a small region, we can clearly see that our solution has much higher resolution than others. This is consistent with the SSIM measure: the SSIM index of SZ is higher than others by 62.8% and 82.5%, respectively. By observing the decompressed images, SZ suffers from a serious data loss as shown in Fig. 3.13 (c) because many data points would be flushed to the same values when the error bound is relatively large. ZFP suffers from the unexpected mosaic effect because it splits the entire dataset into pretty small blocks (4×4×4) and the decompressed data are flushed to the same value in each block when compression ratio is high.

Furthermore, we show the visualization result of the CLOUDf field in Hurricane-ISABELA in Fig. 3.14. Similarly, the decompressed data under our solution has a better visual quality than does SZ or ZFP. Besides some unconspicuous strips (artifacts) in the blue background, SZ also has some distortion in the enlarged region. On the other hand, ZFP has little distortion in the background, but exhibits block-wise effects in the enlarged region. Our solution has little distortion in both the background and local region, which leads to very high overall visual quality (SSIM 0.9966).

In addition to error-bounded lossy compressors, we also present the visual quality of using down-sampling methods as compression and interpolation methods as decompres-


(a) original raw data


(b) our sol. (PSNR=29,SSIM=0.6867)


(c) SZ (PSNR=24.5,SSIM=0.4218)

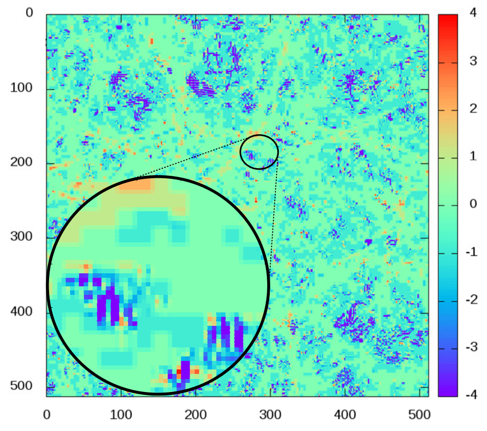
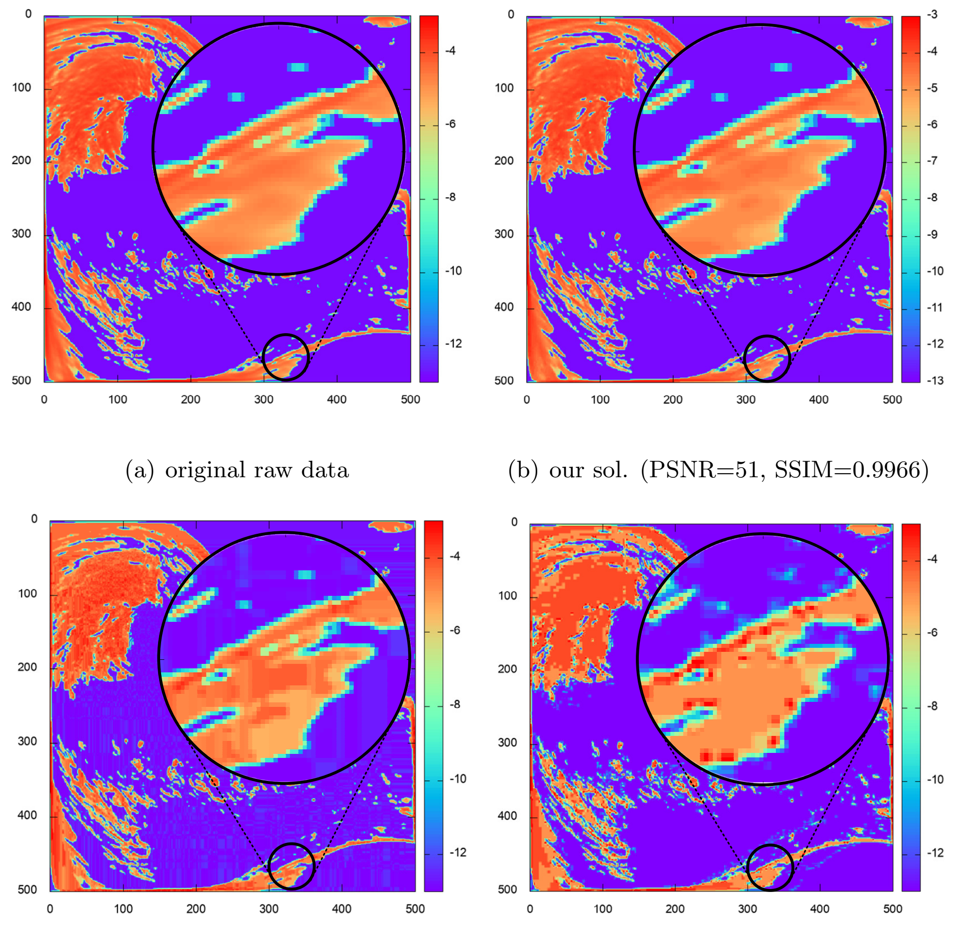

(d) ZFP (PSNR=21.3,SSIM=0.3762)

Figure 3.13: Data Distortion of NYX(dark_matter:slice 100) with CR=58:1

sion (a mechanism widely used in the visualization community) in Fig. 3.15 for comparison. During the compression, one data point would be sampled uniformly every 4 points for each field in each dimension, leading to the compression ratios of 64:1. During the decompression, tricubic interpolation [56] is used to reconstruct the missing points. We can clearly observe that our solution exhibits much better visual quality (see Fig. 3.13 (b) vs. Fig. 3.15 (a); Fig. 3.14 (b) vs. Fig. 3.15 (b)), in that the downsampling+interpolation method over-


(a) original raw data

(b) our sol. (PSNR=51, SSIM=0.9966)

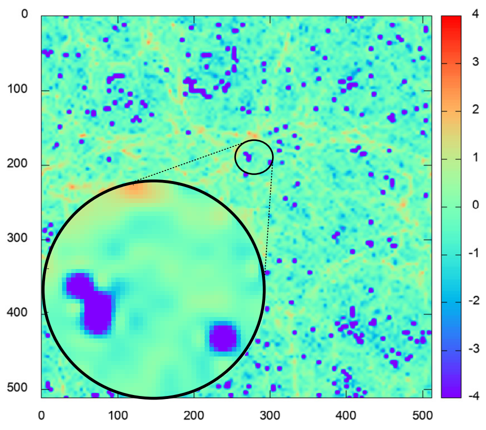
(c) SZ (PSNR=29.9, SSIM=0.6573)

(d) ZFP (PSNR=22.5, SSIM=0.8893)

Figure 3.14: Data Distortion of Hurricane(CLOUDf:slice 50) with CR=66:1

smoothed the regions with diverse values (Fig. 3.15 (a)) and over-amplified some boundary data points (Fig. 3.15 (b)).

Pearson correlation coefficient has been used to assess the correlation between the original dataset and decompressed dataset [8] by the community. We have validated that the Pearson correlation coefficients under our solution are higher than those of SZ and ZFP in a large majority of cases across all the four applications. We here exemplify


(a) dark_matter_density(CR=64:1,PSNR=18.1, SSIM=0.4345)

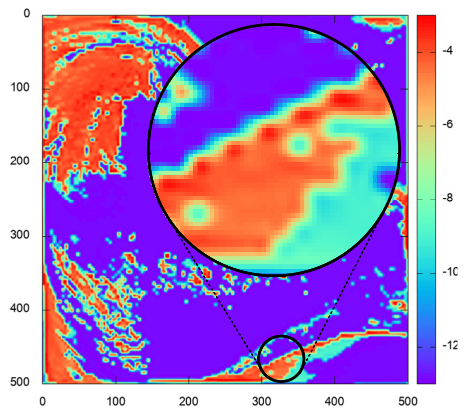

(b) CLOUDf(CR=64:1,PSNR=17.7,SSIM=0.7681)

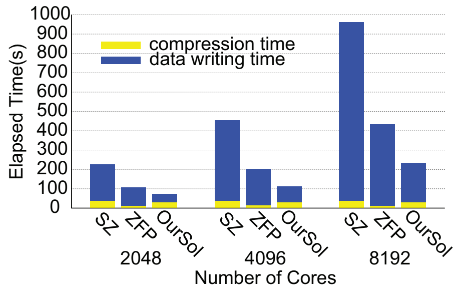
Figure 3.15: Data distortion of uniform down-sampling and tricubic interpolation of NYX dataset with similar compression ratios

the correlation results in Table 3.4 using only NYX data due to the space limitation. We run three compressors and tune their compression ratios to be 60:1∼70:1. The reason we cannot fix the compression ratio across fields is that ZFP exhibits piece-wise compression ratios with various error bounds. We did not use fixed-rate mode for ZFP because it is always worse than its fixed-absolute-error mode in our experiments with respect to the rate-distortion.

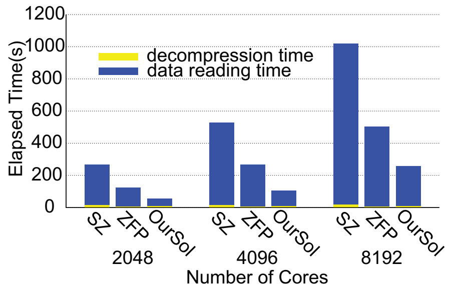
Table 3.4: Pearson Correlation Coefficients of 6 Fields in NYX

fields	**our solution**	**SZ**	**ZFP**	**CR**
dark matter density	0.959274616	0.939890773	0.763353467	58:1
baryon density	0.999537914	0.986076774	0.999529095	66:1
temperature	0.992798653	0.870542883	0.995984391	66:1
velocity_x	0.999961851	0.996793357	0.999871972	70:1
velocity_y	0.999944697	0.997787266	0.999912867	70:1
velocity_z	0.999867946	0.992133964	0.999826937	63:1

We evaluate the overall data dumping/loading performance on the NYX simulation using different lossy compressors with the same level of data distortion. Specifically, we set the PSNR for its fields to 60 except for dark matter density (PSNR=30) and baryon density (PSNR=40), because such a setting already reaches a high visual quality (as exemplified in Fig. 3.13(b)). The evaluation is weak-scaling. Each rank processes 3 GB data and the total data size increases linearly with the number of cores. We assess the performance by running different scales (2,048 cores $\sim$ 8,192 cores), and each core needs to process a total of 3 GB data during the execution. The total data size is up to 24 TB when using 8,192 cores, which may take over six hours to dump. We present the breakdown of the data dumping performance (sum of compression time and data writing time) and data loading performance (sum of data reading time and decompression time) in Fig. 3.16. Since the parallel performance is dominated by the data reading/writing time (to be shown later) and VAPOR, FPZIP and ISABELA have low compression ratios, they exhibit much higher overall data dumping/loading time than the other compressors. Accordingly, we omit their results in the figure to clearly observe the performance difference among the three best solutions (our solution, SZ and ZFP).



(a) Data dumping performance



(b) Data loading performance

Figure 3.16: Performance evaluation using NYX

It is observed that the overall data dumping time under our solution takes only 24% and 54% of the time cost by SZ and ZFP when adopting 8,192 cores, which correspond to 4.12X and 1.86X performance, respectively. The key reason is that our compressor leads to significantly higher compression ratios than the other two compressors when the PSNR is in the range of [30,60], as shown in Fig. 3.12(c). When running the simulation with 8,192 cores, our solution can also obtain 1.95X higher data loading performance (49% lower time cost) than the second best solution (ZFP) does. It is a little higher in comparison with data dumping performance (1.86X), because of the higher decompression rate than the compression rate.

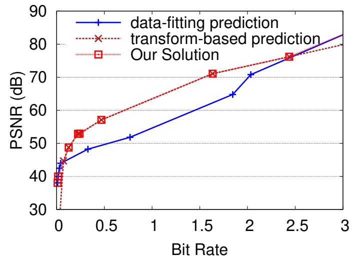
3.5.2 Hybrid Prediction Model

Experimental Setting

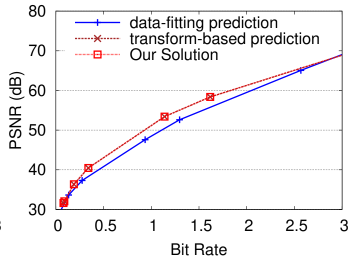
We conducted our experimental evaluation on the Bebop supercomputer [89] at Argonne National Laboratory using 2,048∼8,192 cores (i.e., 64∼256 nodes, each with two Intel Xeon E5-2695 v4 processors and 128 GB of memory, and each processor with 18 cores). The storage system uses General Parallel File Systems (GPFS), which are located on a raid array and served by multiple file servers. The I/O and storage systems are typical high-end supercomputer facilities equipped with two I/O nodes. Such a system is a good case for demonstrating the I/O bottleneck when I/O is saturated, which is a common case when running extremely large-scale applications (e.g., with millions of cores) on cutting-edge supercomputers because I/O nodes are always far fewer than compute nodes. We use the file-per-process mode with POSIX I/O [110] on each process for reading/writing data in parallel.

The application data is from multiple domains including Hurricane Isabel climate simulation [46], NYX cosmology simulation [73], and SCALE-LETKF weather simulation [108]. Each application involves multiple simulation snapshots (or time steps). Without loss of generality, we assessed only meaningful fields with relatively large data sizes (other fields have either constant data or data sizes that are too small). Also, some datasets with closed clustered data in [0, 1] are transformed to its logarithmic domain for better visualization quality, as suggested by the domain scientists. The data sizes per snapshot are 1.3 GB, 3 GB, and 6 GB per core for the three applications, respectively. Thus, the total data sizes per snapshot are 10.4 TB, 24 TB, and 48 TB, respectively, when the execution scale is 8,192 cores in our experiment.

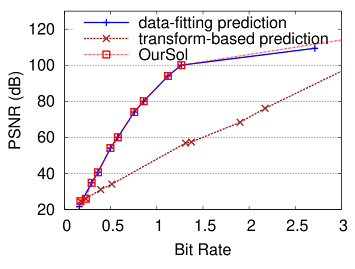
We focus on the improvement in rate distortion and in parallel I/O performance by leveraging our designed lossy compressor, as compared with three state-of-the-art lossy compression methods: SZ2.0 [65], ZFP0.5.4 [67], and an automatic online selection between SZ and ZFP from [96] (called AOS for short). They have been confirmed as the best in



(a) Uf48

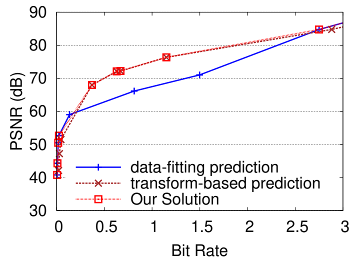


(b) PRECIPf48

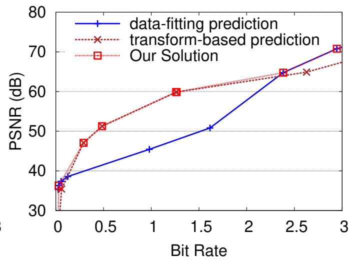


(c) QCLOUDf48

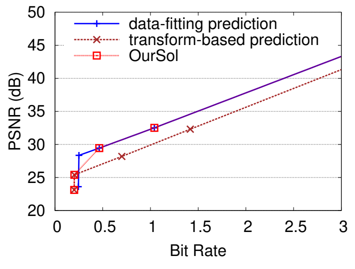
Figure 3.17: Assessment of Bestfit Selection Algorithm on Hurricane Isabel



(a) velocity_x

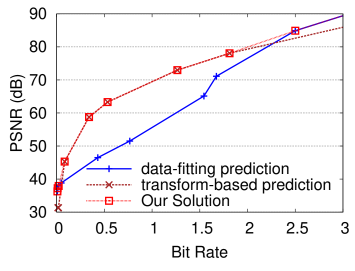


(b) baryon_density

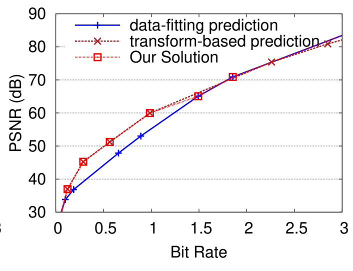


(c) dark_matter_density

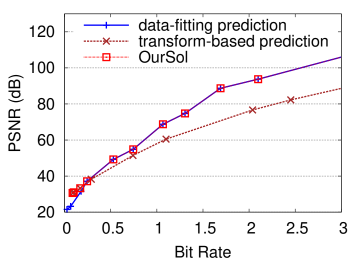
Figure 3.18: Assessment of Bestfit Selection Algorithm on NYX



(a) RH



(b) QR



(c) QI

Figure 3.19: Assessment of Best-Fit Selection Algorithm on SCALE-LETKF

class [23,70,96]. Since the compression ratio will dominate the I/O performance in a large-scale environment (as discussed in Section 4.4.1), we also investigated the rate distortion of different compressors. We note that we did not evaluate error bounds as compression quality because domain scientists care more about overall statistical errors, as mentioned in Section 4.4.1.

In what follows, we first assess our selection algorithm in terms of the overall rate distortion results, that is, the bit rate vs. data distortion (assessed via PSNR). Next we compare the overall rate distortion of our compressor with that of other state-of-the-art compressors, and we investigate the satisfactory level of data distortion by visualizing the decompressed data compared with the original raw data in high resolution. We then report on a series of parallel experiments to evaluate the I/O performance gain of our lossy compressor against other state-of-the-art compressors.

Assessment of Our Bestfit Prediction Selection Algorithm

In this subsubsection, we analyze the effectiveness of our best-fit prediction selection algorithm proposed in Section 3.4.3. We present the rate distortion curves of the three solutions based on our improved data-fitting prediction, our improved transform-based prediction, and our best-fit selection algorithm, respectively, in Figs. 3.17, 3.18, and 3.19. We demonstrate three fields for each application because of the limited space; other fields lead to the similar results. Based on the definitions of PSNR and bit rate given in Section 4.4.1, the more left a curve is located, the higher its overall compression quality is. We can see that our selection algorithm can always select the best-fit predictor in the given bit rate range, such that the rate distortion result is optimized for any point in this range. In

Fig. 3.17(a), for instance, our solution selects the data-fitting predictor when the bit-rate is lower than 0.1 or higher than 2.5 and the transform-based predictor otherwise, leading to the best rate distortion for all bit rates. On the other hand, our solution is able to select the data-fitting predictor when it is better than the transform-based predictor, as shown in 3.17 (c). Therefore, our solution can lead to a much better result when all the fields in the dataset are considered. We note that our solution does not fully overlap with the data-fitting predictor because it does not yield the same error bound (due to the selection mechanism) as the data-fitting predictor for one or two points on the switching boundary. They would have the same result, however, when the error bound in the selector is restricted to the same value.

