Precision Cascade: A novel algorithm for multi-precision extreme compression

J. Gonzalez[1], J. Lauret[2], Y. Ying[3], G. Van Buren[2], M. Burtscher[4], P. Canal[5], I. A. Cali[3], R. Nunez[1]

[1] Accelogic LLC, Weston, Florida, USA
[2] Brookhaven National Laboratory, Upton, New York, USA
[3] MIT, Cambridge, Massachusetts, USA
[4] Texas State University, San Marcos, Texas, USA
[5] Fermi National Laboratory, Batavia, Illinois, USA

E-mail: `kying@alum.mit.edu`

Abstract. Lossy compression algorithms are incredibly useful due to powerful compression results. However, lossy compression has historically presented a trade-off between the retained precision and the resulting size of data compressed with a lossy algorithm. Previously, we introduced BLAST, a state-of-the-art compression algorithm developed by Accelogic. We presented results that demonstrated BLAST can achieve a compression factor that undeniably surpasses compression algorithms currently available in the ROOT framework. However, the leading concern of utilizing the lossy compression technique is the delayed realization that more precision is necessary. This precision may have been irretrievably lost in an effort to decrease storage size. Thus, there is immense value in retaining higher precision data in reserve. Though, in the era of exabyte computing, it becomes extremely inefficient and costly to duplicate data stored at different compressive precision values. A tiered cascade of stored precision optimizes data storage and resolves these fundamental concerns.

Accelogic has developed a game-changing compression technique, known as "Precision Cascade", which enables higher precision to be stored separately without duplicating information. With this novel method, varying levels of precision can be retrieved, potentially minimizing live storage space. Preliminary results from STAR and CMS demonstrate that multiple layers of precision can be stored and retrieved without significant penalty to the compression ratios and (de)compression speeds, when compared to the single-precision BLAST baseline.

In this paper, we will present the integration of Accelogic's "Precision Cascade" into the ROOT framework, with the principal purpose of enabling high-energy physics experiments to leverage this state-of-the-art algorithm with minimal friction. We also present our progress in exploring storage reduction and speed performance with this new compression tool in realistic examples from both STAR and CMS experiments and feel we are ready to deliver the compression algorithm to the wider community.

1. Introduction

Lossy compression algorithms have been long applied to sound and image processing, but sophisticated algorithms have struggled to infiltrate the physics world. However, the need for data compression algorithms has increased drastically with the growth of large-scale numerical data, especially in the High Energy and Nuclear Physics (HENP) community [1]. Accelogic, LLC has pioneered a novel compression theory, known as "Compressive Computing" [2, 3]. At the root of this theory lies a radically new concept known as "sensitivity", which has changed our understanding of numerical precision, and introduced intelligent lossy compression to the realm of physics.

At the forefront of this revolution is the idea that most data in practice is comprised of many "zero-information-bearing bits", or "zibbits", which are bits that carry zero or insignificant information. Therefore, one can make the argument that removing such bits is not a true loss, deeming compression techniques targeted at them to be only "quasi-lossy". The reason behind this is due to inherent physical limitations in measurement or precision, as well as insignificance of the data, in terms of numerical scale [1, 4].

Using these concepts, Accelogic has engineered a powerful compression algorithm library, known as BLAST, offering stunning lossy and lossless compression algorithms. Previously, we demonstrated that Accelogic's lossy compression surpasses common compression algorithm baselines, such as gzip [5] and float16. We performed case studies on datasets from the STAR and CMS experiments using BLAST. The CMS study was performed on binary files, while the STAR study was performed within ROOT, an object-oriented data analysis framework commonly used for HENP [6]. In both case studies with STAR and CMS, the BLAST algorithms (at acceptable compression levels that offer mid-precision compression) are able to outperform gzip by approximately 2-4$\times$ [4].

We emphasize that our previous results prove lossy compression works. However, we also determined that compression levels above a certain threshold of loss may deteriorate the data too much, such that the lossy dataset results in strong deviations from the original dataset. Such instances voice concerns over too much precision irretrievably lost.

