DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

DOE Award Final Report

DOE award number: DE-SC0022223
Sponsoring program office: Advanced Scientific Computing Research
Recipient: Dr. Martin Burtscher
Project title: Automatic Generation of Algorithms for High-Speed Reliable Lossy Data Compression
Project directors: Dr. Margaret Lentz and Dr. Hal Finkel
Team members: Dr. Sheng Di (co-PI) and Dr. Franck Cappello (senior advisor)

Public Summary

Fast reliable data compression is urgently needed for many leading-edge scientific instruments and for exascale high-performance computing applications because they produce vast amounts of data at extremely high rates. The goal of this project has been to develop a framework named LC that is able to automatically generate high-speed lossless and reliable lossy compression and decompression algorithms that can be customized for different kinds of data.

The resulting LC framework is freely available on GitHub. To achieve high-speed operation, LC outputs optimized and parallelized CPU and GPU implementations of the generated algorithms. To ensure the quality of lossily compressed data, LC guarantees the user-provided error bound. To be able to customize the compression algorithm to various use cases, LC can synthesize millions of different algorithms and automatically search for the one that works best for the given data.

We have already employed LC to create state-of-the-art lossless and lossy compressors for scientific data as well as leading lossless compressors for images. We hope that LC and the customized, fast, reliable, and CPU/GPU-compatible compression algorithms that it can generate will greatly benefit the many scientific applications that need not only high trustworthiness but also high performance.

1 First Project Outcome: LC Framework

The primary outcome of this project is the LC framework. It can automatically generate customized state-of-the-art lossless and guaranteed-error-bounded lossy data compression algorithms for individual files or groups of files. The resulting compressors and decompressors are parallelized and produce bit-for-bit identical results on CPUs and GPUs. The open-sourced framework and a step-by-step tutorial on how to use and optionally extend it are freely available on GitHub [1].

LC is able to create millions of distinct algorithms and has synthesized leading lossless compressors for integer [2] and floating-point data [3] as well as guaranteed-error-bounded lossy compressors for floating-point data [4]. These compressors are both fast and compress well and are generally not similar to prior algorithms. The performance of some of these algorithms is described in more detail in the next sections.

LC consists of the following three main parts:

- Component library
- Preprocessor library
- Framework code

Both libraries contain code to perform data transformations (encoders) and their inverses (decoders) for CPU and GPU execution. The components are lossless whereas the preprocessors include the guaranteed-error-bounded lossy quantizers. The available transformations and their operation are described in the tutorial as well as in a paper [5]. The user can easily extend these libraries as explained in the tutorial. The LC framework takes preprocessors and components from these libraries and chains them into a pipeline to build a compression algorithm. It similarly chains the corresponding decoders in the opposite order to build the matching decompression algorithm. Figure 1 illustrates this process.

LC's libraries are, in part, the result of a detailed analysis of many preexisting compression algorithms. We broke these algorithms down into their constituent parts, generalized them, and implemented them using