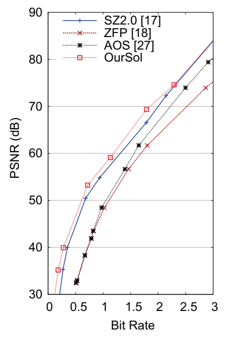
Rate Distortion of Our Solution versus State-of-the-Art Methods

We present the rate distortion curves (bit rate vs. PSNR) for the three applications in Figs. 3.20, 3.21, and 3.22. In each figure, we present the overall result (with the same error bounds for all fields) in the first subfigure and demonstrate the results for two specific fields in the other two subfigures/SC19. One can clearly see that our solution always outperforms other compressors in compression quality. The key reason is that our new encoding scheme for the transform-based predictor is more efficient than the traditional embedded encoding. Also, it can automatically switch to the data-fitting predictor when the data-fitting predictor performs well as our sampling strategy can effectively select the best-fit one. Note that AOS is worse than SZ2.0 in some cases because it builds on the Lorenzo predictor in SZ1.4 and its mechanism cannot be applied to the hybrid design in SZ2.0 directly. Also note that the overall improvement on all fields for each application appears not as high as the individual gain on each field, because we can only adopt the same error bound to perform the compression for each field, such that the PSNR values across different fields have a high variance, leading to a mitigated overall PSNR. This situation would not happen when people compress the data based on the statistical distortion (PSNR) instead of the same error bound for different fields, which is a usual case in the real world.

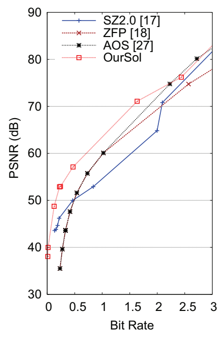
Parallel Performance

Before performing the evaluation of parallel I/O performance with different lossy compressors, we need to determine the acceptable level of data distortion. To this end, we plot the data using a visualization tool for each field involved in our experiment and select the distortion level (PSNR) with a high visual quality for each field. We demonstrate two examples in Figure 3.23 and Figure 3.24, respectively. In the NYX simulation, for instance, the reconstructed data of the velocity_x field has a fairly high visual quality (even visualizing it with a higher resolution) when the PSNR is about 68, whereas its visual quality decreases with lower PSNR. By comparison, for the QS field in the SCALE-LETKF simulation, PSNR=46 is enough for getting a good visual quality (see Fig. 3.24), whereas PSNR=32 causes a noticeable distortion with high resolution. In our experiments, we tune the PSNR to the same value for different lossy compressors. This value is set to $60 \sim 70$ for normal data (e.g. NYX velocity_x) and $40 \sim 50$ for the logarithmic data (e.g. SCALE-LETKF QS).

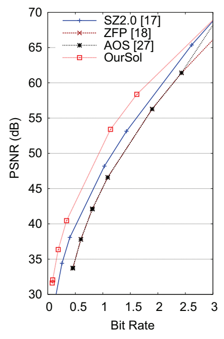
[1]exact number in the brackets



(a) Overall

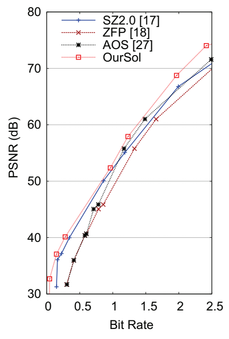


(b) Uf48

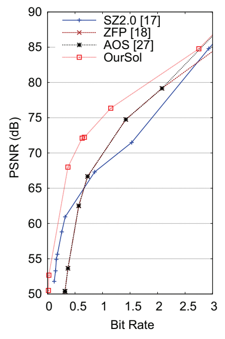


(c) PRECIPf48

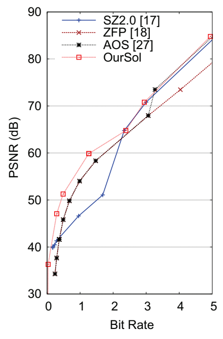
Figure 3.20: Rate Distortion in Hurricane Isabel Simulation



(a) Overall

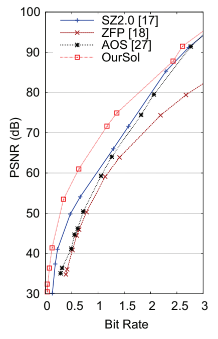


(b) velocity_x

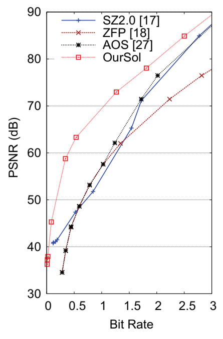


(c) baryon_density

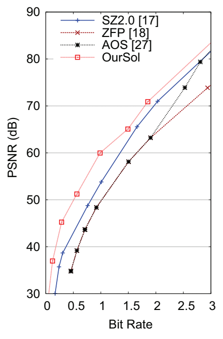
Figure 3.21: Rate Distortion in NYX Simulation



(a) Overall

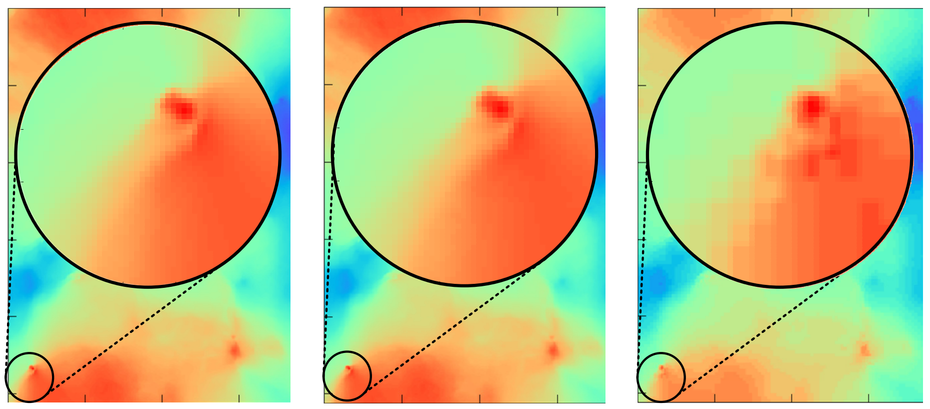


(b) RH



(c) QR

Figure 3.22: Rate Distortion in SCALE-LETKF Simulation



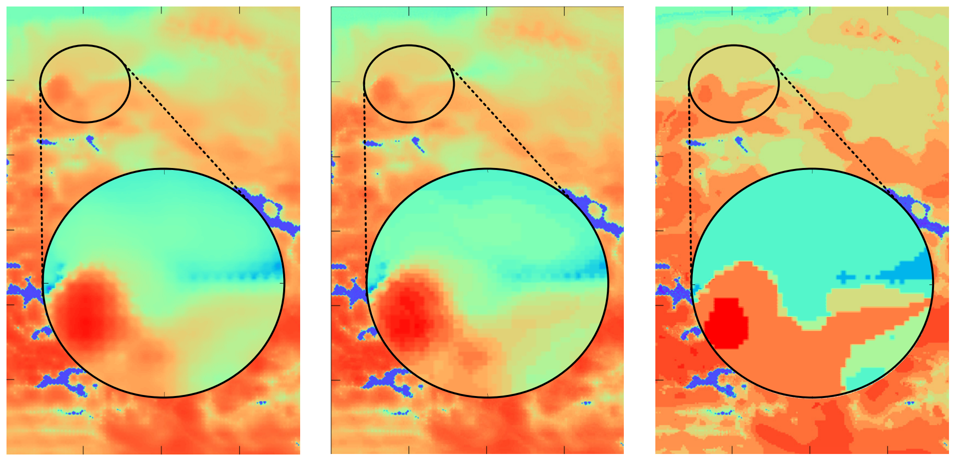
(a) original raw data (b) dec_data (PSNR=68) (c) dec_data (PSNR=52)

Figure 3.23: Visualization of NYX (velocity_x) by Comparing Raw Data and Reconstructed Data with Different PSNRs

Table 3.5: Compression Ratios (Raw Size over Compressed Size) and Memory Overhead

	SZ2.0	ZFP0.5.4	AOS	OurSol	OurSol_b
Hurricane Isabel	36.83	29.46 ($\ll$1%)	30.04	54.5	51.69 (1.1%)
NYX	53.78	46.51 ($\ll$1%)	46.51	103.7	96.92 (1.6%)
SCALE-LETKF	51.72	41.26($\ll$1%)	43.82	109.5	102.1 (1.1%)
Memory Overhead	100%	Constant[1]	100%	100%	Constant[1]

Table 3.5 presents the compression ratios when compressing the three application datasets by four compressors with the same data distortion (PSNR). The table shows that our solution leads to the highest compression ratio in all cases. In absolute terms, its compression ratio is higher than that of the others by 48∼85%, 93∼123%, and 112∼165%, respectively. We also show the memory overhead of these different approaches. Since SZ requires extra space to store the frequencies in the Huffman tree, it incurs 100% memory overhead. On the other hand, ZFP only has constant memory overhead since it processes the



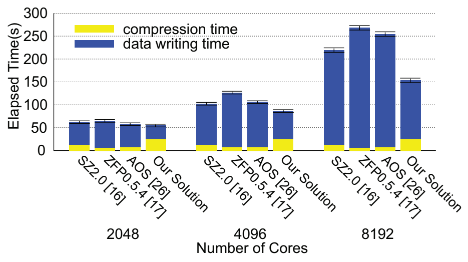
(a) original raw data (b) dec_data (PSNR=46) (c) dec_data (PSNR=32)

Figure 3.24: Visualization of SCALE-LETKF (QS) by Comparing Raw Data and Reconstructed Data with Different PSNRs

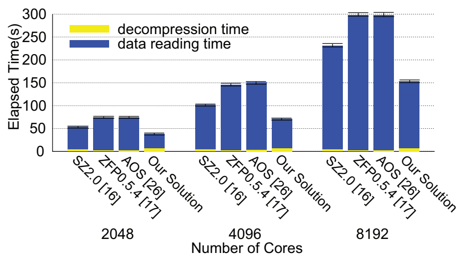
data block by block. AOS follows either SZ or ZFP, so it has 100% memory overhead in the worst case. Our vanilla solution adopts the SZ prediction scheme as it is, leading to 100% overhead. In fact, the memory overhead can be significantly reduced by dividing the data into small blocks and performing the compression block by block, such that the memory overhead can be limited to the block size (constant). We show the result of this blockwise optimization with 128x128x128 block division (64x64x64 for Hurricane to keep around 1% memory overhead) in the last column "OurSol_b". It has constant memory overhead with slightly lower compression ratio (less than 7%) due to the overhead of the Huffman tree in each block. But it still keeps the dominance in compression ratios, compared with other existing lossy compressors.

As discussed in Section 4.4.1, the data dumping/loading performance with lossy compression includes both I/O performance of the parallel file system and compression/decompression performance of the compressor. For large-scale executions when I/O is saturated, however, the compression ratio will dominate the overall data dumping/loading performance for weak-scaling problems, which is often the case for scientific simulations. The reason is that the I/O throughput will remain constant while the total I/O size will increase linearly with the scale. On the other hand, the compression/decompression time will not change because the workload on each rank/node will remain the same even when the scale increases because of limited memory capacity per rank/node (i.e., weak-scaling case). Therefore, the compression ratio will always be the dominant factor for the total I/O performance of current parallel systems when the scale becomes large, as confirmed in our following experiments.

We present the overall data-dumping time and data-loading time for the three applications (Hurricane Isabel climate simulation, NYX cosmological simulation, and SCALE-LETKF weather simulation), in Figs. 3.25, 3.26, and 3.27, respectively. Specifically, we launched 2,048 ranks ($\sim$ 8,192 ranks) to dump and load the total data in parallel. The original raw data writing times would be over 2 hours, 5 hours, and 10 hours, respectively, because of the extremely large data size (10 TB$\sim$48 TB) to process. We ran each case five times and record the compression/decompression time and writing/reading time, as well as the error bars of the total time. The three figures/SC19 clearly show that the solution with our new compressor improves the total I/O performance by 60X than the original I/O performance without any compressor. Moreover, our solution outperforms the other three solutions significantly, especially when the execution scale reaches 8,192 cores. The performance gains over other solutions are approaching the compression ratios with execution scale (or total data size), as is consistent with our analysis. For instance, for the data-dumping performance of the SCALE-LETKF application, our solution outperforms

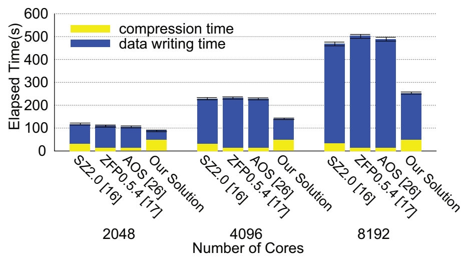


(a) Data dumping performance

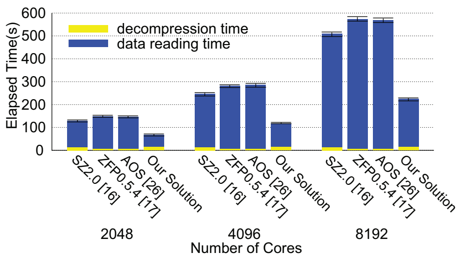


(b) Data loading performance

Figure 3.25: Parallel Performance on Hurricane Isabel Simulation

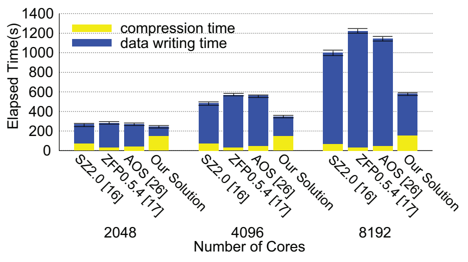


(a) Data dumping performance

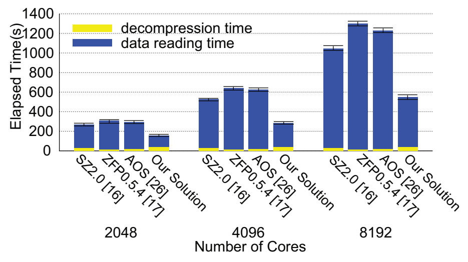


(b) Data loading performance

Figure 3.26: Parallel Performance on NYX Simulation



(a) Data dumping performance



(b) Data loading performance

Figure 3.27: Parallel Performance on SCALE-LETKF Simulation

others by 9.6∼15%, 39.1∼64%, and 72.5∼110%, respectively, which gets closer and closer to the gains on the compression ratios (112%∼165%). For the NYX simulation, our solution outperforms the other solutions by 17.8∼30% in dumping 6 TB of data by 2,048 cores, and it increases up to 85.3∼94% for dumping 24 TB data by 8,192 cores. Moreover, our solution may get further I/O performance gains over other solutions at larger execution scales when I/O is saturated, since the compression time is constant whereas the I/O increases nearly linearly with scale.

3.6 Summary

In this work, we seek to optimize the rate distortion of error-bounded lossy compression by adopting an adaptive, hybrid prediction framework that combines the prediction-based compression model and transform-based compression model. We improve the coefficient encoding efficiency for both models and also optimize the estimation accuracy to select the best-fit predictor. We perform our evaluation using large real-world simulation datasets across different scientific domains on a supercomputer with up to 8,192 cores. Experiments demonstrate that our solution leads to higher compression ratios compared with other lossy compressors, leading to significantly improved I/O performance.

Chapter 4

Enhancing Lossy Compression Efficiency for Relative Error Bound and I/O Performance

4.1 Introduction

Today's scientific research is in urgent need of efficient error-bounded lossy compressors due to the extremely large volumes of data produced in high performance computing (HPC) simulations and applications. Although the previous chapter proposes several algorithms to improve the efficiency of error-bounded lossy compression, these are still discrepancies between the compressors and the demands from domain scientists. On the one hand, the proposed approaches either target PSNR or absolute error bound, whereas a bunch of scientific applications [43, 73] require relative error bound at least for some of the data fields. On the other hand, these algorithms focus on how to maximize the compression ratios given the error bound or distortion instead of how to get the best I/O performance. In this chapter, we tackle these two problems by leveraging an efficient preconditioner for applications demanding relative error bound and an adaptive compression framework targeting I/O performance. Our contributions are summarized as follows.

- We formalize the transformation problem between absolute error bound and relative error bound in the context of lossy data compression mathematically. We also solve this problem and find the unique mapping function, which is consistent with using logarithmic change for error measurement.

- We investigate the impact of the selection of different logarithmic bases on the compression quality for SZ and ZFP, respectively. We prove that various bases lead to the similar compression results theoretically.

- We propose an efficient pointwise relative-error-bounded lossy compression algorithm by combining the logarithmic data transform scheme and state-of-the-art absolute-error-bounded compressors. Specifically, we integrate the transformation scheme into both SZ and ZFP.

- We propose an optimized data dumping performance model that can effectively represent the writing performance with different execution scales and data sizes.

- We analyze our proposed performance model in terms of the lossy compression technique, which is a fundamental guideline to develop an efficient algorithm for optimizing the data dumping performance.

- We develop an adaptive lossy compression framework with a series of optimization strategies to improve the dumping performance for the scientific simulations with error-bounded lossy compressors. The optimized framework has two critical steps: compression quality estimation and online optimization of compression settings.

4.2 Background

4.2.1 Lossy Compression with Relative Error Bound

In comparison with the absolute error bound that has been widely used to control the data distortion by existing state of the art lossy compressors [67, 92], pointwise relative error bound is significant for many scientific applications. Unlike absolute-error-bounded compression (i.e., the difference between each decompressed data value and its corresponding original value must be bounded within a constant number), pointwise relative-error-bounded compression means that the compression error of each data point should be bounded within a constant percentage of its data value. That is, the smaller the data value, the lower the absolute error bound to be applied on the data point. Obviously, the pointwise relative error bound can preserve more details in regions with small values. On the other hand, some application users demand pointwise relative error bound based on the physical meaning of the simulation. According to cosmologists (such as the users and developers of HACC and NYX), for example, the higher a particle's velocity is, the larger the compression error it can tolerate.

Pointwise relative error bound is much tougher to deal with than absolute error bound according to the principles of lossy compressions. SZ [32,33,92], for example, predicts the value for each data point based on the consecutiveness of the dataset and control the compression errors by a linear-scaling quantization method. Since the bin size is fixed for each data point, the compression errors can be bounded within a constant value (i.e., absolute error bound). However, such a design cannot adapt to the demand of pointwise relative error bound, which requires diverse error controls on different data points. In order to enable SZ to support point-wise relative error bound, a blockwise strategy was proposed [33]: the entire data is split into multiple non-overlapped data blocks each adopting an absolute-error-bounded compression where the error bound is calculated by using the minimal value in the block. This strategy may significantly degrade the compression ratio, especially when the minimal values in some blocks are far smaller than other data values in those blocks, because the absolute error bounds would be too small to get a high compression ratio. The developers of ZFP [67] proposed an approximate compression mode (called *precision*) to achieve a similar effect with pointwise relative error bound. However, it cannot fully respect the pointwise relative error bound (to be presented later) because the data transformation and embedded encoding adopted in the compression are designed for the optimization of rate distortion, that is, peak signal-to-noise ratio (PSNR) vs. bit-rate (the number of bits used to represent one data point on average).

