Fast Operations on Compressed Arrays Without Decompression

HARVEY DAM, University of Utah
GANESH GOPALAKRISHNAN (ADVISOR), University of Utah

ACM Reference Format:
Harvey Dam and Ganesh Gopalakrishnan (advisor). 2023. Fast Operations on Compressed Arrays Without Decompression. 1, 1 (September 2023), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

In modern scientific computing and machine learning systems, data movement has overtaken compute as the performance bottleneck, thus motivating the wider adoption of lossy data compression. Unfortunately, state-of-the-art floating-point array compressors such as SZ [1] and ZFP [2] require decompression before most operations can be performed on the data. In this work, **our contribution is to show that compression methods can be designed to allow efficient operations on compressed arrays.** In particular, compression methods that consist of only linear transformations and quantization allow certain operations on compressed arrays without decompression. We develop such a compression method, called PyBlaz, the first compression method we know that can **compress arbitrary-dimensional arrays** and **directly operate on the compressed representation**, with **all stages running on GPUs**.

1.1 The PyBlaz Compression Process

The compression process in PyBlaz consists of data type conversion, blocking, orthonormal transform, binning, pruning, and flattening. Each step is detailed in the poster. Decompression consists of the compression steps in reverse. As PyBlaz is a lossy compressor, only blocking and flattening are exactly invertible because they merely rearrange elements. The other steps incur some floating-point rounding loss or additional loss due to binning or pruning. A detailed explanation of each compression step will appear in an upcoming paper.

1.2 Compression Ratio

The achieved compression ratio depends on user settings, but mostly on the index type used for binning and the pruning mask. While the floating-point type does not impact compression ratio as much as the other settings, using a lower precision can speed up subsequent operations. Without pruning, a compression ratio of approximately 4 can be achieved by using 16-bit integers for bin indices, or 8 with 8-bit integers. Pruning half the indices would again approximately double the final compression ratio.

Authors' addresses: Harvey Dam, University of Utah; Ganesh Gopalakrishnan (advisor), University of Utah.

1.3 Compressed-Space Operations

Compressed-space operations take advantage of the linearity and orthonormality of the compression steps, which preserves dot products and linear relations between elements during compression. In this way, scalar functions that use information about dot products can be composed with other functions to form more sophisticated transformations. For example, the structural similarity index measure (SSIM) is used in image processing as a proxy for the visual similarity between two images, ranging from 0 to 1 with 1 being most similar. The SSIM is calculated using a weighted product of the normalized means, variances, and covariances of two arrays. We can obtain these components using compressed-space operations (Algorithm 1). We also have a fast and approximate decompression process that avoids most decompression steps, providing a rough view of the decompressed array with the granularity of the block size on which arbitrary decompressed-space operations can be performed. Details about these algorithms and others will be described to the poster audience.

Algorithm 1: Structural Similarity Index Measure

Data: compressed arrays $A = \{\mathbf{s}, \mathbf{i}, N_1, F_1\}, B = \{\mathbf{s}, \mathbf{i}, N_2, F_2\}$, luminance stabilizer s_l, contrast stabilizer s_c, luminance weight w_l, contrast weight w_c, structure weight w_s
Result: the SSIM between A and B
$\mu_A \leftarrow \text{Mean}(A)$;
$\mu_B \leftarrow \text{Mean}(B)$;
$\sigma_A^2 \leftarrow \text{Variance}(A)$;
$\sigma_B^2 \leftarrow \text{Variance}(B)$;
$\sigma_A \leftarrow \sqrt{\sigma_A^2}$;
$\sigma_B \leftarrow \sqrt{\sigma_B^2}$;
$\sigma_{AB} \leftarrow \text{Covariance}(A, B)$;
$l \leftarrow \dfrac{2\mu_A\mu_B + s_l}{\mu_A^2 + \mu_B^2 + s_l}$;
$c \leftarrow \dfrac{2\sigma_A\sigma_B + s_c}{\sigma_A^2 + \sigma_B^2 + s_c}$;
$s \leftarrow \dfrac{\sigma_{AB} + \frac{s_c}{2}}{\sigma_A\sigma_B + \frac{s_c}{2}}$;
return $l^{w_l} c^{w_c} s^{w_s}$;

1.4 Performance Assessment of PyBlaz Compression, Decompression, and Compressed-Space Operations

We developed PyBlaz with the GPU as the *primary* computing device in mind. Thus, every step of compression, decompression, and compressed-space operations takes at most logarithmic time (due to *max* or *sum* operations) with sufficient threads. We have found PyBlaz to be faster than ZFP with CUDA [2], as shown in the poster. Also, our preliminary experiments show that PyBlaz is competitive with SZ3, the latest version of SZ [1]. However, we reiterate that only PyBlaz supports operations directly on compressed arrays.

1.5 Applications

We emphasize that PyBlaz is developed with goals entirely different from that of other floating-point array compressors. We intend for PyBlaz to be used where keeping arrays compressed while still operating on them is a priority, not in situations that demand high compression ratio and low error bounds. We show the following application in the poster.

Suppose a scientific simulation has multiple possible implementations, with some being more expensive and more precise than others. One may wish to test several cheaper implementations to search for an acceptable trade-off of speed and precision. Compressed forms of these simulation outputs can be stored to avoid data movement, both between compute and storage nodes, and between compute devices and the host. The host can then decide whether some simulation has diverged too much from the reference using a compressed-space scalar function such as the L_2 norm, avoiding decompression, and avoiding using additional memory.

1.6 Limitations and Future Work

As error characteristics are usually expected in discussions of compressors, we intend to provide an analysis of error introduced by the binning and pruning steps in terms of the relevant compression settings, i.e. the number of bins, pruning mask, and orthonormal transform function.

We will also investigate how PyBlaz performs in its intended applications, such as in artificial neural network inputs and parameters. In a space where the state-of-the-art is trending toward lower precision and more compact representations, we will study how our compressed-space operations can contribute to more efficient implementations of backpropagation (e.g. through addition and scalar multiplication) and regularization components in loss functions (e.g. through L_2 norm).

REFERENCES

[1] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2018. An Efficient Transformation Scheme for Lossy Data Compression with Point-Wise Relative Error Bound. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 179–189. https://doi.org/10.1109/CLUSTER.2018.00036

[2] Peter Lindstrom. 2014. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics* 20 (08 2014). https://doi.org/10.1109/TVCG.2014.2346458