Introduction

Seismic data compression is neither new (Wood, 1974) nor neglected in current research (Zhang, 2017). However, the need for it is increasing and dedicated open-source tools for performing it have not been available until now. We discuss the motivations and methods behind Equinor's new open-source Python seismic compression library: seismic-zfp (https://github.com/equinor/seismic-zfp)

Seismic data has been a fundamental component of exploration geophysics for decades. Contemporary technologies have been used for the recording, storage, transmission and processing of these data throughout this time. Today the technology landscape sees a major shift towards cloud computing, driving cost efficiency through centralization of compute and storage hardware by cloud vendors who then rent it out by the second or by the byte. At the same time analytics methods continue increasing their compute intensity, with Machine Learning – in particular Deep Neural Networks (DNNs) – leading the charge.

There has always been a need to balance the costs of high data volume with the expected value which may be extracted from such data, both in terms of storage costs and processing costs. Cloud computing comes with many built-in advantages, including elastic scalability, protection against data-loss and professionally managed security. However, cloud vendors charge users by the Gigabyte-month of storage, or the Terabyte of data egress. This means we cannot take full advantage of the cost savings available through cloud computing if we continue to use uncompressed 32-bit floating point representation for seismic data everywhere it is used.

Although the cost of cloud storage can be large, it is often outweighed by the cost of cloud processing. A single V100 GPU with 14Tflops of compute power may be up to $10000/month for continuous usage training a DNN. Therefore, quite aside from minimizing the cost of data storage, it is vitally important that seismic data can be accessed as fast as it can be handled by this colossal compute power. Waiting on several terabytes to be shifted around the cloud is an infeasibly expensive option. Therefore, maximizing available bandwidth within the cloud is a further motivation to compress seismic data.

There is one additional aspect of Machine Learning which will motivate our design choices in building a system for compressing seismic data. This is the need for training our algorithms on disparate, bordering on random, subsets of the data. This arbitrary sub-volume access pattern necessitates the ability to perform decompression on very granular portions of the compressed volume.

Method

Seismic-ZFP is based on the zfp floating point compression algorithm (Lindstrom, 2014). This algorithm is implemented in a BSD-licensed open-source C/C++ library for lossy compression/decompression of regular arrays of 32 or 64-bit floating point numbers with up to 4 dimensions. It compresses 4^n sub-volumes of the input array into a single bit-string, which may be truncated to yield either fixed-rate, fixed-precision or fixed-accuracy in the compressed representation.

For the purposes of seismic data, we shall focus on the fixed-rate compression, where we may specify how many bits of storage are used to represent each 4^n sub-volume. For example, in the case of 3D seismic images originally in 32-bit representation our initial quantity of data for each sub-volume will be 4x4x4 x 32-bits = 256 bytes. Given a bit rate of 2-bits per voxel this would then use 16 bytes when compressed, i.e. a 16:1 compression ratio.

The possibility for regularity is one of the key aspects of seismic data which we will take advantage of. It may always be represented as a regular n-dimensional array, with zeroed out traces filling areas in space, offset or azimuth with no signal. While this can be wasteful of storage space for certain survey geometries while uncompressed, the tradeoff will be paid back post-compression.

Regularity and fixed bitrate guarantee us two things:

- Neighboring data along any dimension of the array will also be neighboring it in the compressed representation. This enables the compression algorithm to take advantage of correlation between neighboring samples.
- Any piece of data can be found in the compressed representation from knowledge of its location in the n-dimensional space we are considering.

Once a seismic volume is regularized and compressed into 4^n sub-volumes there is one more choice to be made before storing to disk, namely the order in which the sub-volumes are stored. Modern computer storage hardware, both HDD and SSD, have standardized on 4KB disk blocks. This means 4096-bytes is the minimum amount of data which can be written or read in a single transaction, and this is several times the 16-bytes we want to use to represent each 4^n sub-volume. Therefore, we should choose an order which packs sub-volumes that will commonly be accessed together into the same disk block. Given conflicting requirements of common access patterns for seismic data it is pragmatic to enable the disk layout of the compressed volume to be flexible. With the good compression ratios on offer, it may even be feasible to store separate copies for different access patterns. Optimal layouts for trace/inline/crossline and z-slice read patterns are presented in Figures 1 & 2 respectively.