4.2.2 Lossy Compression for I/O Performance

Improving the overall I/O performance of lossy compression is important in scientific simulations because the prohibitive I/O time is the main reason for preventing scientific simulations from scaling up. The goal is to provide users with an error-bounded lossy compressor of possibly optimal output throughput. The key challenge that we address is how to optimize the trade-off between compression overhead, compression ratio when the quality of the data is guaranteed. This trade-off poses difficult challenges because of the antagonistic effects of these aspects: better compression ratio typically leads to higher compression overhead. Therefore, in order to optimize the I/O performance, we need to find the sweet spot that can minimize the sum between the compression overhead and the write overhead to the PFS without violating the desired quality properties. While compression can be applied in an embarrassingly parallel fashion, writes to the PFS are subject to a limited global I/O bandwidth. Therefore, it is non-trivial to find such a sweet spot. To solve this problem, we introduce an novel adaptive approach that relies on I/O performance modeling to characterize the write overhead to the PFS under variable number of ranks and data size per rank. Then, using this performance model and data sampling, we propose a strategy to choose an optimal lossy compression algorithm and fine-tune it such that it has the lowest dumping time.

4.3 An Efficient Transformation Scheme for Relative Error Bound

In this part, we propose to leverage the logarithmic transformation to change a relative-error-bound compression problem to an absolute-error-bound compression problem, which coud serve as a preconditioner for scenarios demanding relative error bound.

4.3.1 Mathematical Foundation of the Transformation Scheme

In this section, we leverage a specialized data transformation scheme that can convert the pointwise relative-error-bounded lossy compression problem to an absolute-error-bounded lossy compression problem. That is, under our designed mapping scheme, any absolute-error-bounded lossy compressor can be enabled to support pointwise relative error bound such that more details can be preserved in the regions with small data values. In what follows, we first derive an efficient data mapping/transformation scheme and then prove that the mapping method is the unique solution to the data conversion between the absolute error bound and pointwise relative error bound. We also analyze how to adjust the absolute error bound for the lossy compression to strictly respect the pointwise relative error bound, considering the impact of the possible round-off errors raised during the mapping.

Theories of the Transformation Scheme

The research problem can be formulated as follows: find a bijective and continuous function (denoted by f) that can map the original dataset (denoted by x) to another domain $f(x)$ such that the pointwise relative error bound (denoted by b_r) can be mapped to an absolute error bound (denoted by b_a) using another mapping function g (i.e., $b_a = g(b_r)$). Note that the mapping function (g) used to convert the error bound is different from the mapping function (f) used to transform the data, which will be discussed later in more detail. Moreover, the mapping function f should be bijective because we need to map the data reconstructed by the absolute-error-bounded compressor back to the original data domain (i.e., x=$f^{-1}(f(x))$) during the decompression. It also needs to be continuous because otherwise the mapping function may affect the continuity of the original data, degrading the data prediction accuracy in turn. With such a continuous bijective mapping function, the original data can be mapped to another domain and then compressed by the corresponding absolute error bound b_a. In the decompression phase, the data will be mapped back to the original domain via the corresponding inverse function, and the pointwise relative error bound will automatically hold. We derive Theorem 3 in order to search for the most effective mapping function.

Theorem 3. *Given a dataset whose data values are denoted by x, Equation (4.1) is a sufficient condition of transforming it to another data domain by a mapping function f, such that if the transformed data are compressed with an absolute error bound $g(b_r)$, the corresponding inverse mapping of the decompressed transformed data will always be bounded by the pointwise relative error bound b_r in the original domain.*

$$\frac{f^{-1}(f(x)+g(b_r))-x}{x}=b_r \tag{4.1}$$

Proof. The pointwise relative-error-bounded compression regarding the mapping function can be formalized as

$$\frac{|f^{-1}(f(x)+\epsilon)-x|}{|x|}\leq b_r$$

where $\epsilon\in[-g(b_r),\, g(b_r)]$ refers to compression error, f is the target forward-mapping function to be applied before the compression, and f^{-1} refers to the inverse function to be applied after the decompression.

Since f and f^{-1} are bijective and continuous, they must be strictly monotonic. Without loss of generality, we denote x as a positive value, and $f(x)$ is a monotonically increasing function (in fact, if $x < 0$, it can be mapped to $-x$ first and then we can derive the same result by the following derivation). Since f is a monotonic function and $x > 0$, we have $f^{-1}(f(x) + \epsilon)$ must be always in interval $[f^{-1}(f(x) - g(b_r)),\ f^{-1}(f(x) + g(b_r))]$.

In order to reach the maximum compression ratio, the absolute compression error ϵ and the pointwise relative compression error are expected to be equal to their bounds (i.e., $g(b_r)$ and b_r) at the same time. This leads to $\frac{f^{-1}(f(x)+g(b_r))-x}{x} = b_r$ and $\frac{f^{-1}(f(x)-g(b_r))-x}{x} = -b_r$. Although g is defined only in $[0, +\infty)$ because of the pointwise relative error bound $b_r \geq 0$, we can merge the two equations to Formula (3) if we introduce a definition $g(x) = -g(-x)(\forall x < 0)$ without loss of generality. $\square$

This theorem indicates that the pointwise relative error bound can be transformed to an absolute error bound as long as Equation (3) holds for the data-mapping function f and the error-bound-mapping function g. In what follows, we derive a theorem (Theorem (5)) to find the corresponding functions.

Before proposing the theorem, we recall a critical lemma based on the theory of functional equations.

Lemma 4. *The exponential function is the only continuous and nonconstant function that satisfies $f(x + y) = f(x)f(y)$, where x and y are both real numbers.*

Theorem 5. $f(x) = \log_{base} x + \mathrm{C}$ *is the unique mapping function that satisfies the Equation (4.1), where* $\mathrm{base} > 1$ *and* $\mathrm{C} = f(1) \in R$*. In this situation, the mapping function g of the absolute error bound is $b_a = g(b_r) = \log_{base}(1 + b_r)$.*

Proof. We rewrite Equation (4.1) and apply f to both sides

$$f(x) + g(b_r) = f((1 + b_r)x).$$

Let us set $x = 1$. Then the function g can be represented as $g(b_r) = f(1 + b_r) - f(1)$. Let $y = 1 + b_r$, and substitute g in the above equation. Then we have

$$f(x) + f(y) - f(1) = f(xy)$$

Let $h(x) = f(x) - f(1)$, then we can derive $h^{-1}(x) = f^{-1}(x + f(1))$. According to the above equation, we can get:

$$h(x) + h(y) = f(x) + f(y) - 2f(1) = f(xy) - f(1) = h(xy).$$

Applying h^{-1} to the left-most and right-most sides of the above equation, we get the following equation:

$$h^{-1}(h(x) + h(y)) = xy = h^{-1}(h(x))h^{-1}(h(y)).$$

Let $x' = h(x)$ and $y' = h(y)$. The equation turns out to be $h^{-1}(x' + y') = h^{-1}(x')h^{-1}(y')$. Denoting $h^{-1}(1) = base$ leads to $h^{-1}(x) = base^x$ according to Lemma 4. Thus, $h(x) = \log_{base} x$. Let $f(1) = C$. We have $f(x) = h(x) + f(1) = \log_{base} x + C$. Accordingly, $b_a = g(b_r) = f(1 + b_r) - f(1) = \log_{base}(1 + b_r)$. □

Without loss of generality, C can be set to 0 because of two factors. On the one hand, C just adds a static shift to the mapped data, which means that it has no effect on the prediction accuracy of the Lorenzo predictor. On the other hand, more perturbations may be introduced to the result because of round-off errors if C is nonzero. Therefore we fix $C = 0$ in our implementation.

As previously mentioned, the logarithmic function is used for relative error measurement in some scientific domains such as economics and ocean observations. The theory presented in this paper, in addition, further indicates that the logarithmic function family is the unique mapping function for transformation between relative error and absolute error in the context of lossy data compression.

Round-off Error Considerations

Because of the inexact representation of floating-point arithmetic, we cannot produce exact calculation results. Thus, there is also round-off error while applying the mapping function f and f^{-1}. When this error is taken into consideration, the pointwise relative error bound may no longer be respected. In this subsection we analyze how to control this error.

Now that we have our mapping function $f(x) = \log_a x$ and its reverse function $f^{-1}(x) = a^x$. To find the error bound in the logarithmic domain such that the original point-wise error bound can be respected, we have the following lemma.

Lemma 6. *The absolute error derived in Theorem 5 should be adjusted to* $b'_a = log_a(1 + b_r) - \max_x |\log_a x|\varepsilon_0$ *to respect the pointwise relative error bound considering round-off errors, where* $\max_x |\log_a x|$ *is the maximum absolute value of the mapped data* $\log_a x$ *and* ε_0 *is the round-off error introduced to* $f(x)$ *because of machine precision.*

Proof. For any data point x, its decompressed value x_d will be

$$x_d = f^{-1}(f(x)(1 + \varepsilon_0) + \epsilon) = a^{(1+\varepsilon_0)\log_a x+\epsilon} = xa^{\varepsilon_0 \log_a x+\epsilon}$$

where $\epsilon \in [-b'_a, b'_a]$ is the compression error of data in the transformed domain (i.e., logarithmic data domain). Similar to the analysis above, in order to respect the error bound, the absolute error bound b'_a should be set as follows: $a^{\varepsilon_0 \log_a x+b'_a} \leq 1 + b_r$ for any data point x. Therefore, we have $b'_a = g'(b_r) = \log_a(1 + b_r) - \max_x |log_a x|\varepsilon_0$. □

We set ε_0 to machine epsilon in our implementation. This setting can already have all of the data points strictly bounded within the specified error bounds during the compression in our evaluation, as will be presented later.

4.3.2 Impact of Base Selection

In this section, we investigate the impact of base selection for the logarithmic mapping solution in Theorem 5. Specifically, we prove that different logarithmic bases lead to similar compression results; therefore, simply changing the logarithmic base cannot improve the compression quality.

Impact of Base Selection on SZ

Review of SZ. To understand the impact of base selection on SZ, we need to review the principle of the SZ compressor. SZ [92] is a prediction-based lossy compressor. It consists of three stages during the compression. In the stage I, it predicts each data value in the dataset according to some deterministic prediction model. Then, it applies a linear-scaling quantization on the prediction errors such that all the floating-point values could be converted to a set of integer quantization codes. In stage II, SZ adopts a customized Huffman encoder constructed in terms of the quantization codes and performs the compression. Stage III applies GZIP to the encoded data to further improve the compression ratio. This step is optional depending on how hard the data is to be compressed and the error bound specified. During the decompression, SZ first decompresses the data by GZIP and the Huffman encoder and then recovers the data by the prediction model with the decoded prediction errors. For the data prediction, SZ adopts the Lorenzo predictor [47] with a limited number of neighbors by default since more neighbors may cause degraded prediction accuracy because of the impact of the decompressed values.

Logarithmic Base Analysis. We notice that the selection of different bases for the log mapping function does not affect clearly the compression quality of SZ. In absolute terms, we have the following lemma.

Lemma 7. *For SZ lossy compressor with Lorenzo predictor and linear-scaling quantization, different bases will lead to the same prediction accuracy, if the arithmetic operations lead to the exact results.*

Proof. Suppose a mapping function $f(x) = \log_a x$ is used to compress the same 1D dataset. Then the quantization index i should satisfy Equation (4.2), when predicting the next point x_1 by the previous data point x_0.

$$\log_a x_1 = \log_a x_0 + q \log_a (1 + b_r) \tag{4.2}$$

Thus, q can be solved as follows.

$$q = \frac{\log_a x_1 - \log_a x_0}{\log_a (1 + b_r)} = \frac{\log_a \frac{x_1}{x_0}}{\log_a (1 + b_r)} = \log_{1+b_r} \frac{x_1}{x_0}$$

Therefore, the quantization index is independent of the mapping base a if the arithmetic operations lead to exact results. This also holds for 2D and 3D Lorenzo predictors since the quantization index can be computed as follows.

$$q_{2D} = \frac{\log_a x_{01} + \log_a x_{10} - \log_a x_{00} - \log_a x_{11}}{\log_a (1 + b_r)} = \log_{1+b_r} \frac{x_{01} x_{10}}{x_{00} x_{11}}$$

$$\begin{aligned} q_{3D} = & \frac{\log_a x_{001} + \log_a x_{010} + \log_a x_{100} + \log_a x_{111}}{\log_a (1+b_r)} \\ & - \frac{\log_a x_{000} + \log_a x_{011} + \log_a x_{101} + \log_a x_{110}}{\log_a (1+b_r)} \\ & = \log_{1+b_r} \frac{x_{001}x_{010}x_{100}x_{111}}{x_{000}x_{011}x_{101}x_{110}} \end{aligned}$$

□

However, the floating-point arithmetic operations may cause nonexact results because of round-off errors, introducing certain deviation to the distribution of quantization index codes. Fortunately, we can derive a strict bound for the quantization index with different bases as follows.

Theorem 8. *Given two coding integer values (q_0 and q_1) at ith quantization index bin based on two different bases, the difference in between is bounded by $|\log_{1+b_r}(1-b_r)-1|$, $3|\log_{1+b_r}(1-b_r)-1|$, $7|\log_{1+b_r}(1-b_r)-1|$ for the 1D, 2D, and 3D Lorenzo predictions, respectively.*

Proof. Let us start with the 1D case. Without loss of generality, assume that the decompressed data of two bases are x_{1d} and $x_{1d'}$. Then the difference of their quantization indices q_0 and q_1 can be derived according to Lemma 7.

$$q_0 - q_1 = \log_{1+b_r} \frac{x_1}{x_{0d}} - \log_{1+b_r} \frac{x_1}{x_{0d'}} = \log_{1+b_r} \frac{x_{0d'}}{x_{0d}}$$

Since $(1-b_r)x_0 \le x_{0d} \le (1+b_r)x_0$ and $(1-b_r)x_0 \le x_{0d'} \le (1+b_r)x_0$, we have

$$\log_{1+b_r} \frac{(1-b_r)x_0}{(1+b_r)x_0} \le q_0 - q_1 \le \log_{1+b_r} \frac{(1+b_r)x_0}{(1-b_r)x_0}.$$

It can be simplified to

$$\log_{1+b_r}(1-b_r) - 1 \le q_0 - q_1 \le 1 - \log_{1+b_r}(1-b_r).$$

As for the 2D and 3D Lorenzo predictions, the difference involves 3 and 7 multiplication of decompressed data. Thus they are bounded by

$$\log_{1+b_r}(\frac{1-b_r}{1+b_r})^3 \le q_0 - q_1 \le \log_{1+b_r}(\frac{1+b_r}{1-b_r})^3$$

$$\log_{1+b_r}(\frac{1-b_r}{1+b_r})^7 \le q_0 - q_1 \le \log_{1+b_r}(\frac{1+b_r}{1-b_r})^7,$$

which corresponds to the given bounds in the theorem. □

Impact of Base Selection on ZFP

Review of ZFP. Similarly, we first go over how ZFP works as an error-bounded lossy compressor and the metrics to assess the effectiveness of its orthogonal transform. ZFP [67] adopts a largely different method to compress a floating-point dataset compared with SZ. Briefly, it involves two critical steps: an invertible blockwise transform on the input data and an embedded encoding for the transformed coefficients. Specifically, it divides the whole dataset into independent blocks, aligns the exponents, and turns the floating-point representations into fixed-point representations. Then, ZFP applies a data transform in each block. Finally, it performs an embedded encoding on the transformed data (or transformed coefficients). Such a design obtains an optimized rate distortion (i.e., PSNR vs. bit-rate), although it may not maximize the compression ratio given an error bound because it conservatively overestimates the errors in the maximum bit-plane computation for the purpose of strictly respecting the error bound.

The transform is the most critical part in ZFP compression, and it is the key factor of the final compression quality. Its effectiveness can be determined by two metrics, *decorrelation efficiency* and *coding gain*, as introduced in [26,67]. Therefore, we analyze the impact of the logarithmic base selection according to the two metrics, defined as follows.

Definition 9. *Considering that each entry in a block is a random variable, the decorrelation efficiency η and coding gain γ can be defined by*

$$\eta = \frac{\sum_i \sigma_{ii}^2}{\sum_i \sum_j \sigma_{ij}^2} \quad \gamma = \frac{\sum_i \sigma_{ii}^2}{n(\prod_i \sigma_{ii}^2)^{\frac{1}{n}}}, \tag{4.3}$$

where σ_{ij} is the element in the ith row and jth column of the covariance matrix of transformed coefficients and n is the number of elements in a block.

Logarithmic Base Anaylsis. With these two metrics, we can analyze the impact of different logarithmic bases on the quality of ZFP compression with the following lemma.

Lemma 10. *Decorrelation efficiency η and coding gain γ will be unchanged across different logarithmic bases.*

Proof. For simplicity, we only discuss 1D cases. The multi-dimensional cases can be deducted accordingly. We use A to denote any generic transform matrix.

Denote $X = (X_1, \ldots, X_n)^T$ as random variables of origin data in each block, $V = (V_1, \ldots, V_n)^T$ as the random variables of coefficients. The data will turn out to be $Y = \log_a X = \frac{\ln X}{\ln a}$ after the logarithmic transformation. Correspondingly, the coefficients will be $V = AY$. More specifically, let us denote $A = (A_1, \ldots, A_n)^T$, where A_i is the ith row vector of A. Then the covariance between any of two variables can be computed by

$$\begin{aligned}\sigma_{ij} &= cov(V_i, V_j) = E(V_i V_j) - E(V_i)E(V_j) \\ &= E(Y^T A_i^T A_j Y) - E(Y) A_i^T A_j^T E(Y) \\ &= E((\frac{\ln X}{\ln a})^T A_i^T A_j \frac{\ln X}{\ln a}) - E((\frac{\ln X}{\ln a})^T) A_i^T A_j E(\frac{\ln X}{\ln a}) \\ &= \frac{1}{(\ln a)^2}[E((\ln X)^T A_i^T A_j \ln X) - E(\ln X)^T A_i^T A_j E(\ln X)].\end{aligned}$$

Thus, the logarithmic base a serves only as a multiplicand factor. During the computation of η and γ as shown in Equation (4.3), $\frac{1}{(\ln a)^2}$ can be extracted as a common factor of both numerator and denominator and thus canceled. Therefore, the base selection will not affect the decorrelation efficiency and coding gain of ZFP. $\square$

As such, we have proved that the ZFP transform will work equally well on different logarithmic bases. In fact, different logarithmic bases just apply different multiplicands to the transformed coefficients. Under these circumstances, the compression quality of ZFP will hardly change, as will be validated in the evaluation section.