2. Precision Cascade

In the existing landscape, the user would need to lossy compress duplicates of the dataset, at various compression levels, in order to store multiple copies of varying precision. However, the duplication of data wastes storage space, defeating the purpose of these extreme compression algorithms. Thus, these concerns beg the question: *can we retrieve varying levels of precision without being repetitive?*

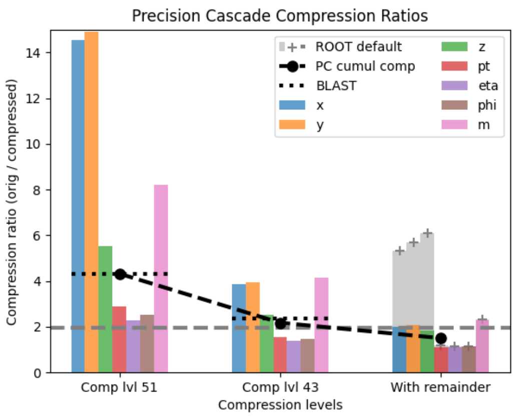
Accelogic has developed a novel technique, known as Precision Cascade, in response to this question. Precision Cascade allows higher precision to be stored separately without duplicating information, by storing higher precision bits in separate files. These files can then be later combined with lower precision files in order to retrieve higher levels of precision, while the user retains control over defining the levels of precision.

3. Results on CMS and STAR datasets

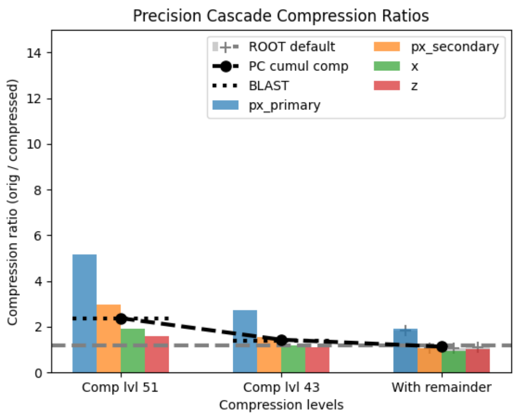
We assessed the performance of the algorithm on subsets of data from the STAR and CMS experiments. Using ROOT, we applied the Precision Cascade compression algorithms to extracted branches from particle data within the experiment datasets, and compared the results against the inside-of-ROOT single-precision BLAST baseline, as well the default ROOT compression algorithm. In the following section, we demonstrate results for the three metrics we assessed: compression ratio, compression speed, and decompression speed.

3.1. Compression Ratio

The primary metric for assessing a compression algorithm's performance is the compression ratio, which represents how much larger the original file is than the compressed (ratio between original and compressed). The compression ratios for Precision Cascade are shown against the BLAST and ROOT baselines in Figure 1 below.



(a) CMS Compression Ratios



(b) STAR Compression Ratios

Figure 1: Comparison of compression ratio against BLAST single-precision baseline and ROOT default compression algorithms

The multi-colored bars represent the compression ratio of individual columns of stored particle information in the dataset for the various tiers of Precision Cascade precision, as labeled on the x-axis. The overall Precision Cascade compression ratio is shown as the black dot, and is compared against the BLAST single-precision baseline at each precision level, denoted by the dotted black line. The lossless ROOT compression baseline is shown in gray bars and markers for the individual columns, and the overall ROOT baseline compression ratio is shown as the gray dashed line.

In both CMS and STAR, we demonstrate that Precision Cascade at the first compression tier, level 51, is approximately double that of the ROOT default, and the same as the BLAST default. Even at the second compression tier, level 43, Precision Cascade slightly outperforms the ROOT default, although it decreases in performance relative to the BLAST baseline at compression level 43. This can be attributed to overhead for Precision Cascade. Retrieving the entire dataset (losslessly) demonstrates similar or worse levels of compression relative to the ROOT baseline. We emphasize that our results demonstrate there is non-negligible overhead for storing precision in different locations, but this may be worthwhile, as the lossy compression still outperforms the ROOT baseline.

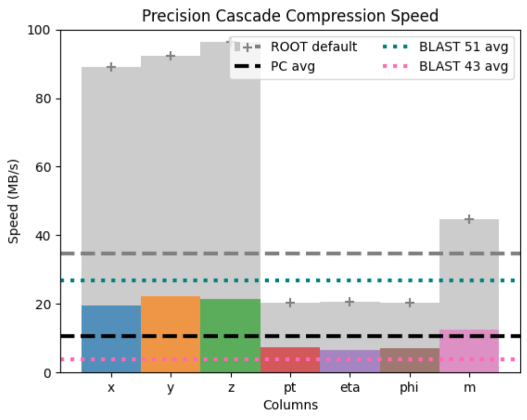
3.2. Compression Speed

Secondary to compression ratio, we also assessed speed as a measure of performance. The compression speed is shown in Figure 2. With the multi-precision compression algorithm, the compression occurs all at once, thus we are unable to assess the compression speeds of each compression tier individually.