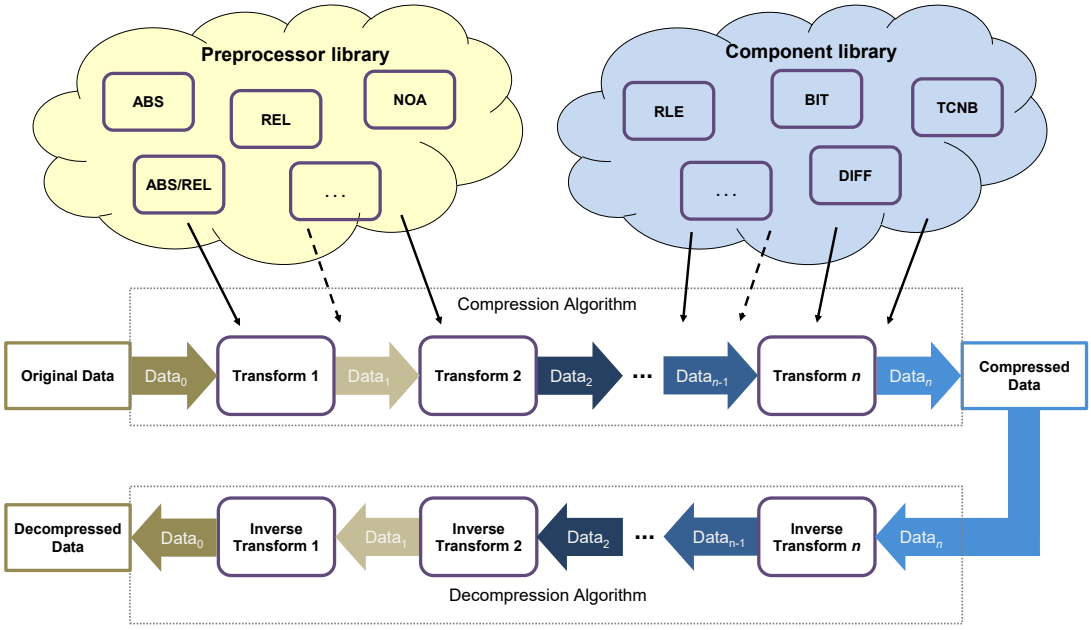

Figure 1: LC's process of chaining (i.e., pipelining) n data transformations to form a custom compression algorithm and the inverses of those transformations to form the matching decompression algorithm.

a common interface such that each component can be given a block of input data, which it transforms into an output block of data for input to the next component. This makes it possible to chain components, allowing LC to generate millions of compression-algorithm candidates from a small set of components.

Even without any preprocessors, LC can generate k^n unique algorithms, where k is the number of components in the library and n is the chain length, because every position in the chain can hold any one of the k components. Note that the order of the components matters and that duplicate components are allowed and can be useful. As an example, chains with just 4 components selected from LC's library of about 70 components result in $70^4 = 24$ million possible compression algorithms — a vast search space in which to discover previously unknown, superior compression algorithms as outlined in the following sections. Including preprocessors increases the size of the search space even further.

1.1 General Features

LC can automatically search for effective compression algorithms by testing combinations of user-selected sets of components in each pipeline stage. It supports exhaustive search for the best algorithm in its search space as well as a genetic-algorithm-based search for cases where the exhaustive search would take too long [6]. In addition, the user can optionally supply a regular expression to reduce the size of the search space. LC is able to search for the best algorithm based solely on compression ratio or based on both compression ratio and throughput. In the latter case, it outputs the Pareto front, that is, a set of algorithms that represent different compression-ratio versus speed tradeoffs.

LC can generate algorithms for CPUs and GPUs. The resulting implementations are deterministic and bit-for-bit compatible, meaning the user may compress a file on either a CPU or a GPU and decompress it on either a CPU or a GPU [7]. The CPU code is written in C++ and parallelized using OpenMP. The GPU code is written in CUDA for NVIDIA GPUs and in HIP for AMD GPUs [8]. Once a suitable algorithm has been found, the user can employ LC's code generator to produce a standalone compressor and decompressor for that algorithm that does not require the framework.

Most of the components and preprocessors in LC's libraries support 1-, 2-, 4-, and 8-byte word sizes [5].

Both libraries are user customizable and extensible, meaning users are able to add their own data transformations by following the API outlined in the tutorial. LC then includes the new transformations in its search for a good compression algorithm and can also use them in the code generator.

1.2 Lossy-Mode Features

In addition to lossless algorithms, LC is able to generate lossy algorithms for 32-bit single- and 64-bit double-precision floating-point data in IEEE 754 format. It supports absolute, relative, normalized absolute, and combined absolute and relative error bounds [7]. Moreover, it guarantees that these point-wise error bounds are not violated by losslessly encoding any value that it cannot quantize within the user-provided error bound [4]. It supports all floating-point values, including infinities, not-a-number (NaN), and denormals. Each quantizer provides two modes, one that replaces the lost bits by zeros and another that replaces them by random bits to minimize autocorrelation between the errors.

2 Second Project Outcome: PFPL

We have used LC to generate several leading compressors. For example, it has been instrumental in the creation of PFPL, a lossy compressor for single- and double-precision floating-point data [4]. PFPL guarantees point-wise absolute (ABS), relative (REL), and normalized-absolute (NOA) error bounds. In fact, it is currently the only available compressor that guarantees the error bound on all three error-bound types. It is also one of very few compressors that runs on both CPUs and GPUs.

