Progress toward Accelogic compression in ROOT

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

1. Fermi National Laboratory, Batavia, Illinois
2. Brookhaven National Laboratory, New York
3. Accelogic LLC, Weston, Florida
4. MIT, Boston, Massachusetts
5. Texas State University, San Marcos, Texas

pcanal@fnal.gov

Abstract. For the last 7 years, Accelogic pioneered and perfected a radically new theory of numerical computing codenamed "Compressive Computing", which has an extremely profound impact on real-world computer science [1]. At the core of this new theory is the discovery of one of its fundamental theorems which states that, under very general conditions, the vast majority (typically between 70% and 80%) of the bits used in modern large-scale numerical computations are absolutely irrelevant for the accuracy of the end result. This theory of Compressive Computing provides mechanisms able to identify (with high intelligence and surgical accuracy) the number of bits (i.e., the precision) that can be used to represent numbers without affecting the substance of the end results, as they are computed and vary in real time. The bottom-line outcome will be to provide state-of-the-art compression algorithms --and accompanying software libraries-- able to surpass the performance of the compression engines currently available in the ROOT [7] framework. The resulting technology has the capability to enable substantial economic and operational gains (including speedup) for High Energy and Nuclear Physics data storage/analysis. In our initial studies, a factor of nearly x4 (3.9) compression was achieved with RHIC/STAR data where ROOT compression managed only x1.4 [6].

As a collaboration of experimental scientists, private industry, and the ROOT Team, our aim is to capitalize on the substantial success delivered by the initial effort and produce a robust technology properly packaged as an open-source tool that could be used by virtually every experiment around the world as means for improving data management and accessibility.

In this contribution, we will present our efforts integrating our concepts of "functionally lossless compression" within the ROOT framework implementation, with the purpose of producing a basic solution readily integrated into HENP applications. We will also present our progress applying this compression through realistic examples of analysis from both the STAR and CMS experiments.

1. Accelogic's BLAST Compression Algorithms

Accelogic LLC has developed the so-called theory of Compressive Computing, which enables the injection of unprecedented speedups in large scale numerical HPC software. One of the key elements behind the success of Compressive Computing is the development of a new theory (and accompanying practice) of precision which departs from the traditional theory of numerical error through a powerful

new concept called "sensitivity" [1-3]. Numerical sensitivity plays a role which is dual and complementary to that of numerical error, and it presents itself naturally when the goal is to identify and compress (or truncate) useless bits. When applied to the problem of data storage, these concepts lead to compression techniques with unprecedented compression factors for large-scale numerical data (like the data stored in Nuclear Physics facilities). We call these techniques "quasi lossy". Compressive Computing teaches us that most data computed in practice carries intrinsically a majority of bits that bear no information at all (so-called "zero-information-bearing bits" or simply "zibbits"). The argument can be made that if bits carry zero or insignificant information, then losing their content is not a true "loss". Bits might carry zero or insignificant information due to obvious factors such as limitations in measurement/precision or insignificance of the data (do we care of the gram level for measuring the weight of an elephant?). Another important reason for the ubiquitous presence of zibbits in real life is the not-so-obvious factor of "sensitivity propagation" [1-3].

The library codes provided by Accelogic's Compressive Computing library, called *COMPRESSIA*,[1] offer four product lines with different application fields, namely communication acceleration, memory-access acceleration, data-storage enhancement, and graph-database acceleration. In this paper, we are concerned about the data-storage enhancement product line, codenamed BLAST. The BLAST compression engine offers (sensitivity-based) tunable parameters controlling the level of lossiness for its data compression and decompression algorithms. Finding the right value of these parameters for a given use case requires evaluation of the effect on the end result to find the optimal compression without information loss. BLAST also provides lossless algorithms that can provide unprecedented compression factors when targeted to Nuclear Physics facility data.

