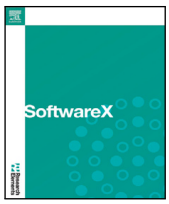
SoftwareX 24 (2023) 101590

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

MGARD: A multigrid framework for high-performance, error-controlled data compression and refactoring

Qian Gong [a,*], Jieyang Chen [b], Ben Whitney [f], Xin Liang [c], Viktor Reshniak [a], Tania Banerjee [d], Jaemoon Lee [d], Anand Rangarajan [d], Lipeng Wan [e], Nicolas Vidal [a], Qing Liu [g], Ana Gainaru [a], Norbert Podhorszki [a], Richard Archibald [a], Sanjay Ranka [d], Scott Klasky [a]

[a] Oak Ridge National Laboratory, USA
[b] University of Alabama at Birmingham, USA
[c] University of Kentucky, USA
[d] University of Florida, USA
[e] Georgia State University, USA
[f] University of Wisconsin Eau Claire, USA
[g] New Jersey Institute of Technology, USA

ARTICLE INFO

Keywords:
Error-controlled data compression
Data refactoring
I/O acceleration
Derived quantities preservation

ABSTRACT

We describe MGARD, a software providing MultiGrid Adaptive Reduction for floating-point scientific data on structured and unstructured grids. With exceptional data compression capability and precise error control, MGARD addresses a wide range of requirements, including storage reduction, high-performance I/O, and in-situ data analysis. It features a unified application programming interface (API) that seamlessly operates across diverse computing architectures. MGARD has been optimized with highly-tuned GPU kernels and efficient memory and device management mechanisms, ensuring scalable and rapid operations.

Code metadata

Current code version	1.5.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00502
Permanent link to Reproducible Capsule	codeocean.com/capsule/4683587
Legal Code License	Apache-2.0 license
Code versioning system used	git
Software code languages, tools, and services used	C++, CUDA, HIP, SYCL, OPENMP
Compilation requirements, operating environments	Software: NVCOMP, ZSTD. Hardware: NVIDIA GPU, AMD GPU, x86 CPU, ARM CPU, Power CPU
If available Link to developer documentation/manual	github.com/CODARcode/MGARD/blob/master/README.md
Support email for questions	jchen3@uab.edu or gongq@ornl.gov

1. Motivation and significance

In today's scientific landscape, large-scale scientific applications generate an overwhelming volume of data, surpassing the capabilities of network and storage systems. For instance, the Square Kilometer Array (SKA) telescope, designed to explore radio-waves from the early universe, is projected to deliver around 600 Petabytes of data per year to a network of SKA Regional Centers for ingestion and storage [1]. Despite this data deluge, modern parallel file systems (PFS) exhibit limited aggregated bandwidth, typically measured in several Terabytes per second. The throughput of Wide Area Network (WAN) for long-distance data transmission is even sluggish, usually in the range of several hundred Megabytes per second. A parallel trend has also emerged in artificial intelligence community, marked by growing demands for storage and memory resource to support the training of increasingly deeper, wider, and non-linear deep neural networks (DNN). Additionally, the efficiency of DNN operations is hindered by

* Corresponding author.
E-mail address: gongq@ornl.gov (Qian Gong).

https://doi.org/10.1016/j.softx.2023.101590
Received 31 July 2023; Received in revised form 6 October 2023; Accepted 14 November 2023
Available online 22 November 2023

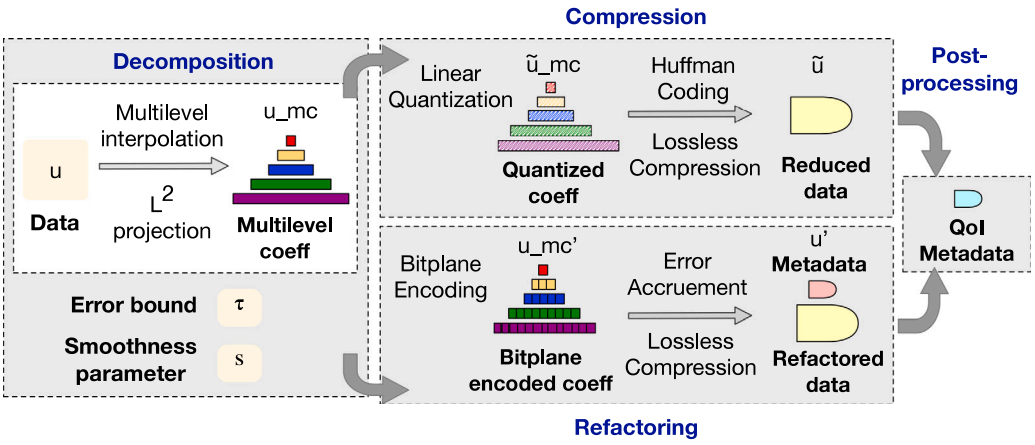


Fig. 1. The software pipeline overview illustrating the two primary functionalities of MGARD — compression and refactoring, both with precision error control.

rising communication costs associated with sharing model parameters during distributed training.