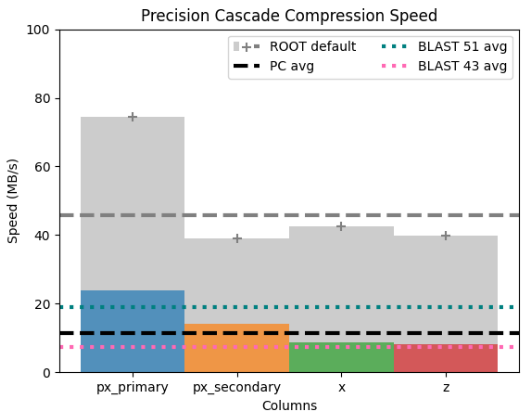
Compared to the ROOT baseline, we see a clear decrease in performance by approximately 3×. Compared to the BLAST single-precision baseline, the Precision Cascade speed counterintuitively outperforms the higher precision (level 43) BLAST speed, but manages to underperform relative to the lower precision (level 51) BLAST. Thus, we conclude that utilizing lossy compression may result in slower compression speeds compared to lossless ROOT baselines.

3.3. Decompression Speed

Alongside compression speed, we also assessed decompression speed. The decompression speeds for each compression tier are shown in Figure 3.

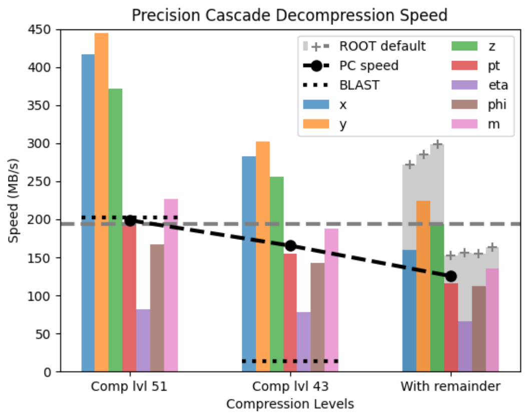


(a) CMS Compression Speeds

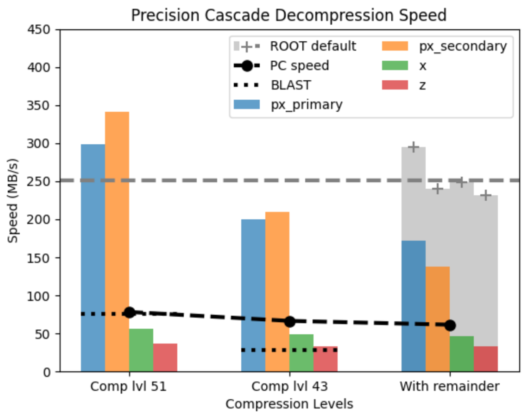


(b) STAR Compression Speeds

Figure 2: Comparison of compression speed against BLAST single-precision baseline and ROOT default compression algorithms



(a) CMS Decompression Speeds



(b) STAR Decompression Speeds

Figure 3: Comparison of decompression speed against BLAST single-precision baseline and ROOT default compression algorithms

As the precision grows, the decompression speed decreases. For CMS, the decompression speed at compression level 51 is slightly faster than the ROOT default, though the higher precision levels fall below baseline. For STAR, all the decompression speeds fall short of the ROOT baseline by approximately 3×. These compression/decompression speeds demonstrate the cost of multi-precision, extreme compression, compared to existing lossless algorithms.

4. Utility

Despite slower (de)compression speeds, there is still great advantage in piece-wise lossy compression. One of the primary benefits of Precision Cascade is that it allows the user to store highly compressed data in expensive fast access storage, and the remaining tiers in cheaper archival storage. Certain analyses that require less precision can quickly access the high compression data, while analyses that require more precision are still able to retrieve it later via archival storage.

Some analyses require more precision, but fewer statistics. With Precision Cascade, the user also can control which files are stored with higher precision. In an experiment with many files, users have the option of storing higher precision files for only a subset of the files comprising the entire dataset.

