DOI: 10.14529/jsfi200101

State of the Art and Future Trends in Data Reduction for High-Performance Computing

Kira Duwe[1]***, Jakob Lüttgau***[1]***, Georgiana Mania***[2]***, Jannek Squar***[1]***,***
Anna Fuchs[1]***, Michael Kuhn***[1]***, Eugen Betke***[3]***, Thomas Ludwig***[3]

Research into data reduction techniques has gained popularity in recent years as storage capacity and performance become a growing concern. This survey paper provides an overview of leveraging points found in high-performance computing (HPC) systems and suitable mechanisms to reduce data volumes. We present the underlying theories and their application throughout the HPC stack and also discuss related hardware acceleration and reduction approaches. After introducing relevant use-cases, an overview of modern lossless and lossy compression algorithms and their respective usage at the application and file system layer is given. In anticipation of their increasing relevance for adaptive and in situ approaches, dimensionality reduction techniques are summarized with a focus on non-linear feature extraction. Adaptive approaches and in situ compression algorithms and frameworks follow. The key stages and new opportunities to deduplication are covered next. An unconventional but promising method is recomputation, which is proposed at last. We conclude the survey with an outlook on future developments.

Keywords: data reduction, lossless compression, lossy compression, dimensionality reduction, adaptive approaches, deduplication, in situ, recomputation, scientific data set.

Introduction

Breakthroughs in science are increasingly enabled by supercomputers and large scale data collection operations. Applications span the spectrum of scientific domains from fluid-dynamics in climate simulations and engineering, to particle simulations in astrophysics, quantum mechanics and molecular dynamics, to high-throughput computing in biology for genome sequencing and protein-folding. More recently, machine learning augments the capabilities of researchers to sift through large amounts of data to find hidden patterns but also to filter unwanted information.

Unfortunately, the development of technologies for storage and network throughput and capacity does not match data generation capabilities, ultimately making I/O a now widely anticipated bottleneck. This is only in part a technical problem, as technologies to achieve arbitrary aggregate throughput performance and capacity exist but cannot be deployed economically. Besides being too expensive at the time, their energy consumption poses a challenge in exascale systems adding to the operational cost [51]. As data centers are expected to become a major contributor to global energy consumption [133], data reduction techniques are an important building block for efficient data management.

Also, to capture as much data as possible as efficiently as possible, they are going to become far more important in the future. Especially as multiple projections across scientific domains estimate increasing data volumes for a variety of reasons: For one, Data generation capabilities are on the rise, due to improving compute capabilities as supercomputers become more powerful but also more broadly available. The increase in CPU performance encourages researchers to increase model resolution, but also to consider more simulations for uncertainty quantification,

[1]Hamburg University, Hamburg, Germany
[2]Deutsches Elektronen-Synchrotron (DESY), Hamburg, Germany
[3]German Climate Computing Center (DKRZ), Hamburg, Germany

both increasing the amount of generated data. Higher-resolution instrument and detector outputs, in addition to an exploding number of small scale measurement stations, are a second factor leading to increased data ingest. This shows in particular in remote sensing for earth observation and astronomy as well in detectors used in high-energy physics.

In many scientific contexts, data is stored in self-describing data formats such as NetCDF and HDF5. Some subdomains like bioinformatics may also employ text-based (blast, fasta) formats, that can be handled similarly with respect to compression. But to cope with the magnitude of the data, established techniques such as compression on written data are not sufficient. One alternative growing in popularity are in situ approaches, which bypass the storage systems for many post-processing tasks. Often in situ lossy compression is applied as data is generated, but more advanced in situ analysis relies on triggers preserving only records for essential events. In some cases, data volumes are reduced by several orders of magnitude [60]. CERN, for example, employs both combinatorial in situ and off situ techniques to filter and to track the particles produced in the proton-proton collision [15]. Selecting only the interesting events worthy of offline analysis means keeping 1/200000 events, which occur every second [60]. In situ processing can also benefit data visualization. At the German Climate Computing Center (DKRZ), use cases for in situ techniques include data reduction as well as feature detection, extraction, and tracking, which are also used to steer simulation runs [123]. Uptake in situ methods makes image data formats more relevant. Thus, our survey also covers data reduction for images.

Besides in place reduction at the application layer, optimization and data reduction opportunities can be found along the data path. A typical HPC stack spans multiple I/O layers including parallel distributed files systems, the network, and node-local storage. Additional leveraging points can be found in the memory hierarchy, spanning caches, RAM, NVRAM, NVMe down to HDDs and tape for long-term storage.

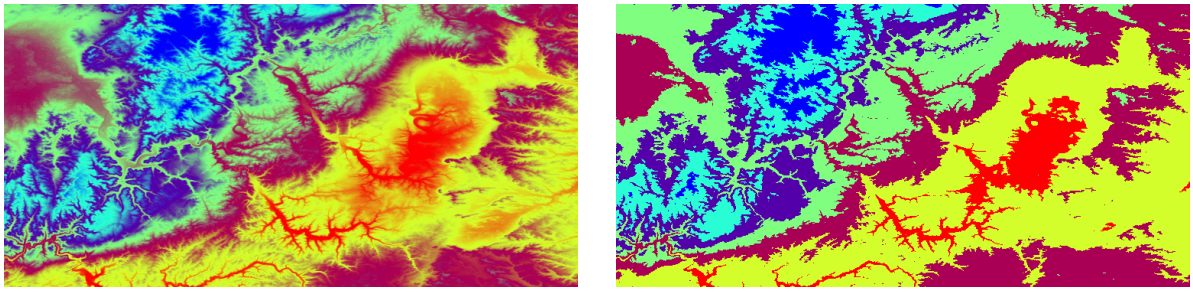
This survey covers a large variety of data reduction techniques. Mathematical backgrounds are, therefore, not covered in-depth. Additional details about fundamental compression techniques are discussed by Li et al. [82]. We extend their work by presenting additional frameworks, tools, and algorithms for compression. Some of them emerged in the last two years. In addition, we evaluate a more extensive variety of data reduction techniques considering dimension reduction, adaptive approaches, deduplication, in situ analysis, and recomputation.

The remainder of this paper is structured as follows: In section 1, an overview of lossless as well as lossy compression algorithms is given. Section 2 introduces dimensionality reduction techniques, forming the basis for the adaptive approaches described in section 3. In section 4, deduplication is explained. Section 5 details the algorithms and frameworks for in situ analysis. Recomputation approaches are collected in section 6. An outlook on future developments is provided in section 7.

1. Compression

Compression reduces the size of a dataset by removing redundancies to maximize its entropy. The ratio between the original and compressed data is denoted as the compression ratio (CR), where a higher ratio is better. Lossless compression is used whenever the reconstruction has to be byte-exact and has been widely explored in scientific domains. In general, the decompression is faster than the compression, which often fits the typical HPC workflow: Data is calculated and compressed once but read and decompressed several times [93]. Compression can be integrated into different system layers, such as the application layer or the file system and block layer. While

the user can decide whether to deal with lossy or lossless compression at the application layer, transparent compression within the system out of the user's control has to be always lossless. Data like source codes or binaries must be compressed only lossless since their reconstruction must be accurate.



(a) Before truncation (32-bit floating-point) (b) After truncation (8-bit integer)

Figure 1. Truncation reduces data size (conversion from 32-bit floats into 8-bit integers results in a factor of 4) but also leads to a visible degredation of data quality

Lossless compression usually provides a poor CR for HPC applications due to their extensive usage of floating-point data [77]. Relief can be achieved by using lossy compression since it allows to trade in data quality for data size. Many techniques for lossy compression have been developed with a focus on multimedia data such as audio or image data, but they can be reused for scientific data too [82]. However, losing data quality is not acceptable for all users and limits possible use cases. Figure 1 gives an example of a crude quantization: The original 32-bit floating-point data is mapped to 8-bit integers. The file size reduction is about a factor of 4 and comes at the price of a visible degradation of the data quality.

1.1. Lossless Compression

Lossless compression at the storage end is one of the obvious use cases in HPC systems, which aims for more storage capacity. While some of the local file systems like btrfs [83], NTFS or ZFS [154] provide compression services, the distributed parallel file systems used in the HPC field are often limited concerning this service. Ceph and GlusterFS both run in user space and offer server-side compression. Spectrum Scale (previously GPFS) by IBM also offers compression of cold data with zlib and recently introduced LZ4 for sequential read workload [13]. The most popular HPC file system Lustre indirectly benefits from compression when using the ZFS backend, which supports multiple compression algorithms. The data is then kept compressed not only on disk, which saves storage space but also in ZFS's own cache, which allows more diskless I/O. Lately, ZFS also gained support for hardware compression [62].

Compression can also improve the relative network throughput, I/O completion time and therefore speed up application runtimes. IOFSL is a project of Argonne National Laboratory aiming to provide a software layer at the file system interface. It has been extended by compression services, which can increase network bandwidth [152]. Another improvement for network throughput has been achieved by implementing an optimization of the two-phase collective I/O technique for ROMIO, the most popular MPI-IO implementation [45]. By reducing the data from group sources, Mellanox switches allow the collective operation processing to be offloaded and therefore improve the network speed in-flight.

Furthermore, hardware compression is used for better CPU utilization [21, 29]. Also, the energy costs of running an HPC cluster can be reduced by compression [10, 72].

Another use case apart from I/O and persistent data is transparent compression of working data within the application. zram is a kernel module that creates compressed block devices and compresses the process memory without its knowledge [99]. It is commonly used for temporary files or as a swap disk, while live data stays uncompressed. The CR is limited since only single pages are compressed. As the benefit of transparent compression depends on the specific data, it is not a universal approach. Additional studies have been done to reduce the performance gap between processor and memory calls by introducing cache compression [9, 33]. A wide range of scientific applications and tools already involves compression mechanisms for computation or out-processing, like LAMMPS, HDF5 and MapReduce [40, 120].

There are different potential usages of lossless compression with different requirements on the algorithms. The following sections describe the basics of entropy- and dictionary-based compression and mention the most popular and useful lossless algorithms for the HPC field.

1.1.1. Entropy-based

Entropy-based encoding works by replacing unique symbols within the input data with a unique and shorter prefix code. The more often a symbol occurs, the shorter the prefix should be to ensure the best CR [94]. The most known techniques are arithmetic and Huffman codings. Huffman coding replaces words in a way that no word is a prefix of any other word in the system [111]. It creates a binary tree of nodes, which represents the symbols and their frequencies. At first, the data has to be scanned and the frequencies must be calculated. The nodes are either inserted into the binary tree or combined depending on the frequencies [20]. While Huffman coding splits input data into components that are encoded separately, arithmetic coding encodes the entire input into a number. This coding aims to compress close to the entropy limit while Huffman coding performs badly when the probabilities do not equal fractions with powers of two in their denominators [25]. Both codings are rarely used as stand-alone algorithms but can be adapted and combined with more elaborated techniques, some of which follow.

1.1.2. Dictionary-based

Another popular method is based on dictionaries and aims for partitioning the original input data to phrases (non-overlapping subsets of the original data) and the corresponding, possibly shortest codewords. This encoding is also known as substitution and has two main stages: dictionary construction (finding phrases and codewords) and parsing (replacing phrases by codewords) [125]. The dictionary has to be available for both the compressor and the decompressor. Dictionary codes can be classified into static and dynamic (or sometimes adaptive) constructs. Static dictionaries are created before the input processing and stay the same for the complete run. In case of the dynamic method, the dictionary is updated during parsing and the two stages (construction and parsing) mostly interleave. Byte pair encoding is a simple way of compression where the most common symbols are replaced by a symbol, different from the original alphabet [134]. The table of replacements is called *dictionary*.

LZ Family Lempel-Ziv is one of the most known dynamic dictionary compression methods. *LZ77* assumes and exploits that data is likely to be repeated. When repeated, a word can be

replaced by a pointer to the last occurrence accompanied by the number of matched characters [132]. The dictionary is then a part of the previously encoded sequences. The input is analyzed through a sliding window, that consists of search and look ahead buffers. LZ77 is suffix-complete, which means any suffix of a phrase is a phrase itself. The performance is limited by the number of comparisons needed for finding the matching pattern.