4.3.3 Implementation

In this section, we discuss how to implement the pointwise relative error bound by combining the logarithmic data transform scheme and the existing state-of-the-art absolute-error-bounded lossy compressors.

The pseudo-code of the logarithmic-mapping-based lossy compression in terms of pointwise relative error bound is presented as Algorithm 6.

In the algorithm, we first calculate the required absolute error bound (denoted by b'_a) based on the given pointwise relative error bound b_r, with a consideration of the possible round-off errors to be introduced during the mapping operation due to the machine epsilon (line 1), according to Lemma 6. Then, the algorithm performs the data transformation based on the logarithmic function (line 2$\sim$8). If the original data point's value is equal to 0, we will map it to the lower-bound exponent of the floating-point data value range minus $2b'_a$. Specifically, the minimal positive values of a single-precision number and a

Algorithm 6 LOGARITHMIC-MAPPING BASED LOSSY COMPRESSION FOR POINT-WISE RELATIVE ERROR BOUND

Input: a dataset (denoted by D), user-specified point-wise relative error bound b_r

Output: compressed data stream in form of bytes

```
1: $b'_a = log_a(1 + b_r) - \max_x |\log_a x|\varepsilon_0$; /*Calculate the absolute error bound in the transformed data domain*/
2: signs[$\|D\|$] = {0}, $P = 1$
3: for (each data point $D_i$ in the dataset $D$) do
4:     if ($D_i == 0$) then
5:         $d_i = \log_2(\text{min}) - 2b'_a$, where min refers to the minimum floating-point value; /*set $d_i$ to be two error bounds less than the lower-bound exponent of the data value range*/
6:     else
7:         if ($D_i > 0$) then
8:             Compute $d_i = \log_2(D_i)$; /*Perform the data mapping*/
9:         else
10:            Compute $d_i = \log_2(-D_i)$;
11:            $P = 0$, sign[i] = 1;
12:        end if
13:    end if
14: end for
15: if ($P$==0) then
16:     compress signs with gzip
17: end if
18: Perform the compression of the transformed dataset $\{d_i\}$ by a lossy compressor (such as SZ or ZFP) using the absolute error bound $b'_a$;
19: Output the compressed data stream in bytes;
```

double-precision number are 2^{-2^7}=2^{-127} and $2^{-2^{11}}$=2^{-1024}, respectively, because their IEEE 754 representations [51] adopt 1+7 bits and 1+11 bits to represent a signed exponent, respectively. In the decompression phase, the data values decompressed by the absolute-error-bounded compressor in the transformed domain will be back-transformed to 0, as long as their reconstructed values are lower than or equal to the minimal positive values minus b'_a. Such a design can ensure that almost all the zero-value data points will be decompressed to an exact zero number, unlike the SZ 1.4 that may reconstruct the zeros to be close-to-zero numbers approximately. After the data transformation step (line 3∼8), the algorithm will perform the absolute-error-bounded lossy compression over the transformed dataset, by leveraging an existing compressor such as SZ and ZFP (line 19).

4.4 Improving Data Dumping via Adaptive Compressor Selection

In this part, we discuss how to improve the overall data dumping performance with system information and an adaptive compression framework.

4.4.1 I/O Model with Compression

Our target is to optimize the data dumping performance by lossy compression such that the simulation data can be dumped to a PFS as fast as possible, while still guaranteeing that the data is valid for user's post-analysis. This contrasts other research with multi-objectives regarding lossy compression [67, 69, 70].

Adding compression to the I/O performance model makes the problem a trade-off, since the compression introduces extra time on compression while saving cerntain time on data writing because of reduced data size. In the following text, we extend the I/O model by taking the impact of error-bounded lossy compression into consideration.

The error-bounded lossy compressor allows users to compress the data based on a specific error setting (absolute error bound, relative error bound, or PSNR), and its impact on the original I/O model can be formulated by some functions of the given error setting. In our analysis, we focus on absolute error bound (denoted as eb) because it has been widely used in the community. In general, compression ratio, compression rate and decompression rate are determined by the absolute error bound. Suppose the lossy compressor achieves a compression ratio[1] of $\rho(eb) : 1$ and the minimum compression rate

[1]Compression ratio is the ratio of original data size to compressed size.

is denoted as $f_{comp}(eb)$ GB/s. Since each rank compresses its local data independently, there is no communication cost across ranks (except the negligible Allreduce operation for synchronization which determines the final size), thus the overall data dumping time can be formulated as $t_{dump}(eb) = \frac{MN/\rho(eb)}{f_{write}(M/\rho(eb),N)} + \frac{M}{f_{comp}(eb)}$. The overall data dumping rate with respect to the given error bound eb can be derived as Formula (4.4).

$$Rate_{dump}(eb) = (\frac{MN/\rho(eb)}{f_{write}(M/\rho(eb), N)} + \frac{M}{f_{comp}(eb)})^{-1} \tag{4.4}$$

Based on the above formula, we note that the key factors affecting the compression-based data dumping rate include data writing rate function f_{write}, compression ratio ρ, and compression rate f_{comp}. Even though the execution scale is fixed, there is still a complicated interplay among them. Although the intermediate cases are complicated, we can interpret some extreme cases intuitively. Considering each rank handles the same amount of simulation data, f_{write} would be much larger than f_{comp} if the simulation runs in a relatively small scale, such that a faster compressor would be more favored. On the other hand, if the execution scale is extremely large, f_{write} could be close to the peak performance of the file system (a situation with saturated I/O). In this situation, larger N would increase writing time linearly, in which a compressor featuring higher compression ratio should be considered. Therefore, having a good knowledge of the I/O information is essential to designing an adaptive compressor with the best I/O performance.

In fact, our model is not restricted to only the I/O use-case. It can be extended to any other scenarios associated with compression involving a resource having a performance that can be modeled as a roof-line. For instance, if one wants to leverage compression to accelerate the data transfer between memory and cache, the time t_{dump} could be replaced by the data transferring time. Similarly, if compression is used in data communication, t_{dump} corresponds to the communication time for the message.

4.4.2 I/O Performance Model of Parallel File Systems under Write Concurrency

In this section, we introduce an I/O performance model for parallel file systems to address the concurrent data dumping scenario we explore in this paper. Specifically, lossy compression usually leads to similar compression ratios for similar data, which means all application processes write roughly the same amount of data concurrently. Therefore, the question our performance model aims to answer is what I/O bandwidth can be achieved if N application processes (ranks) need to write M GB of data simultaneously to the parallel file system?

To answer this question, we introduce a methodology to build a performance model that we illustrate using the Theta [90] supercomputer, ranked 24th as of November 2018 in the Top 500 [101] and featuring a 10 PB Lustre parallel file system. The same methodology can be applied for other supercomputers and parallel file system configurations.

As a first step, we analyze the write throughput on Theta under weak scalability (increasing number of writers) and different data size per writer. We assume the application consists of N ranks, each of which needs to dump M GB of data per snapshot. Although the parallel file system has a theoretical peak I/O bandwidth P, it is not enough to simply approximate the total write time as $M \cdot N/P$. This is because the degree of write concurrency and the size per writer can lead significant variability even when the total size $M \cdot N$ remains the same. Furthermore, the behavior of the PFS can become even more complex when its peak bandwidth is not saturated. To account for these factors, we consider the overall write bandwidth as a function of the file size per node (x) and the number of ranks (N): $f(x, N)$ GB/s. In this case, the time dump the whole snapshot onto the PFS can be approximated as $t_w = \frac{MN}{f(M,N)}$.

To obtain f, we perform an I/O performance profiling with different execution scales (i.e., we consider a set of usual values for N). Through experimentation, we observed that f tends to grow linearly with the logarithm of M up to a threshold (below the peak bandwidth P), after which it remains constant. These experimental results are illustrated in Figure 4.1. Such an observation is consistent with the intuitive behavior of parallel file systems, as the peak bandwidth is hardly achievable given overheads and limits that could be set by the system administrator. Therefore, we propose to formulate f using a roof-line model, as shown in Formula (4.5).

$$f(x) = \begin{cases} ax + b & x < x_t \\ c & x \geq x_t \end{cases} \tag{4.5}$$

where x refers to the logarithm of the data size per rank and x_t is the turning point indicating the saturation of the I/O bandwidth. To obtain the coefficients a, b, c, we use least-square fitting of the experimental data (which uses a variable size from 2^1 MB to 2^9 MB and covers the typical file sizes generated by lossy compression). Then, once the coefficients were determined, we can use f to predict the performance for any data size.

Unfortunately, the point of saturation x_t is not the same for all N. Therefore, we assume that x_t is a function of N, which leads to a separate optimization problem for all usual values of N we consider (denoted by i). The process of determining the parameters in the roof-line model can be formulated as Equation (4.6).

$$f_i(x) = \begin{cases} ax + b & x < x_i \\ c & x \geq x_i \end{cases} \tag{4.6}$$

where $\{x_i\}$ and $\{y_i\}$ are referred to the turning point (i.e., the profiling case i) and the corresponding I/O bandwidth, respectively. Specifically, for each execution scale with various data sizes per rank, the least-square optimization problem can be formulated as follows.

We need to identify the coefficients a, b and c that can minimize $\sum_j (f_i(x_j) - y_j)^2$, where i is assumed to be the profiling case number (as shown in Formula (4.7)).

$$\min_{a,b,c} \sum_j (f_i(x_j) - y_j)^2 \tag{4.7}$$

We denote the corresponding least-square optimized coefficients as a_i^*, b_i^*, and c_i^*, respectively. Then, the optimal profiling case number (denoted by i^*) can be determined by the following equation (i.e., the one leading to the least-square).

$$i^* = \arg\min_i \{ \min_{a_i^*, b_i^*, c_i^*} \sum_j (f_i(x_j) - y_j)^2 \}$$

Then, the optimal turning point (denoted by x_t^*) can be calculated as follows:

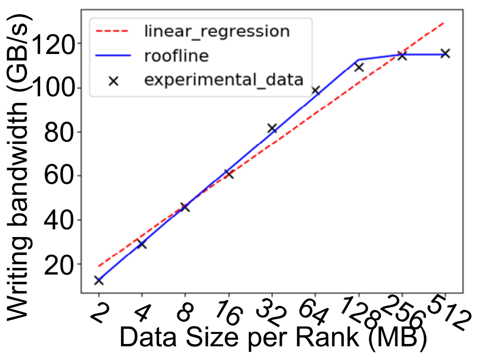
$$x_t^* = \frac{c^* - b^*}{a^*}$$

where a^*, b^* and c^* refer to the corresponding coefficients with the turning point being set to the profiling case i^*. The final optimal I/O model is presented in Formula (4.8)

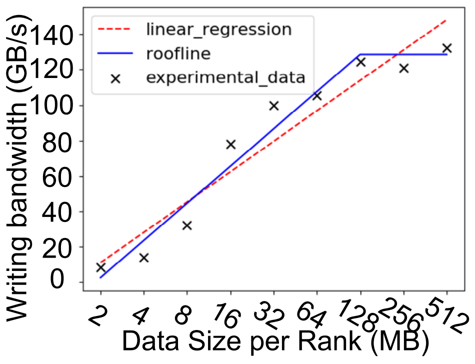
$$f^*(x) = \begin{cases} a^* x + b^* & x < x_t^* \\ c^* & x \geq x_t^* \end{cases} \tag{4.8}$$

There is a specific situation we have to deal particularly with during the above model optimization. In fact, the above calculation might result in a situation that c^* is smaller than $a^* x_t^* + b^*$, implying that the I/O saturated case has lower bandwidth than does the I/O unsaturated simulation with unsaturated I/O, which is a little counter-intuitive.

Hence, we need to do an adjustment to our I/O model to respect the ground-truth. If this counter-intuitive situation occurs, we set c to $a^*x_t^* + b^*$ in the roof-line model and recompute the error and corresponding parameters. This can always give us a valid roof-line model, where the constant part is always no less than the maximal of the linear part. Fig. 4.1 shows the experiments and the resulting roof-line model for the data writing bandwidth on theta with 256, and 512 nodes. The roof-model (blue curve) does provide a better trend compared to vanilla least-square model (red curve) according to this figure.



(a) Write BW (256 nodes)



(b) Write BW (512 nodes)

Figure 4.1: Write/Read bandwidth (BW) on Theta

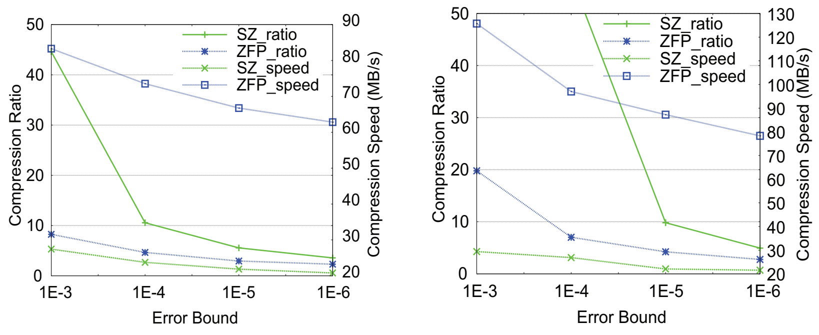
Note that the above-mentioned I/O performance profiling just needs to be done once for each supercomputer and our optimized I/O model based on the profiling results can be reused for all the applications running on the supercomputer. Thus, the I/O performance profiling is not overhead that needs to be paid every time an application is running.

4.4.3 Adaptive Lossy Compression Framework

In this section, we present the design of our adaptive lossy compression framework for getting the best overall parallel data dumping performance. In the following text, we first introduce the design motivation, and then present our solution in details. Because of limited space, we only show the examples done on the Theta supercomputer in this section as it represents the modern supercomputers with fast I/O.

Design Motivation

Our target is to design an adaptive framework that can maximize the data dumping performance by automatically selecting the bestfit compressor, given the basic profiling information of the I/O environment (such as sampled I/O bandwidth based on execution scale and data sizes). In our framework, we use error bound as our metric. Then, we choose two state-of-the-art lossy compressors - SZ and ZFP, because they have been considered the best two error-bounded lossy compressors on different datasets in literature [65, 67]. In general, SZ has higher compression ratios for fixed error bounds while ZFP has higher compression rate, especially because of their different compression principles. SZ contains four critical steps: (1) predict each data value, (2) perform linear-scaling quantization, (3) a customized variable-length encoding, and (4) optional lossless compression such as Zstd [116]. By comparison, ZFP contains five critical steps: (1) align values in each block to a common exponent; (2) convert floating-point values to a fixed-point representation; (3)decorrelation by applying orthogonal transforms; (4) order transform coefficients; and (5) embedded encoding.



(a) velocity_x (b) dark_matter_density

Figure 4.2: Performance of different compressors on two typical fields in NYX dataset on Theta

Figure 4.2 shows the compression ratio and compression speed for two example fields in the NYX dataset. From this figure, we can see that SZ leads to much higher compression ratios given the error bound, especially on some easy-to-compress dataset such as dark_matter_density. The compression ratio could be 10X higher than that of ZFP. However, this does not imply it will always lead to better overall data dumping performance. This is because the compression speed of SZ is only one fourth of that of ZFP, such that the performance gain achieved from data reduction cannot beat over the compression time cost especially when the the data size is small (e.g. in small execution scale or with large compression ratios). As such, selecting the bestfit compressor is critical to the overall data dumping performance. Furthermore, we also perform some optimization steps and fine-tuning on the compressors to achieve higher data dumping performance.

Design Overview

We present the design overview of our adaptive lossy compression framework in Figure 4.3. It has a sampling stage at the very beginning to collect the information for the different compressors. After that, we have a decision process to determine which compressor to use and whether lossless compression should be performed on the compressed data. If the selected compressor is SZ, we perform further optimization regarding the tradeoff between compression ratio and compression speed. Specifically, we re-design the key stages in the SZ compression and offer more options to fine-tune the compressor. We also perform a fast parameter search to get the best parameter settings. We then perform the compression using the selected bestfit compression strategy (either ZFP or fine-tuned SZ). Finally, we get the maximum size of the compressed data and pack all the data segments to the multiple of 64KB to maximize the I/O bandwidth, as required by the Lustre file system on Theta (stripe size) for the best performance, and write the files into the parallel file system.