Compression has emerged as a promising solution to address the challenges posed by storage and I/O bandwidth limitations. The ideal compression approaches seek to reduce data size by several orders of magnitude while preserving its fidelity for reliable scientific use. The ability to refactor data into a multi-scale representation that aligns with the hierarchical architecture of storage tiers is also highly desirable. However, the presence of random mantissa with the floating-point representation of scientific data limits the compression ratios [2,3] with conventional entropy-based lossless compressors [4–7]. Alternative data reduction approaches, like sparse output rates, have their limitations too, potentially overlooking valuable scientific insights in unsaved timesteps.

Recently, lossy compression has garnered increased attention due to its effectiveness in reducing data stored in floating-point precision. A typical lossy compressor involves decorrelation, precision truncation, and lossless encoding steps, along with mathematical theories to control data distortion. An ideal lossy compressor for scientific data reduction should possess the following features: (1) strict error control with respect to different norms, (2) high throughput to avoid I/O bottlenecks, (3) portability on mainstream computing processors, (4) the ability to handle data defined on various grid structures, and (5) the capability to refactor data into multi-scales.

In this regard, several state-of-the-art lossy compressors have been developed. SZ [8], ZFP [9], TTHERSH [10], and FPZIP [11] offer APIs accepting L^2 or/and L^∞ error bound settings. SZ offers additional error controls for several types of quantities of interest (QoIs), including polynomials, logarithmic mapping, weighted sum, and critical point/isosurface [12,13]. In terms of the throughput, although SZ and ZFP provide high-performance libraries – cuSZ [14] and cuZFP [15] – on NVIDIA GPUs, they only support single precision and fixed-rate compression mode separately, resulting in limited usability and lower compression ratios. Moreover, these GPU-based compressors lack out-of-core support, requiring users to manually tile and fit data into GPU memory, impacting throughput performance. Additionally, existing error-bounded lossy compressors (e.g., SZ, ZFP, FPZIP, TTHERSH) are limited to data defined on uniformly spaced grids up to four dimensions.

Addressing these challenges, our present paper describes MGARD: the MultiGrid Adaptive Reduction for Data [16–18] a high-performance framework designed for compressing and refactoring scientific data defined on various grid structures while ensuring precise error control. By decomposing floating-point data into a hierarchical representation on multigrid and applying quantization, MGARD achieves exceptional compression capabilities for scientific data. Importantly, the induced information loss during compression is mathematically guaranteed by finite element theories, ensuring the trustworthiness of the compressed data for a wide range of scientific applications. MGARD offers refactoring functionality as an alternative to lossy compression for applications requiring near-lossless storage and the flexibility to access data in various scales. It supports refactoring data into a set of components representing hierarchical resolutions and precision, enabling users to incrementally retrieve and recompose them to any accuracy on demand. Moreover, MGARD's state-of-the-art implementation supports compressing and refactoring data defined on various mesh topologies and offers multi-resolution and multi-precision parametrization options. It delivers high performance and scalability on leadership high-performance computing (HPC) facilities, such as Summit and Frontier. Previous works have shown that the high-throughput compression on GPU helps accelerate the training of large-scale DNNs by reducing the communication latency [19]. Furthermore, DNNs trained using data reduced by error-bounded compressors exhibit little or no accuracy loss [20,21].

MGARD consists of GPU and CPU kernels. Implemented in C++11 [22], OpenMP [23], CUDA [24], HIP [25], and SYCL [26], MGARD leverages platform portability and embraces modern software engineering practices, including unit testing and continuous integration. The framework provides a unified application programming interface (API) with a level of abstraction focused on data reduction and reconstruction in scientific workflows. With built-in compile-time auto-tuning and runtime adaptive scheduling techniques, users can expect the best performance across different computing architectures. MGARD is part of the United States Department of Energy (DOE) Exascale Computing Project (ECP) software technology stack for data reduction [27,28], which solidifies its position as a crucial component in the advancement of data reduction technologies.

2. Software design

As illustrated in Fig. 1, the inputs to MGARD API consist of a data array `u`, user-prescribed error bound(s) τ, and a smoothness parameter `s`, which defines the norm of error bounds. MGARD comprises two primary modules: data compression and refactoring. Both modules start with a common practice, recursively decomposing `u` into a sequence of approximations at various levels of the multi-resolution hierarchy. This decomposition generates a multilevel representation, `u_mc`, which is better suited for compression and refactoring purposes.

The compression module involves a quantization stage where each component of `u_mc` is approximated by a multiple of a quantization bin width [29,30]. This linear quantization effectively transforms floating-point data into integers, facilitating efficient coding and ensuring that

the specified error bound for `u_mc` is met. On the other hand, the refactoring module encodes `u_mc` into precision segments at different levels of the multi-resolution hierarchy, utilizing bitplane encoding [31]. Both compression and refactoring modules employ the same set of error estimators for accuracy control, which are analogous to the posterior error estimators used in numerical analysis. These error estimators consider quantization errors or precision segments of multilevel coefficients as inputs, allowing error control in various metrics, norms, and linear Quantities of Interest (QoIs) [16–18].