LZ77's successor *LZ78* constructs the dictionary differently. It begins with a single symbol entry in the dictionary, which grows by concatenating the first symbol of the following input after every parsing step. This algorithm uses greedy parsing, replacing the longest phrase with a prefix match by a codeword. In opposite to LZ77, LZ78 is prefix-complete.

Due to the explicit dictionary, the patterns are potentially held until the end of the input, which results in ever-growing dictionary buffers. There are several approaches on how to limit their sizes or optimize the dictionary building. These modifications of the general method and the development of optimizations continue today. The development of variants to the original method and respective optimizations continue today. All the modifications are very fast at decompressing, just like their ancestors.

LZ77 Variants *LZSS* is an algorithm developed by Storer and Szymanski that improves the look-ahead buffer by storing it in a circular queue and introduces a binary tree for the search buffer. *LZS* is based on LZSS and uses Huffman encoding for the length-distance pair [126]. *DEFLATE* is based on LZSS but uses a chained hash table to find duplicated sequences. The matched lengths and distances are compressed with two Huffman trees. There are hardware implementations of a novel adaptive version of DEFLATE [141] and an FPGA approach [48]. The DEFLATE format is used in ZIP, gzip, Zopfli, zlib and many other algorithms, which also have hardware implementations [1, 116]. *LZJB* is based on *LZRW1*, which uses hash tables among other techniques. While LZRW1 could overrun buffers by either reading past the input or writing past the output, LZJB does not. As a result, it is more suitable for file system usage, e.g., in ZFS [155]. It also allows a larger match length with smaller look-behind buffer and is, therefore, faster with less memory usage. *LZMA* is the default algorithm used in the 7z compression software and is similar to DEFLATE but uses delta filtering with range, instead of Huffman encoding [81]. MAFISC is an HDF5 compression filter based on LZMA. *LZX* uses a history buffer up to 2 MiB and combines Huffman coding techniques with shorter codes. The three most recent matches are then compressed with LZMA [100].

Snappy is developed by Google and was open-sourced in 2011. The input is cut into fix sized (64 KiB) blocks and the encoder is byte-oriented. Work on an FPGA version of Snappy has been done [118]. *LZFSE* is an Apple compressor with finite-state entropy, which combines a dictionary compression scheme with a technique based on asymmetric numeral systems [137]. *Brotli* is the Google approach to replace Zopfli and is not DEFLATE-compatible [8]. The compressed file is represented by a collection of meta-blocks. These are composed of a data part, which is simply compressed by LZ77 and a header describing how to decode the data part.

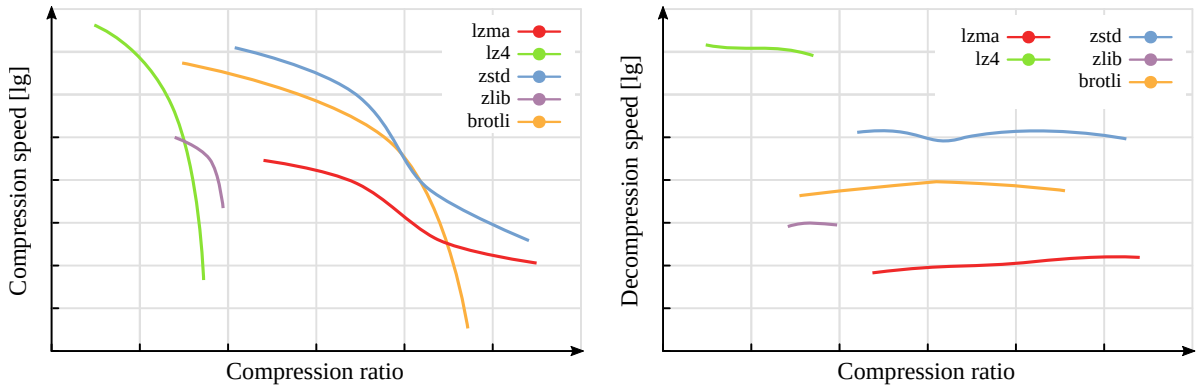
Bloclz is the default algorithm in Blosc, a high-performance compressor optimized for binary data. It is based on FastLZ, which itself is inspired by *LZV* and *LZF* algorithms. They all favor CPU efficiency over CR. *LZO* uses a quick hash table for lookups and has additional optimizations to output tokens [160]. The multi-core performance is explored in an additional work [69]. *LZ4* is another LZ77 variant with a fixed, byte-oriented encoding with no other codings. It provides an extremely fast decompressor, which can reach up to half of the memcpy throughput.

LZ4 has further variants. One is the fast mode that trades CR for compression speed and the high compression (HC) mode. There are also modifications for real-time hardware as well as general hardware implementations [80, 90]. *Lizard*, previously called LZ5, uses expansions to LZ4 and can optimally combine them with Huffman coding. It is also a part of the Blosc compressor library. *Zstd* supports a large search window (eight times larger than zlib) and involves an entropy coding stage, using fast Finite State Entropy or Huffman coding. The implementation works well with modern processors and compilers and is multi-threaded [131, 140].

LZ78 Variants One of the modifications is *LZW*, introduced by the initial authors of LZ78 and Terry Welch [151]. A dictionary is initialized to all possible symbols and the input is then processed symbol by symbol and concatenated to a string that is searched in the dictionary.

This process continues as long as matches are found and the dictionary is updated to the new concatenation when the word was missing in the dictionary [130]. The GIF encoding is based on LZW. There are also hardware-accelerated versions of LZW [127]. *LZWL* is an LZW extension for working with syllables or complete words. Other variants of this encoding are *LZMW*, *LZAP* and *LZWL*. LZMW does not reinitialize the full dictionary like LZW does, but removes the last used phrase instead. Here, concatenation is not performed on one new symbol and the match but on one match with another match [76]. LZAP is another modification of LZW, which adds all the prefixes of the unknown word instead of a concatenation of one prefix with this word. It allows for better CR with potentially faster dictionary growth and more frequent updates [142].

Other compression algorithms do not use dictionaries but smartly combine several techniques, especially in order to achieve a high CR. One of such is *bzip2*, which compresses files using the Burrows-Wheeler block-sorting compression algorithm [17], MTF (move to front), RLE (run-length encoding) of MTF result, Huffman coding, Unary base-1 encoding of Huffman table selection, Delta encoding of Huffman-code bit lengths and sparse bit arrays. Due to its low performance, parallelization of bzip2 has been explored in [53].



(a) (b)

Figure 2. Qualitative comparison of some compression algorithms regarding the de-/compression speed and ratio (the larger, the better) for different compression levels [140]

Intel's *QAT* (QuickAssist Technology) provides hardware-based algorithms for cryptography and compression. De-/compression is offloaded to the QAT module (available on chipset or external PCIe cards). QAT can speed up different algorithms like DEFLATE, LZS and can be extended for many other algorithms. The data has to be physically contiguous, which is a hard requirement exceedingly few systems are able to fulfill without additional efforts. Hardware

compression has the potential to be much faster, but due to limited buffers (mostly one kernel page of 4 KiB), software solutions may still preferable [62].

1.1.3. Efficiency

Depending on the needs within the HPC systems, the most suitable algorithm varies. While the application and network layer mostly require very fast algorithms, the storage backend can afford slower throughput for a higher CR but greatly benefits from a fast decompressor for the read performance. Figure 2 shows a few of the algorithms measured on a specific data set with different compression levels. The results are extremely dependent on the input data but often allow a qualitative insight into the performance nevertheless.

1.2. Lossy Compression

Data in HPC is mostly generated by numerical simulations of natural processes, which follow the principle of locality so that adjacent data is likely to be highly correlated. Floating-point data is hard to compress nonetheless since it often already features high entropy.

Applying lossless compression on large data sets does not always satisfy external requirements such as guaranteed I/O performance for live visualization or limited assigned storage, due to long compression times or low CRs. If a controlled loss of data quality is an option, lossy compression can be applied instead. Using lossy compression, CRs of over 400:1 are possible [41] – even though a CR beyond 64:1 will degrade precision of reconstructed data [84]. Lossy compression does not always outperform lossless compression in case of constrained error margins [93]. Lossy compression is common in computer graphics [7, 16] and used by many visualizations primarily meant to be interpreted by humans but are not fed back to numerical computations.

Unbiased compression methods which stay within the data's noise or analysis's error margin can be applied without degradating quality [40, 73]. More use cases for lossy compression include the reduction of I/O time or the acceleration of checkpoint handling [30]. Lossy compression methods are usually used in a multilayer-compression approach, though specific algorithms reduce the data size sufficiently on their own: First, lossy compression reduces the data diversity so that the lossless compressors applied afterwards work more efficiently. There are different approaches to combining these techniques as well as distinctions in their implementation.

Many file formats add native support for lossy compression like for example NetCDF4/HDF5 [40], GRIB2, XTC/TNG [95], and Zarr.

1.2.1. Methods

Preprocessing These methods are easy to apply but lag behind regarding the CR or quality of the reconstructed data. A naive lossy compression step is *truncatation*, which simply omits the least significant bits. In the case of floating-points, these are the last bits of the mantissa.

More subtle approaches are *Bit Shaving* and *Bit Grooming*, which do not change the data type itself but tamper with the bit representation of floating-point data. Similar to truncation, Bit Shaving modifies the most insignificant bits by changing them to zero. It thereby induces a bias as the new values always underestimate the original values. To correct this bias Bit Grooming applies a bitmask, in which zeros and ones are alternating to balance the error [159]. For a given number of significant digits of a float to be preserved, Bit Grooming tends to spare too

many bits from being changed. A potential remedy for this is *Digit Rounding*, which substitutes the static bitmask of Bit Grooming with a dynamic and more granular bitmask [40].

Normalization can be used to save bits within floats, as values close to zero require fewer bits in the mantissa while preserving range and precision.

Another method is *quantization*, where the values of the original data are first subdivided into intervals and each point of an interval is then mapped onto the same representative. In literature, the representatives are called *codewords* and the interval scheme *codebook*. The mapping can be done fore single values (*scalar quantization*) or a set of values (*vector quantization*). Special attention should be paid to how the codebook is constructed because it is more computational intensive compared to the lookup.

A collection of algorithms to generate codebooks is presented in [63]. If the codebook generates intervals of varying length, the error bound cannot be guaranteed; to control the error the intervals' length must be uniform [143].

Linear packing uses quantization by mapping normalized 4-Byte floats onto 2-Byte integers. Preserving the original dynamic range, a linear transformation between the original and modified data is stored in addition. While this introduces overhead, the data size is still about halved in return. This approach can be further improved by *Layer-packing*, which applies linear packing on slices of multi-dimensional data. The slicing is performed alongside the so-called thick dimension, i.e. the dimension with the highest range of values. For example, many variables of a 3D atmospheric model undergo heavy changes along the vertical dimension but are comparatively quite stagnant at the same height. In this case, the dataset would be split into horizontal slices. The precision of the linear packed slices is improved at the cost of additional overhead [136].

Transform compression This kind of method relies on a frequency transformation of the spatio-temporal original data signal into a new domain. The coefficients of the transformed signal may then be classified regarding their significance. While no loss of precision is applied in theory because transformations are invertible, transformations on floating-point data typically lead to rounding errors. In literature those transformations are therefore also addressed as *near-lossless* methods [82]. Transform compression becomes definitely lossy as soon as insignificant small coefficients are eliminated. The most important transformation methods are variants of discrete Fourier transforms (fast Fourier transform [FFT], discrete cosine transform [DCT], modified discrete cosine transform [MDCT], discrete sine transform [DST], modified discrete sine transform [MDST]) and wavelet transforms. An extensive overview of these transformation methods can be found in [149].

Prediction A good derivation of a data predictor allows estimating the value of adjacent data. To reproduce the original data, the predictor and the differences, also called residuals, need to be stored, which are minimal if the predictor has been well fit. Those residuals feature less entropy and, if they are small enough, they can be represented by smaller datatypes [114]. If the residuals are preprocessed before their compression, this method becomes also irreversible.

Some relevant schemes to perform the prediction are linear predictors (e.g. *Lorenzo predictor* [64], *mean-integrated Lorenzo predictor* [84]), *differential pulse-code modulation* [32], and *motion compensation* [139]. The latter one is of special interest for molecular dynamics since the simulated particles move along trajectories or stand still and can, therefore, be approximated by for example a low-polynomial function [114].