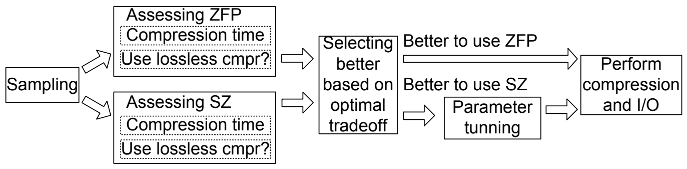


Figure 4.3: Design Overview of Adaptive Lossy Compression Framework

Table 4.1: Variable Definitions

N	Number of ranks used
M	File size on each rank
t_{comp}	Lossy compression time
t_{ll_comp}	Lossless compression time
ρ_l	Lossy compression ratio
ρ_{ll}	Lossless compression ratio
$f_{write}(x)$	Write bandwidth collected from previous section

We list the key notations to be used in Table 4.1. Based on the definition of these notations, we can infer the total dumping performance with/without the compressor as well as with/without lossy compression by Equation (4.4) as follows:

$$Rate_{dump} = \frac{f_{write}(M)}{MN}$$

$$Rate_{dump\_l} = \frac{1}{\frac{MN/\rho_l}{f_{write}(M/\rho_l)} + t_{comp}}$$

$$Rate_{dump\_ll} = \frac{1}{\frac{MN/\rho_{ll}}{f_{write}(M/\rho_{ll})} + t_{comp} + t_{ll\_comp}}$$

Table 4.2: Accuracy of the Uniform Sampling Approach

			t_{comp}			t_{ll_comp}			ρ_l			ρ_{ll}		
field	**eb**	**compressor**	**real**	**estimated**	**error**	**real**	**estimated**	**error**	**real**	**estimated**	**error**	**real**	**estimated**	**error**
velocity_x	1e-3	SZ	12.36	11.66	-5.7%	3.62	2.91	-19.6%	14.92	15.81	6.0%	19.33	21.60	11.7%
		ZFP	6.11	5.74	-6.2%	1.45	0.94	-35.2%	8.24	8.61	4.6%	8.52	8.89	4.4%
	1e-4	SZ	12.92	12.36	-4.3%	0.54	0.67	24.9%	11.29	11.44	1.3%	11.82	11.77	-0.4%
		ZFP	6.95	6.38	-8.1%	1.37	1.43	4.4%	4.65	4.87	4.7%	4.65	4.87	4.7%
	1e-5	SZ	14.10	13.69	-2.9%	0.53	0.75	41.4%	5.76	5.71	-0.9%	5.76	5.78	0.3%
		ZFP	7.66	6.95	-9.3%	2.14	1.98	-7.3%	2.94	3.08	4.7%	2.94	3.08	4.7%
	1e-6	SZ	15.53	17.28	11.2%	0.86	1.36	58.5%	3.59	3.41	-5.2%	3.60	3.56	-1.1%
		ZFP	8.14	7.63	-6.3%	2.70	2.71	0.6%	2.31	2.41	4.8%	2.30	2.41	4.8%
dark matter density	1e-3	SZ	6.07	5.76	-5.1%	0.15	0.22	46.2%	31.41	32.81	4.5%	578.95	557.89	-3.6%
		ZFP	4.05	3.87	-4.5%	2.79	2.44	-12.7%	19.80	20.64	4.2%	24.98	25.72	3.0%
	1e-4	SZ	7.17	6.83	-4.8%	1.36	1.17	-14.0%	15.06	15.75	4.5%	60.40	61.81	2.3%
		ZFP	5.26	4.88	-7.2%	7.83	6.88	-12.1%	7.03	7.36	4.7%	7.59	7.93	4.5%
	1e-5	SZ	12.44	12.11	-2.6%	2.98	0.83	-72.0%	9.27	9.55	3.0%	9.70	9.90	2.1%
		ZFP	5.86	5.31	-9.3%	11.46	10.45	-8.8%	4.24	4.44	4.7%	4.37	4.57	4.7%
	1e-6	SZ	14.87	15.26	2.6%	0.89	0.91	2.7%	4.91	4.83	-1.6%	4.92	4.97	1.0%
		ZFP	6.52	5.79	-11.2%	12.93	9.87	-23.7%	2.77	2.90	4.8%	2.77	2.90	4.8%

Block-wise data sampling and statistics estimation

According to the analysis in the previous section, we need to know the compression ratio and compression speed to compute the corresponding dumping performance given the I/O information and error settings. However, it is very hard to find some surrogate function to do so without data information since the compression ratio as well as the compression speed may depend on specific datasets and/or different error settings significantly. Some of existing works [96] adopted sampling methods to estimate the compression ratio roughly for these compressors. However, they overestimated the compression ratios for SZ as they used the entropy formula which represents an ideal situation that is hardly reached by the compressor (even when SZ is equipped with outstanding lossless compressors such as Zstd) in practice. For the performance estimation of ZFP, the existing sampling methods either overlooked the meta data in the embedded encoding or did not study the impact of lossless compression. Furthermore, they did not have any study on the compression speed since they focus on on compression ratios.

In our approach, we extend the sampling idea to estimate both the compression ratios and compression speed for the two compressors. Apart from the compression ratio estimation, we notice that sampling can offer an effective compression speed approximation based on our in-depth analysis of the compression algorithms. ZFP, for example, performs a customized transform, fixed point alignment and embedded encoding individually on each 4x4x4 block. Its compression time is linear to the number of blocks as well as the time to compress each block. The sampling method would give us a quite accurate estimation in this case. As for SZ, the most time-consuming stage would be the block-wise prediction and quantization, together with the following Huffman encoding. The operations in the block-wise prediction and quantization would be the same for each data block, which also implies linearity in the compression time. As for the Huffman encoding, the sampling would also give good approximations as long as the sampled data leads to similar distributions of quantization index (which likely appears as long as the sampling is not biased).

The detailed sampling method we designed is described as follows. We need to sample the data in common multiples (12x12x12) of 6x6x6 and 4x4x4 because the default 6x6x6 block setting in SZ would usually lead to a better result. To restrict the overhead, we restrict the sampled data to be around 1% of the original data and this already leads to a great performance gain (to be shown later). Specifically, we use a uniform sampling method in the granularity of 12x12x12 block. That is, we sample one 12x12x12 block from every certain number of blocks along each dimension of the data. The distance is fixed for every dimension and is restricted by the 1% requirement. For example, we have 512x512x512 data in the NYX dataset, which corresponds to 42x42x42 blocks. Then, we slightly increase the sampling until around 1% sampling rate is reached. In this case, we will sample 1 every 4 blocks along each dimension, leading to around 1.7% sampling rate. This sampling approach would produce a deterministic result, which is also the reason why we did not adopt the randomize algorithms.

Table 4.2 shows the sampling accuracy of the statistics (listed in Table 4.1) on two typical fields in the NYX dataset. Please note that the SZ result here denotes the improved SZ with fastest setting (to be detailed later) as we need to compare this version with ZFP in our final solution. The column “real”, “estimated” and “error” stand for the real execution time, estimated execution time from sampling, and the corresponding relative error of estimated time with respect to the real time, respectively. From this table, we can see that the uniform sampling approach already get a satisfactory accuracy as most of the error for t_{comp}, ρ_l and ρ_{ll} is within 5%. The only exception is t_{ll_comp}, which indicates the lossless compression time on the lossy compressed data. This is because the lossless compression time relies heavily on the data layout so it expresses less linearity on the data size. However, on the one hand, as lossless compression time only takes a small percentage compared to the SZ compression (around 5% on average), this inaccuracy would have little impact on the final result for SZ. On the other hand, although the lossless compression time may be higher than the compression time of ZFP (e.g. error bound 1E−5 and 1E−6 in dark_matter_density), our framework would make an easy decision to not apply the lossless compression due to the high time overhead and little benefit in the compression ratio in normal cases. All in all, this sampling approach would provide us descent results, as will be validated in the evaluation section.

Adjustments and Optimizations on SZ

We further improve the performance and flexibility of the SZ compressor by applying some adjustments and optimizations to either improve the performance or provide more chances to get better speed-ratio tradeoffs because of more options we create. This is inspired by the fact that the performance gain actually can be further improved if we choose SZ, compared with the best selection of original SZ and ZFP. From our experiments, we obverse that SZ usually "over-compress" the data with respect to the overall dumping performance. For example, SZ spends 20 seconds to compress the velocity_x field in the

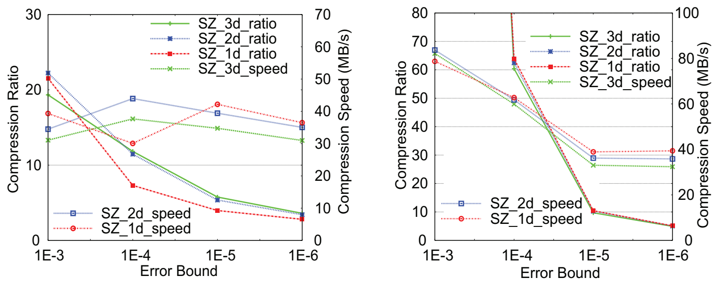
NYX dataset by 40X when the error bound is set to 1E-3, leading to less than 4 seconds in writing with 256 nodes on Theta due to the significant reduced data size thus 24s total dumping time. On the other hand, if we can use less computation to compress the data with a lower compression ratio, say 10 seconds to compress the data by 10X, we end up with around 9 seconds on data writing and thus 19 seconds on total data dumping time (because 40X compression corresponds to around 54GB/s writing performance while 10X compression corresponds to around 90GB/s in terms of Figure 4.1). In this case, we improve the overall performance of SZ by 26% due to 2X faster compression speed with 4X lower compression ratio. As an important contribution of this paper, we explore how to optimize SZ to get the best ratio-speed tradeoffs in different situations.

Online Determination on Lossless Compression. Although SZ provides a configuration file allowing users to decide whether to apply lossless compression, it cannot make an online decision automatically. By contrast, our adaptive framework is able to decide whether to use the lossless compression stage in SZ under different I/O environment, as presented in previous section. We note it here as it is a good example to offer the flexibility for the ratio-speed tradeoff.

Lower Precision Operations Alignment. In our previous SZ implementation, we adopt double precision for most of the computation as it gives more accurate result than the single-precision, avoiding the round-off errors effectively. Thus, when the data is in single precision, there would be many implicit type conversions between double and float, which may incur noticeable performance overhead. This is especially observed for the KNL nodes on Theta. Therefore, we align all the variables used in the SZ prediction and quantization to the data precision and eliminate the conversion overhead and accelerate the computation since single-precision operations are usually faster than the double precision operations, especially in the KNL nodes on Theta. Furthermore, this alignment would have little impact on the final compression ratio because single-precision is accurate enough to perform the error-bounded lossy compression for single-precision datasets.

Low Dimensional Prediction and Unpredictable Data Re-organization for Dependency Removal. We also re-design the prediction scheme in SZ to provide higher feasibility to tradeoff between compression ratios and compression speeds. Specifically, We study the ratio-speed tradeoffs for the prediction dimensions in SZ and remove the dependencies to let the compiler have a better optimization for the code.

We consider to revise the 3D prediction in the Lorenzo [47] predictor for higher performance as the heavy dependencies in the Lorenzo predictor greatly prevents the compiler from optimizing the code. In the 3D Lorenzo predictor, each data point depends on all its 7 adjacent neighbors, making it impossible for parallelism. To alleviate this problem, we propose to use lower dimension prediction for less dependencies and less computations. For instance, if 2D prediction is used for 3D data, the dependency only exist along the 2D plane. Then, the operations along the other dimension could potentially be done using some instruction-level-parallelism techniques. Furthermore, if 1D prediction is used, the operations along the other two dimensions could both be done in parallel. Besides, SZ also have a special strategy for unpredictable data (which is far away from predicted value and have to be stored separately). We also isolate this process to remove the dependencies for the same purpose. Consequently, this adjustment could lead to worse compression ratio as higher dimensional prediction usually leads to better accuracy, which requires us to make a careful selection among them.



(a) velocity_x (b) dark_matter_density

Figure 4.4: Performance of different predict dimensions on two typical fields in NYX dataset

The compression/decompression speed and the corresponding compression ratio for prediction using different dimensions are shown in Figure 4.4. From this figure, we can see that the 3D prediction can usually leads to the best compression, while it has the worse compression/decompression speed due to more computation operations and less potential for parallelization, which may not be the best option in some cases. While 1D prediction is unstable sometimes (probably due to the different number of unpredictable data), it always leads the compression speed when the error bound is relatively small. For instance, we would rather use 1d prediction for the dark_matter_density field, as it has 20% higher compression speed while keeping almost the same compression ratio.

Simplified parameter search

Besides the optimizations on performance, we do a simplified parameter search to the reach the best configuration for SZ as it is a highly tunable compressor with many parameters such as block sizes, prediction dimensions, max number of quantization index, etc. However, as each parameter combination requires a synchronization on all the ranks (otherwise we may have some ranks with high compression speed and low compress ratio while others with low compression speed and high compression ratio, which gives us the worst result), the number of searches should be minimal. After numerous analysis and experiments, we identify block size and prediction dimension as the most important parameters in SZ. We then use a heuristic to explore how to reach the best parameter setting on the two parameters with minimal overhead.

Block size and all Lorenzo prediction. The latest SZ 2.0 [65] adopts a block-wise design which automatically selects between a Lorenzo predictor and a linear-regression based predictor for each block. It uses a block size of 6 by default, as it seems to be the most reasonable option across different datasets and error settings [65]. However, it is never the optimal in any case. Generally speaking, a small block size (but not too small) is preferred when the error bound is loose (e.g. 1E-2) as it can predict data more accurately. However, as smaller block sizes introduce larger overhead, it will affect the performance when the error bound is tight (e.g. 1E-5). On the other hand, we find that the best setting of tight error bounds is to use Lorenzo prediction in all the blocks, as it eliminates all the necessary computation related to regression (which potentially increases the speed) and does not waste any space on storing the regression coefficients and the corresponding meta data (which potentially increases the ratio). Therefore, we select the better one between the default block size 6 and all Lorenzo prediction (in fact it is the extreme case when the block size is the data size and Lorenzo predictor is chosen for the only block), which can already lead to optimal solutions with relatively tight error bounds. For the loose error bound, we stick to the block size 6 as 1) we can reuse the sampled data; 2) it has similar performance to the optimal block size; and 3) the benefit from slighter higher compression ratio would not be much because the I/O time is not dominant in this case and the I/O bandwidth will drop.

Prediction dimension search. Another key parameter is prediction dimension discussed in the previous subsection, as it could lead to up to 20% difference in compression time and 50% compression ratios between 1D and 3D predictions as shown in Figure 4.4. Therefore, we do another parameter search along the prediction dimension to get the optimal one. As 1D prediction is the fastest, we will start there and check how it is compared to 2D prediction. If it is better, we will return 1D prediction as the optimal solution. Otherwise, we will continue to search between 2D prediction and 3D prediction.

Algorithm 7 Heuristic to explore best parameter setting for SZ

Input: sampled data D, I/O information io, estimated time t_0 for fastest setting, fastest setting s_0
Output: Optimal SZ parameter setting

```
 1: $s_1$ = SZ_WITH_REGRESSION_AND_1D_PRED;
 2: $t_1 = estimate\_time(D, io, s_1)$; $sync()$; /*Estimate the time for the regression setting*/
 3: if ($t_0 < t_1$) then
 4:     $s_1 = s_0$;$s_1.pred\_dim = 2$; /*Explore 2D prediction*/
 5:     $t_1 = estimate\_time(D, io, s_1)$; $sync()$;
 6:     if ($t_0 < t_1$) then
 7:         return $s_0$; /*Return setting with 1D prediction when 1D prediction is better than 2D prediction*/
 8:     else
 9:         $s_0.pred\_dim = 3$; /*Explore 3D prediction*/
10:         $t_0 = estimate\_time(D, io, s_0)$; $sync()$;
11:         if ($t_0 < t_1$) then
12:             return $s_0$; /*Return setting with 3D prediction*/
13:         else
14:             return $s_1$; /*Return setting with 2D prediction*/
15:         end if
16:     end if
17: else
18:     return $s_1$ /*Return the regression setting*/
19: end if
```

Algorithm 7 concludes our parameter search approach. The *sync* operation in the pseudo code indicates the MPI_Allreduce operation to collect the best selection from all the ranks. It will gather the votes for the best selection and then collect the corresponding lossless settings. As we are given the profile for SZ with fastest setting (when compared with ZFP), we perform SZ with the regression setting (block size 6 with 3D prediction) on the sample data to gather the information. We then select the one with the lower estimated time as before. If the regression setting works better, we will use the default regression setting directly as further parameter adjustment would make little difference to the final result. On the other hand, if the all Lorenzo setting works better, it indicates either the error bound is relatively low (the compression ratio is relatively high) or the compression time is more important. In this case, we will continue search along prediction dimensions to find the best one. Please note that we will stop exploration for 3D prediction when 1D prediction is better than 2D prediction as it is very unlikely to be better than 1D prediction in this case. This early stopping criterion will save one execution on the sample data.

4.5 Experimental Evalutions

In this section, we conduct experiments with datasets from scientific simulations to demonstrate the effectiveness of our approaches.

4.5.1 Preconditioner for Relative Error Bound

In this section, we compare our approaches with four state-of-the-art lossy compressors providing pointwise relative error bounds: ISABELA [55], SZ PW_REL mode (denoted SZ_PWR) [33, 92], ZFP precision mode (denoted ZFP_P) [67], and FPZIP [69]. We also demonstrate the effectiveness of pointwise relative error from the perspective of visual quality by comparing it with the absolute-error-bound mode in SZ (SZ_ABS). To distinguish from the existing SZ and ZFP compressors, we name our approaches SZ_T and ZFP_T for SZ and ZFP with our transformation scheme, respectively.

Experiment Setup

We conduct our experimental evaluations on a supercomputer [89] using up to 4,096 cores (i.e., 128 nodes, each with two Intel Xeon E5-2695 v4 processors and 128 GB of memory, and each processor with 16 cores). The storage system uses General Parallel File Systems (GPFS). These file systems are located on a raid array and served by multiple file servers. The I/O and storage systems are typical high-end supercomputer facilities. We use the file-per-process mode with POSIX I/O [110] on each process for reading/writing data in parallel. The HPC application data are from multiple domains, namely, HACC cosmology simulation [43], CESM-ATM climate simulation [53], NYX cosmology simulation [73], and Hurricane ISABEL simulation [46]. Each application involves many simulation snapshots (or time steps). We assess only meaningful fields with relatively large data sizes (other fields have constant data or too small data sizes). Table 4.3 presents all the 101 fields across these simulations. The data sizes per snapshot are 3.1 GB, 1.9 GB, 1.2 GB, and 3 GB for the four applications, respectively.

Table 4.3: Simulation fields used in the evaluation

Application	# Fields	Dimensions	Examples
HACC	3	280953867	velocity_x, velocity_y, velocity_z
CESM-ATM	79	1800×3600	CLDHGH, CLDLOW ···
NYX	6	512×512×512	dark_matter_density, temperature ···
Hurricane	13	100×500×500	CLOUDf48, Uf48 ···

Impact of Base Selection

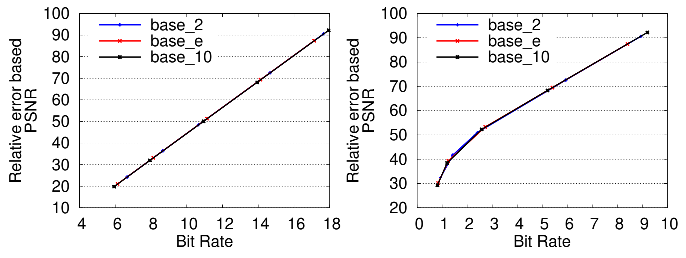
We choose two representative fields, dark_matter_density and velocity_x, in NYX to demonstrate the influence of different logarithmic bases on the final result. Dark_matter_density is a typical use case for pointwise relative error. A large majority (84%) of its data is distributed in [0, 1], and the rest is distributed in [1, 1.378E+4]. Simply applying the absolute value will result in huge distortion when users need to focus on the densest data in [0, 1]. Velocity_x, on the other hand, has usually large values with positive/negative signs indicating directions. Pointwise relative error is also needed when accurate directions of the 3D velocity are required for each point. We evaluate 3 most widely used logarithmic bases: 2, e and 10 on both SZ and ZFP on these two fields.

Table 4.4 shows the compression ratio of SZ with six different pointwise relative error bounds ranging from 10^{-4} to 0.3. According to the table, the different logarithmic bases impact only 1% and 3% on the final compression ratio on average for the two fields, respectively. This variance is less when the pointwise relative error bound is small because of larger number of quantization intervals and tighter bound (Theorem 8) on the difference of quantization index at that time, resulting in a low ratio in frequency difference of the

Table 4.4: Compression ratio of different bases for SZ_T on 2 fields in NYX

fields	**dark_matter_density**			**velocity_x**		
log_bases	**2**	**e**	**10**	**2**	**e**	**10**
0.0001	2.033	2.036	2.036	4.235	4.202	4.254
0.001	2.724	2.725	2.585	7.647	7.509	7.482
0.01	3.842	3.843	3.847	13.047	13.115	13.131
0.1	6.298	6.249	6.307	20.788	18.171	20.079
0.2	7.619	7.595	7.562	22.623	23.090	24.635
0.3	8.529	8.427	8.541	29.696	28.799	29.361

Huffman tree. When the pointwise relative error bound grows, the variance becomes slightly larger because of the smaller number of quantization intervals and the looser bound.