The rest of this section compares the compression ratio, compression speed, and decompression speed of PFPL to that of leading lossy compressors from the literature, including cuSZp [9], MGARD-X [10], SPERR [11], SZ3 [12, 13, 14], and ZFP [15]. Since most of them do not support REL (and NOA is a scaled version of ABS), we only show results for ABS. We evaluated the compressors on the 1E-1, 1E-2, 1E-3, and 1E-4 error bounds, all of which exhibit the same general trends [4]. Therefore, we only show results for 1E-3 here. Our evaluation system runs Fedora 37 and contains an AMD Ryzen Threadripper 2950X CPU with 16 cores that can simultaneously run 32 threads. The GPU in this system is an NVIDIA RTX 4090 with 16,384 processing elements. We also tested the codes on another system with a different CPU and GPU, which yielded similar trends.

Figures 2 and 3 show scatter plots of the compression ratio versus the compression throughput and of the compression ratio versus the decompression throughput, respectively. The Pareto front [16] is marked in blue, highlighting the best compressors. Note that both axes are logarithmic, meaning seemingly small differences are actually quite large. We used the 67 single-precision SDRBench inputs [17] that all listed compressors support and on which they do not violate the error bound by more than $1.5\times$. We excluded SZ2 [18] because its successor SZ3 outperforms it. For each included compressor, we show results for all supported modes, that is, serial, parallel CPU (OMP), and/or parallel GPU (CUDA) execution.

For both compression and decompression, $\text{PFPL}_{\text{CUDA}}$ is on the Pareto front even though the underlying algorithm and the code have been automatically synthesized. Note that the serial, OpenMP, and CUDA implementations yield the same compression ratio since they produce bit-for-bit identical output. Only SPERR and the two versions of SZ3 compress more than PFPL because they are CPU-only compressors that use GPU-unfriendly transformations to boost their compression ratio. For the same reason, the OpenMP version of SZ3 compresses significantly less than serial SZ3, that is, the serial version includes well-compressing transformations that are not parallelism friendly. The remaining compressors are both slower and compress less than $\text{PFPL}_{\text{CUDA}}$, except $\text{cuSZp}_{\text{CUDA}}$ decompresses more quickly. In particular, $\text{PFPL}_{\text{CUDA}}$'s throughput is two orders of magnitude higher than that of the CPU-based compressors on the Pareto front. Moreover, $\text{PFPL}_{\text{CUDA}}$ compresses over 4 times more than $\text{cuSZp}_{\text{CUDA}}$, the only other GPU compressor on the Pareto front. PFPL_{OMP} is the fastest parallel CPU code. Without $\text{PFPL}_{\text{CUDA}}$, it would be on the Pareto front.

The LC synthesized PFPL versions uniquely combine a high throughput with a high compression ratio. The remaining approaches on the Pareto front either compress well but are slow or have a high throughput

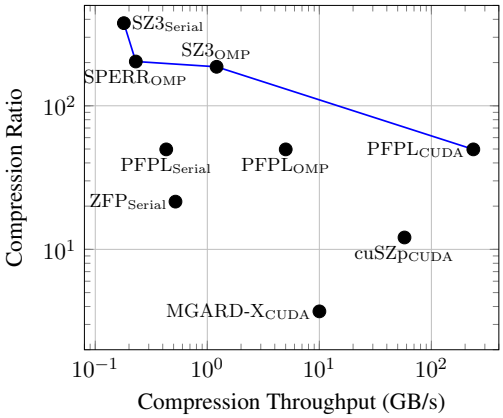

Figure 2: Geometric-mean compression ratio vs. compression throughput with an ABS error bound of 1E-3 on SDRBench inputs, including Pareto front

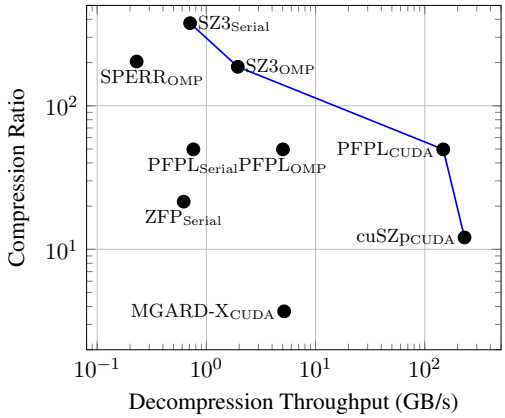

Figure 3: Geometric-mean compression ratio vs. decompression throughput with an ABS error bound of 1E-3 on SDRBench inputs, including Pareto front

but low compression ratio. PFPL is highly effective even though it supports key features that the other compressors lack. (1) It is fully CPU/GPU compatible, which is otherwise only the case for MGARD-X, but MGARD-X does not support REL and does not guarantee the error bound. (2) It supports the ABS and REL error-bound types, which is otherwise only the case for SZ2, but SZ2 does not guarantee the error bound on REL and only supports CPUs. (3) It guarantees the error bound for all supported types, which is otherwise only the case for SZ3, but SZ3 does not support REL and only runs on CPUs.