In the final stage, the quantization and precision segments obtained from the compression and refactoring modules are compressed through lossless encoding and written to disk as a self-describing buffer containing all the necessary parameters for decompression and recomposition. The compressed/refactored representation may also undergo post-processing, when the preservation of non-linear QoIs is required. The refactoring module includes an additional step that collects errors in the precision segments of the multilevel coefficients. The recomposition module, operating in an inverse procedure to refactoring, employs a greedy algorithm to determine the retrieval order of precision segments. This strategy aims to fetch the most significant segment across all levels based on the previously accrued error estimators.

2.1. APIs

MGARD is designed with two levels of APIs to support the integration with different user applications and IO libraries.

2.1.1. High-level APIs

The high-level APIs offer an all-in-one compression and refactoring solution, providing users with a seamless integration experience with MGARD. Key features of the high-level APIs include:

- Unified APIs: MGARD offers a single set of APIs for compressing and refactoring. MGARD automatically optimizes the reduction and refactoring kernels for the targeted GPU or CPU architectures during the software installation stage, and utilizes the same APIs across various systems, enhancing code portability and ease of use.
- Self-describing format: The output of compression and refactoring APIs includes all the necessary information required by a decompressor/re-compositor to read and reconstruct data correctly. This encompasses vital details such as the code's version, error bounds employed, data topologies, and the type of lossless encoders utilized.
- Unified memory buffers on CPU/GPUs: MGARD automatically detects the locations of input/output buffers and handles the host-to-device data transfer internally, eliminating manual setup.
- Multi-device out-of-core processing: The high-level APIs can automatically detect and leverage multiple accelerator devices on a system. MGARD also boasts with an out-of-core optimization to manages memory overflow and inter-device data transfer. These functionalities are crucial for large-scale data processing, where GPUs often have smaller memory capacities compared to host CPUs.

2.1.2. Low-level APIs

The low-level APIs offer users complete control over the compression and refactoring processes, empowering them to customize the functionality based on their specific application needs. Key features of the low-level APIs include:

- Highly customizable code pipeline: The low-level APIs expose individual functions for each step within compression and refactoring, such as memory management and sub-operations. This level of granularity allows users to construct their own highly optimized compression/refactoring pipelines tailored to their application's requirements.
- Device asynchrony: The low-level APIs allow users to pipeline computation and cross-device data transfer so they will execute asynchronously. For example, MGARD operations on GPUs can be overlapped with the application's workload on CPUs. This opens up significant opportunities for users to optimize MGARD in tandem with their application's execution logic, leading to enhanced performance.

The dual-tiered API approach of MGARD ensures that users seeking a quick and easy integration with minimal effort and those requiring granular control over the compression and refactoring processes are both catered.

2.2. Software architecture

MGARD is meticulously designed to be highly functional, performant, portable, and extendable. This is achieved through a modularized software architecture with carefully designed abstraction layers for maximum portability. It has been successfully integrated into ADIOS – a high-performance parallel I/O framework with an extensive user community – as an inline compressor. This integration allows ADIOS users to write and compress data using MGARD in a single step. Fig. 2 provides an illustration of MGARD's software architecture. At the foundation of the architecture are device abstractions (green), which ensure the sustained functionality irrespective of underlying hardware features. One layer above (blue), MGARD incorporates a built-in auto-tuning module. This model automatically adjusts performance configurations, such as GPU thread block sizes, shared memory usage, and processor occupancy in the software installation stage, ensuring that MGARD operates efficiently on targeted hardware micro-architectures. The design of MGARD's auto-tuning module draws inspiration from techniques discussed in [32–36], primarily focusing on optimization at the kernel functions level. The subsequent layer (dark yellow) houses the central computation kernels used by the compression and refactoring processes. They serve as the foundational building blocks for MGARD's compression and refactoring pipelines (gray) to assemble with. These functionalities are exposed to users through low-level APIs. They provide users the flexibility to fine-tune compression/decompression pipelines according to their specific application needs. The separation between the low-level and high-level APIs is marked by the inclusion of out-of-core processing and metadata management (dark red). The out-of-core processing dynamically partitions data arrays into chunks that fit within the device memory, serializes the compression of large input arrays, and handles chunk data movement internally. On the other hand, the metadata management layer saves all information required for data reconstruction and recomposition in a self-describing format. The high-level APIs encapsulates underlying complexity into a single line of code for compression, decompression, refactoring, and recomposition separately (as illustrated in examples later presented in Section 3).Users can integrate high-level APIs into their applications without delving into the intricacies of MGARD's data reduction and refactoring processes.