(a) dark_matter_density (b) velocity_x

Figure 4.5: Rate distortion of different bases for ZFP_T on 2 fields in NYX

As mentioned, we prove that different bases do not affect the decorrelation efficiency and coding gain of ZFP. However, since ZFP aims at optimizing the rate distortion given an absolute error bound, it may not keep the same compression ratio because of the different maximum bit-plane computed in embedded coding. Thus, we show the point-wise relative error based rate distortion of ZFP_T in Figure 4.5, in which the PSNR is calculated based on point-wise relative errors with the value range being set to 1. From this figure, we can see that the different bases make little difference in terms of point-wise rate distortion, which is a result that is consistent with our analysis.

Table 4.5: Performance overhead of different bases on 2 fields in NYX

fields	**dark_matter_density**			**velocity_x**		
log_bases	**2**	**e**	**10**	**2**	**e**	**10**
pre-processing time(s)	1.67	1.59	2.23	2.18	2.08	2.74
post-processing time(s)	1.73	2.30	7.11	2.04	2.52	7.35

Table 4.5 shows the overhead of preprocessing and postprocessing steps in our transformation scheme under difference bases. The overhead in preprocessing comes from two aspects: logarithmic mapping on the original dataset and lossless compression on the signs if the dataset is not always positive or negative. Correspondingly, the postprocessing step decompresses the signs if necessary and performs the inverse mapping. Field dark_matter_density has faster preprocessing and postprocessing time compared with velocity_x because its data are always positive and omit the sign compression. Base 10 performs badly during postprocessing because it does not have fast implementation in a standard C library such as base 2 (exp2) and base e (exp). Also it is not competitive on preprocessing, so we do not use it. Although base e is faster than base 2 while preprocessing, it is much slower during the postprocessing step. Thus we use logarithmic base 2 in our implementation, and we fix it for both SZ_T and ZFP_T in the rest part of evaluation.

Table 4.6: Pointwise relative error bound on 2 representative fields in NYX

			dark_matter_density					**velocity_x**				
pwr_eb	**type**	**name**	**settings**	**bounded**	**Avg E**	**Max E**	**CR**	**settings**	**bounded**	**Avg E**	**Max E**	**CR**
1E-3	prediction	ISABELA	1E-3	≈ 100%	4.6E-4	≈ 1E-3	1.35	1E-3	≈ 100%	4.7E-4	≈ 1E-3	1.71
		FPZIP	-p 19	100%	3.4E-4	9.8E-4	2.28	-p 19	100%	3.5E-4	9.8E-4	6.15
		SZ_PWR	-P 1E-3	≈ 100%*	1.2E-4	≈ 1E-3	1.87	-P 1E-3	≈ 100%	4.5E-4	≈ 1E-3	6.77
		SZ_T	-P 1E-3	100%	4.7E-4	9.9E-4	2.72	-P 1E-3	100%	4.9E-4	9.9E-4	7.58
	transform	ZFP_P	-p 26	99.93%*	5.7E-5	2.9E+4	1.5	-p 20	99.94%	2.3E-5	1.5E+2	3.4
		ZFP_T	-p 1E-3	100%	2.2E-5	2.1E-4	1.81	-p 1E-3	100%	2.3E-5	2.1E-4	3.58
1E-2	prediction	ISABELA	1E-2	≈ 100%	4E-3	≈ 1E-2	1.91	1E-2	≈ 100%	2.9E-3	≈ 1E-2	2.42
		FPZIP	-p 16	100%	2.7E-3	7.8E-3	2.89	-p 16	100%	2.8E-3	7.8E-3	10.79
		SZ_PWR	-P 1E-2	≈ 100%*	1.2E-3	≈ 1E-2	2.46	-P 1E-2	≈ 100%	4.5E-3	≈ 1E-2	11.08
		SZ_T	-P 1E-2	100%	4.8E-3	1E-2	3.85	-P 1E-2	100%	5E-3	1E-2	13.49
	transform	ZFP_P	-p 23	99.94%*	5.7E-4	2.5E+5	1.75	-p 16	99.91%	3.4E-4	7.7E+2	5.42
		ZFP_T	-p 1E-2	100%	1.8E-4	1.6E-3	2.18	-p 1E-2	100%	1.8E-4	1.6E-3	5.38
1E-1	prediction	ISABELA	1E-1	≈ 100%	1.8E-2	≈ 1E-1	2.52	1E-1	100%	7.6E-3	1E-1	2.75
		FPZIP	-p 13	100%	2.2E-2	5.9E-2	3.97	-p 13	100%	2.2E-2	5.9E-2	19.08
		SZ_PWR	-P 1E-1	≈ 100%*	1.2E-2	≈ 1E-1	3.37	-P 1E-1	100%	4.5E-2	1E-1	13.73
		SZ_T	-P 1E-1	100%	4.6E-2	1E-1	6.31	-P 1E-2	100%	4.8E-2	1E-1	22.07
	transform	ZFP_P	-p 19	99.91%*	5.7E-3	1.9E+5	2.23	-p 13	99.93%	2.6E-3	2E+5	12.1
		ZFP_T	-p 1E-1	100%	2.8E-3	2.6E-2	3.00	-p 1E-1	100%	2.1E-3	2.5E-2	13.3

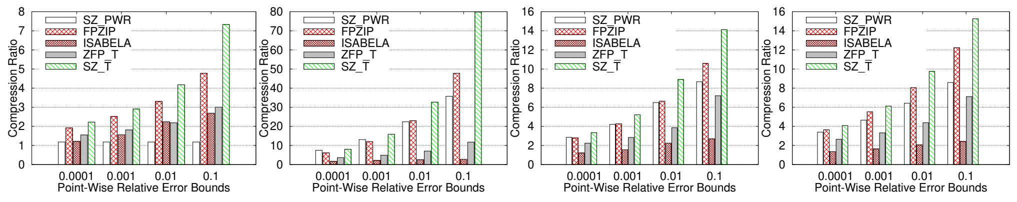
Strict Error Bound Test

In this subsection, we check the maximum pointwise relative errors of our approach and state-of-the-art approaches. As mentioned, ZFP overpreserves the absolute error bound and thus may not be competitive with other compressors given an absolute error bound. Therefore, we select SZ_T as our final solution in order to get the maximal possible the compression ratio with the transformation scheme given the pointwise relative error bound. However, we still compare ZFP_T with the precision mode of ZFP to demonstrate that our transformation scheme can also be used to improved transform-based compressors such as ZFP.

The results of three most widely used pointwise error bounds (0.1, 0.01, 0,001) are shown in Table 4.6. Columns 4 and 9 (settings) indicate the parameters we choose for each compressor. Columns 5 and 10 show the percentage of decompressed data that is strictly bounded by the given error bound. The notation $\approx$ indicates that most of the data is bounded by the error bound, but there exists little data (usually much less than 0.01%) that exceeds the bound, likely because of round-off errors. The notation * in the table indicates that the compressor modifies original 0 in the data. The compressors without the notation * keep the original 0 as it is, such that the decompression has no loss on the values 0. The columns *Avg E* and *Max E* indicate the average and maximum pointwise relative errors, respectively. From this table, one can clearly see that only FPZIP and the compressors under our transformation scheme (SZ_T and ZFP_T) can strictly respect the given error bound and keep the original zeros as they are. Furthermore, the SZ_T compressor also yields the best compression ratio on the two fields, demonstrating its high quality and good efficiency.

We also compare ZFP under our transformation scheme (denoted ZFP_T) with the -p option given by the original ZFP lossy compressor (denoted ZFP_P). Since ZFP_P does not strictly respect the error bound, we set the percentage threshold for bounded data in ZFP_P to 99.9%, in order to keep the same order of average error with ZFP_T. According to the table, ZFP_T outperforms ZFP_P in almost all aspects, demonstrating that our transformation scheme can really improve the compression quality for ZFP as well. Its compression ratio is not as high as those of other compressors because of the over-preserved error bound. If there exist some other transform-based compressors that may lead to a very high compression ratio given specific absolute error bounds, our transformation scheme can also turn them into outstanding compressors respecting pointwise relative error bound.

Compression Ratio & Compression/Decompression Rate

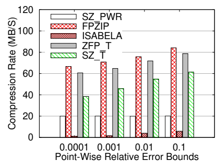


(a) HACC (b) CESM-ATM (c) NYX (d) Hurricane

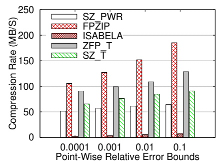
Figure 4.6: Compression ratio on given point relative error bound

Here we showcase the compression performance (compression ratio and compression/decompression rate) of the above lossy compressors. However, tuning the parameter for ZFP_P for each field under each error bound is complicated because it does not respect the error bound. Also, according to two fields in NYX (Table 4.6), ZFP is not as competitive as FPZIP and SZ_PWR in terms of the compression ratio. Thus we do not test ZFP_P for overall performance from this section on.

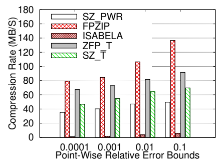
The compression ratios of the lossy compressors on the four datasets are displayed in Figure 4.6. ISABELA usually cannot achieve a high compression ratio because of its high index overhead. SZ_PWR is competitive when the error bound is small, but its performance degrades for larger error bounds. Also, it is not good at sharply varying datasets such as HACC because of the group minimum design. FPZIP is good in most cases, but its performance suffers on 2D datasets, especially when the error bound is small. Our SZ_T almost outperforms all the other compressors by a certain scale under all the tested error bounds. However, ZFP_T does not exhibit a high compression ratio because it overpreserves the error bound.



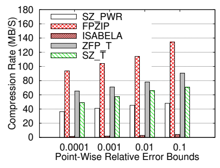
(a) HACC Comp.



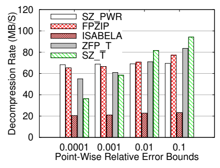
(b) CESM Comp.



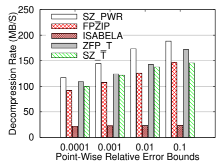
(c) NYX Comp.



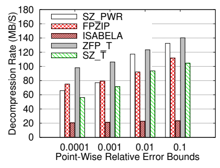
(d) Hurricane Comp.



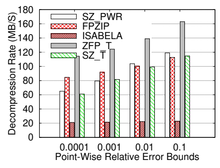
(e) HACC Decomp.



(f) CESM Decomp.



(g) NYX Decomp.



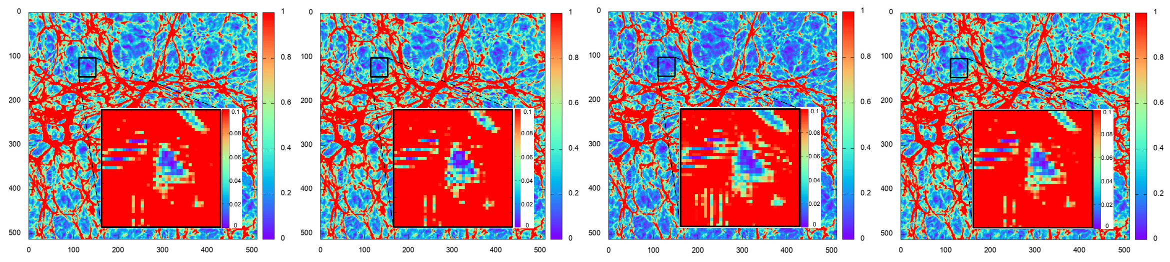
(h) Hurricane Decomp.

Figure 4.7: Compression/decompression rate on given point relative error bound

We also evaluate the compression/decompression rate of these lossy compressors. The results are shown in Figure 4.7. According to this figure, FPZIP leads all the other compressors in all the datasets in terms of compression speed. ZFP_T usually gets the second place because ZFP is faster than SZ. SZ_T is better than SZ_PWR since it does not compute complicated block information. ISABELA is slow because of the high sorting overhead. Regarding decompression, all the compressors except ISABELA exhibit comparable performance. SZ_PWR shows considerable improvement compared with its compression rate because it saves the block information and does not compute it during decompression.

Visualization for Multiprecision and Angle Skews

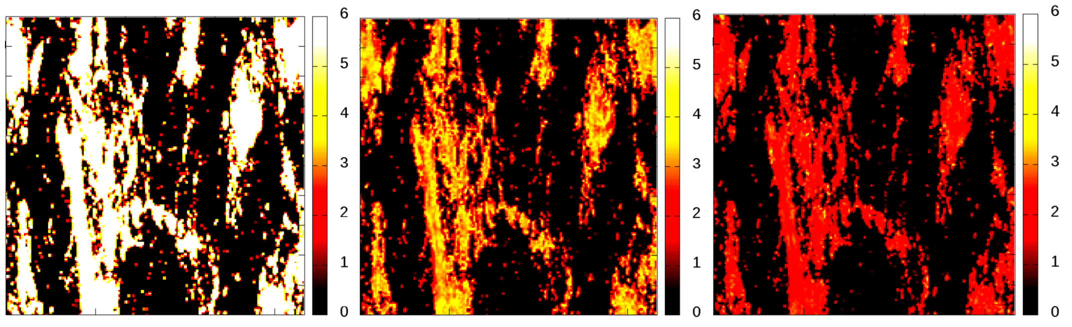
Besides compression performance, compression quality, such as the multiprecision visualization, is also important. Unlike absolute error bound, which requires universal restriction on each data point, pointwise relative error bound provides value-based restrictions that are usually different for different data points. Under this requirement, smaller value will have a smaller error bound on this data point and vice versa. Respecting this error bound is very effective when all the data points are all equally important regardless of their data value; otherwise, the small data value will be easily distorted by the universal restriction. In this section, we analyze the quality of pointwise error bound by visualizing the decompressed data generated by difference compressors.



(a) Original data (b) SZ_ABS (c) FPZIP (d) SZ_T

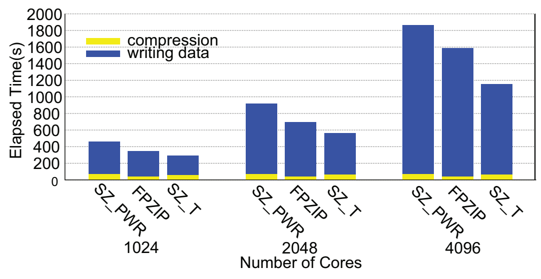
Figure 4.8: Multiprecision distortion of Slice 100 in dark_matter_density (NYX, $512 \times 512 \times 512$) when the compression ratio is 7. The original data is shown in range [0, 1], and enlarged windows are observed with a higher precision [0, 0.1]. Compared with SZ_ABS, FPZIP clearly keeps the features in blue parts (e.g., data in the center). However, it exploits more local loss and adds certain noise to regions between the blue and red parts (i.e., data in the top right and bottom left parts) since its max pointwise relative error is 0.5, which is much larger than that of SZ_T (0.15).

Figure 4.8 shows the multiprecision visualization results of original data, SZ_ABS, FPZIP, and SZ_T on the 100th slice in dark_matter_density fields of the NYX dataset when the compression ratio is set to 7. The absolute-error-bound mode of SZ is used as a comparison to demonstrate the advantages of pointwise relative error bound. Only FPZIP and SZ_T are selected because the other compressors cannot achieve such a compression ratio when the point-wise relative error bound is set to less than 100%. The original images show the visualization of a data range [0, 1] (value greater than 1 is shown as 1). The zoomed-in windows show a more precise range of [0, 0.1]. According to this figure, absolute error bound mode leads to noticeable distortion. The blue region in the center is distorted a lot because the universal restriction on each point is 0.055, which is large for data in range [0, 0.1]. On the other hand, FPZIP keeps the rough features in the center because of pointwise relative error bound, which has tighter restrictions on those regions. However, since it has to relax its pointwise error bound to 0.5 to reach such a compression ratio, there is large distortion for data in range (0.1, 1), leading to the artifacts in the bottom left and top right. As a contrast, SZ_T uses a pointwise relative error bound of 0.15 to get the same compression ratio with little distortion.

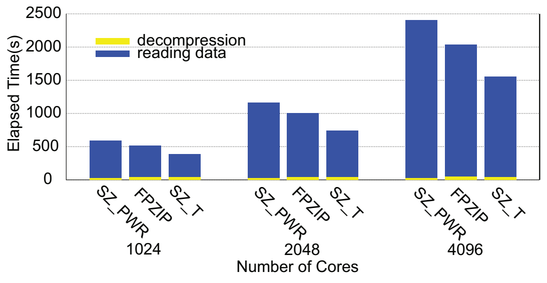


(a) SZ_ABS (b) FPZIP (c) SZ_T

Figure 4.9: Angle skews of different compressors on HACC datasets when compression ratio is 8. The absolute-error-bounded compressor leads to large angle skews because the universal error bound (15 in this case) may greatly affect a small value. SZ_T has better performance because it has a stricter pointwise error bound (0.145) than that of FPZIP (0.334) under the given compression ratio.



(a) Data Dumping Time



(b) Data-loading Time

Figure 4.10: Dumping and loading performance of NYX in parallel execution

We also compare the skewed angles between original data and decompressed data for the velocity fields in HACC. A particle's skewed angle is defined as the angle between its original velocity and its reconstructed velocity in 3D space. It is calculated by $\theta = \arccos \frac{\vec{v}\vec{v_d}}{||\vec{v}||||\vec{v_d}||}$, where $\vec{v}$ is the original velocity in the 3D space and $\vec{v_d}$ is the reconstructed one. Because data points in HACC are scattered in the 3D space, we divide the whole space into $200 \times 200 \times 200$ blocks and compute the average skewed angle values in each block. We present the result at slice 100 in Figure 4.9. The brighter a region is, the larger distortion it has, meaning a worse result. The figure shows that the absolute-error-bounded compressor has larger skewed angles (usually > 6), while pointwise relative-error-bounded compressors have much smaller skewed angles (around 4 for FPZIP and 2 for SZ_T). SZ_T is better than FPZIP because it has a stricter error bound (0.145 versus 0.334) at this compression ratio.