3 Third Project Outcome: FPcompress

We have also used LC to generate four state-of-the-art lossless compressors for scientific floating-point data. We named them SPspeed, SPratio, DPspeed, and DPratio [3]. They all support CPU and GPU execution. The "SP" versions target single-precision data whereas the "DP" versions target double-precision data. The "speed" versions aim to provide a high throughput at a good compression ratio. The "ratio" versions aim to provide a high compression ratio at a good throughput. We use the name "FPcompress" to summarily refer to all four compressors.

This section compares the compression ratio, compression throughput, and decompression throughput of the lossless GPU compressors ANS, Bitcomp, Cascaded, Deflate, Gdeflate, LZ4, Snappy, and ZSTD from the nvCOMP library [19] as well as SPspeed/SPratio [3], MPC [20], and Ndzip [21]. We also evaluated the lossless CPU compressors Bzip2 [22], Gzip [23], SPspeed/SPratio [3], SPDP [24], ZFP [25], FPzip [26], ZSTD [27], and Ndzip [28]. Our evaluation system is the same as above containing an RTX 4090 GPU with 16,384 processing elements and a Ryzen Threadripper 2950X CPU with 16 cores that can run 32 threads.

Figures 4 and 5 show scatter plots of the compression ratio versus the compression throughput and of the compression ratio versus the decompression throughput, respectively, for the GPU codes. Both axes are linear. The Pareto front is again marked in blue. We used 90 single-precision SDRBench files as inputs.

In both figures, the Pareto front includes SPratio, SPspeed, and Bitcomp-i0. SPratio delivers the highest compression ratio while also outperforming many of the other GPU compressors in throughput. SPspeed delivers the highest throughput and a higher compression ratio than most of the other compressors. Note that SPspeed compresses and decompresses at over half a terabyte per second on a single GPU.

Unlike all other tested compressors, the compressors in the nvCOMP library (including Bitcomp-i0) produce multiple compressed chunks that are *not concatenated*, which gives them a speed advantage. Perhaps even more importantly, the nvCOMP compressors only work on GPUs, meaning the compressed data cannot be used on a CPU. In contrast, LC synthesizes fully compatible versions of SPspeed and SPratio that work on CPUs and GPUs and concatenate the compressed data into a single memory block.

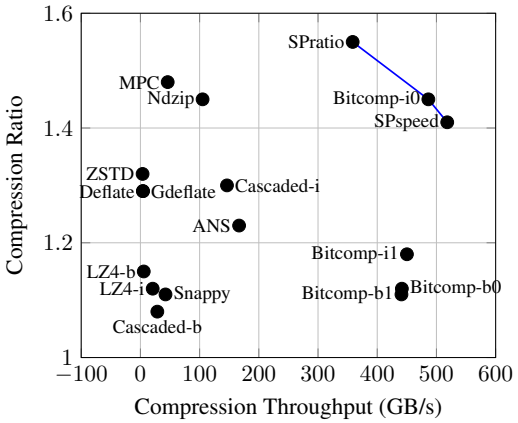

Figure 4: GPU geometric-mean lossless compression ratio vs. compression throughput on SDRBench inputs, including Pareto front

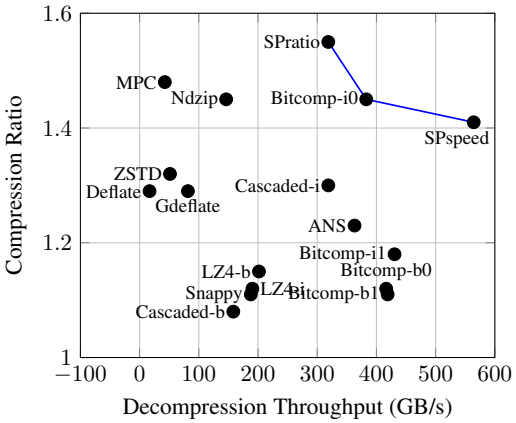

Figure 5: GPU geometric-mean lossless compression ratio vs. decompression throughput on SDRBench inputs, including Pareto front

On the CPU (results shown elsewhere [3]), SPspeed and SPratio also share the Pareto front with one other compressor. Both LC algorithms are faster than the other tested serial and parallel CPU compressors. Moreover, SPratio outperforms all of them in compression ratio except FPzip, which yields the highest geometric-mean compression ratio. However, SPspeed compresses $75\times$ faster and decompresses $55\times$ faster than FPzip.