1.2.2. Selection of lossy compressors

Even though lossy compression is not omnipresent in HPC yet, there are some compressors available. Well-known are SZ and ZFP since they achieve a good performance [41], but we will also discuss other compressors, which stand out in their field. The last four entries are frameworks, which provide a collection of lossy compressors.

SZ [41]: Three steps are performed on the original data. At first, a multidimensional prediction model is applied, such as preceding neighbor fitting that assumes the current value to equal the preceding value. There are also linear and quadratic polynomial fittings. Afterwards, prediction points are quantized. Data, which cannot be fitted by the quantized predictors, is normalized and then truncated. At last, further compression is performed by using DEFLATE.

ZFP [86]: The original 3D data is chunked into $4 \times 4 \times 4$ blocks. All values in a block are normalized to the maximum in this block so that the adjusted values are in the range of $[-1, +1]$. Since high-order floats are eliminated now, the values are transformed into a Q3.60 fixed-point representation, which increases the numerical stability. Then, an orthogonal transform is applied and the resulting coefficients are sorted. Because the transformation results in many insignificant coefficients, they are well compressible.

FPZIP [87]: The data is at first approximated via a Lorenzo predictor and then truncated. Intervals of predictions and residuals are then encoded. Most of the time, ZFP outperforms FPZIP when the lossy compression is used [143].

ISABELA [77]: To smooth the data, it is first sorted and predicted afterwards using B-splines. Since ISABELA does in situ processing, it is discussed in detail in section 5.1.2.

TTHRESH [19]: The Tucker decomposition, a higher-order singular value decomposition, is performed. Then the decomposition core is flattened and the coefficients are compressed. Ballester-Ripoll et al. showed that the degradation of data quality is managed well at high CRs since high peak signal-to-noise ratios are preserved. However, their compression scheme suffers from lower (de-)compression speed and disadvantageous random access times in return. Another drawback is the insufficient support for 2D datasets [30].

NUMARCK [35]: Motion compensation (forward predictive coding) between checkpoints is used and data is then approximated by utilizing machine learning. NUMARCK is therefore discussed in detail in section 3.1.2.

SSEM [128]: This compressor uses a wavelet transform and vector quantization to speedup the general checkpoint creation time of an HPC application.

FRaZ [148]: This framework includes SZ, ZFP and MGARD [3–5]. The lossy compressors can be controlled by setting the upper bound of the absolute error. Some already feature a fixed-rate mode, but usually, this mode comes with a catch, e.g. it neglects the absolute error bound. FRaZ determines the correct setting of the absolute error for an indicated combination of lossy compressor and a particular dataset with only limited overhead.

VAPOR [113]: VAPOR allows to explore and interact with large datasets stepwise. To make this possible, it utilizes a progressive data model to divide the dataset into areas of interest and apply lossy compression (wavelet transforms). Currently, VAPOR3 is being developed and according to its roadmap, additional compressors shall be included [150].

Z-checker [144]: The primary goal of this framework is to allow users checking and understanding the data features and how lossy compression would affect the data quality. The virtualization allows to quickly explore datasets and evaluate the impact of several lossy compressors on specific data and compression properties.

Deep neural network [119]: This approach uses a neural network to analyze dataset features such as the hit ratio of prediction methods in order to estimate the CR for different lossy compressors, currently SZ and ZFP. More details are presented in section 3.2.

2. Dimensionality Reduction

Scientific data such as simulation or detector output usually has a high dimensionality, often requiring reduction for appropriate handling. Ideally, the result of dimensionality or dimension reduction (DR) resembles the intrinsic dimensionality of the data, thus preserving those features necessary for characterization [49]. Besides mitigating the curse of dimensionality [23], DR reduces the multi-collinearity, thereby improving the interpretation of parameters while decreasing the computation time as well as the data size. Also, visualizing results is considerably simplified with a smaller feature space. DR approaches can be separated into feature selection and feature extraction (FE). The former focuses on selecting a characteristic subset while feature extraction, also referred to as feature projection, transforms the data into a representation of fewer dimensions [31]. As neither the geometry of the data nor the intrinsic dimensionality is known, DR is an ill-posed problem enforcing the assumption of certain data properties [98].

Following the taxonomy of DR techniques depicted in Fig. 3 by Maaten et al., FE approaches are split into convex and non-convex techniques. The main difference being that non-convex techniques can model the existence of multiple local optima, while convex approaches require global and local optimum to coincide.

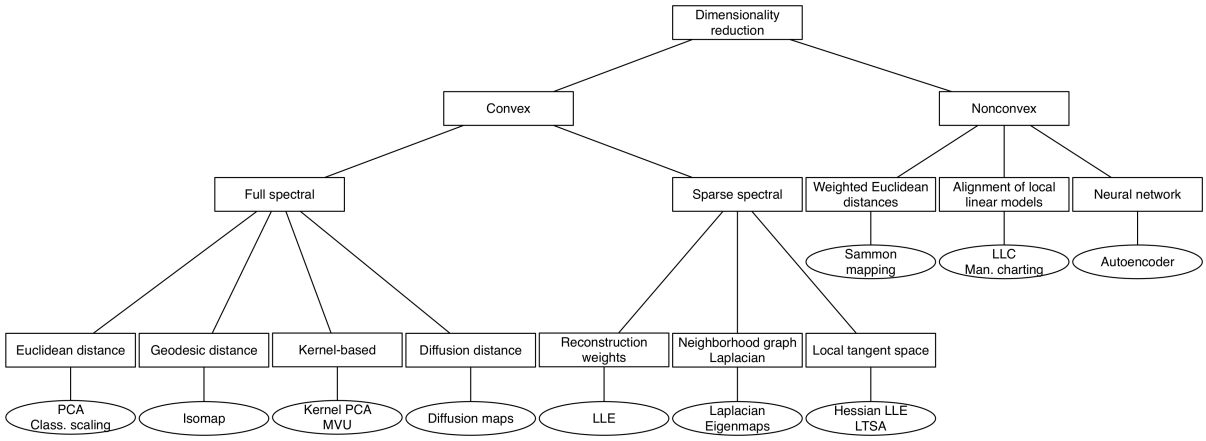


Figure 3. Taxonomy of dimensionality reduction techniques (FE) from [98]

2.1. Feature Selection

Feature selection approaches devide into wrappers, filters, and embedded methods [57]. Prominent wrappers are greedy search strategies, namely *backward elimination* and *forward selection*. The first removes the least promising variables while the latter continuously incorporates variables into larger subsets. Filter methods often focus on features like variance or correlation as they are easy to compute [57]. In contrast to wrappers, they are not adjusted to a specific model and result in a more general selected set. Random forests have proven to be useful for ranking feature importance [37]. Cliff et al. proposed an iterative random forest implementation optimized for HPC. An embedded feature selector based on the linear dependency of input and output is *Least absolute shrinkage and selection operator (Lasso)*. Yamada

et al. proposed *Hilbert-Schmidt Independence Criterion Lasso (HSIC Lasso)* considering also non-linear dependencies [158]. The global optimum of HSIC Lasso can be efficiently determined, making it a scalable technique suitable for very high-dimensional data where the dimensionality is magnitudes higher than the sample size.

2.2. Linear Feature Extraction

Principal Component Analysis (PCA), also called classical scaling, finds a linear low-dimensional projection maximizing the data variance, that is finding a low number of representative linear combinations (principal components) [39]. PCA can either be performed by using the covariance matrix to construct the eigenvectors or through *singular value decomposition (SVD)* of a normalized data matrix. The computational complexity of PCA, determined by the number of datapoints n and their dimensionality D, is $\mathcal{O}(D^3)$, when $D < n$. Therefore, there is a number of proposed optimizations. One is to use the autocorrelation and the large scale structure of data produced by climate science or a similar domain where the values do not vary abruptly over time and neighboring fields are not completely independent [24]. Martel et al. show how the iterative Jacobi method can be used to speed up the eigenvalue decomposition of PCA on HPC systems using hardware acceleration [102].

Several approaches are based on PCA and SVD, such as factor analysis [98], Partial Least Squares (PLS) or Maximum Covariance Analysis (MCA). PLS and MCA proofed valuable for the analysis of multi-temporal datasets [24]. The *Linear Discriminant Analysis (LDA)* is closely related to PCA but focuses on the discrimination between classes in contrast to PCA that does not consider any underlying class structure of the data for the computation of the principal components [103]. Opposed to LDA, the *Generalized discriminant analysis (GDA)* is a nonlinear technique using kernel function operators [22]. *Random Projection (RP)* is based on the Johnson-Lindenstrauss lemma stating it is possible to map vectors of high-dimensional space onto an $\mathcal{O}(n \log n)$ dimensional space while the pairwise distances are approximately preserved [157]. While RP can be computed efficiently, high distortions are possible. An interesting proposal to counter this is to first increase the dimensionality to have a better feature representation and then to reduce it with RP [96]. *Non-negative Matrix Factorization (NMF)* is an approach, often used in domains as astronomy, that has good interpretability due to the non-negative entries in the factorized matrices [31].

2.3. Non-Linear Convex Feature Extraction

Kernel PCA operates in a high-dimensional space constructed through a kernel function [98]. Therefore, the eigenvectors are based on the kernel matrix, not the covariance matrix. *Isomap* improves classical scaling by using the geodesic distance instead of the Euclidean distance, thereby considering the distribution of neighboring data points on a manifold [146]. When data points lie on or near a curved manifold (Swiss roll dataset), the Euclidean distance may differ considerably from their distance over the manifold. However, Isomap is topologically instable, possibly constructing incorrect connections in the neighborhood graph [98]. Also, non-convex manifolds can pose problems. *Maximum Variance Unfolding (MVU)* expands on Kernel PCA by aiming at learning the kernel matrix through defining a neighborhood graph. MVU differs from Isomap because it preserves the local geometry, ultimately trying to unfold the manifold. *Diffusion Maps (DM)* use Markov random walks on the data graph to determine the diffusion

distances in that it is more likely to walk to a point nearby [98]. As it integrates over all paths, the diffusion distance is more robust to noise or short-circuiting than the geodesic distance. The computational complexity is $\mathcal{O}(n^3)$ for Kernel PCA, Isomap and DM, and $\mathcal{O}((nk)^3)$ for MVU with k nearest neighbors. *Local Linear Embedding (LLE)* and *Laplacian Eigenmaps (LE)* are closely related to Kernel PCA and Isomap as they all solve an eigenproblem. Their crucial difference lies in the type and scope of local property preservation. LE differs from *Hessian LLE* only in the way the differential operator on the manifold is defined. These sparse spectral methods can be computed in $\mathcal{O}(pn^2)$, where p is the ratio of nonzero elements to the total number of elements.

2.4. Non-Linear Non-Convex Feature Extraction

Sammon Mapping (SM) improves classical scaling by weighting each pair's input to the cost function so that the local structure is retained better. It has been successfully applied to geospatial data [98]. *Locally Linear Coordination (LLC)* and *Manifold Charting (MC)* globally align local linear models. *T-distributed Stochastic Neighbor Embedding (t-SNE)* is optimized for the visualization of large high-dimensional datasets outperforming SM, LLC and MC [97]. This is accomplished by projecting each data point to a two- or three-dimensional map, such that a high similarity leads to short distances. *Uniform Manifold Approximation and Projection (UMAP)* improves t-SNE by better preserving the global structure by using local Riemann manifold approximations and representing it with a fuzzy topological structure [106]. *Multilayer Autoencoders (AE)* are feed-forward neural networks that force the model to compress the data due to their architecture. AEs consist of two networks, one encoder and one decoder reconstructing the data. Their number of hidden layers is odd and they share weights between the input and output layer [98]. By pretraining, the network with *Restricted Boltzmann Machines (RBM)*, the existence of local optima in the objective function can be dealt with. AEs on their own do not cope well with very high-dimensional data as the number of weights is too large. However, by using PCA beforehand, this limitation can be overcome. A comparison of the respective advantages of AEs in contrast to PCA is given in [47]. The computational complexity depends on the target dimensionality d, the number of iterations i and the number of weights in a neural network w. It is $\mathcal{O}(in^2)$ for SM, $\mathcal{O}(inw)$ for AE, $\mathcal{O}(imd^3)$ for LLC and MC.