We performed considerable performance testing of several key library algorithms on both *outside-of-ROOT data (*i.e., standalone binary files containing data extracted from ROOT trees) and *inside-of-ROOT data* (i.e., typical data inside of ROOT, compressed and decompressed transparently through "root2root" conversion). The tests used convoluted physics signals, so to give a larger and more reliable view of real-life impact than looking at a single variable distribution. We look at both the effect on data volume reduction (Compression ratio) and on compression and decompression speeds. We tested across different data types and different buffer sizes.

2. ROOT Integration

We extended the ROOT I/O library, on an experimental branch, to allow compression of the data using BLAST second-generation algorithms. We also created a standalone repository for BLAST. The branch allows for an active development environment with aim to include more of BLAST's compression schemes in the future (third-generation and beyond). The ROOT code is rapidly evolving to handle the specifics of working with BLAST, e.g., data type in source buffers is relevant.

Any lossy compression is entirely type dependent, so we need to enhance the internal code in the `TTree` class [7] to pass the Branch/Basket [7] data type down to the compression engine. In a ROOT Basket the data is stored in a platform independent format and in particular it is bytes swapped compared to the in-memory format x86 and amd64 architectures. Consequently, we had to enhance the BLAST code to be aware of whether it needed to byte-swap the data before attempting to compress it. Usually, the buffer created by ROOT that contains the data is followed up by an array of 32-bit `integers` describing array lengths or entry boundaries. When passed to the lossy a compression engine that expects `floats` or `doubles`, this resulted in the array of 32-bit `integers` to lose its significance (both because it was interpreted as a floating pointing value and because some of the bits were drops when in this case all the bits are significant). To solve this issue we needed to turn on the ROOT feature that

[1] COMPRESSIA, currently in commercial pre-release stage, is the result of 17 years of maturation of the theory and practice of Compressive Computing at Accelogic. It was made possible thanks to multi-million-dollar research funding provided by ten Phase II SBIR contracts from NASA, the U.S. Department of Energy, and the U.S. Department of Defense. Aiming at maximizing the benefits of this technology to the Nuclear Physics community, and working jointly with Brookhaven and Fermi National Laboratories, Accelogic has committed to providing free licensing of all necessary technologies (including intellectual property) for the seamless integration inside ROOT of key Compressive Computing data storage technologies.

avoid the addition of the `integer` array to the buffer (`ROOT::Experimental::EIOFeatures::kGenerateOffsetMap`).

Selecting the new compression algorithm is done via the usual compression setting interfaces (enumeration value: `ROOT::EAlgorithm::kBLAST`). The meaning of the level is dependent of the data type (single level for `integer` type and an indication of the desired loss for floating point values). If the type of the data is not supported, the buffer is compressed using the default compression engine and level (usually gzip). We are still in the early phase of testing compression within ROOT and so far, it compares well with outside-of-ROOT, but some plots shown here are only using an outside-of-ROOT approach.

3. Results on CMS and STAR data files

One note is that the inside-of-ROOT compression algorithm performed on the STAR dataset has integer tuning parameters ranging from -60 to +10, inclusive, where -60 is the least compressive and +10 is the most compressive. Meanwhile, the outside-of-ROOT algorithm performed on the CMS dataset is limited to a subset of the inside-of-ROOT parameters: {-13, -8, -7, -4, 0}[2]. Similar to the range above, the more negative the value, the higher compression parameter. The compression ratio resulting from various compression parameters is shown in Figure 1 below.

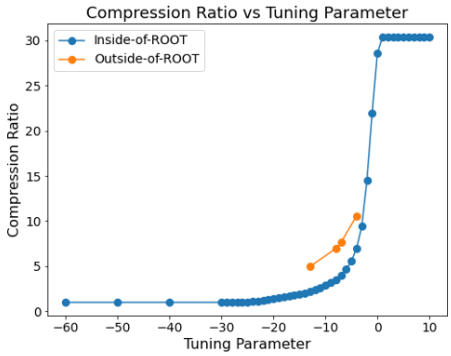


Figure 1: Comparing compression ratios of various tuning parameters recorded for the inside-of-ROOT compression on STAR data, as well as the outside-of-ROOT compression on CMS simulated data

3.1. CMS Experiment study: jets

The CMS team has been exploring *outside-of-ROOT* compression on select columns of CMS-like particle Monte Carlo simulation data. The columns extracted from the simulated events are as follows: particle ID (`integer`) [4], status code (`integer`), transverse momentum (`double`), pseudorapidity (`double`), azimuthal angle (`double`), and mass (`double`).