2.3. Software functionalities

MGARD primarily focuses on two functionalities: compression and refactoring, and mathematically guarantees that the information loss induced by compression and refactoring adheres to user-prescribed error tolerance. The compression functions can promote scientific discoveries by releasing storage burden so simulation/devices can output data at enhanced resolutions/frequencies [37]. They could also accelerate I/O due to MGARD's high-throughput on GPUs. As data volumes and velocities continue to increase, scientists require tools to incrementally retrieve, move, and process reduced data based on scientific priorities and resource constraints. MGARD's refactoring functionality empowers users to make trade-offs among uncertainty, speed, and

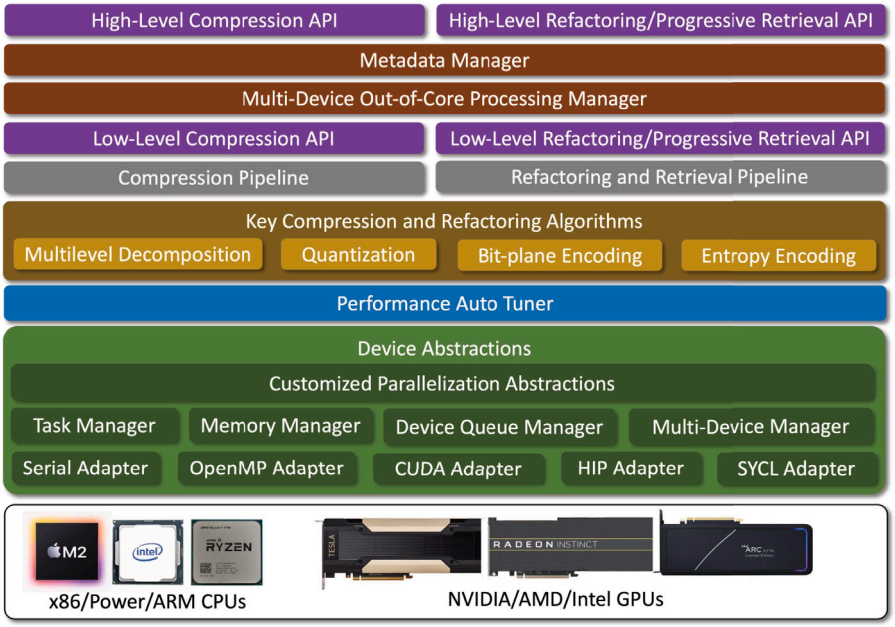


Fig. 2. Software architecture of MGARD. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

resource utilization. Furthermore, scientific data often undergoes a process where it is compressed at one place/device and then transferred to different sites/devices for various analyses. MGARD's unified API facilitates cross-platform data sharing through its design, encompassing functions, and format portability.

3. Illustrative examples

The following examples illustrate MGARD's compression and refactoring APIs. MGARD employs the same set of APIs for backend functions running on various GPU and CPU architectures and will automatically switch to the most optimal processors available.

Listing 1 showcases MGARD's high-level APIs for compression and reconstruction. For compression API, the inputs are data array, shape, data type, configuration (e.g., error messages), error bound, error bound type (e.g., REL or ABS for relative or absolute error), and the smoothness parameter. The outputs are compressed data and bytes, which will be stored in the `compressed_array` and `out_byte`. The compression ratio is obtained by dividing `in_byte` and `out_byte`. The decompression API takes the `compressed_array` and bytes as the inputs and stores the decompressed results in `decompressed_array`. One noteworthy aspect is that MGARD's interface automatically detects the available device memory and location of buffers holding `in_array`, `compressed_array`, and `decompressed_array`. When GPUs devices are used, the high-level APIs dynamically schedules the out-of-core processing and manages host-to-device data transfer internally.

```
#include "mgard/compress_x.hpp"

// prepare data buffers
mgard_x::DIM num_dims = 3;
mgard_x::SIZE n1, n2, n3;
std::vector<mgard_x::SIZE> shape{n1, n2, n3};
mgard_x::SIZE in_byte = n1 * n2 * n3 * sizeof(
    double);
mgard_x::SIZE out_byte;
//... load data into in_array
double *in_array = ...;
void *compressed_array = NULL;
void *decompressed_array = NULL;
// tol: error tolerance
// s: smoothness parameter
double tol = 0.01, s = 0;

// MGARD config parameters
mgard_x::Config config;

// Compressing with high level API
mgard_x::compress(num_dims, mgard_x::data_type::
    Double, shape, tol, s, mgard_x::
    error_bound_type::REL, in_array,
    compressed_array, out_byte, config, false);

// Decompressing with high level API
mgard_x::decompress(compressed_array, out_byte,
    decompressed_array, config, false);
```