In conclusion, an extensive study by Maaten et al. shows that linear and non-linear techniques need to be evaluated on both artificial and real-world datasets as non-linear FE approaches outperform linear ones for complex non-linear data while they perform poorly on natural datasets [98]. They outline future goals to include the development of objective functions that are not impaired by trivial optimal solutions. Also, they suggest that prospective techniques do not base their data representation on neighborhood graphs to model local properties, thereby mitigating the curse of dimensionality. Chao et al. remark that further research has to be done in terms of scalability as well as the ability to cope with missing input values [31]. Furthermore, integrating heterogeneous data will continue to pose a considerable challenge.

3. Adaptive Approaches

Adaptive approaches have the ability to change behavior depending on a specific problem to achieve the best possible results. They may act fully automatic or require some degree of manual intervention. In this section, we consider deep learning-based data reduction techniques

and meta-compressors, whose behavior depend on data. The former learn from data on how to achieve optimal data reduction strategy. The latter decide which is the best compression algorithm for a particular data type.

3.1. Compression Based on Machine Learning

Machine learning based compression involves two key steps: modeling and coding. While coding can be regarded as a solved problem, there is still no optimal solution for modeling. The difficulty in modeling is building the most compact representation of data. A group of compression algorithms applies various machine learning techniques to get closer to the optimum.

3.1.1. Media compression

In the last years, researchers did considerable progress in media compression with machine learning. Some solutions have already impressive characteristics and may be used as alternatives to traditional approaches. They cover lossy, lossless image and video compression.

Full resolution lossy image compression with recurrent neural networks (RNN) by G. Toderici et al. supports variable compression rates [147]. In the experiments, compression with RNNs (LSTM, associative LSTM) outperforms JPEG for most bit rates. End-to-end optimized image compression in [18] is optimized for a better rate-distortion performance than the standard JPEG and JPEG2000 compression. The evaluation shows a better visual image quality at all bit rates. Real-time adaptive lossy image compression outperforms JPEG, JPEG2000 and WebP [121]. CocoNet is a deep learning approach that learns and maps pixel coordinates to colors [27]. A trained CocoNet-network is able to memorize one single picture and can be used for advanced image processing. Altough, CocoNet is used for image representation, it has also a high potential for image compression. A Learned Lossless Image Compression (L3C) outperforms PNG up to 1.4, WebP and JPEG2000 up to 1.08 in compression ratio [110]. The encoder represents a compressed images as as a set of extracted features, which can be assembled again to an image by a trained predictor. The parallel nature of the approach allows a performant implementation on parallel computer architecture. Deep Learning Video Coding (DLVC) is a deep learning approach, that achieves a CR of more than 2.5 compared to H.265/HEVC, at the same quality level [85, 88]. Internally, DVLC uses a set of different deep learning-based, in particular two CNN-based filters, and different non-learning-based coding techniques.

3.1.2. Data compression

There is also a number of data compressors that can be used as general purpose compressors or are already adapted to HPC community needs. DeepZip uses the capability of neural networks to create arbitrary complex mappings [55]. Together with arithmetic encoders, it works as a powerful lossless compression algorithm for sequential data, like text and genomic datasets. In experiments, it outperforms GZip on real data and achieves near-optimal performance on synthetic data. NUMARCK (Northwestern University Machine learning Algorithm for Resiliency and ChecKpointing) exploits the fact that in many scientific applications, subsequent checkpoints contain insignificant changes [35]. K-Means algorithms optimize forward predictive coding, which codes a checkpoint with reference to a past checkpoint, resulting in lossy compression. DeepSZ is a lossy neural network compressor [67]. It involved key steps, like network pruning, error bound assessment, optimization for error bound configuration, and compressed

model generation, featuring a high compression ratio and low encoding time. Compared with other state-of-the-art methods, DeepSZ can improve the compression ratio by up to 1.43, the DNN encoding performance by up to 4.0 (with four Nvidia Tesla V100 GPUs), and the decoding performance by up to 6.2. Neural networks have also been used in [115] to compress data gathered by Internet of Things devices in a lossy fashion.

Machine learning can also be used in ways that are not traditionally called compression. For instance, in [70], machine learning was used to replace traditional data structures for B-trees, hash maps and bloom filters by other types of models, including deep neural networks. Benchmarks on real-world data show, that neural networks can be up to 70% faster and consume order-of-magnitude less memory than B-trees.

3.2. Meta-Compressors

Meta-compressors are high-level data compressors with support of two or more compression algorithms. Algorithm selection can be user-defined, selected according to user requirements (semi-automatic), or can be fully transparent to users (automatic). In automatic and semi-automatic meta-compressors, a decision unit performs usually two tasks. First, it executes a compressibility check on digital data to decide, if compression will be beneficial. Secondly, it selects the most suitable compression algorithm for this particular data set. The most frequently used compressibility checkers are sample-based, but there is also a trend for more advanced deep-learning-based solutions. Sample-based compressibility checkers perform compression tests on sample data and choose an algorithm with the highest compression ratio. Deep-learning-based solutions are more advanced and usually outperform sample-based compressibility checkers [42, 119]. To predict the compressibility, a supervised model is trained with example data of a compression algorithm. A trained model is able to do a pre-analysis of data and check if it will benefit from compression or not, without costly compression. Deep-learning-based decision units can potentially be integrated into file systems with multi-algorithms support. Currently, there is no support in major HPC file systems.

C-Blosc2 is a high-performance lossless meta-compressor optimized for binary data [11]. It supports BloscLz, LZ4, LZ4HC, Zstd, Lizard, and zlib compression algorithms. Aside from achieving the best compression ratios, it is designed for fast data transmission to processor cache and speed-up memory-bound computations. The support of a 64-bit address space allows C-Blosc2 to access large sparse and sequential data, either in-memory or on-disk. Scientific Compression Library (SCIL) supports lossy and lossless compression [74]. Users set high-level instructions on how to handle data, e.g., they can define accuracy, absolute/relative tolerance, significant digits/bits, or relative error for lossy compression. Adaptive Compression Scheme (ACOMPS) is a relatively young research project [138]. It selects the best lossless (LZO, ZLIB, BZIP2, FPC, ISOBAR) or lossy (ZFP, SZ, ISABELA) compression method independently for each variable in a dataset. Another promising research on online selection of two popular lossy compression algorithms, ZFP and SZ, was done in [145].

4. Deduplication

Strategies to reduce data can also extend to a higher-level perspective than byte streams, files or objects. Especially, for large scale data management systems and data centers, opportunities to discover large chunks of identical data exist [108, 156]. A common strategy often referred to

as *deduplication* is therefore to hash and index data, and then only reference already existing fragments instead of keeping duplicate copies.

In scientific computing and HPC, redundant or duplicate data can occur in a variety of contexts [108]. Many scientific workflows utilize similar data. Input data is often downloaded by individual users and kept in their project or user directories. Across a large user base, this can amount to significant data volumes. Similar strategies are already employed by large email systems, where emails going to multiple inboxes are not stored multiple times on the server. Kaiser et al. note potential savings across scientific domains ranging from 37% to 99%, with a particularly dominant factor being null regions [68]. Not limited to HPC are backups and database snapshots, which effectively implement deduplication for incremental backups, although they can also be coupled with lower-level strategies. Similarly, images of software environments, as used with virtual machines (VMs) and containerization, typically can share identical system files and directory structures [75]. When base system images are provided, it is possible to only store one copy of the base image instead of keeping copies for each user. A related application comes with software stacks outside of containerization. Here, tooling support for building software like Spack provides the opportunity to offer flexibility to customize software environments, while relying on a selection of maintained site-wide packages as much as possible [50].

Finally, deduplication is commonly used to speed up or avoid unnecessary data transmission. As such wide area network (WAN) optimizations often include deduplication support [112]. In a scientific context, this, for example, becomes relevant as data is replicated between sites through national and international scientific networks.

Taking a closer look at how deduplication is typically implemented, Xia et al. identify five key stages common across many approaches that still apply today [156]. Data is typically first split up into smaller chunks to improve the chance of finding matches. The most important factor here is the granularity of chunks, as such it is common to find both block-level as well as file/object-level deduplication. In the next phase, actual data gets typically hashed to obtain a fingerprint, which allows efficient comparison even across a network. A hash is defined as a function that maps an arbitrary sequence of data to a fixed-size value. To minimize collisions, and thus risking to loose data, cryptographic hash functions are typically being employed. The fingerprints are then stored into index structures to allow efficient lookup. Finally, it is common to store compressed variants of the chunks to leave no opportunity to save space unused. To store and receive the (compressed) chunks, some data management on top of actual storage media is required. There can also be operational benefits to deduplication, which can help with the endurance of SSDs as unnecessary write cycles that damage the cells can be avoided.

Applying deduplication does introduce overheads for keeping index structures and to perform matching. ZFS deduplication, for example, requires additional memory to hold lookup tables [154]. Different storage solutions employ both software-, hardware-based or hybrid approaches, in addition to inline and out-of-band deduplication. Software-based approaches used to offer more flexibility at the cost of performance. Research into approaches using FPGAs such as CIDR might allow hardware-accelerated deduplication without sacrificing flexibility [6].

There is a number of security and privacy considerations to be aware of when employing deduplication. Deduplication can reduce data safety as the impact of data corruption increases. Some cloud providers used deduplication, which allowed to feign ownership of a file by transmitting a wrong hash, and this way extract sensible data. Research into proof of ownership methods offers strategies to mitigate this [59]. Similarly, when dealing with encrypted data, deduplication

cannot be applied across users as logically identical data would have different signatures. This notion of local and global deduplication applies also to scientific data, as reductions can be performed at the application, the node or the system level with different trade-offs [68].

5. In Situ Processing

Idle cycles are usually available on computing nodes due to the discrepancy between the computing capabilities and the I/O transfer rates. A solution to address this inefficiency is to use the available idle nodes for performing different computations on the data, which is already in memory. This process is called *in situ processing*.

5.1. Algorithms and Techniques

In situ algorithms usually focus on data analysis and data reduction for different purposes including logging, filtering, compression and visualization. Despite the fact that some of the generic algorithms could potentially be applied in situ, the majority of them do not satisfy the main restriction imposed by this context: use the available CPU(s) and available memory while keeping the impact on the application's performance at a negligible level. Therefore, specific techniques were developed or adapted for this environment. This section will look into some of them together with their applicabilities.

5.1.1. Data analysis

Efficient in situ data analysis techniques include feature extraction, feature tracking and region growing techniques. These are particular to each application because they depend on the data structure and data flow. Machine learning solutions for dimensionality reduction, clustering or classification are gaining much traction in recent years. Using deep convolutional autoencoders for in situ data reduction proves good results for feature extraction from large carbon particle simulation datasets [89].

5.1.2. Data compression

Data is usually compressed for visualization and storage reduction. Due to a high level of entropy, the simulated data is a difficult candidate for the majority of the general compression algorithms presented in the previous sections. A study of state-of-the-art compression algorithms and their efficiencies for in situ environments recommends lossy methods like FPZIP, scalar quantization, Discrete Fourier and Wavelet Transforms and tailored in situ methods like ISOBAR [129] and ISABELA [38, 78]. The last two approaches are briefly explained next.

In Situ Orthogonal Byte Aggregate Reduction (ISOBAR) uses a preconditioner to enhance the performance of a general lossless compressor. The main components of the ISOBAR preconditioner are the analyzer, which evaluates the data compressibility at the byte level and the partitioner, which splits the data into compressible bytes, incompressible bytes, and metadata. The compressible chunks are then compressed using a lossless compressor chosen by the EUPA-selector based on an efficient linearization strategy and user's preference for either a high compression ratio or high compression throughput. In the end, the merger reassembles the compressed bytes, the incompressible bytes and the metadata into the final output.

In situ Sort-And-B-spline Error-bounded Lossy Abatement (ISABELA) obtains a higher degree of data reduction than the lossless ISOBAR but at the price of accuracy. The method splits the data into fixed-sized windows and applies a preconditioner to sort the data from each window into a monotonically increasing curve. These curves are later approximated using cubic B-splines, which can be easily modeled with a low number of coefficients.

Wavelet compression paired with state-of-the-art floating-point compressors, MPI and OpenMP, provides a performant parallel and distributed solution for in situ compression [58]. Reallocating the I/O resources dynamically based on the coefficient magnitudes of the wavelet method and Shannon entropy leads to a new method that displays good results for simulations with imbalanced data complexity [101].