The workflow for this compression testing has been to extract the desired values from the original ROOT tree generated from the Monte Carlo simulated events and write the values to a binary file to feed into the *outside-of-ROOT* compression algorithm. Once the binary files are compressed, we decode the values from the output file and compute any particle quantities, such as the track momentum, necessary for the jet clustering algorithm. Finally, we produce the jets from the particle-level quantities by running the anti-kT jet clustering algorithm [5].

It is worth noting that while there are only six columns per simulated particle extracted from the original dataset, we must include one additional column, representing the event number of the particle, in order to keep clear divisions of entries. Thus, the compression is performed and assessed on seven columns in the binary file.

To assess the quality of the jets resulting from compressed data, we inspect two jet substructure observables, jet shape and jet fragmentation function, and compare the results with jets from the original, uncompressed data. A visual description of each substructure is shown in Figure 2. Jet shape shows the

[2] While this is the entire range, for the following study, only a subset of both the outside-of-ROOT and inside-of-ROOT parameters were selected and studied.

transverse momentum as a function of radial distance from the jet axis. On the other hand, jet fragmentation function describes the longitudinal energy distribution of tracks within a jet.

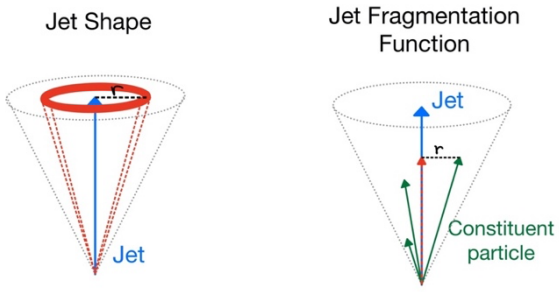


Figure 2: Visuals representing jet shape and jet fragmentation function

The jet shape of the simulated jet sample is shown in Figure 3, and the jet fragmentation function of the same sample is shown in Figure 4. In both plots, the compression with tuning parameters {-13, -8, -7} appear to be very close to the original, deviating only at most 1.327% from the original value for jet shape, and at most 1.081% from the original value for jet fragmentation. Further analysis is required for conclusive understanding, but these results demonstrate potential for data integrity post-compression. However, observing the results for parameter -4, we notice that there are clear deviations from the original data, which lead to different conclusions about the distribution of energy within jets in the sample, rendering this compression too lossy and not viable. Using this compression parameter, the maximal deviation is 17.44% for jet shape, and 145.8% for jet fragmentation. It is worth noting that at even the least compressive *outside-of-ROOT* compressor parameter, X = -13, BLAST's compression ratios on the double columns (p_T, η, ϕ, m) range from *~4-15x* improvement, while gzip only provides *~2-4x* improvement.

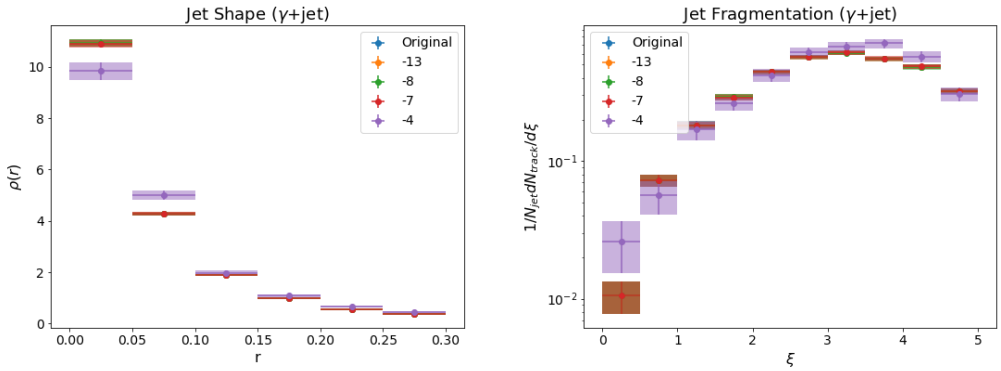


Figure 3/4: Jet shape and jet fragmentation function for photon-jet samples. While the compression parameters {-13, -8, -7} appear to give results near the original value, the compression parameter –4 returns results with unacceptable deviation.