4 Publications and other Project Outcomes

This project yielded the following 12 peer-reviewed scientific publications, including several in top venues.

- The first DCC'24 paper [29] presents an ML approach to predict effective compression algorithms for a given dataset. In particular, it shows that the compression ratios of a few fast compression algorithms make good features for classifying datasets and selecting good sophisticated compressors for them.
- The second DCC'24 paper [2] describes LICO, another compression algorithm that we designed with the help of LC. Unlike PFPL and PFcompress, LICO targets image data (i.e., integers), highlighting the versatility of LC. It is lossless and faster at compression and decompression than bzip2, gzip, Zstd, PNG, TIFF, and JPEG2000. It also compresses the tested images more than all of them except JPEG2000, which is over $45\times$ slower than LICO.
- The ESSA'24 paper [6] explains how LC can be used to create a separate, customized compressor for each file. Doing so boosts the compression ratio above what any single compressor can achieve, which is enabled by a compression-algorithm synthesis tool like LC.
- The CoDaC'24 paper [7] provides insight into how LC manages to guarantee the user-specified point-wise error bound for ABS, REL, and NOA. It further shows how LC guarantees bit-for-bit identical output on CPUs and GPUs. No other current compressor supports all these features.
- The ASPLOS'25 paper [3] presents the four FPcompress algorithms described above. They represent the current state of the art in lossless compression of floating-point data, in particular on GPUs.
- The ISPASS'25 paper [5] details the data transformations in LC's component library and measures how important the various transformations are. For example, it studies which transformations tend to occur in which stage of the compression pipeline, how they rank relative to each other, and what the impact on the compression ratio is when removing certain transformations from LC's library.
- The IPDPS'25 paper [4] presents the guaranteed-error-bounded lossy PFPL algorithm described above. It is a state-of-the-art compressor for scientific floating-point data on GPUs.
- The first DRBSD'25 paper [30] describes a new resolution-based progressive compression approach. It achieves competitive compression ratios against traditional compression methods and employs LC

to generate tailored compression algorithms for each progressive level.

- The second DRBSD'25 paper [8] evaluates over 100,000 GPU compressors generated by LC that are compiled with NVCC, Clang, and HIPCC. It demonstrates that the performance of the same compressor and decompressor can vary significantly depending on which compiler is used.
- The third DRBSD'25 paper [31] studies the compressibility of scientific floating-point datasets in Posit and IEEE-754 representation using several lossless compressors from the literature as well as some generated by LC. The results show that Posits are nearly as compressible as standard floats and indicate that special-purpose compressors for Posits may be desirable, including lossy versions.
- The SC'25 paper [32] describes the state of practice in lossy HPC compression. It is a summary of a workshop that examined application needs and compressor capabilities, including those of LC. It presents 24 takeaways and outlines gaps not addressed by current production compressors.
- The IPCCC'25 paper [33] introduces a new homomorphic approach called omni-homomorphic compression that is based on the aforementioned PFPL algorithm. Omni-homomorphic compression delivers orders of magnitude higher compression ratios and faster speeds than prior techniques while still making it possible to operate on the compressed data directly without the need for decompression.

5 Software Releases

This project has resulted in the following five open-source software releases. They are all freely available on GitHub under the 3-clause BSD license, which makes it simple to re-use, re-distribute, and produce derivatives of our work.

- The LC Framework is the main deliverable of this project [1]. It can automatically synthesize effective lossy and lossless compressors. It contains serial C++ and parallel C++/OpenMP code for CPUs as well as parallel CUDA and soon to be released HIP code for NVIDIA and AMD GPUs, respectively.
- The PFPL code [34] implements our fast and well-compressing lossy floating-point compressor that guarantees the error bound and is fully CPU/GPU compatible.
- The SPspeed, SPratio, DPspeed, and DPratio codes [35] implement four fast and well-compressing lossless compressors for floating-point data that are also fully CPU/GPU compatible.
- The LICO code [36] implements our fast and effective compressor for images. It supports serial and parallel CPU execution.
- The oHC code [37] provides a sample implementation of our omni-homomorphic compression approach. It includes serial and parallel CPU code as well as GPU code.