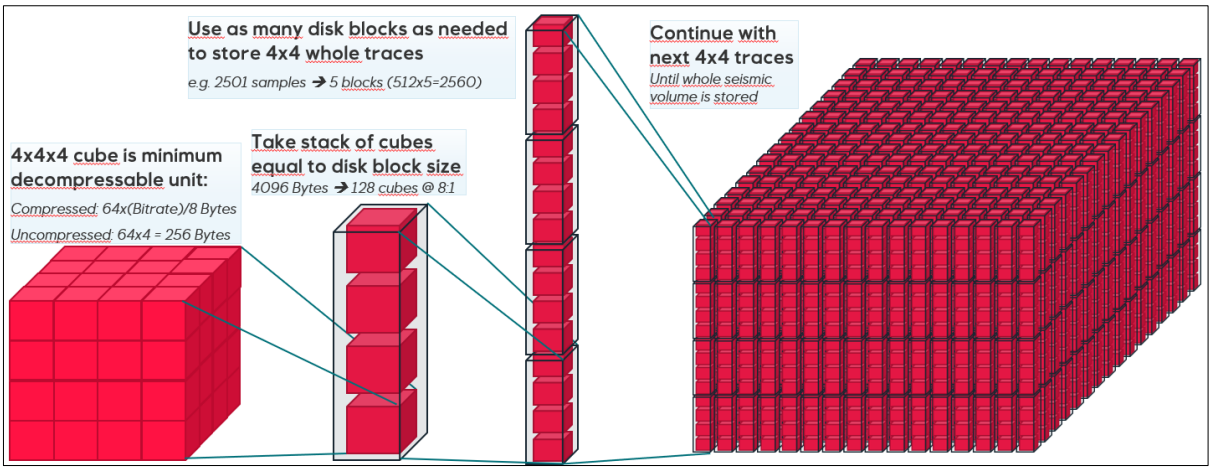

Figure 1. *Packing 4x4x4 compressed sub-volumes of 3D seismic data into disk blocks and arranging on disk to enable optimal trace/inline/crossline reading.*

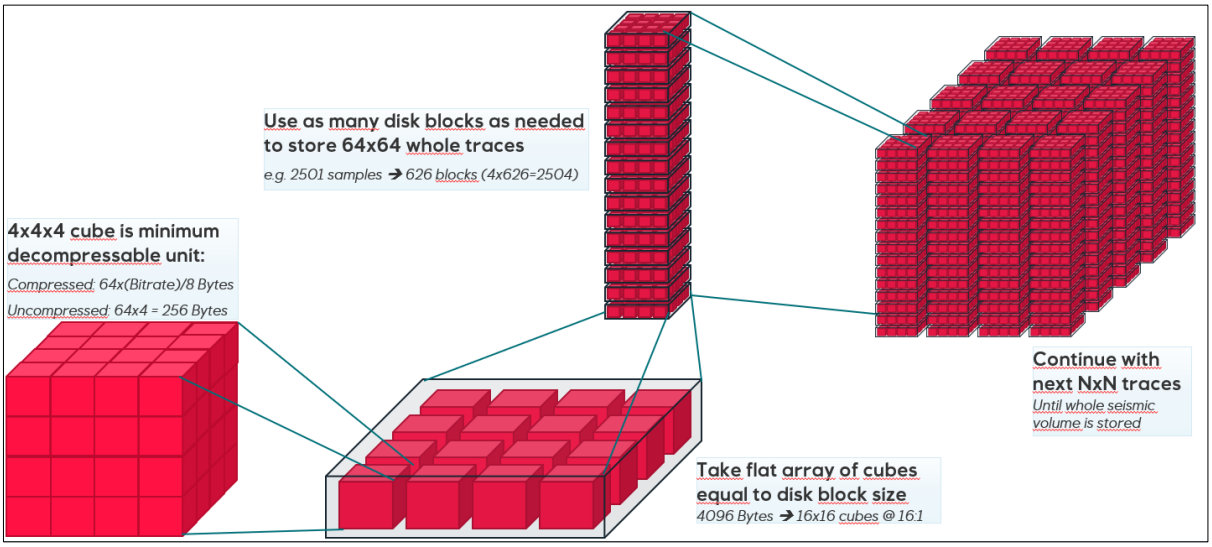

Figure 2. *Packing 4x4x4 compressed sub-volumes of 3D seismic data into disk blocks and arranging on disk to enable optimal z-slice reading.*