3.2. STAR Experiment study

3.2.1. Invariant mass information retention and loss

We looked at various metrics of information retention vs. BLAST `float` compressor parameter X with STAR data on invariant mass distribution of K^0_s (only showing two metrics here to keep the message clear). With BLAST, we found essentially full information retention at a compression of *~x3.0* for `floats` representing particle momenta components used in calculating the invariant mass, which gzip and others compressed at only *~x1.07* (what we get with the existing ROOT). Examining metrics like this are a path for tuning optimal compressor parameters, but metrics will be specific to the science of interest.

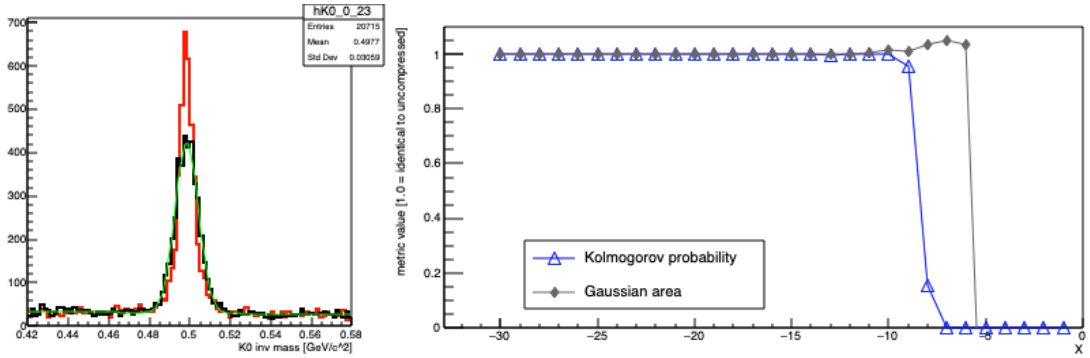


*Figure 5/6: Example of information lost in a degraded K^0_s inv. mass peak (**black**) vs. the original distribution (**red**). The green curve is a Gaussian plus background fit to the reconstructed mass peak after BLAST compression. The ratio of the compressed Gaussian area to uncompressed is shown in the right plot as a possible metric that ideally should be 1.0, along with the unbinned Kolmogorov probability measured between the original and BLAST-compressed invariant masses.*

3.2.2. (De)compression speed comparisons

In the graphs below, we select the highest compression setting of each lossless encoder, along with two compression parameter choices for BLAST: no noticeable changes at all (X = -27), and with no/negligible loss of information (X = -10). Performance is shown vs. the size of the compression buffer in bytes, with speeds reported as the number of 32-bit `floats` (de)compressed per second using consistent conditions across all compressors. We can see that BLAST with no/negligible loss of information reaches a significantly higher compression ratio while being quite competitive in terms of both compression and decompression speed, actually accelerating with more aggressive compression.