6 Training

This project has provided research training to six students.

- Noushin Azami has been one of the main implementers of LC. Her work focused on the lossless aspects of the framework [2, 3, 5, 6], which represents the bulk of her 2024 PhD dissertation.
- Alex Fallin is the other main implementer of LC. His work focuses on the guaranteed-error-bounded lossy aspects [4, 7, 33], which make up a large part of his PhD dissertation. He is expected to graduate in May 2026.
- Kayla Wesley worked on some components of LC, specifically a new parallelization of the LZ77 algorithm. The results of her work are described in her master's thesis. She graduated in 2022.
- Mackenzie Toliver used LC to find effective compression algorithms for small datasets. The results of her research are described in her undergraduate honor's thesis. She graduated in 2024.
- Brandon Burtchell has worked on an ML approach to predict good compressors in LC's vast search space [29], a progressive compression approach that uses LC to compress each progression level with a different pipeline [30], and a performance comparison of different compressors when compiled with different compilers [8]. He included these projects in his PhD proposal, which he recently defended.

- Andrew Rodriguez has used LC to generate per-file customized compression algorithms to boost the compression ratio [6] as well as on a comparison of compression ratios between scientific data encoded in IEEE 754 format and Posit format [31]. He included these two projects in his PhD proposal, which he also recently defended.

Bibliography

[1] Noushin Azami, Alex Fallin, Brandon Burtchell, Andrew Rodriguez, Benila Jerald, Yiqian Liu, Anju Mongandampulath Akathoott, and Martin Burtscher. LC Git Repository. `https://github.com/burtscher/LC-framework`, 2025. Accessed: 2025-12-23.

[2] Noushin Azami, Rain Lawson, and Martin Burtscher. LICO: An Effective, High-Speed, Lossless Compressor for Images. In *Proceedings of the 2024 Data Compression Conference*, 2024.

[3] Noushin Azami, Alex Fallin, and Martin Burtscher. Efficient Lossless Compression of Scientific Floating-Point Data on CPUs and GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 395–409, New York, NY, USA, 2025. Association for Computing Machinery.

[4] Alex Fallin, Noushin Azami, Sheng Di, Franck Cappello, and Martin Burtscher. Fast and Effective Lossy Compression on GPUs and CPUs with Guaranteed Error Bounds. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 874–887, June 2025.

[5] Noushin Azami and Martin Burtscher. Identifying Important Data Transformations for Synthesizing Effective Lossless Compressors. In *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 286–296, 2025.

[6] Andrew Rodriguez, Noushin Azami, and Martin Burtscher. Adaptive Per-File Lossless Compression of Floating-Point Data. In *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 423–430, 2024.

[7] Alex Fallin and Martin Burtscher. Lessons Learned on the Path to Guaranteeing the Error Bound in Lossy Quantizers. In *Workshop on Correct Data Compression*, 2024.

[8] Brandon Alexander Burtchell and Martin Burtscher. Characterizing the Performance of Parallel Data-Compression Algorithms across Compilers and GPUs. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, page 269–278, New York, NY, USA, 2025. Association for Computing Machinery.

[9] Yafan Huang, Sheng Di, Xiaodong Yu, Guanpeng Li, and Franck Cappello. cuszp: An ultra-fast gpu error-bounded lossy compression framework with optimized end-to-end performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'23, Denver, CO, USA, 2023. Association for Computing Machinery.

[10] Xin Liang et al. MGARD+: Optimizing multilevel methods for error-bounded scientific data reduction. `https://arxiv.org/abs/2010.05872`, 2020. Online.

[11] Shaomeng Li, Peter Lindstrom, and John Clyne. Lossy scientific data compression with sperr. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1007–1017, 2023.

[12] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447, 2018.

[13] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023.

[14] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1643–1654, 2021.

[15] James Diffenderfer, Alyson L. Fox, Jeffrey A. Hittinger, Geoffrey Sanders, and Peter G. Lindstrom. Error analysis of zfp compression for floating-point data. *SIAM Journal on Scientific Computing*, 41(3):A1867–A1898, 2019.