A more novel method employs Generative Adversarial Network to compress data from computational fluid dynamics simulations [91]. The authors use the discriminative neural network to compress the data on the compute nodes while the generative network handles the reconstruction on the visualization nodes.

5.1.3. Data visualization

In situ data visualization is extremely important in large, complex applications because it allows not only following the real-time simulation but also steering it if required. Kress describes the in situ visualization technologies and surveys the most popular libraries and frameworks with their advantages and disadvantages in [71]. The main technologies are briefly compared next.

Table 1. The briefly comparison of the main technologies

Technology / Characteristic	Tightly Coupled	Loosely Coupled
Visualization and simulation share the CPU(s)	yes	no
Visualization and simulation share the memory	yes	no
Data duplication (double RAM usage)	no	yes
Visualization needs network access to retrieve the data	no	yes
Coordination between visualization and simulation	minimal	intensive
Computational steering capabilities	yes	limited
Visualization added costs	RAM and CPU	nodes and network
Scalable	no	yes
Fault tolerance	no	yes

There is a hybrid approach combining flexibility and computational steering support. The choice which of the three technologies is more appropriate for a given use case is based on the hardware architecture, the available resources and the particularities of the simulation data.

A comprehensive comparison between the two main visualization tools which set the ground for the majority of the in situ frameworks – ParaView [2] and VisIt [36] is described in [122].

5.2. Frameworks and Infrastructure

While in situ processing is best performed at the application level of the software stack, middleware and toolkits can increase their performance by facilitating the data extraction. Examples of such frameworks include EPIC [44] and Freeprocessing [46]. A solution based on I/O layer components was proposed in [26]. Remote direct memory access (RDMA) can also contribute by enabling zero-copy, high-throughput and low-latency networking for HPC clusters [38].

Well-known frameworks focused on visualization include ParaView Catalyst [14], Strawman [79], VisIt LibSim [153] and Damaris/Viz [43]. Besides visualization capabilities, some frameworks offer computational steering too. CUMULVS [52] and ISAAC [105] are just two of them. Other in situ scenarios like clustering, covered by KeyBin2 [34] and compression, covered by CubismZ [58] complete the list of functionalities offered by this environment.

6. Recomputation and Reproducibility

A rather unconventional approach for reducing the amount of data that has to be stored is recomputation, that is, instead of storing experiment or simulation results within a storage system, they are recomputed on demand. This technique can be combined with in situ methods.

To be able to recompute results, a hard prerequisite is reproducibility. Reproducing experiments requires all necessary input data, software, scripts etc. to be archived and documented. Popper is a convention for creating reproducible scientific publications and makes use of best practices established open-source software development [66]. While Popper is tuned towards scientific publications, this also includes all experiments that have been performed for such a publication and can, therefore, be easily adapted for general experiments. Experiments may either be deployed locally with Docker [61] or in the cloud with Ansible [104].

The ReScience initiative has launched a new peer-reviewed journal that tries to tackle replication problems by using a new publication approach [124]. The whole review and publication process is hosted on GitHub and anyone with a GitHub account is able to comment on a submitted publication. Reviewers then try to replicate the submitted results. The authors define reproducibility as the ability to arrive at the same results as a publication when using the same code and data as used for the publication itself. Replication, however, means that new code can be written for a computational model or method to obtain the same results.

The DataONE Data Package standard provides a specification for packaging together data, software and visualizations as well as accompanying metadata [107]. In combination with the WholeTale project, arbitrary computational results can be reproduced [28]. For instance, the computing environment can be described with a Dockerfile that can be used by interested researchers to rebuild the exact environment.

There are also domain-specific frameworks and toolkits. For instance, [65] introduces an R-based framework called climate4R that aims to standardize the tools used for accessing, harmonizing and post-processing climate data. It provides access to both local and remote data and features built-in support for a wide range of remote data sources. SwarmRob is a toolkit for making robotics research reproducible [117]. In addition to providing a toolkit, the authors propose a new workflow that is separated into research and review phases. In the research phase, authors specify needed services in a Container Definition File as well as the services' configurations and interactions in an Experiment Definition File. The experiment can then be performed by the SwarmRob toolkit using these two files. In the review phase, reviewers are able use the same infrastructure to reproduce the authors' results. DUGONG is a preconfigured Docker container for bioinformatics and computational biology research [109]. It packages over 3,000 dependencies and applications, including Jupyter Notebook. By providing a common software base for research, results can be reproduced more easily.

Scientific applications often have a multitude of dependencies that have to be provided in specific configurations and versions that are not provided by operating system distributions. Therefore, scientists often have to resort to installing dependencies from source manually. Their

manual processes are problematic with regard to reproducibility. A convenient way to manage all dependencies required by the actual application is package managers. EasyBuild helps manage complex scientific software stacks by providing recipes for popular packages and additional features [12]. Spack works in a similar way but specializes in supporting combinatorial builds such that software can be used easily with a wide range of different configurations and versions [50]. Moreover, Spack computes hashes for each of its packages, further helping reproducibility.

Singularity is a container platform that allows running unprivileged users to run containers on typical HPC systems due to its integration with common batch schedulers such as SLURM [54]. It has built-in support for signing and verification of containers as well as portability and reproducibility. Even though applications running within the containers are isolated from the host environment, Singularity still allows access to host hardware such as GPUs and InfiniBand for improved performance. The authors of [135] make use of Singularity and Spack to provide an in situ software stack that can be moved between HPC machines easily. These containers can then be run on systems that support executing containers. Such an approach may also be used for reproducibility by providing the full container image to interested third parties.

In addition to containerizing application code, input and output data also needs to be handled since it is typically read from or written to a traditional shared storage system. Data Pallets are an approach for encapsulating output data within containers [92]. These Data Pallets can be then be passed from one step of a workflow to the next. Unique container and dependency IDs are used for provenance. Moreover, no application changes are necessary since the workflow system can take care of managing Data Pallets.

To summarize, there are several approaches to improve the reproducibility of experiments. However, to be able to recompute data, it is necessary to be able to regenerate bit-identical results. This introduces additional requirements since even changes to the underlying hardware architecture might cause minuscule changes in the output data. If bit-identical results are not necessary, available reproducibility approaches can already be used to recompute data even over longer periods of time. Care must be taken to ensure that applications stay executable by updating the infrastructure and archived artifacts to the current state of the art. Therefore, recomputation introduces additional overhead regarding software and data management.

Conclusion

As data volumes continue to grow, but the gap between computational throughput and I/O bandwidth widens, data reduction techniques are becoming more important. Many established technologies continue to remain highly relevant such as lossless/lossy compression or deduplication, which are routinely applied across different layers. In some areas, general-purpose solutions seem unlikely to meet future requirements, which is why research increasingly explores specialized and adaptive approaches that find the most appropriate representation depending on the data. As new systems and applications are designed with asynchronous processing pipelines mind, in situ approaches are gaining momentum. Often, they allow reducing data volumes by several orders of magnitude by filtering, aggregation or transformation at the edge or in transit. This paradigm shift in programming and the promotion of reproducible research also allows considering recomputation as a viable alternative for data that is accessed infrequently.

Many of the covered technologies do not exist in isolation but are combined. Compression remains one of the most popular and obvious methods that can be employed throughout the entire HPC stack in software and hardware. As a result, hardware acceleration is expected

to considerably improve performance and help reduce overheads. The overall performance of lossless compression algorithms often depends on the input, requiring research into adaptive approaches. When applying lossy compression, scientists are often hesitating which information can be safely discarded without interfering with the intentions of their workflow and without hurting unanticipated use too much. Here, keeping provenance information is important, but adoption is primarily hindered by a lack of guidance on the disadvantages and advantages of lossy compression for users. To spare users from needing to evaluate which specialized approaches are most suitable, frameworks to automate this decision are gaining popularity, but research into adaptive data-aware solutions is needed.

Currently, adaptive approaches can be categorized into methods using heuristics to predict and pick the best performing methods on the one hand and methods to transform data using machine learning or dimensionality reduction on the other. Predictors achieve good performance on specific data sets but are typically trained on a selection of training datasets, which introduces a bias and thus limits generalization. This also extends to methods for adaptive data transformation and dimensionality reduction, where most methods exploit specific structural properties but overlook others. A promising exception are approaches based on autoencoders which learn a compact representation without supervision. Specialized methods are useful, but a dialog in the community to curate representative datasets for data reduction is needed too.

Even though great progress has been achieved in the last 10 years with respect to in situ techniques, the disparity between computational throughput and I/O bandwidth has not been solved yet. Firstly, efforts are still being done in increasing the compression rates for the lossless algorithms and to bound the precision loss in lossy algorithms. Secondly, the HPC clusters become more heterogeneous with the addition of GPUs. This forces the in situ algorithms to be redesigned in order to efficiently use the new hardware which otherwise would not meet the low CPU usage requirements. Last but not least, deep learning gets increasingly more attention due to the very promising results obtained by the usage of the generative models. While continuing the efforts to improve well-known data analysis and visualization mechanisms, new ideas like in situ virtual reality are starting to emerge [56]. To enable further improvement, the development of additional scientific benchmarks to evaluate data reduction techniques are necessary.

Acknowledgements

Parts of this publication were enabled by the following projects: CoSEMoS[4], funded by the German Research Foundation (DFG) under grant KU 3584/1-1; the Helmholtz graduate school for the structure of matter DASHH[5]; the Intel Parallel Computing Center on Enhanced Adaptive Compression in Lustre, funded by Intel[6]; i_SSS[7], funded by BASF SE. Bull Cooperation with the research department of DKRZ to improve I/O for climate applications on the Mistral HPC.

[4] `https://cosemos.de`
[5] `https://www.dashh.org/`
[6] `https://wr.informatik.uni-hamburg.de/research/projects/ipcc-l/start`
[7] `https://wr.informatik.uni-hamburg.de/research/projects/i_sss/start`

References

1. Abdelfattah, M.S., Hagiescu, A., Singh, D.: Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL. In: McIntosh-Smith, S., Bergen, B. (eds.) Proceedings of the International Workshop on OpenCL, IWOCL 2013 & 2014, 13-14 May 2013, Georgia Tech, Atlanta, GA, USA / 12-13 May 2014 Bristol, UK. pp. 4:1–4:9. ACM (2014), DOI: 10.1145/2664666.2664670

2. Ahrens, J.P., Geveci, B., Law, C.C.: ParaView: An End-User Tool for Large-Data Visualization. In: Hansen, C.D., Johnson, C.R. (eds.) The Visualization Handbook, pp. 717–731. Academic Press / Elsevier (2005), DOI: 10.1016/b978-012387582-2/50038-1

3. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel techniques for compression and reduction of scientific data - the univariate case. Computat. and Visualiz. in Science 19(5-6), 65–76 (2018), DOI: 10.1007/s00791-018-00303-9

4. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel Techniques for Compression and Reduction of Scientific Data - The Multivariate Case. SIAM J. Scientific Computing 41(2), A1278–A1303 (2019), DOI: 10.1137/18M1166651

5. Ainsworth, M., Tugluk, O., Whitney, B., et al.: Multilevel Techniques for Compression and Reduction of Scientific Data-Quantitative Control of Accuracy in Derived Quantities. SIAM J. Scientific Computing 41(4), A2146–A2171 (2019), DOI: 10.1137/18M1208885

6. Ajdari, M., Park, P., Kim, J., et al.: CIDR: A cost-effective in-line data reduction system for terabit-per-second scale SSD arrays. In: 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, 16-20 Feb. 2019, Washington, DC, USA. pp. 28–41. IEEE (2019), DOI: 10.1109/HPCA.2019.00025

7. Akenine-Möller, T., Ström, J.: Graphics Processing Units for Handhelds. Proceedings of the IEEE 96(5), 779–789 (2008), DOI: 10.1109/JPROC.2008.917719

8. Alakuijala, J., Farruggia, A., Ferragina, P., et al.: Brotli: A general-purpose data compressor. ACM Trans. Inf. Syst. 37(1), 4:1–4:30 (2019), DOI: 10.1145/3231935

9. Alameldeen, A.R., Wood, D.A.: Adaptive Cache Compression for High-Performance Processors. In: 31st International Symposium on Computer Architecture, ISCA 2004, 19-23 June 2004, Munich, Germany. pp. 212–223. IEEE Computer Society (2004), DOI: 10.1109/ISCA.2004.1310776