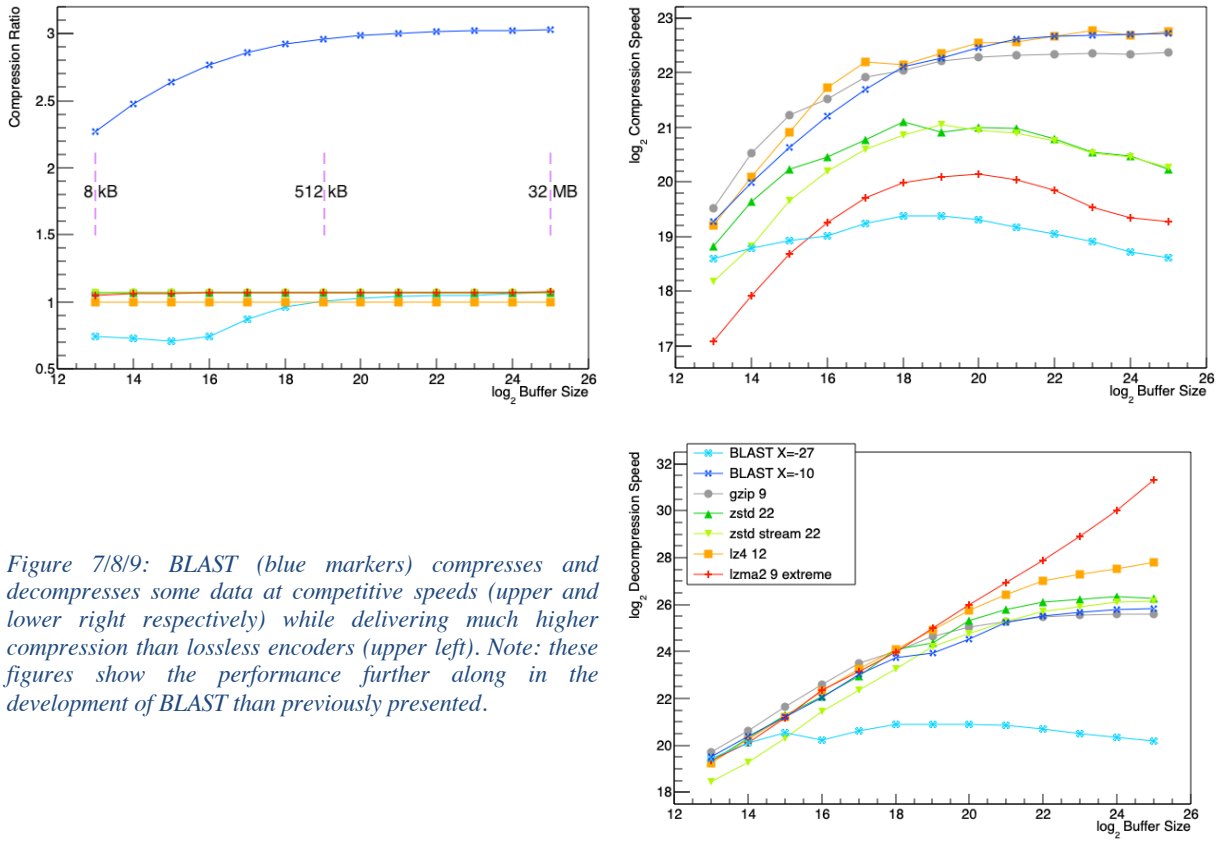
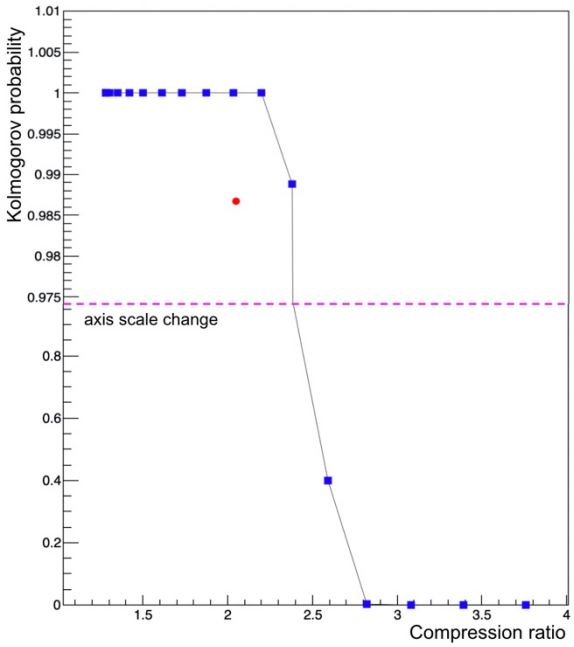


Figure 7/8/9: BLAST (blue markers) compresses and decompresses some data at competitive speeds (upper and lower right respectively) while delivering much higher compression than lossless encoders (upper left). Note: these figures show the performance further along in the development of BLAST than previously presented.