[16] Kaisa Miettinen. Theoretical analysis of multiobjective optimization. *Doctoral Dissertation, Acta Polytechnica Scandinavica, Mathematics and Computing Series, No. 67*, 1998.

[17] Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. SDRBench: Scientific data reduction benchmark for lossy compressors. In *International Workshop on Big Data Reduction (IEEE IWBDR20) in conjunction with IEEE International Conference on Big Data (IEEE BigData20)*, 2020.

[18] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447, 2018.

[19] nvcomp: Nvidia gpu data compression library. `https://github.com/NVIDIA/nvcomp`, 2025. Accessed: 2025-09-24.

[20] A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher. MPC: A Massively Parallel Compression Algorithm for Scientific Data. In *2015 IEEE International Conference on Cluster Computing*, pages 381–389, Sept 2015.

[21] Fabian Knorr, Peter Thoman, and Thomas Fahringer. ndzip-gpu: efficient lossless compression of scientific floating-point data on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[22] Julian Seward. bzip2 and libbzip2. *avaliable at http://www.bzip.org*, 1996.

[23] Gzip: The gnu data compression utility. `https://www.gnu.org/software/gzip/`. Accessed: 2025-09-24.

[24] S. Claggett, S. Azimi, and M. Burtscher. SPDP: An Automatically Synthesized Lossless Compression Algorithm for Floating-Point Data. In *Proceedings of the 2018 Data Compression Conference*, DCC '18, pages 337–346, 2018.

[25] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 20(12):2674–2683, 2014.

[26] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 12(5):1245–1250, 2006.

[27] inikep. lzbench. `https://github.com/inikep/lzbench`, 2025. Accessed: 2025-09-24.

[28] Fabian Knorr, Peter Thoman, and Thomas Fahringer. ndzip: A high-throughput parallel lossless compressor for scientific data. In *2021 Data Compression Conference (DCC)*, pages 103–112. IEEE, 2021.

[29] Brandon Alexander Burtchell and Martin Burtscher. Using Machine Learning to Predict Effective Compression Algorithms for Heterogeneous Datasets. In *2024 Data Compression Conference (DCC)*, pages 183–192, 2024.

[30] Brandon Alexander Burtchell and Martin Burtscher. Building n-Dimensional Trees for Resolution-Based Progressive Compression. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, page 307–313, New York, NY, USA, 2025. Association for Computing Machinery.

[31] Andrew Rodriguez and Martin Burtscher. On the Compressibility of Floating-Point Data in Posit and IEEE-754 Representation. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, page 300–306, New York, NY, USA, 2025. Association for Computing Machinery.

[32] Franck Cappello, Robert Underwood, Yuri Alexeev, Alison Baker, Ebru Bozdağ, Martin Burtscher, Kyle Chard, Sheng Di, Kyle Gerard Felker, Paul Christopher O'Grady, Hanqi Guo, Yafan Huang, Peng Jiang, Sian Jin, Petter Johansson, Shaomeng Li, Xin Liang, Erik Lindahl, Peter Lindstrom, Zarija Lukić, Magnus Lundborg, Danylo Lykov, Masaru Nagaso, Kento Sato, Amarjit Singh, Seung Woo Son, Shihui Song, William Tang, Dingwen Tao, Jiannan Tian, Kazutomo Yoshii, and Kai Zhao. What to Support When You're Compressing: The State of Practice Gaps and Opportunities for Scientific Data Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, page 1966–1979, New York, NY, USA, 2025. Association for Computing Machinery.

[33] Alex Fallin and Martin Burtscher. Omni-Homomorphic Compression for Large Scientific Datasets. In *44th IEEE International Performance Computing and Communications Conference*, November 2025.

[34] Alex Fallin and Martin Burtscher. PFPL Git Repository. `https://github.com/burtscher/PFPL`, 2025. Accessed: 2025-12-23.

[35] Noushin Azami and Martin Burtscher. FPcompress Git Repository. `https://github.com/burtscher/FPcompress`, 2025. Accessed: 2025-12-23.

[36] Noushin Azami and Martin Burtscher. LICO Git Repository. `https://github.com/burtscher/LICO`, 2025. Accessed: 2025-12-23.

[37] Alex Fallin and Martin Burtscher. oHC Git Repository. `https://github.com/burtscher/oHC`, 2025. Accessed: 2025-12-23.