Parallel Evaluation

We still use the NYX dataset to demonstrate the data-dumping and data-loading performance in parallel execution. Since ISABELA and ZFP_T have a much lower compression ratio (ISABELA also has a relatively lower compression rate), we do not involve them in the parallel execution. For the rest of the compressors (SZ_PWR, FPZIP, SZ_T), we fix the pointwise error bound to 0.01 for all six fields in NYX datasets. We evaluate the three compressors on three scales (1,024 cores $\sim$ 4,096 cores). Each rank in the evaluation needs to process 3 GB of data, which corresponds to a total of 3 TB $\sim$ 12 TB of data. We plot the breakdown of data-dumping time (compression and writing) and data loading time (reading and decompression) in Figure 4.10. As a comparison, the original data needs about 0.7 $\sim$ 2.8 hours and 1 $\sim$ 4 hours for dumping and loading, respectively. From this figure, we can observe that our transformation scheme is able to achieve $1.62X$, $1.38X$ dumping performance and $1.55X$, $1.31X$ loading performance over SZ_PWR and FPZIP on 4k cores, respectively, thanks to the higher compression ratio and acceptable compression/decompression rates. Also, we tend to have more advantages when the scale continues to increase.

4.5.2 Adaptive Compression Framework for I/O Performance

In this section, we present the performance evaluation results on two supercomputers to show the efficiency of our adaptive compression framework.

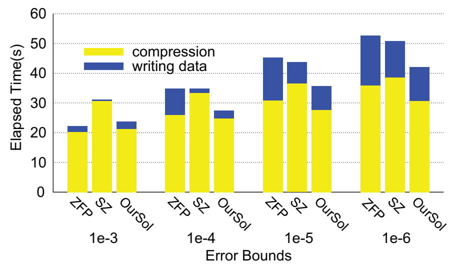
Experimental Setting

We conducted our experimental evaluation on two supercomputers with different scales. We first evaluate our framework on Bebop [89], a medium-scale supercomputer with relatively low bandwidth I/O facilities. Bebop has 1024 public nodes, each with two Intel Xeon E5-2695 v4 processors and 128GB of memory, and each processor with 18 cores. It uses General Parallel File Systems (GPFS) equipped with 2 I/O nodes, providing around 2GB/s I/O bandwidth. Bebop is a good example for extreme case when I/O is saturated all the time (which is usually the case for scientific simulations), as its 2 I/O nodes are always used. We also validate our framework on Theta [90], a supercomputer with relatively low CPU frequency (KNL nodes) and high I/O bandwidth. Each node on Theta is a 64-core processor with 16GB high bandwidth in-package memory and 192 GB of DDR4 RAM. It is also equipped with 10 PB Lustre file system [1] with a total bandwidth of 172 GB/s for write and 240 GB/s for read, which is representative of realistic systems with high-performance I/O. We use MPI-IO [100] for all the experiments as it is efficient and widely used. The application data is from 2 typical scientific simulations, namely NYX [73] cosmology simulation and SCALE-LETKF [108] weather simulation. Each application involves six data fields, each of which contains 512MB/540MB data on each rank for the two applications, respectively. This yields to a total of 96TB/101.25TB data on Theta when the scale is 512 nodes.

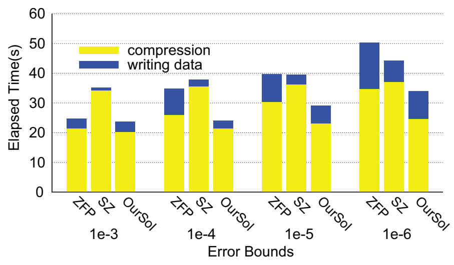
We focus on the overall dumping performance on the parallel file systems, including both compression time and writing time. We applied our solution to two state-of-the-art lossy compressors - ZFP [67] that often has higher compression speed and SZ [65] which generally leads to higher compression ratios. In the following sections, we first show the dumping performance of our adaptive framework compared to these two state-of-the-art compressors, and then we show a detailed breakdown of the time and overhead.

Dumping performance on Bebop

We first present the dumping performance on the Bebop cluster. Specifically, we first show the result on only 1 node. We choose this scale because it is the turning point on selecting between ZFP and SZ for most of the fields. It happens at such a low scale because 1) ZFP does not have as much dominance in speed as that on Theta; 2) I/O bandwidth in Bebop is very low such that compression ratio would easily dominate.

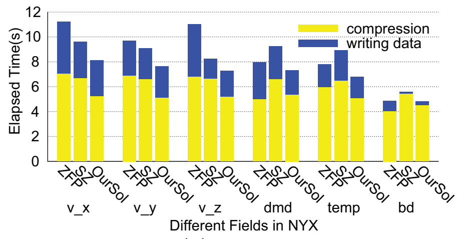


(a) NYX

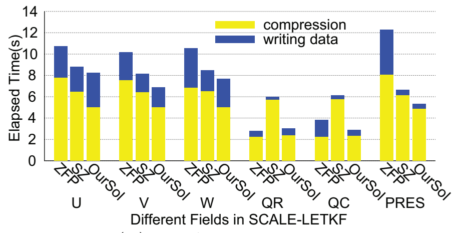


(b) SCALE-LETKF

Figure 4.11: Dumping performance of different error bounds (1E-3 ∼ 1E-6) on Bebop with 1 node (32 cores)



(a) NYX

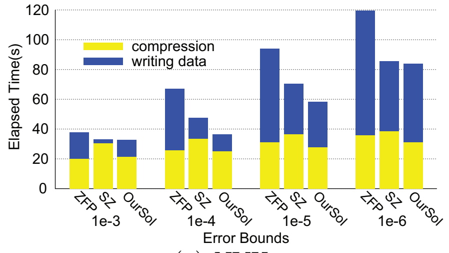


(b) SCALE-LETKF

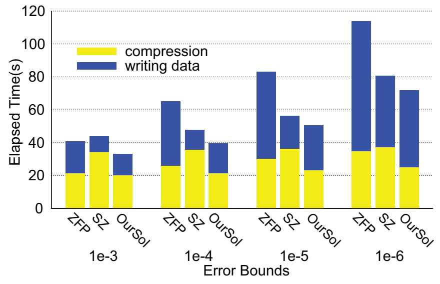
Figure 4.12: Detailed breakdown of I/O Performance on each field (error bound 1E-6) on Bebop with 1 nodes (32 cores)

The overall dumping performance on Bebop with 1 node is shown in Figure 4.11. From this figure, we can see that the compression time takes the major part of the total dumping time. Even in this case, ZFP can only be better than SZ when the error bound is large because it only has slightly higher compression speed than that of SZ, which could not compensate for the loss in writing time on this platform. On the other hand, our solution always leads to the shortest dumping time except for the error bound 1E-3 on NYX dataset, because the adaptive framework is able to switch to the most suitable option for the different fields in the dataset. As for the error bound 1E-3 on NYX dataset, our solution selects ZFP correctly for all the fields, with a slight overhead because of the sampling and other decision process in our design.

We also show the detailed time breakdown on the six fields of the two datasets in Figure 4.12, where we can easily figure out how our adaptive framework makes the decision. For instance, the adaptive framework selects SZ for all the fields, and it discards the lossless compression in SZ for the last 4 fields. Therefore, although our solution may have higher writing time, it has much lower compression time thanks to our optimization of parameter settings and the removal of lossless compression.



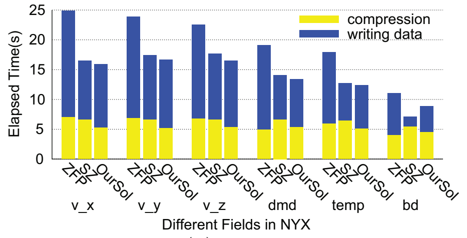
(a) NYX



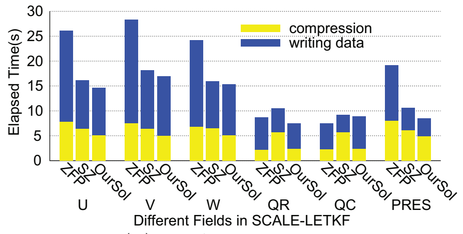
(b) SCALE-LETKF

Figure 4.13: Dumping performance of different error bounds (1E-3 $\sim$ 1E-6) on Bebop with 4 nodes (128 cores)

We then perform the experiments with 4 nodes on Bebop in 4.13. In this case, our solution also leads to the best overall dumping performance. Please note that our solution costs more time than SZ in the "bd" field, as shown in Figure 4.14. This is due to the fact that I/O bandwidth is shared thus not stable. When we do the profiling, we get the writing speed in Bebop is around 2 GB/s. However, when the data is dumped, the



(a) NYX



(b) SCALE-LETKF

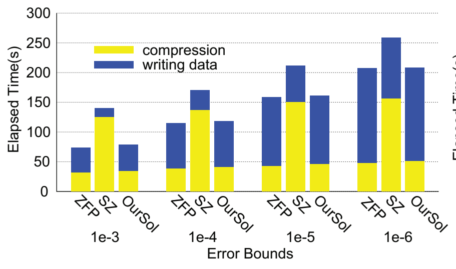
Figure 4.14: Detailed breakdown of dumping performance on each field (error bound 1E-6) on Bebop with 4 nodes (128 cores)

actual writing speed is only 1.05 GB/s, which leads to the wrong decision for discarding the lossless compression stage in our solution. However, this situation only occurs around the turning point, which would not incur much overhead even when the decision is wrong (even in this case when the writing speed is inaccurate by 50%). Generally speaking, our solution would work better in a relatively stable environment like Theta, as will be discussed in next section.

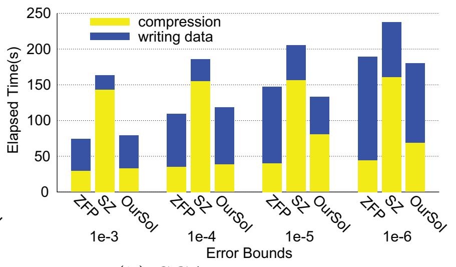
Dumping performance on Theta

We perform similar experiments on Theta. Unlike Bebop, the turning point for choosing SZ and ZFP is around 256 $\sim$ 512 nodes. Therefore, we show the result on Theta with 256 nodes and 512 nodes, respectively. Please note that we pad the compressed data to multiple of 64KB and set the maximal padded size as the stripe size (which is the real compressed size) for best performance, as required by the Lustre file system on Theta.

The overall dumping performance on Theta with 256 nodes (16,384 cores) is displayed in Figure 4.15. In this case, the total data to be dumped in a single snapshot would be 48TB and 51TB, costing $\sim$410s and $\sim$440s on data writing, respectively. For each

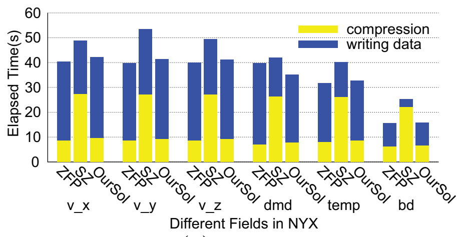


(a) NYX

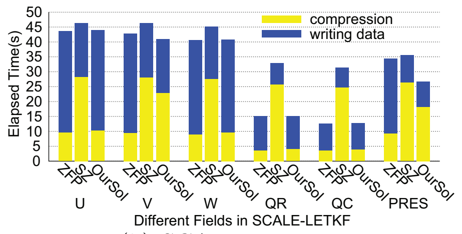


(b) SCALE-LETKF

Figure 4.15: Dumping performance of different error bounds (1E-3 ∼ 1E-6) on Theta with 256 nodes (16,384 cores)



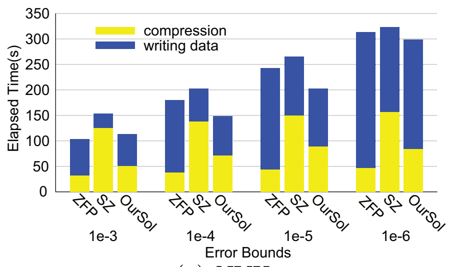
(a) NYX



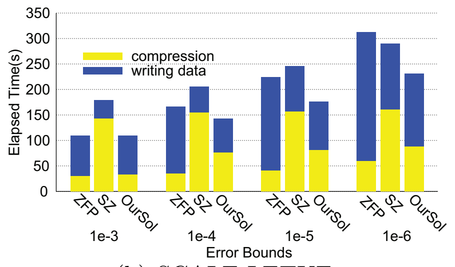
(b) SCALE-LETKF

Figure 4.16: Detailed breakdown of dumping performance on each field (error bound 1E-6) on Theta with 256 nodes (16,384 cores)

dataset, the writing bandwidth is 115GB/s according to Figure 4.1). In this situation, the compression time for SZ would easily become larger than the writing time, as shown in this figure. Then, our adaptive framework would prefer ZFP in most cases, as shown in the detailed performance breakdown on each field in Figure 4.16. From this figure, we can see that our solution always selects ZFP for the NYX dataset, leading to similar execution time compared with ZFP. Please note that our solution would have a low overhead due to the sampling and synchronization process. It is around 5% when the error bound is 1E-3 and reduces to less than 2% when the error bound is 1E-6 on the two datasets, due to the fact that the percentage of compression time over total dumping time would decrease with the decreasing error bound (low error bound leads to low compression ratio thus high writing time). Nevertheless, our solution outperforms ZFP when the error bound is tight, since our framework accurately picks four fields (the first, second, third and last) out of the six fields in which the optimized SZ is faster than ZFP considering the overall dumping performance.

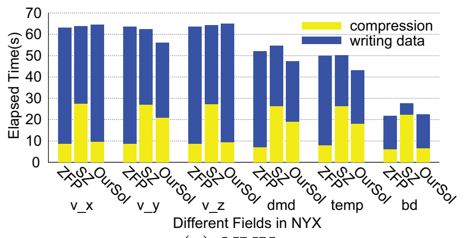


(a) NYX

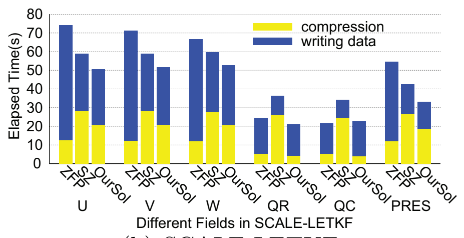


(b) SCALE-LETKF

Figure 4.17: Dumping performance of different error bounds (1E-3 $\sim$ 1E-6) on Theta with 512 nodes (32,768 cores)



(a) NYX



(b) SCALE-LETKF

Figure 4.18: Detailed breakdown of dumping performance on each field (error bound 1E-6) on Theta with 512 nodes (32,768 cores)

We then show the result with slightly increased scale in Figure 4.17, by using 512 nodes (32,768 cores) on Theta. Note that the experiments here are all weak-scaling thus the file size to be dumped is doubled. Compression ratios become more important in this case because of the increased time on writing, especially when the error bound is relatively small. In this case, our adaptive solution would select the optimized SZ for most of the time. As shown in Figure 4.18, it chooses to use the optimized SZ for five out of six fields in NYX and four out of six fields in SCALE-LETKF, when the error bound is set to 1e-5. This decision leads to over 20% performance gain in these fields (e.g. the first, second, third and fifth field in NYX and the first, second, third and the last field in SCALE-LETKF). As these fields dominate the total dumping time, the final performance gains for our solution are 20% and 27%, respectively, compared to the second-best compressor for the two datasets in this case.

4.6 Summary

In this section, we propose two methods to enhance the lossy compression efficiency according to different demands. We propose to use logarithmic transform as a preditioner for scientific data requiring relative error bound and leverage the existing state of the art compressors with slight modifidications. Then, we design an adaptive lossy compression framework selecting best-fit lossy compressor and optimal compression settings for different scales at runtime. The proposed methods further improve lossy compression effciency with respect to different demands, making lossy compression more efficient under these demands.

Chapter 5

Conclusions

To tolerate soft errors while enabling low overhead, high fault coverage and timely fault detection and correction in fast Fourier Transform computations, we proposed an online ABFT scheme for FFT by taking advantage of divide-and-conquer nature of FFT algorithm. We divide a large FFT computation into smaller ones and apply the offline ABFT to the small FFTs with reused checksum generations. Unlike traditional offline ABFT schemes that can only detect and correct error at the end of the computation, our design can detect and correct errors in each decomposed FFT which is much finer-grained than the original FFT, leading to timely error recovery. We also apply various optimizations including serial ones such as checksum re-design and parallel ones such as communication-computation overlap. Experimental results demonstrate that the proposed scheme introduces less overhead in error-free executions, and improves the computing efficiency by 2X over existing schemes when errors occur. Futhermore, it has higher fault coverage due to higher numerical stability.

To improve the quality of error-bounded lossy compression for better serving scientific applications, we propose adaptive compression algorithms with multiple prediction methods and compression models, as well as strategies to select the best-fit prediction method and compression model in terms of the data features in different regions or fields of the dataset. We evaluate our solution using 100+ fields across 4 well-known HPC simulations, by comparing to six existing state-of-the-art lossy compressors (SZ, ZFP, TTHRESH, VAPOR, FPZIP and ISABELA). Experiments demonstrate that our the multi-predictor design can achieve up to 8x compression ratio as that of other compressors with the same PSNR. By running the three best lossy compressors (SZ, ZFP and our solution) using up to 8,192 cores, our solution has 1.86X dumping performance and 1.95X loading performance over the second best compressor because of the significant reduction of data size. In addition, the proposed compression algorithm with hybrid compression models further improves the compression quality thanks to the improved coefficient encoding algorithms and the accurate selection algorithms.

To meet the different requirements and objectives from scientific applications, we propose two methods to enhance the lossy compression efficiency with respect to the relative error bound requirement and I/O performance. For the relative error bound, we formulate the bidirectional mapping problem between relative error bound and absolute error bound in order to use the existing lossy compressors in their best compression mode (absolute error mode). Under the specific constraints of lossy compression, we conclude that the logarithm in base 2 is the most effective transform from relative error bound to absolute error bound. We also prove the uniqueness of the logarithmic transform to solve our initial mapping problem. Experiments demonstrate that the proposed solution can significantly improve the compression ratios (up to 60%) under the demand of relative error bound. Then, we design an adaptive lossy compression framework selecting best-fit lossy compressor and optimal compression settings for different scales at runtime. The proposed solution achieves up to 27% performance gain over other approaches on Theta supercomputer with 32k cores. These methods further improve lossy compression effciency with respect to different demands from scientific applications.

Bibliography

[1] Lustre file system. `http://lustre.org/`. Online.

[2] Mark Ainsworth, Scott Klasky, and Ben Whitney. Compression using lossless decimation: analysis and application. *SIAM Journal on Scientific Computing*, 39(4):B732–B757, 2017.

[3] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science*, 19(5-6):65–76, 2018.

[4] Francesc Alted. Blosc, an extremely fast, multi-threaded, meta-compressor library, 2017.

[5] Anna Antola, R Negrini, MG Sami, and Nello Scarabottolo. Fault tolerance in fft arrays: time redundancy approaches. *Journal of VLSI signal processing systems for signal, image and video technology*, 4(4):295–316, 1992.

[6] Rizwan A Ashraf, Roberto Gioiosa, Gokcen Kestor, Ronald F DeMara, Chen-Yong Cher, and Pradip Bose. Understanding the propagation of transient errors in hpc applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 72. ACM, 2015.