Listing 1: MGARD data compression and decompression API example

Listing 2 demonstrates how to refactor and incrementally recompose data using MGARD's high-level APIs. The refactoring API returns a metadata file and the compressed resolution/precision segments. Lines 23-42 illustrate the recomposition process. It commences with a coarse representation of the data, then sequentially retrieves partial segments that lead to the next level of precision/resolution.

```
#include "mgard/mdr_x.hpp"
...

// prepare data buffers
mgard_x::DIM num_dims = 3;
mgard_x::SIZE n1, n2, n3;
std::vector<mgard_x::SIZE> shape{n1, n2, n3};
mgard_x::SIZE in_byte = n1 * n2 * n3 * sizeof(
    double);
mgard_x::SIZE out_byte;
//... load data into in_array
double *in_array = ...;

mgard_x::Config config;
mgard_x::MDR::RefactoredMetadata
    refactored_metadata;
```

```
mgard_x::MDR::RefactoredData refactored_data;

// Refactor with high level API
mgard_x::MDR::MDRefactor(D, mgard_x::data_type::
    Double, shape, in_array, refactored_metadata
    , refactored_data, config, false);

// Save refactored_metadata and refactored_data
    to files
...

mgard_x::MDR::ReconstructedData
    reconstructed_data;
// Read in refactored_metadata from file
...
// Progressively reconstruct for each error
    bound
for (double tol : tolerances) {
    // Specify error bound and smoothness
    parameter for each subdomain
    for (auto &metadata : refactored_metadata.
    metadata) {
          metadata.requested_tol = tol;
          metadata.requested_s = s;
    }
    // Determine required data components for
    reconstruction
    mgard_x::MDR::MDRequest(refactored_metadata,
     config);
    // Read in required data components from
    files
    ...
    // Reconstruct with high level API
    mgard_x::MDR::MDReconstruct(
    refactored_metadata, refactored_data,
    reconstructed_data, config, false,
    original_data);

    // reconstructed_data now contains
    progressively reconstructed data
    double out_data = reconstructed_data.data;
}
```

Listing 2: MGARD data refactoring API example

4. Application impact

The MGARD team has worked with application scientists from a variety of research communities to alleviate their storage and I/O challenges.

4.1. Plasma physics

- **XGC:** The X-point included Gyrokinetic Code (XGC) is a fusion physics code specialized in simulating plasma dynamics in the edge region of a tokamak reactor [38,39]. We compressed the 5D particle distribution function (pdf) generated by XGC simulating an ITER-scale experiment [40], and evaluated the errors in five derived QoIs (density, parallel/vertical temperatures, and two flux surface averaged momentums). Fig. 3 illustrates that the MGARD with QoI post-processing can reduce the data storage for up to 200× and 290× with the relative L^2 errors in all QoIs below 1×10^{-14} and 1×10^{-8} separately, whereas the compression without QoI optimization exhibits a relative L^2 error of approximately 1×10^{-2} given the same compression ratios. Noted that λ represents the set of Lagrange multipliers obtained through a convex optimization program aiming to reduce QoI errors in each sub data-domain. λ can be further quantized or truncated to increase compression ratios. Readers can find more MGARD studies on XGC simulation data in [41–43].

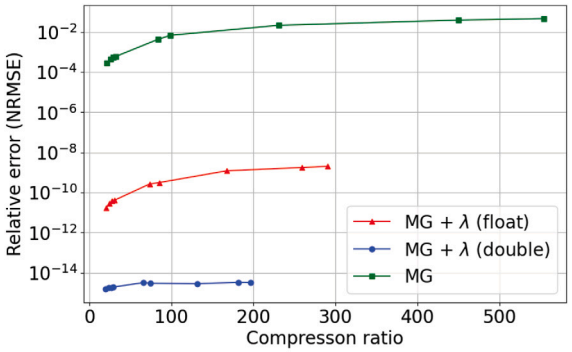


Fig. 3. Illustration of errors in QoIs derived from the XGC f-data lossy compressed by MGARD with QoI post-processing.

4.2. Earth and cosmological science

- **NYX:** NYX is an AMR-based cosmological hydrodynamics simulation code developed at Lawrence Berkeley National Laboratory [44]. Fig. 4 presents the compression and decompression throughput of MGARD and the GPU implementation of two other state-of-the-art lossy compressors: cuSZ and ZFP-CUDA. The throughput data was obtained from the Summit supercomputer [45], where each compute node hosts six NVIDIA V100 GPUs. For our evaluation, we fed each GPU with 15 GB of NYX data, using a relative L^2 error bound of 1×10^{-3} for data compression. Throughout the evaluation, MGARD surpassed other GPU-accelerated lossy compressors in terms of performance due to its efficient compression kernels and multi-GPU pipeline optimization. Fig. 5 illustrates how data compression accelerated I/O throughput in NYX simulations. Using the same setting as the experiments in Fig. 4, we compare the combined time spent on compression/decompression and reading/writing the reduced data against the time spent on reading and writing the uncompressed data. The results suggest that data compression can effectively reduce the I/O cost, and MGARD exhibits the most significant improvement among the three lossy compressors.
- **E3SM:** The Energy Exascale Earth System Model is a state-of-the-science Earth's climate model used to investigate energy-relevant science [46]. Due to storage constraints, E3SM currently outputs model data at the 6-hourly interval instead of the physical temporal resolution, which is 15 min. In Fig. 6(b) [37], the tropical cyclone (TC) tracks detected from data outputted at an hourly interval are compared with TC tracks obtained from the same set of data, lossy compressed using MGARD with distinct error bounds tailored to regions with varying degrees of turbulence. Concurrently, Fig. 6(a) illustrates TC tracks detected from data outputted at a 6-hourly rate. Despite the lossy compression of hourly data requiring only 1/4 of the storage compared to the uncompressed 6-hourly data, a notable enhanced accuracy is achieved.