10. Alforov, Y., Ludwig, T., Novikova, A., et al.: Towards Green Scientific Data Compression Through High-Level I/O Interfaces. In: 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018, 24-27 Sept. 2018, Lyon, France. pp. 209–216. IEEE (2018), DOI: 10.1109/CAHPC.2018.8645921

11. Alted, F.: Blosc2-Meets-Rome. `https://blosc.org` (2019), accessed: 2020-02-17

12. Alvarez, D., Cais, A.Ó., Geimer, M., et al.: Scientific Software Management in Real Life: Deployment of EasyBuild on a Large Scale System. In: 2016 Third International Workshop on HPC User Support Tools, HUST@SC 2016, 13 Nov. 2016, Salt Lake City, UT, USA. pp. 31–40. IEEE Computer Society (2016), DOI: 10.1109/HUST.2016.009

13. Amlekar, S.: Compression support in Spectrum Scale 5.0.0. https://developer.ibm.com/storage/2018/01/11/compression-support-spectrum-scale-5-0-0/ (2018), accessed: 2020-02-20

14. Ayachit, U., Bauer, A.C., Geveci, B., et al.: ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In: Weber, G.H. (ed.) Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 25–29. ACM (2015), DOI: 10.1145/2828612.2828624

15. Azzurri, P.: Track Reconstruction Performance in CMS. Nuclear Physics B - Proceedings Supplements 197(1), 275–278 (2009), DOI: 10.1016/j.nuclphysbps.2009.10.084

16. Baker, A.H., Hammerling, D., Turton, T.L.: Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data. Comput. Graph. Forum 38(3), 517–528 (2019), DOI: 10.1111/cgf.13707

17. Balkenhol, B., Kurtz, S.: Universal Data Compression Based on the Burrows-Wheeler Transformation: Theory and Practice. IEEE Trans. Computers 49(10), 1043–1053 (2000), DOI: 10.1109/12.888040

18. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end Optimized Image Compression. CoRR abs/1611.01704 (2016), http://arxiv.org/abs/1611.01704

19. Ballester-Ripoll, R., Lindstrom, P., Pajarola, R.: TTHRESH: Tensor Compression for Multidimensional Visual Data. CoRR abs/1806.05952 (2018), http://arxiv.org/abs/1806.05952

20. Barbay, J.: Optimal Prefix Free Codes with Partial Sorting. Algorithms 13(1), 12 (2020), DOI: 10.3390/a13010012

21. Barr, K.C., Asanovic, K.: Energy-aware lossless data compression. ACM Trans. Comput. Syst. 24(3), 250–291 (2006), DOI: 10.1145/1151690.1151692

22. Baudat, G., Anouar, F.: Generalized Discriminant Analysis Using a Kernel Approach. Neural Computation 12(10), 2385–2404 (2000), DOI: 10.1162/089976600300014980

23. Bellman, R., Lee, E.: History and development of dynamic programming. IEEE Control Systems Magazine 4(4), 24–28 (1984), DOI: 10.1109/MCS.1984.1104824

24. Bogaardt, L., Goncalves, R., Zurita-Milla, R., et al.: Dataset Reduction Techniques to Speed Up SVD Analyses on Big Geo-Datasets. ISPRS Int. J. Geo-Information 8(2), 55 (2019), DOI: 10.3390/ijgi8020055

25. Bookstein, A., Klein, S.T.: Is Huffman coding dead? Computing 50(4), 279–296 (1993), DOI: 10.1007/BF02243872

26. Boyuka II, D.A., Lakshminarasimhan, S., Zou, X., et al.: Transparent in Situ Data Transformations in ADIOS. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, 26-29 May 2014, Chicago, IL, USA. pp. 256–266. IEEE Computer Society (2014), DOI: 10.1109/CCGrid.2014.73

27. Bricman, P.A., Ionescu, R.T.: CocoNet: A deep neural network for mapping pixel coordinates to color values. CoRR abs/1805.11357 (2018), http://arxiv.org/abs/1805.11357

28. Brinckman, A., Chard, K., Gaffney, N., et al.: Computing environments for reproducibility: Capturing the "Whole Tale". Future Generation Comp. Syst. 94, 854–867 (2019), DOI: 10.1016/j.future.2017.12.029

29. Canal, R., González, A., Smith, J.E.: Very low power pipelines using significance compression. In: Wolfe, A., Schlansker, M.S. (eds.) Proc. of the 33rd Annual IEEE/ACM Int. Symposium on Microarchitecture, MICRO 33, 10-13 Dec. 2000, Monterey, California, USA. pp. 181–190. ACM/IEEE Computer Society (2000), DOI: 10.1109/MICRO.2000.898069

30. Cappello, F., Di, S., Li, S., et al.: Use cases of lossy compression for floating-point data in scientific data sets. IJHPCA 33(6) (2019), DOI: 10.1177/1094342019853336

31. Chao, G., Luo, Y., Ding, W.: Recent Advances in Supervised Dimension Reduction: A Survey. Machine Learning and Knowledge Extraction 1(1), 341–358 (2019), DOI: 10.3390/make1010020

32. Chen, K., Ramabadran, T.V.: Near-lossless compression of medical images through entropy-coded DPCM. IEEE Trans. Med. Imaging 13(3), 538–548 (1994), DOI: 10.1109/42.310885

33. Chen, X., Yang, L., Dick, R.P., et al.: C-Pack: A High-Performance Microprocessor Cache Compression Algorithm. IEEE Trans. VLSI Syst. 18(8), 1196–1208 (2010), DOI: 10.1109/TVLSI.2009.2020989

34. Chen, X., Benson, J., Peterson, M., et al.: KeyBin2: Distributed Clustering for Scalable and In-Situ Analysis. In: Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, 13-16 Aug. 2018, Eugene, OR, USA. pp. 34:1–34:10. ACM (2018), DOI: 10.1145/3225058.3225149

35. Chen, Z., Son, S.W., Hendrix, W., et al.: NUMARCK: machine learning algorithm for resiliency and checkpointing. In: Damkroger, T., Dongarra, J.J. (eds.) International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, 16-21 Nov. 2014, New Orleans, LA, USA. pp. 733–744. IEEE Computer Society (2014), DOI: 10.1109/SC.2014.65

36. Childs, H., Brugger, E., Whitlock, B., et al.: Visit. In: Bethel, E.W., Childs, H., Hansen, C.D. (eds.) High Performance Visualization - Enabling Extreme-Scale Scientific Insight. Chapman and Hall / CRC computational science series, CRC Press (2012), DOI: 10.1201/b12985-21

37. Cliff, A., Romero, J., Kainer, D., et al.: A High-Performance Computing Implementation of Iterative Random Forest for the Creation of Predictive Expression Networks. Genes 10(12), 996 (2019), DOI: 10.3390/genes10120996

38. Critchlow, T., van Dam, K.K.: Data-Intensive Science. CRC Press (2013)

39. Cunningham, J.P., Ghahramani, Z.: Linear dimensionality reduction: survey, insights, and generalizations. J. Mach. Learn. Res. 16, 2859–2900 (2015), http://dl.acm.org/citation.cfm?id=2912091

40. Delaunay, X., Courtois, A., Gouillon, F.: Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files. Geoscientific Model Development 12(9), 4099–4113 (2019), DOI: 10.5194/gmd-12-4099-2019

41. Di, S., Cappello, F.: Fast Error-Bounded Lossy HPC Data Compression with SZ. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, 23-27 May 2016, Chicago, IL, USA. pp. 730–739. IEEE Computer Society (2016), DOI: 10.1109/IPDPS.2016.11

42. Diederich, M., Doerk, T., Muehge, T., et al.: Decision-based data compression by means of deep learning technologies (2018), `https://patentswarm.com/patents/US20190221192A1`, application US 20180277068 A1

43. Dorier, M., Sisneros, R., Peterka, T., et al.: Damaris/Viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In: Geveci, B., Pfister, H., Vishwanath, V. (eds.) IEEE Symposium on Large-Scale Data Analysis and Visualization, LDAV 2013, 13-14 Oct. 2013, Atlanta, Georgia, USA. pp. 67–75. IEEE Computer Society (2013), DOI: 10.1109/LDAV.2013.6675160

44. Duque, E.P., Hiepler, D.E., Haimes, R., et al.: EPIC - An Extract Plug-In Components Toolkit for In-Situ Data Extracts Architecture. DOI: 10.2514/6.2015-3410

45. Filgueira, R., Singh, D.E., Pichel, J.C., et al.: Exploiting data compression in collective I/O techniques. In: Proceedings of the 2008 IEEE International Conference on Cluster Computing, 29 Sept.-1 Oct. 2008, Tsukuba, Japan. pp. 479–485. IEEE Computer Society (2008), DOI: 10.1109/CLUSTR.2008.4663811

46. Fogal, T., Proch, F., Schiewe, A., et al.: Freeprocessing: Transparent in situ Visualization via Data Interception. In: Amor, M., Hadwiger, M. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, Swansea, Wales, UK. pp. 49–56. Eurographics Association (2014), DOI: 10.2312/pgv.20141084

47. Fournier, Q., Aloise, D.: Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods. In: 2nd IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019, 3-5 June 2019, Sardinia, Italy. pp. 211–214. IEEE (2019), DOI: 10.1109/AIKE.2019.00044

48. Fowers, J., Kim, J., Burger, D., et al.: A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs. In: 23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, 2-6 May 2015, Vancouver, BC, Canada. pp. 52–59. IEEE Computer Society (2015), DOI: 10.1109/FCCM.2015.46

49. Fukunaga, K., Olsen, D.R.: An Algorithm for Finding Intrinsic Dimensionality of Data. IEEE Trans. Computers 20(2), 176–183 (1971), DOI: 10.1109/T-C.1971.223208

50. Gamblin, T., LeGendre, M.P., Collette, M.R., et al.: The Spack package manager: bringing order to HPC software chaos. In: Kern, J., Vetter, J.S. (eds.) Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 40:1–40:12. ACM (2015), DOI: 10.1145/2807591.2807623

51. Geist, A., Reed, D.A.: A survey of high-performance computing scaling challenges. IJHPCA 31(1), 104–113 (2017), DOI: 10.1177/1094342015597083

52. Geist II, G.A., Kohl, J.A., Papadopoulos, P.M.: Cumulvs: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications. IJHPCA 11(3), 224–235 (1997), DOI: 10.1177/109434209701100305

53. Gilchrist, J.: Parallel data compression with bzip2. In: Proc. of the 16th IASTED int. conf. on parallel and distributed computing and systems. vol. 16, pp. 559–564 (2004)

54. Godlove, D.: Singularity: Simple, secure containers for compute-driven workloads. In: Furlani, T.R. (ed.) Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), PEARC 2019, 28 July-1 Aug. 2019, Chicago, IL, USA. pp. 24:1–24:4. ACM (2019), DOI: 10.1145/3332186.3332192

55. Goyal, M., Tatwawadi, K., Chandak, S., et al.: DeepZip: Lossless Data Compression using Recurrent Neural Networks. CoRR abs/1811.08162 (2018), http://arxiv.org/abs/1811.08162

56. Gupta, A., Günther, U., Incardona, P., et al.: A Proposed Framework for Interactive Virtual Reality In Situ Visualization of Parallel Numerical Simulations. CoRR abs/1909.02986 (2019), http://arxiv.org/abs/1909.02986

57. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection. J. Mach. Learn. Res. 3, 1157–1182 (2003), http://jmlr.org/papers/v3/guyon03a.html

58. Hadjidoukas, P.E., Wermelinger, F.: A Parallel Data Compression Framework for Large Scale 3D Scientific Data. CoRR abs/1903.07761 (2019), http://arxiv.org/abs/1903.07761

59. Halevi, S., Harnik, D., Pinkas, B., et al.: Proofs of ownership in remote storage systems. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, 17-21 Oct. 2011, Chicago, Illinois, USA. pp. 491–500. ACM (2011), DOI: 10.1145/2046707.2046765

60. Halkiadakis, E.: Proceedings for TASI 2009 Summer School on "Physics of the Large and the Small": Introduction to the LHC experiments (2010)