3.2.3. Other lossy compression techniques



We also compared BLAST to existing data types available in ROOT that embed a lossy compression into the transition between in-memory and binary-platform-independent formats (`Float16_t` and `Double32_t`). STAR has also employed common bit-packing, where 32-bit `floats` are compressed by mapping a pre-defined range of values into the range of 16-bit `shorts` through linear scaling plus type casting. Both approaches can then be followed by a subsequent lossless compression step. BLAST outperformed both techniques with the STAR data, retaining as much information at higher compression ratios: 2.37 vs. 2.05 in the `Float16_t` with gzip test (shown in Figure 10), and 2.75 vs. 2.36 in the `short` with gzip test.

Figure 10: BLAST (blue) retains information in a distribution as measured by an unbinned Kolmogorov probability as well as `Float16_t` with gzip (red) at a higher compression ratio.

4. Open issues and step forward.

We have several additional challenges and tasks to tackle. The release of comprehensive licenses for both the software and the associated intellectual property is a crucial step, which is expected to be done immediately after project closing (2022). This would include free unlimited permission for ROOT to use and integrate both the patents and the codes (i.e., redistribute sources as part of ROOT releases). The patent process seems to be on schedule so far. In the interim, binary library distribution is envisioned.

The ZIG capability (ability to save the data in incremental compression strength spread over multiple files / branches) has already been implemented and tested successfully. ZIG is believed to be key to build end-user's ease and confidence in using lossy compressions, and we expect to be reporting on it soon.

Instrumentation for automatic guessing of the optimal tuning value is not provided for now. Data compression with the potential for information loss should not be done blindly; the subtleties of compression and data organization in experiment-specific formats may require performing *once* similar analyses as STAR and CMS have done here to understand the case-by-case impacts. However, we made the key finding that for both CMS and STAR (two independent experiments with different analysis data formats) the same range of "*X*" lossiness leads to maximum compression with a negligible loss of information.

5. Conclusions

The libraries are ready for integration and in final testing phase. The distribution will be done at first via pre-built binary. The compression ratios achieved exceed those of classic compression algorithms: from x2 to x4 better compression ratios. Static alternatives, such as the use of `Float16_t` / `Double32_t` [7], have the substantial problem that they may render acceptable results for some data, and unacceptable ones for other data. In our experiments, even with the above serious concern, the usage of these alternatives is still inferior in performance to BLAST, with BLAST showing a 16% superiority in compression ratio, and also maintaining the ability to be used without class declaration changes.

6. Acknowledgments

This material is based upon work partially supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics SBIR 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] Juan G. Gonzalez, Santiago A. Fonseca, Rafael C. Nunez. *Arrangements for communicating data in a computing system using multiple processors*. U.S. Patent # 10965744, Granted March 30, 2021.

[2] Juan G. Gonzalez, Santiago A. Fonseca, Rafael C. Nunez. *Arrangements for communicating data in a computing system using multiple processors*. U.S. Patent # 10305980, Granted May 28, 2019.

[3] Juan G. Gonzalez, Santiago A. Fonseca, Rafael C. Nunez. *Arrangements for storing more data in faster memory when using a hierarchical memory structure*. U.S. Patent # 10114554, Granted October 30, 2018.

[4] Particle Physics Group, et al. *Review of Particle Physics*, Physics Letters B 667:1 (2008), doi 10.1016/j.physletb.2008.07.018

[5] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. *The anti-kt jet clustering algorithm*. Journal of High Energy Physics (2008), doi 10.1088/1126-6708/2008/04/063

[6] Jérôme Lauret, Juan Gonzalez, Gene Van Buren, Rafael Nuñez, Philippe Canal and Axel Naumann. *Extreme Compression for Large Scale Data Store*, EPJ Web of Conferences 245, 06024 (2020), doi 10.1051/epjconf/202024506024

[7] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework,Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86