4.3. Radio astronomy

- **SKA:** The Square Kilometer Array (SKA) [47] hosts two of the world's largest radio telescope arrays, archiving approximately 300 petabytes of data per year. Early exploration work has indicated that MGARD can compress radio astronomy data by approximately 20× without introducing structural artifacts. Ongoing efforts aim to integrate data reduction into the Casacore Table Data System's I/O pipeline.

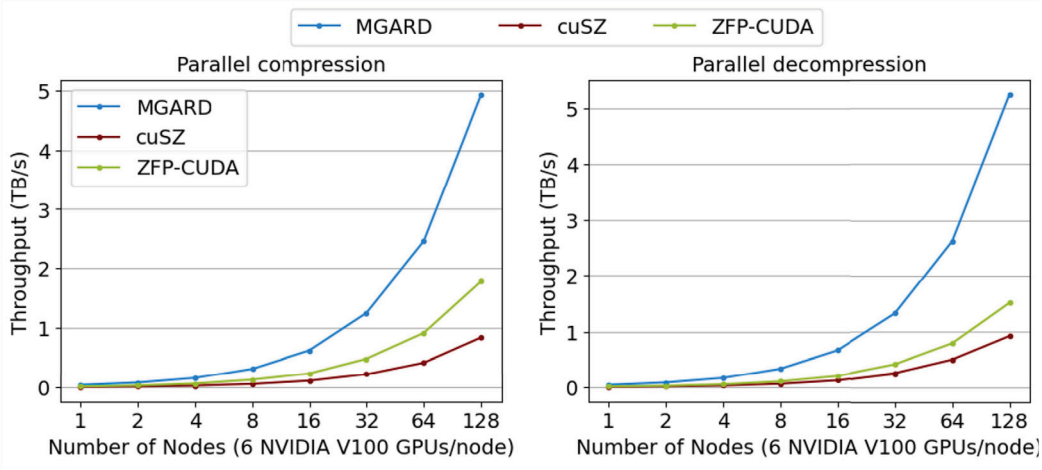


Fig. 4. Comparing the throughput performance of compression and decompression provided by MGARD, cuSZ, and ZFP-CUDA on OLCF Summit nodes, using NYX data and a relative error bound of 1×10^{-3}.

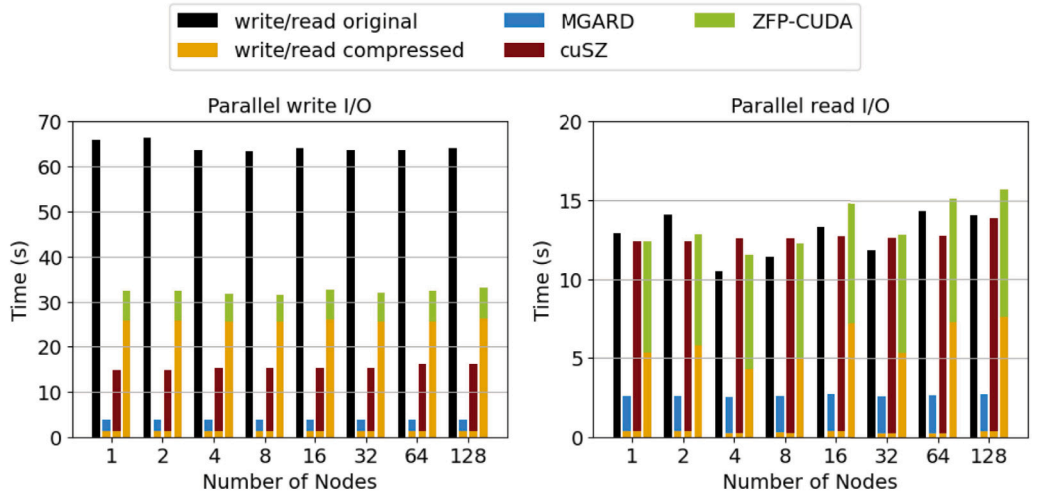


Fig. 5. Comparing the end-to-end I/O time for reading and writing both compressed and uncompressed NYX data using MGARD, cuSZ, and ZFP-CUDA. Each node accommodates six NVIDIA V100 GPUs.

By showcasing the impact of MGARD in diverse applications, it is evident that MGARD significantly tackles data storage and I/O challenges in the workflow of large-scale scientific experiments while ensuring the preservation of vital scientific insights.

5. Conclusion

MGARD has been designed to tackle storage, I/O, and data analysis challenges for scientific applications. With novel multilevel decomposition, advanced encoding, and rigorous error control techniques, MGARD can compress data into a greatly reduced representation or refactor the data into a format supporting incremental retrieval. A well-developed mathematical foundation allows MGARD to provide error bounds not just on the raw data but also on QoIs derived from the lossy reduced data. With the mathematically proved theories and solid empirical evaluations, MGARD provides compression that will not compromise the scientific validity and utility of data. The refactoring capability of MGARD serves as an alternative to the single-error-bounded compression for users who require near-lossless data storage but may retrieve data at varied precisions/resolutions. Beyond trustworthiness, MGARD can accelerate data movement and in-situ data analytics with its extensively optimized CPU and GPU implementations, and is portable so that data compression and refactoring can operate on mainstream computing processors.