61. Higgins, J., Holmes, V., Venters, C.C.: Orchestrating Docker Containers in the HPC Environment. In: Kunkel, J.M., Ludwig, T. (eds.) High Performance Computing - 30th Int. Conf., 12-16 July 2015, Frankfurt, Germany. Lecture Notes in Computer Science, vol. 9137, pp. 506–513. Springer (2015), DOI: 10.1007/978-3-319-20119-1_36

62. Hu, X., Wang, F., Li, W., et al.: QZFS: QAT Accelerated Compression in File System for Application Agnostic and Cost Efficient Data Storage. In: Malkhi, D., Tsafrir, D. (eds.) 2019 USENIX Annual Technical Conference, USENIX ATC 2019, 10-12 July 2019, Renton, WA, USA. pp. 163–176. USENIX Association (2019), https://www.usenix.org/conference/atc19/presentation/hu-xiaokang

63. Huang, C., Harris, R.W.: A comparison of several vector quantization codebook generation approaches. IEEE Trans. Image Processing 2(1), 108–112 (1993), DOI: 10.1109/83.210871

64. Ibarria, L., Lindstrom, P., Rossignac, J., et al.: Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. Comput. Graph. Forum 22(3), 343–348 (2003), DOI: 10.1111/1467-8659.00681

65. Iturbide, M., Bedia, J., Garcia, S.H., et al.: The R-based climate4R open framework for reproducible climate data access and post-processing. Environmental Modelling and Software 111, 42–54 (2019), DOI: 10.1016/j.envsoft.2018.09.009

66. Jimenez, I., Sevilla, M., Watkins, N., et al.: The Popper Convention: Making Reproducible Systems Evaluation Practical. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, 29 May-2 June 2017, Orlando / Buena Vista, FL, USA. pp. 1561–1570. IEEE Computer Society (2017), DOI: 10.1109/IPDPSW.2017.157

67. Jin, S., Di, S., Liang, X., et al.: DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression. CoRR abs/1901.09124 (2019), http://arxiv.org/abs/1901.09124

68. Kaiser, J., Gad, R., Süß, T., et al.: Deduplication Potential of HPC Applications' Checkpoints. In: 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, 12-16 Sept. 2016, Taipei, Taiwan. pp. 413–422. IEEE Computer Society, DOI: 10.1109/CLUSTER.2016.32

69. Kane, J., Yang, Q.: Compression Speed Enhancements to LZO for Multi-core Systems. In: Panetta, J., Moreira, J.E., Padua, D.A., et al. (eds.) IEEE 24th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2012, 24-26 Oct. 2012, New York, NY, USA. pp. 108–115. IEEE Computer Society (2012), DOI: 10.1109/SBAC-PAD.2012.29

70. Kraska, T., Beutel, A., Chi, E.H., et al.: The Case for Learned Index Structures. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, 10-15 June 2018, Houston, TX, USA. pp. 489–504 (2018), DOI: 10.1145/3183713.3196909

71. Kress, J.: In Situ Visualization Techniques for High Performance Computing. http://www.cs.uoregon.edu/Reports/AREA-201703-Kress.pdf (2017), accessed: 2020-01-23

72. Kuhn, M., Kunkel, J., Ludwig, T.: Data Compression for Climate Data. Supercomputing Frontiers and Innovations 3(1), 75–94 (2016), DOI: 10.14529/jsfi160105

73. Kumar, A., Zhu, X., Tu, Y., et al.: Compression in Molecular Simulation Datasets. In: Sun, C., Fang, F., Zhou, Z., et al. (eds.) Intelligence Science and Big Data Engineering - 4th International Conference, IScIDE 2013, 31 July-2 Aug. 2013, Beijing, China, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8261, pp. 22–29. Springer (2013), DOI: 10.1007/978-3-642-42057-3_4

74. Kunkel, J., Novikova, A., Betke, E.: Towards Decoupling the Selection of Compression Algorithms from Quality Constraints An Investigation of Lossy Compression Efficiency. Supercomputing Frontiers and Innovations 4(4) (2017), DOI: 10.14529/jsfi170402

75. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: Scientific containers for mobility of compute. PLOS ONE 12(5), 1–20 (2017), DOI: 10.1371/journal.pone.0177459

76. Lakhani, G.: Reducing coding redundancy in LZW. Inf. Sci. 176(10), 1417–1434 (2006), DOI: 10.1016/j.ins.2005.03.007

77. Lakshminarasimhan, S., Shah, N., Ethier, S., et al.: Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, 29 Aug.-2 Sept. 2011, Bordeaux, France, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6852, pp. 366–379. Springer (2011), DOI: 10.1007/978-3-642-23400-2_34

78. Lakshminarasimhan, S., Shah, N., Ethier, S., et al.: ISABELA for effective in situ compression of scientific data. Concurrency and Computation: Practice and Experience 25(4), 524–540 (2013), DOI: 10.1002/cpe.2887

79. Larsen, M., Brugger, E., Childs, H., et al.: Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In: Weber, G.H. (ed.) Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV 2015, 15-20 Nov. 2015, Austin, TX, USA. pp. 30–35. ACM (2015), DOI: 10.1145/2828612.2828625

80. Lee, S.M., Jang, J.H., Oh, J., et al.: Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression. IEICE Electronic Express 14(11), 20170399 (2017), DOI: 10.1587/elex.14.20170399

81. Li, B., Zhang, L., Shang, Z., et al.: Implementation of LZMA compression algorithm on FPGA. Electronics Letters 50(21), 1522–1524 (2014), DOI: 10.1049/el.2014.1734

82. Li, S., Marsaglia, N., Garth, C., et al.: Data Reduction Techniques for Simulation, Visualization and Data Analysis. Comput. Graph. Forum 37(6), 422–447 (2018), DOI: 10.1111/cgf.13336

83. Li, W., Yao, Y.: Accelerate Data Compression in File System. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., et al. (eds.) 2016 Data Compression Conference, DCC 2016, 30 March-1 April 2016, Snowbird, UT, USA. p. 615. IEEE (2016), DOI: 10.1109/DCC.2016.24

84. Liang, X., Di, S., Tao, D., et al.: Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In: Abe, N., Liu, H., Pu, C., et al. (eds.) IEEE International Conference on Big Data, Big Data 2018, 10-13 Dec. 2018, Seattle, WA, USA. pp. 438–447. IEEE (2018), DOI: 10.1109/BigData.2018.8622520

85. Lin, J., Hu, Y., Liu, D.: Deep Learning-Based Video Coding (DLVC). `http://dlvc.bitahub.com/` (2020), accessed: 2020-02-20

86. Lindstrom, P.: Fixed-Rate Compressed Floating-Point Arrays. IEEE Trans. Vis. Comput. Graph. 20(12), 2674–2683 (2014), DOI: 10.1109/TVCG.2014.2346458

87. Lindstrom, P., Isenburg, M.: Fast and Efficient Compression of Floating-Point Data. IEEE Trans. Vis. Comput. Graph. 12(5), 1245–1250 (2006), DOI: 10.1109/TVCG.2006.143

88. Liu, D., Li, Y., Lin, J., et al.: Deep Learning-Based Video Coding: A Review and A Case Study. CoRR abs/1904.12462 (2019), http://arxiv.org/abs/1904.12462

89. Liu, Q., Hazarika, S., Patchett, J.M., et al.: Deep Learning-Based Feature-Aware Data Modeling for Complex Physics Simulations. CoRR abs/1912.03587 (2019), http://arxiv.org/abs/1912.03587

90. Liu, W., Mei, F., Wang, C., et al.: Data Compression Device Based on Modified LZ4 Algorithm. IEEE Trans. Consumer Electronics 64(1), 110–117 (2018), DOI: 10.1109/TCE.2018.2810480

91. Liu, Y., Wang, Y., Deng, L., et al.: A novel in situ compression method for CFD data based on generative adversarial network. J. Visualization 22(1), 95–108 (2019), DOI: 10.1007/s12650-018-0519-x

92. Lofstead, J.F., Baker, J., Younge, A.: Data Pallets: Containerizing Storage for Reproducibility and Traceability. In: Weiland, M., Juckeland, G., Alam, S.R., et al. (eds.) High Performance Computing - ISC High Performance 2019 International Workshops, 16-20 June 2019, Frankfurt, Germany, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11887, pp. 36–45. Springer (2019), DOI: 10.1007/978-3-030-34356-9_4

93. Lu, T., Liu, Q., He, X., et al.: Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data. In: 2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, 21-25 May 2018, Vancouver, BC, Canada. pp. 348–357. IEEE Computer Society (2018), DOI: 10.1109/IPDPS.2018.00044

94. Lu, Z.M., Guo, S.Z.: Chapter 1 - Introduction. In: Lu, Z.M., Guo, S.Z. (eds.) Lossless Information Hiding in Images, pp. 1–68. Syngress (2017), DOI: 10.1016/B978-0-12-812006-4.00001-2

95. Lundborg, M., Apostolov, R., Spångberg, D., et al.: An efficient and extensible format, library, and API for binary trajectory data from molecular simulations. Journal of Computational Chemistry 35(3), 260–269 (2014), DOI: 10.1002/jcc.23495

96. Ma, C., Jung, J., Kim, S., et al.: Random projection-based partial feature extraction for robust face recognition. Neurocomputing 149, 1232–1244 (2015), DOI: 10.1016/j.neucom.2014.09.004

97. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. Journal of Machine Learning Research 9, 2579–2605 (2008)

98. van der Maaten, L., Postma, E., van den Herik, J.: Dimensionality reduction: a comparative review. Journal of Machine Learning Research 10(66-71), 13 (2009)

99. Magenheimer, D.: In-kernel memory compression. https://lwn.net/Articles/545244/ (2013), accessed: 2020-02-20

100. Mahoney, M.: Data Compression Explained. http://mattmahoney.net/dc/dce.html#Section_524 (2013), accessed: 2020-02-20

101. Marsaglia, N., Li, S., Belcher, K., et al.: Dynamic I/O Budget Reallocation For In Situ Wavelet Compression. In: Childs, H., Frey, S. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2019, 3-4 June 2019, Porto, Portugal. pp. 1–5. Eurographics Association (2019), DOI: 10.2312/pgv.20191104

102. Martel, E., Lazcano, R., López, J.F., et al.: Implementation of the Principal Component Analysis onto High-Performance Computer Facilities for Hyperspectral Dimensionality Reduction: Results and Comparisons. Remote Sensing 10(6), 864 (2018), DOI: 10.3390/rs10060864

103. Martínez, A.M., Kak, A.C.: PCA versus LDA. IEEE Trans. Pattern Anal. Mach. Intell. 23(2), 228–233 (2001), DOI: 10.1109/34.908974

104. Masek, P., Stusek, M., Krejci, J., et al.: Unleashing Full Potential of Ansible Framework: University Labs Administration. In: 22nd Conference of Open Innovations Association, FRUCT 2018, 15-18 May 2018, Jyväskylä, Finland. pp. 144–150. IEEE (2018), DOI: 10.23919/FRUCT.2018.8468270

105. Matthes, A., Huebl, A., Widera, R., et al.: In situ, steerable, hardware-independent and data-structure agnostic visualization with ISAAC. Supercomputing Frontiers and Innovations 3(4), 30–48 (2016), DOI: 10.14529/jsfi160403

106. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. CoRR abs/1802.03426 (2018), https://arxiv.org/abs/1802.03426

107. Mecum, B.D., Jones, M.B., Vieglais, D., et al.: Preserving Reproducibility: Provenance and Executable Containers in DataONE Data Packages. In: 14th IEEE International Conference on e-Science, e-Science 2018, 29 Oct.-1 Nov. 2018, Amsterdam, The Netherlands. pp. 45–49. IEEE Computer Society (2018), DOI: 10.1109/eScience.2018.00019

108. Meister, D., Kaiser, J., Brinkmann, A., et al.: A study on data deduplication in HPC storage systems. In: Hollingsworth, J.K. (ed.) SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, 11-15 Nov. 2012, Salt Lake City, UT, USA. p. 7. IEEE/ACM (2012), DOI: 10.1109/SC.2012.14

109. Menegidio, F.B., Jabes, D.L., de Oliveira, R.C., et al.: Dugong: a Docker image, based on Ubuntu Linux, focused on reproducibility and replicability for bioinformatics analyses. Bioinformatics 34(3), 514–515 (2018), DOI: 10.1093/bioinformatics/btx554