Furthermore, another large advantage of Precision Cascade is that the user can rebuild the original dataset, losslessly, if necessary. For example, the user may need to recalibrate precision utilizing the original dataset. Alternatively, the user may realize that too much precision is lost in the lossy tiers. However, this precision is retrievable with Precision Cascade.

5. ROOT Integration

BLAST has already been integrated and tested in ROOT. The BLAST compression suite can be applied to all branches containing homogeneous numerical types (e.g. split branches), with the exception of Double32_t and Float16_t, which are already lossy and thus not supported. BLAST is lossy for floating point branches and lossless for integer branches.

Precision Cascade is supported only for floating type values, namely doubles and floats. The output files are named according to the cascade tier and the suffix is customizable. Currently, the format is `${original_filename}_precisioncascade_[1,2,3,etc.].root`. When reading the output file, all cascade files are automatically used if present, invisible to the user. For example, immediately after writing to the output, reading back would be done with full precision. If the last cascade tier file is removed, then reading back would be done with high precision (this equates to compression level 43 in the CMS/STAR case study above).

A code example for Precision Cascade usage is shown below. The user must define a vector of strictly decreasing `Int_t` values representing the compression levels for the tiers of Precision Cascade. The user then defines a `PrecisionCascadeCompressionConfig`, with parameters for the compression suite (`kBLAST`), the levels defined initially, and a boolean representing whether or not to keep the residuals that allow the user to rebuild the original file. The user then applies these compression settings to the desired branch to compress.

```
std::vector<Int_t> levels = { 51, 43 };
ROOT::PrecisionCascadeCompressionConfig targetConfig(
    ROOT::RCompressionSetting::EAlgorithm::kBLAST,
    levels,
    true /* Keep also the residual file */ );
...
lossy_branch->SetCompressionSettings(targetConfig);
```

6. Open Issues

From a technical standpoint, all pieces of BLAST and Precision Cascade are ready to deploy to the community. A pending license would grant free unlimited permission for ROOT to use and integrate the codes for non-profit/academic communities (i.e. redistribute sources as part of ROOT releases). In the interim, binary library distributions are available for the early adopter.

7. Conclusion

Derived from the theory of ”Compressive Computing”, BLAST has demonstrated outperformance of ROOT compression, in terms of compression factor, in many cases. For certain analyses, lossy compression retains enough precision for the data to still be useful.

One of the main concerns surrounding use of lossy compression has beenthe irretrievable nature of the ”lost” bits, rendering data useless if later on, the user needs more precision. Precision Cascade is a novel algorithm that solves this concern by allowing users to save ”lost” bits in separate files, such that a user can retrieve higher precision data if necessary. With Precision Cascade, we hope to build the community’s confidence in utilizing lossy compression algorithms and minimize the friction in doing so. We are hoping to release the BLAST algorithm suite in ROOT soon to the community, so please stay tuned!

8. Acknowledgments
This work was supported by the U.S. Department of Energy, SBIR/STRR Program of the Office of science, Nuclear Physics under Award Number DE-SC0018521, as well as other contracts/grants from NASA, the U.S. Department of Defense, and the U.S. Department of Energy, and private support from Intellectual Property Systems, LLC and Accelogic, LLC.

References

[1] Lauret, Jérôme, Gonzalez, Juan, Van Buren, Gene, Nuñez, Rafael, Canal, Philippe, and Naumann, Axel, "Extreme compression for large scale data store," *EPJ Web Conf.*, vol. 245, p. 06024, 2020.

[2] J. G. Gonzalez, S. A. Fonseca, and R. C. Nunez, "Arrangements for communicating data in a computing system using multiple processors," 5 2019.

[3] J. G. Gonzalez, S. A. Fonseca, and R. C. Nunez, "Arrangements for communicating data in a computing system using multiple processors," 3 2021.

[4] J. Lauret, J. Gonzalez, P. Canal, G. Buren, M. Burtscher, I. Cali, R. N. nez, and Y. Ying, "Root files improved with extreme compression." ACAT 2021, November 2021.

[5] P. Deutsch and J.-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3." RFC 1950 (Informational), May 1996.

[6] R. Brun and F. Rademakers, "Root — an object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 389, no. 1, pp. 81–86, 1997. New Computing Techniques in Physics Research V.