[7] Woody Austin, Grey Ballard, and Tamara G Kolda. Parallel tensor compression for large-scale scientific data. In *2016 IEEE International Parallel and Distributed Processing Symposium*, pages 912–922, 2016.

[8] Allison H Baker, Haiying Xu, John M Dennis, Michael N Levy, Doug Nychka, Sheri A Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 203–214. ACM, 2014.

[9] Rafael Ballester-Ripoll, Peter Lindstrom, and Renato Pajarola. Tthresh: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics*, 2019.

[10] P Banerjee et al. Algorithm-based fault detection for signal processing applications. *Computers, IEEE Transactions on*, 39(10):1304–1308, 1990.

[11] Prithviraj Banerjee, Joe T Rahmeh, Craig Stunkel, VS Nair, Kaushik Roy, Vijay Balasubramanian, Jacob Abraham, et al. Algorithm-based fault tolerance on a hypercube multiprocessor. *Computers, IEEE Transactions on*, 39(9):1132–1145, 1990.

[12] Leonardo Arturo Bautista-Gomez and Franck Cappello. Detecting and correcting data corruption in stencil applications through multivariate interpolation. In *CLUSTER*, pages 595–602, 2015.

[13] Patrick G Bridges, Kurt B Ferreira, Michael A Heroux, and Mark Hoemmen. Fault-tolerant linear solvers via selective reliability. *arXiv preprint arXiv:1206.1390*, 2012.

[14] Greg Bronevetsky and Bronis de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 155–164. ACM, 2008.

[15] Martin Burtscher and Paruj Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):18–31, Jan 2009.

[16] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications*, page 1094342019853336, 2019.

[17] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Mark Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 2009.

[18] Marc Casas, Bronis R de Supinski, Greg Bronevetsky, and Martin Schulz. Fault resilience of the algebraic multi-grid solver. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 91–100. ACM, 2012.

[19] Jieyang Chen, Hongbo Li, Sihuan Li, Xin Liang, Panruo Wu, Dingwen Tao, Kaiming Ouyang, Yuanlai Liu, Kai Zhao, Qiang Guan, et al. Fault tolerant one-sided matrix decompositions on heterogeneous systems with gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 68. IEEE Press, 2018.

[20] Jieyang Chen, Xin Liang, and Zizhong Chen. Online algorithm-based fault tolerance for cholesky decomposition on heterogeneous systems with gpus. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 993–1002. IEEE, 2016.

[21] Zhengzhang Chen, Seung Woo Son, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. NUMARCK: machine learning algorithm for resiliency and

checkpointing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 733–744. IEEE Press, 2014.

[22] A Chien, P Balaji, P Beckman, N Dun, A Fang, H Fujita, K Iskra, Z Rubenstein, Z Zheng, R Schreiber, et al. Versioned distributed arrays for resilience in scientific applications: Global view resilience. *Journal of Computational Science*, 2015.

[23] Jong Youl Choi, Choong-Seock Chang, Julien Dominski, Scott Klasky, Gabriele Merlo, Eric Suchyta, Mark Ainsworth, Bryce Allen, Franck Cappello, Michael Churchill, Philip E. Davis, Sheng Di, Greg Eisenhauer, Stéphane Ethier, Ian T. Foster, Berk Geveci, Hanqi Guo, Kevin A. Huck, Frank Jenko, Mark Kim, James Kress, Seung-Hoe Ku, Qing Liu, Jeremy Logan, Allen D. Malony, Kshitij Mehta, Kenneth Moreland, Todd Munson, Manish Parashar, Tom Peterka, Norbert Podhorszki, Dave Pugmire, Ozan Tugluk, Ruonan Wang, Ben Whitney, Matthew Wolf, and Chad Wood. Coupling wxascale multiphysics applications: Methods and lessons learned. In *Proceedings of IEEE International Conference on eScience*, 2018.

[24] Yoon-Hwa Choi and Miroslaw Malek. A fault-tolerant fft processor. *Computers, IEEE Transactions on*, 37(5):617–621, 1988.

[25] Steven Claggett, Sahar Azimi, and Martin Burtscher. Spdp: An automatically synthesized lossless compression algorithm for floating-point data. In *the 2018 Data Compression Conference*, pages 337–346, 2018.

[26] Roger J Clarke. Transform coding of images. *Astrophysics*, 1985.

[27] John Clyne, Pablo Mininni, Alan Norton, and Mark Rast. Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, 9(301):1–29, 2007.

[28] Thomas H Cormen and David M Nicol. Performing out-of-core ffts on parallel disk systems. *Parallel Computing*, 24(1):5–20, 1998.

[29] Teresa Davies and Zizhong Chen. Correcting soft errors online in lu factorization. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 167–178. ACM, 2013.

[30] L Peter Deutsch. GZIP file format specification version 4.3, 1996.

[31] Sheng Di and Franck Cappello. Adaptive impact-driven detection of silent data corruption for hpc applications. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2809–2823, 2016.

[32] Sheng Di and Franck Cappello. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium*, pages 730–739. IEEE, 2016.

[33] Sheng Di, Dingwen Tao, Xin Liang, and Franck Cappello. Efficient lossy compression for scientific data based on pointwise relative error bound. *IEEE Transactions on Parallel and Distributed Systems*, 30(2):331–345, 2018.

[34] James Elliott, Mark Hoemmen, and Frank Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *IPDPS*, pages 1193–1202, 2014.

[35] Ian T. Foster, Mark Ainsworth, Bryce Allen, Julie Bessac, Franck Cappello, Jong Youl Choi, Emil M. Constantinescu, Philip E. Davis, Sheng Di, Zichao Wendy Di, Hanqi Guo, Scott Klasky, Kerstin Kleese van Dam, Tahsin M. Kurç, Qing Liu, Abid Malik, Kshitij Mehta, Klaus Mueller, Todd Munson, George Ostrouchov, Manish Parashar, Tom Peterka, Line Pouchard, Dingwen Tao, Ozan Tugluk, Stefan M. Wild, Matthew Wolf, Justin M. Wozniak, Wei Xu, and Shinjae Yoo. Computing just what you need: Online data analysis and reduction at extreme scales. In *European Conference on Parallel Processing*, pages 3–19. Springer, 2017.

[36] Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

[37] Matteo Frigo and Steven G Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[38] Hongyi Fu and Xuejun Yang. Fault tolerant parallel fft using parallel failure recovery. In *Computational Science and Its Applications, 2009. ICCSA'09. International Conference on*, pages 257–261. IEEE, 2009.

[39] A Geist. How to kill a supercomputer: Dirty power, cosmic rays, and bad solder. *IEEE Spectrum*, 2016.

[40] W Morven Gentleman and Gordon Sande. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578. ACM, 1966.

[41] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

[42] Ali Murat Gok, Sheng Di, Alexeev Yuri, Dingwen Tao, Vladimir Mironov, Xin Liang, and Franck Cappello. PaSTRI: A novel data compression algorithm for two-electron integrals in quantum chemistry. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2018.

[43] Salman Habib, Vitali A. Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Venkatram Vishwanath, Tom Peterka, Joseph A. Insley, David Daniel, Patricia K. Fasel, and Zarija Lukic. HACC: extreme scaling and performance across diverse architectures. *Communications of the ACM*, 60(1):97–104, 2016.

[44] Kuang-Hua Huang and Jacob A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, 33(6):518–528, 1984.

[45] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[46] Hurricane ISABEL simulation data. `http://vis.computer.org/vis2004contest/data.html`, 2019. Online.

[47] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer Graphics Forum*, volume 22, pages 343–348. Wiley Online Library, 2003.

[48] Luc Jaulmes, Marc Casas, Miquel Moretó, Eduard Ayguadé, Jesús Labarta, and Mateo Valero. Exploiting asynchrony from exact forward recovery for due in iterative solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 53. ACM, 2015.

[49] Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. Deepsz: A novel framework to compress deep neural networks by using error-bounded lossy compression. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pages 159–170. ACM, 2019.

[50] Jing-Yang Jou, Jacob Abraham, et al. Fault-tolerant fft networks. *Computers, IEEE Transactions on*, 37(5):548–561, 1988.

[51] William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[52] Toyohisa Kaneko and Bede Liu. Accumulation of round-off error in fast fourier transforms. *Journal of the ACM (JACM)*, 17(4):637–654, 1970.

[53] J. E. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. C. Bates, G. Danabasoglu, J. Edwards, M. Holland, P. Kushner, J. F. Lamarque, D. Lawrence, K. Lindsay, A. Middleton, E. Munoz, R. Neale, K. Oleson, L. Polvani, and M. Vertenstein. The Community Earth System Model (CESM) large ensemble project: A community resource for studying climate change in the presence of internal climate variability. *Bulletin of the American Meteorological Society*, 96(8):1333–1349, 2015.

[54] Jeongnim Kim, Andrew D Baczewski, Todd D Beaudet, Anouar Benali, M Chandler Bennett, Mark A Berrill, Nick S Blunt, Edgar Josué Landinez Borda, Michele Casula, David M Ceperley, Simone Chiesa, Bryan K Clark, Raymond C Clay, Kris T Delaney, Mark Dewing, Kenneth P Esler, Hongxia Hao, Olle Heinonen, Paul R C Kent, Jaron T Krogel, Ilkka Kylänpää, Ying Wai Li, M Graham Lopez, Ye Luo, Fionn D Malone, Richard M Martin, Amrita Mathuriya, Jeremy McMinis, Cody A Melton, Lubos Mitas, Miguel A Morales, Eric Neuscamman, William D Parker, Sergio D Pineda Flores, Nichols A Romero, Brenda M Rubenstein, Jacqueline A R Shea, Hyeondeok Shin, Luke Shulenburger, Andreas F Tillack, Joshua P Townsend, Norm M Tubman, Brett Van Der Goetz, Jordan E Vincent, D ChangMo Yang, Yubo Yang, Shuai Zhang, and Luning Zhao. QMCPACK: an open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19):195901, 2018.

[55] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

[56] Francois Lekien and J Marsden. Tricubic interpolation in three dimensions. *International Journal for Numerical Methods in Engineering*, 63(3):455–471, 2005.

[57] Dong Li, Zizhong Chen, Panruo Wu, and Jeffrey S Vetter. Rethinking algorithm-based fault tolerance with a cooperative software-hardware approach. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2013.

[58] Shaomeng Li, Stanislaw Jaroszynski, Scott Pearse, Leigh Orf, and John Clyne. Vapor: A visualization package tailored to analyze simulation data in earth system science. 2019.

[59] Shaomeng Li, Nicole Marsaglia, Christoph Garth, Jonathan Woodring, John Clyne, and Hank Childs. Data reduction techniques for simulation, visualization and data analysis. In *Computer Graphics Forum*, volume 37, pages 422–447. Wiley Online Library, 2018.

[60] Sihuan Li, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Optimizing lossy compression with adjacent snapshots for n-body simulation data. In *2018 IEEE International Conference on Big Data*. IEEE, 2018.

[61] Sihuan Li, Hongbo Li, Xin Liang, Jieyang Chen, Elisabeth Giem, Kaiming Ouyang, Kai Zhao, Sheng Di, Franck Cappello, and Zizhong Chen. Ft-isort: efficient fault tolerance for introsort. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 71. ACM, 2019.

[62] Xin Liang, Jieyang Chen, Dingwen Tao, Sihuan Li, Panruo Wu, Hongbo Li, Kaiming Ouyang, Yuanlai Liu, Fengguang Song, and Zizhong Chen. Correcting soft errors online in fast fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 30. ACM, 2017.

[63] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression quality based on an optimized hybrid prediction model. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 33. ACM, 2019.

[64] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. Efficient transformation scheme for lossy data compression with point-wise relative error bound. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 179–189, 2018.

[65] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data*. IEEE, 2018.

[66] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. Improving performance of data dumping with lossy compression for scientific simulation. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[67] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[68] Peter Lindstrom. Error distributions of lossy floating-point compressors. *Joint Statistical Meetings*, pages 2574–2589, 2017.

[69] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[70] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Matthew Wolf, Tong Liu, and Zhenbo Qiao. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 348–357, May 2018.

[71] LZ4. `https://github.com/lz4/lz4`, 2019. Online.

[72] Ibrahim Numanagic, James K Bonfield, Faraz Hach, Jan Voges, Jorn Ostermann, Claudio Alberti, Marco Mattavelli, and S Cenk Sahinalp. Comparison of high-throughput sequencing data compression tools. *Journal of Nature Methods*, 13:1005–1008, 2016.

[73] NYX simulation. `https://amrex-astro.github.io/Nyx`, 2019. Online.

[74] Choong Gun Oh and Hee Yong Youn. On concurrent error detection, location, and correction of fft networks. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 596–605. IEEE, 1993.

[75] Choong Gun Oh, Hee Yong Youn, and Vijay K Raj. An efficient algorithm-based concurrent error detection for fft networks. *Computers, IEEE Transactions on*, 44(9):1157–1162, 1995.

[76] Laercio L Pilla, P Rech, F Silvestri, Christopher Frost, Philippe Olivier Alexandre Navaux, M Sonza Reorda, and Luigi Carro. Software-based hardening strategies for neutron sensitive fft algorithms on gpus. *Nuclear Science, IEEE Transactions on*, 61(4):1874–1880, 2014.

[77] EXAALT project. https://www.exascaleproject.org/project/exaalt-molecular-dynamics-at-the-exascale-materials-science/, 2019. Online.

[78] EXAFEL project. https://www.exascaleproject.org/project/exafel-data-analytics-exascale-free-electron-lasers/, 2019. Online.

[79] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conference (DCC'06)*, pages 133–142. IEEE, 2006.

[80] Piyush Sao and Richard W. Vuduc. Self-stabilizing iterative solvers. In *ScalA*, pages 4:1–4:8, 2013.

[81] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. Exploration of lossy compression for application-level checkpoint/restart. In *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 914–922, 2015.

[82] Scientific Data Reduction Benchmark. https://sdrbench.github.io/, 2019. Online.

[83] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. Characterizing the impact of soft errors on iterative methods in scientific computing. In *ICS*, pages 152–161, 2011.

[84] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *ICS*, pages 69–78, 2012.

[85] Jerome M Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[86] Joseph Sloan, Rakesh Kumar, and Greg Bronevetsky. Algorithmic approaches to low overhead fault detection for sparse linear algebra. In *DSN*, pages 1–12, 2012.

[87] Seung Woo Son, Zhengzhang Chen, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Data compression for the exascale computing era-survey. *Supercomputing Frontiers and Innovations*, 1(2):76–88, 2014.

[88] Miroslav Stoyanov and Clayton Webster. Numerical analysis of fixed point algorithms in the presence of hardware faults. *SIAM Journal on Scientific Computing*, 37(5):C532–C553, 2015.

[89] Bebop supercomputer. Available at https://www.lcrc.anl.gov/systems/resources/bebop, 2019. online.

[90] Theta supercomputer. Available at https://www.alcf.anl.gov/theta, 2019. online.

[91] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. In-depth exploration of single-snapshot lossy compression techniques for n-body simulations. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 486–493. IEEE, 2017.

[92] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium*, pages 1129–1139. IEEE, 2017.

[93] Dingwen Tao, Sheng Di, Hanqi Guo, Zizhong Chen, and Franck Cappello. Z-checker: A framework for assessing lossy compression of scientific data. *The International Journal of High Performance Computing Applications*, 0(0), 2017.

[94] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Fixed-psnr lossy compression for scientific data. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 314–318. IEEE, 2018.

[95] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Improving performance of iterative methods by lossy checkponting. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 52–65. ACM, 2018.

[96] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Optimizing lossy compression rate-distortion from automatic online selection between SZ and ZFP. *IEEE Trans. Parallel Distrib. Syst.*, 30(8):1857–1871, 2019.

[97] Dingwen Tao, Shuaiwen Leon Song, Sriram Krishnamoorthy, Panruo Wu, Xin Liang, Eddy Z Zhang, Darren Kerbyson, and Zizhong Chen. New-sum: A novel online abft scheme for general iterative methods. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 43–55. ACM, 2016.

[98] DL Tao and Carlos R. P. Hartmann. A novel concurrent error detection scheme for fft networks. *Parallel and Distributed Systems, IEEE Transactions on*, 4(2):198–221, 1993.

[99] David Taubman and Michael Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice.* Springer Publishing Company, Incorporated, 2013.

[100] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, IOPADS '99, pages 23–32, New York, NY, USA, 1999. ACM.

[101] Top500. `https://www.top500.org/lists/2018/11/`, 2018. Online.

[102] Andy Turner. Parallel i/o performance. `https://www.archer.ac.uk/training/virtual/2017-02-08-Parallel-IO/2017_02_ParallelIO_ARCHERWebinar.pdf`, 2019. Online.

[103] S. Crusan V. Vishwanath and K. Harms. Parallel i/o on mira. `https://www.alcf.anl.gov/files/Parallel_IO_on_Mira_0.pdf`, 2019. Online.

[104] Gregory K Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

[105] Sying-Jyan Wang and Niraj K Jha. Algorithm-based fault tolerance for fft networks. *Computers, IEEE Transactions on*, 43(7):849–854, 1994.

[106] Zhou Wang and Alan C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, Jan 2009.

[107] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[108] SCALE-LETKF weather model. `https://github.com/gylien/scale-letkf`, 2019. Online.

[109] C Weinstein. Roundoff noise in floating point fast fourier transform computation. *IEEE Transactions on Audio and Electroacoustics*, 17(3):209–215, 1969.

[110] Brent Welch. POSIX io extensions for HPC. In *4th USENIX Conference on File and Storage Technologies (FAST05)*, 2005.

[111] Panruo Wu and Zizhong Chen. Ft-scalapack: Correcting soft errors on-line for scalapack cholesky, qr, and lu factorization routines. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 49–60. ACM, 2014.

[112] Panruo Wu, Nathan DeBardeleben, Qiang Guan, Sean Blanchard, Jieyang Chen, Dingwen Tao, Xin Liang, Kaiming Ouyang, and Zizhong Chen. Silent data corruption resilient two-sided matrix factorizations. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 415–427. ACM, 2017.

[113] Panruo Wu, Chong Ding, Longxiang Chen, Feng Gao, Teresa Davies, Christer Karlsson, and Zizhong Chen. Fault tolerant matrix-matrix multiplication: correcting soft errors on-line. In *Proceedings of the second workshop on Scalable algorithms for large-scale systems*, pages 25–28. ACM, 2011.

[114] Panruo Wu, Qiang Guan, Nathan DeBardeleben, Sean Blanchard, Dingwen Tao, Xin Liang, Jieyang Chen, and Zizhong Chen. Towards practical algorithm based fault tolerance in dense linear algebra. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 31–42. ACM, 2016.

[115] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[116] Zstd. https://github.com/facebook/zstd/releases, 2019. Online.