110. Mentzer, F., Agustsson, E., Tschannen, M., et al.: Practical Full Resolution Learned Lossless Image Compression. CoRR abs/1811.12817 (2018), http://arxiv.org/abs/1811.12817

111. Moffat, A.: Huffman Coding. ACM Comput. Surv. 52(4), 85:1–85:35 (2019), DOI: 10.1145/3342555

112. Muthitacharoen, A., Chen, B., Mazières, D.: A Low-Bandwidth Network File System. In: Marzullo, K., Satyanarayanan, M. (eds.) Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, 21-24 Oct. 2001, Chateau Lake Louise, Banff, Alberta, Canada. pp. 174–187. ACM (2001), DOI: 10.1145/502034.502052

113. Norton, A., Clyne, J.P.: The VAPOR Visualization Application. In: Bethel, E.W., Childs, H., Hansen, C.D. (eds.) High Performance Visualization - Enabling Extreme-Scale Scientific Insight. Chapman and Hall / CRC computational science series, CRC Press (2012), DOI: 10.1201/b12985-25

114. Ohtani, H., Hagita, K., Ito, A.M., et al.: Irreversible data compression concepts with polynomial fitting in time-order of particle trajectory for visualization of huge particle system. Journal of Physics: Conference Series 454, 012078 (2013), DOI: 10.1088/1742-6596/454/1/012078

115. Park, J., Park, H., Choi, Y.: Data compression and prediction using machine learning for industrial IoT. In: 2018 International Conference on Information Networking, ICOIN 2018, 10-12 Jan. 2018, Chiang Mai, Thailand. pp. 818–820 (2018), DOI: 10.1109/ICOIN.2018.8343232

116. Plugariu, O., Gegiu, A.D., Petrica, L.: FPGA systolic array GZIP compressor. In: 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). pp. 1–6. IEEE (2017), DOI: 10.1109/ECAI.2017.8166387

117. Pörtner, A., Hoffmann, M., Zug, S., et al.: SwarmRob: A Docker-Based Toolkit for Reproducibility and Sharing of Experimental Artifacts in Robotics Research. In: IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018, 7-10 Oct. 2018, Miyazaki, Japan. pp. 325–332. IEEE (2018), DOI: 10.1109/SMC.2018.00065

118. Qiao, Y., Fang, J., Hofstee, H.P.: An FPGA-based Snappy Decompressor-Filter (2018), DOI: 10.13140/RG.2.2.30215.44962

119. Qin, Z., Wang, J., Liu, Q., et al.: Estimating Lossy Compressibility of Scientific Data Using Deep Neural Networks. IEEE Letters of the Computer Society 3(1), 5–8 (2020), DOI: 10.1109/LOCS.2020.2971940

120. Rattanaopas, K., Kaewkeeree, S.: Improving Hadoop MapReduce performance with data compression: A study using wordcount job. In: 2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON, 27-30 June 2017, Phuket, Thailand. pp. 564–567 (2017), DOI: 10.1109/ECTICon.2017.8096300

121. Rippel, O., Bourdev, L.D.: Real-Time Adaptive Image Compression. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, 6-11 Aug. 2017, Sydney, NSW, Australia. pp. 2922–2930 (2017), http://proceedings.mlr.press/v70/rippel17a.html

122. Rivia, M., Caloria, L., Muscianisia, G., et al.: In-situ Visualization: State-of-the-art and Some Use Cases. http://www.prace-ri.eu/IMG/pdf/In-situ_Visualization_State-of-the-art_and_Some_Use_Cases-2.pdf (2012), accessed: 2020-02-20

123. Röber, N., Engels, J.F.: In-Situ Processing in Climate Science. In: Weiland, M., Juckeland, G., Alam, S.R., et al. (eds.) High Performance Computing - ISC High Performance 2019 International Workshops, 16-20 June 2019, Frankfurt, Germany, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11887, pp. 612–622. Springer (2019), DOI: 10.1007/978-3-030-34356-9_46

124. Rougier, N.P., Hinsen, K., Alexandre, F., et al.: Sustainable computational science: the ReScience initiative. PeerJ Computer Science 3, e142 (2017), DOI: 10.7717/peerj-cs.142

125. Sahinalp, S.C., Rajpoot, N.M.: Chapter 6 - Dictionary-Based Data Compression: An Algorithmic Perspective. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 153–167. Communications, Networking and Multimedia, Academic Press, San Diego (2003), DOI: 10.1016/B978-012620861-0/50007-3

126. Salomon, D.: Data compression - The Complete Reference, 4th Edition. Springer (2007)

127. Samanta, R., Mahapatra, R.: An Enhanced CAM Architecture to Accelerate LZW Compression Algorithm. In: 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems, VLSID'07, 6-10 Jan. 2007, Bangalore, India. pp. 824–829. IEEE (2007), DOI: 10.1109/VLSID.2007.34

128. Sasaki, N., Sato, K., Endo, T., et al.: Exploration of Lossy Compression for Application-Level Checkpoint/Restart. In: 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, 25-29 May 2015, Hyderabad, India. pp. 914–922. IEEE Computer Society (2015), DOI: 10.1109/IPDPS.2015.67

129. Schendel, E.R., Jin, Y., Shah, N., et al.: ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression. In: Kementsietsidis, A., Salles, M.A.V. (eds.) IEEE 28th International Conference on Data Engineering, ICDE 2012, 1-5 April 2012, Washington, DC, USA. pp. 138–149. IEEE Computer Society (2012), DOI: 10.1109/ICDE.2012.114

130. Setia, A., Ahlawat, P.: Enhanced LZW Algorithm with Less Compression Ratio. In: Proceedings of Int. Conf. on Advances in Computing. pp. 347–351. Springer India, New Delhi (2012), DOI: 10.1007/978-81-322-0740-5_41

131. Shadura, O., Bockelman, B.P.: ROOT I/O compression algorithms and their performance impact within run 3. CoRR abs/1906.04624 (2019), `http://arxiv.org/abs/1906.04624`

132. Shanmugasundaram, S., Lourdusamy, R.: A Comparative Study Of Text Compression Algorithms. ICTACT Journal on Communication Technology 1(3), 68–76 (2011), DOI: 10.21917/ijct.2011.0062

133. Shehabi, A., Smith, S., Sartor, D., et al.: United States Data Center Energy Usage Report (2016), DOI: 10.2172/1372902

134. Shibata, Y., Kida, T., Fukamachi, S., et al.: Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching (1999), `https://pdfs.semanticscholar.org/1e94/41bbad598e181896349757b82af42b6a6902.pdf`

135. Shudler, S., Ferrier, N.J., Insley, J.A., et al.: Spack meets singularity: creating movable in-situ analysis stacks with ease. In: Moreland, K., Garth, C., Bethel, E.W., et al. (eds.) Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV@SC 2019, 18 Nov. 2019, Denver, Colorado, USA. pp. 34–38. ACM (2019), DOI: 10.1145/3364228.3364682

136. Silver, J., Zender, C.: The compressionerror trade-off for large gridded data sets. Geoscientific Model Development 10, 413–423 (2017), DOI: 10.5194/gmd-10-413-2017

137. Simone, S.D.: Apple Open-Sources its New Compression Algorithm LZFSE (2016), https://www.infoq.com/news/2016/07/apple-lzfse-lossless-opensource/, accessed: 2020-02-20

138. Singhal, S., Sussman, A.: Adaptive Compression to Improve I/O Performance for Climate Simulations. https://web.njit.edu/~qliu/assets/adaptive-compression-scheme(acomps).pdf (2017), accessed: 2020-02-17

139. Srinivasan, R., Rao, K.R.: Predictive Coding Based on Efficient Motion Estimation. IEEE Trans. Communications 33(8), 888–896 (1985), DOI: 10.1109/TCOM.1985.1096398

140. Szorc, G.: Better Compression with Zstandard. https://gregoryszorc.com/blog/2017/03/07/better-compression-with-zstandard (2017), accessed: 2020-02-17

141. Tahghighi, M., Mousavi, M., Khadivi, P.: Hardware implementation of a novel adaptive version of Deflate compression algorithm. In: 2010 18th Iranian Conference on Electrical Engineering, 11-13 May 2010, Isfahan, Iran. pp. 566–569. IEEE (2010), DOI: 10.1109/IRANIANCEE.2010.5507007

142. Tajul, T.K., Bhuiyan, S.R., Habib, A.: Enhancement of LZAP (Lempel Ziv All Prefixes) Compression Algorithm. In: 2018 4th International Conference on Electrical Engineering and Information Communication Technology, iCEEiCT. pp. 69–73 (2018), DOI: 10.1109/CEEICT.2018.8628148

143. Tao, D., Di, S., Chen, Z., et al.: Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. CoRR abs/1706.03791 (2017), http://arxiv.org/abs/1706.03791

144. Tao, D., Di, S., Guo, H., et al.: Z-checker: A framework for assessing lossy compression of scientific data. IJHPCA 33(2) (2019), DOI: 10.1177/1094342017737147

145. Tao, D., Di, S., Liang, X., et al.: Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP. IEEE Trans. Parallel Distrib. Syst. 30(8), 1857–1871 (2019), DOI: 10.1109/TPDS.2019.2894404

146. Tenenbaum, J.B., Silva, V.D., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319–2323 (2000), https://science.sciencemag.org/content/sci/290/5500/2319.full.pdf

147. Toderici, G., Vincent, D., Johnston, N., et al.: Full Resolution Image Compression with Recurrent Neural Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 21-26 July 2017, Honolulu, HI, USA. pp. 5435–5443 (2017), DOI: 10.1109/CVPR.2017.577

148. Underwood, R., Di, S., Calhoun, J.C., et al.: FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data. CoRR abs/2001.06139 (2020), https://arxiv.org/abs/2001.06139

149. Vetterli, M., Kovacevic, J.: Wavelets and Subband Coding. Prentice Hall Signal Processing Series, Prentice Hall (1995)

150. Visualization and Analysis Software Team: VAPOR product roadmap. Tech. rep., NCAR (2017), https://ncar.github.io/vapor2website/sites/default/files/VAPORRoadmap.pdf

151. Welch, T.A.: A Technique for High-Performance Data Compression. IEEE Computer 17(6), 8–19 (1984), DOI: 10.1109/MC.1984.1659158

152. Welton, B., Kimpe, D., Cope, J., et al.: Improving I/O Forwarding Throughput with Data Compression. In: 2011 IEEE International Conference on Cluster Computing, CLUSTER, 26-30 Sept. 2011, Austin, TX, USA. pp. 438–445. IEEE Computer Society (2011), DOI: 10.1109/CLUSTER.2011.80

153. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In: Kuhlen, T.W., Pajarola, R., Zhou, K. (eds.) Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK. Proceedings. pp. 101–109. Eurographics Association (2011), DOI: 10.2312/EGPGV/EGPGV11/101-109

154. Widianto, E.D., Prasetijo, A.B., Ghufroni, A.: On the implementation of ZFS (Zettabyte File System) storage system. In: 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE. pp. 408–413 (2016), DOI: 10.1109/ICITACEE.2016.7892481

155. Williams, R.N.: An Extremely Fast Ziv-Lempel Data Compression Algorithm. In: Storer, J.A., Reif, J.H. (eds.) Proceedings of the IEEE Data Compression Conference, DCC 1991, 8-11 April 1991, Snowbird, Utah, USA. pp. 362–371. IEEE Computer Society (1991), DOI: 10.1109/DCC.1991.213344

156. Xia, W., Jiang, H., Feng, D., et al.: A Comprehensive Study of the Past, Present, and Future of Data Deduplication. Proceedings of the IEEE 104(9), 1681–1710 (2016), DOI: 10.1109/JPROC.2016.2571298

157. Xie, H., Li, J., Xue, H.: A survey of dimensionality reduction techniques based on random projection. CoRR abs/1706.04371 (2017), http://arxiv.org/abs/1706.04371

158. Yamada, M., Jitkrittum, W., Sigal, L., et al.: High-Dimensional Feature Selection by Feature-Wise Kernelized Lasso. Neural Computation 26(1), 185–207 (2014), DOI: 10.1162/NECO_a_00537

159. Zender, C.S.: Bit Grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+). Geoscientific Model Development 9(9), 3199–3211 (2016), DOI: 10.5194/gmd-9-3199-2016

160. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Information Theory 23(3), 337–343 (1977), DOI: 10.1109/TIT.1977.1055714