Declaration of competing interest

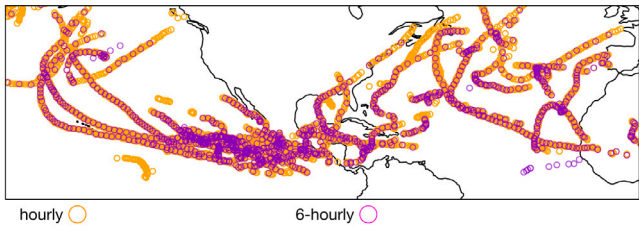
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

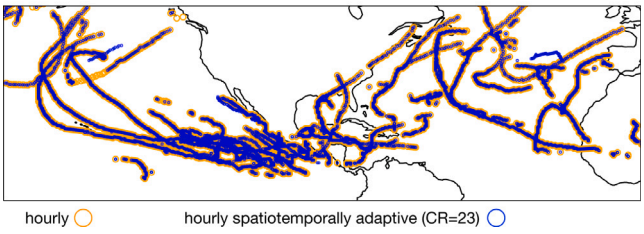
Data will be made available on request.

Acknowledgments

This research was supported in part by the Exascale Computing Project CODAR (17-SC-20-SC) of the US Department of Energy (DOE), the DOE's Advanced Scientific Research Office (ASCR) research project SIRIUS-2, United States, and the DOE's RAPIDS-2 SciDAC project under contract number DE-AC05-00OR22725. In addition, this research used



(a) Visualize the TC tracks found in hourly and 6-hourly data (temporal decimation rate = 6).



(b) Visualize the TC tracks found in hourly and hourly spatiotemporally compressed data (compression ratio = 23).

Fig. 6. Global distributing of TC tracks detected in hourly, 6-hourly, and spatiotemporally adaptive reduced hourly data over one year time span.

resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of DOE under Contract Numbers DE-AC05-00OR22725.

References

[1] Sánchez-Expósito S, Luna S, Garrido J, Moldón J, Verdes-Montenegro L, Darriba L. SKA regional centre prototype at IAA-CSIC: building an open science platform based on cloud services. 2021.
[2] Son SW, Chen Z, Hendrix W, Agrawal A, Liao W-k, Choudhary A. Data compression for the exascale computing era-survey. Supercomput Front Innov 2014;1(2):76–88.
[3] Lindstrom P, Isenburg M. Fast and efficient compression of floating-point data. IEEE Trans Vis Comput Graph 2006;12(5):1245–50.
[4] Burtscher M, Ratanaworabhan P. FPC: A high-speed compressor for double-precision floating-point data. IEEE Trans Comput 2008;58(1):18–31.
[5] Collet Y. RFC 8878: Zstandard compression and the'application/zstd'media type. RFC Editor; 2021.
[6] Deutsch P, et al. GZIP file format specification version 4.3. 1996.
[7] The nvCOMP library provides fast lossless data compression and decompression using a GPU, URL https://github.com/NVIDIA/nvcomp.
[8] Zhao K, Di S, Dmitriev M, Tonellot T-LD, Chen Z, Cappello F. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In: 2021 IEEE 37th international conference on data engineering. IEEE; 2021, p. 1643–54.
[9] Lindstrom P. Fixed-rate compressed floating-point arrays. IEEE Trans Vis Comput Graph 2014;20(12):2674–83.
[10] Ballester-Ripoll R, Lindstrom P, Pajarola R. TTHRESH: Tensor compression for multidimensional visual data. IEEE Trans Vis Comput Graph 2019;26(9):2891–903.
[11] Lindstrom PG. Fpzip. Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States); 2017.
[12] Liang X, Di S, Cappello F, Raj M, Liu C, Ono K, et al. Toward feature-preserving vector field compression. IEEE Trans Vis Comput Graphics 2022.
[13] Jiao P, Di S, Guo H, Zhao K, Tian J, Tao D, et al. Toward quantity-of-interest preserving lossy compression for scientific data. Proc VLDB Endow 2022;16(4):697–710.
[14] Tian J, Di S, Zhao K, Rivera C, Fulp MH, Underwood R, et al. Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data. 2020, arXiv preprint arXiv:2007.09625.
[15] Experimental CUDA port of zfp compression, URL https://github.com/mclarsen/cuZFP.
[16] Ainsworth M, Tugluk O, Whitney B, Klasky S. Multilevel techniques for compression and reduction of scientific data–the univariate case. Comput Vis Sci 2018;19(5):65–76.
[17] Ainsworth M, Tugluk O, Whitney B, Klasky S. Multilevel techniques for compression and reduction of scientific data—The multivariate case. SIAM J Sci Comput 2019;41(2):A1278–303.
[18] Ainsworth M, Tugluk O, Whitney B, Klasky S. Multilevel techniques for compression and reduction of scientific data–quantitative control of accuracy in derived quantities. SIAM J Sci Comput 2019;41(4):A2146–71.
[19] Zhou Q, Anthony Q, Xu L, Shafi A, Abduljabbar M, Subramoni H, et al. Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication. In: 2023 IEEE international parallel and distributed processing symposium. IEEE; 2023, p. 134–44.
[20] Grabek J, Cyganek B. An impact of tensor-based data compression methods on deep neural network accuracy. Ann Comput Sci Inf Syst 2021;26:3–11.
[21] Jin S, Zhang C, Jiang X, Feng Y, Guan H, Li G, et al. Comet: A novel memory-efficient deep learning training framework by using error-bounded lossy compression. 2021, arXiv preprint arXiv:2111.09562.
[22] Stroustrup B. The C++ programming language. Pearson Education; 2013.
[23] The OpenMP programming model, URL https://www.openmp.org.
[24] The CUDA programming language, URL https://developer.nvidia.com/cuda-toolkit.
[25] The HIP programming language, URL https://docs.amd.com/projects/HIP/en/docs-5.3.0/user_guide/programming_manual.html.
[26] The SYCL programming language, URL https://www.khronos.org/sycl/.
[27] Kothe D, Lee S, Qualters I. Exascale computing in the United States. Comput Sci Eng 2018;21(1):17–29.
[28] Messina P. The exascale computing project. Comput Sci Eng 2017;19(3):63–7.
[29] Tao D, Di S, Chen Z, Cappello F. 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. IEEE; 2017, p. 1129–39.
[30] Liang X, Whitney B, Chen J, Wan L, Liu Q, Tao D, et al. Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction. IEEE Trans Comput 2021;71(7):1522–36.
[31] Schwartz JW, Barker RC. Bit-plane encoding: A technique for source encoding. IEEE Trans Aerosp Electron Syst 1966;(4):385–92.
[32] Jiang C, Snir M. Automatic tuning matrix multiplication performance on graphics hardware. In: 14th International conference on parallel architectures and compilation techniques. IEEE; 2005, p. 185–94.
[33] Tillet P, Cox D. Input-aware auto-tuning of compute-bound HPC kernels. In: Proceedings of the international conference for high performance computing, networking, storage and analysis. 2017, p. 1–12.
[34] Li Y, Dongarra J, Tomov S. A note on auto-tuning GEMM for GPUs. In: Computational science–ICCS 2009: 9th international conference Baton Rouge, la, USA, May 25-27, 2009 Proceedings, Part I 9. Springer; 2009, p. 884–92.
[35] Cuenca J, Giménez D, González J. Architecture of an automatically tuned linear algebra library. Parallel Comput 2004;30(2):187–210.
[36] Whaley RC, Dongarra JJ. Automatically tuned linear algebra software. In: SC'98: Proceedings of the 1998 ACM/IEEE conference on supercomputing. IEEE; 1998, p. 38.
[37] Gong Q, Zhang C, Liang X, Reshniak V, Chen J, Rangarajan A, et al. Spatiotemporally adaptive compression for scientific dataset with feature preservation – A case study on simulation data with extreme climate events analysis. In: Proceedings of the 19th IEEE International Conference on E-Science. 2023.
[38] Chang C-S, Ku S. Spontaneous rotation sources in a quiescent tokamak edge plasma. Phys Plasmas 2008;15(6):062510.
[39] Ku S, Chang C-S, Diamond PH. Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. Nucl Fusion 2009;49(11):115021.
[40] Claessens M. ITER: The giant fusion reactor. Springer; 2020.
[41] Gong Q, Liang X, Whitney B, Choi JY, Chen J, Wan L, et al. Maintaining trust in reduction: Preserving the accuracy of quantities of interest for lossy compression. In: Smoky Mountains Computational Sciences and Engineering Conference. Springer; 2021, p. 22–39.
[42] Lee J, Gong Q, Choi J, Banerjee T, Klasky S, Ranka S, et al. Error-bounded learned scientific data compression with preservation of derived quantities. Appl Sci 2022;12(13):6718.
[43] Banerjee T, Choi J, Lee J, Gong Q, Wang R, Klasky S, et al. An algorithmic and software pipeline for very large scale scientific data compression with error guarantees. In: 2022 IEEE 29th international conference on high performance computing, data, and analytics. IEEE; 2022, p. 226–35.
[44] Sexton J, Lukic Z, Almgren A, Daley C, Friesen B, Myers A, et al. Nyx: A massively parallel amr code for computational cosmology. J Open Source Softw 2021;6(63):3068.
[45] Summit Supercomputer, URL https://www.olcf.ornl.gov/summit.
[46] Caldwell PM, Mametjanov A, Tang Q, Van Roekel LP, Golaz J-C, Lin W, et al. The DOE E3SM coupled model version 1: Description and results at high resolution. J Adv Modelling Earth Syst 2019;11(12):4095–146.
[47] van Diepen GN. Casacore table data system and its use in the MeasurementSet. Astron Comput 2015;12:174–80.