Once compressed, a seismic data volume may be read and decompressed using the Python API of the seismic-zfp library. This may be either as a whole, or in the commonly used sub-volumes of inlines/crosslines/z-slices, or indeed in arbitrary sub-volumes denoted by a range in each of inline/crossline/sample-number. Unless these ranges begin and end on exact multiples of 4 there will inevitably be overhead in the number of voxels decompressed. However, this inefficiency is more than mitigated by two factors:

- Far less data is being read from disk, as high compression ratios are suitable for certain applications.
- Superfluous decompressed data is cached, given a high likelihood of reuse.
  (e.g. reading inlines #2, #3 & #4 after reading inline #1)

Seismic data is also commonly accompanied by a certain quantity of meta-data, stored in trace and file headers in SEG-Y format. The seismic-zfp library also preserves both file and trace header values when a compressed seismic file is created from SEG-Y, and is placed en masse at the end of the file as floating-point numpy arrays. This enables efficient reading and avoids the misalignment of data which would result from interleaving header information.

Examples

In order to be confident in any results obtained while using seismic-zfp, a thorough evaluation of the suitability of this lossy compression scheme should be undertaken. One way of performing this evaluation is to investigate the frequency spectra of the same underlying data represented by seismic-zfp or otherwise (Figure 3).

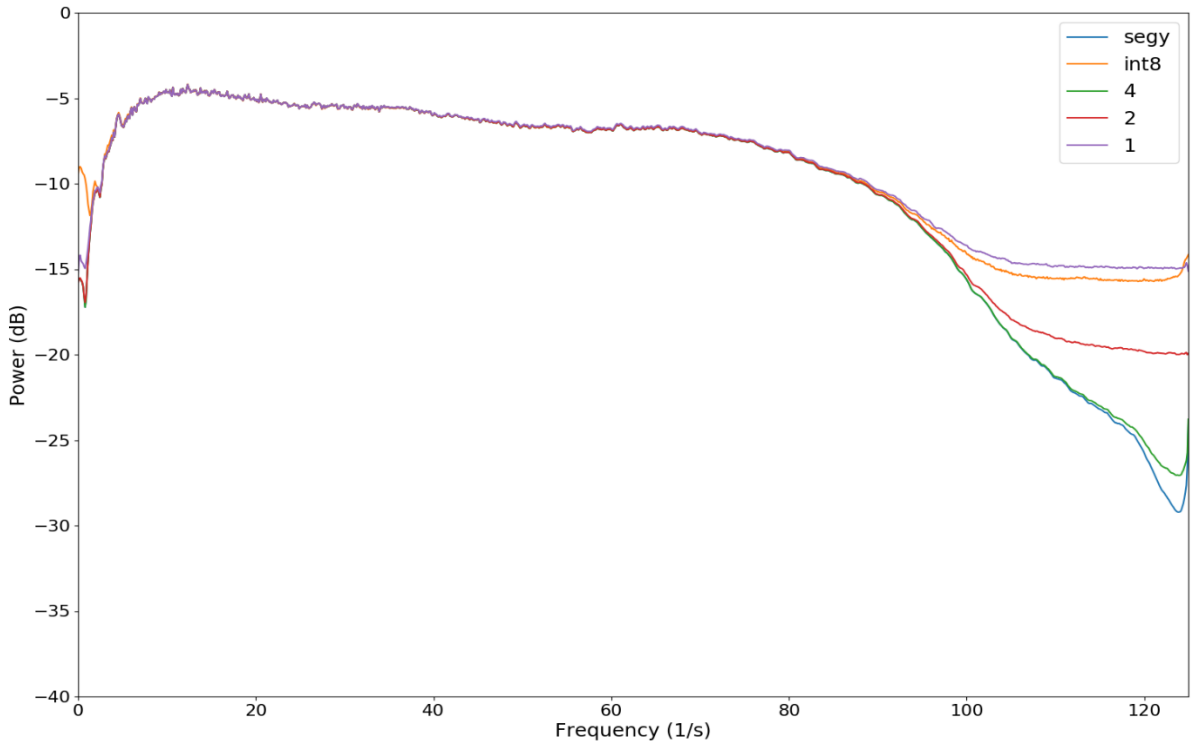

Figure 3: *The frequency spectrum of one 4ms-sampled inline from a modern marine seismic survey when stored as:*

(a - blue)	*32-bit floating point SEG-Y*
(b - orange)	*Scaled 8-bit integers*
(c - green)	*4-bit-per-voxel seismic-zfp*
(d - red)	*2-bit-per-voxel seismic-zfp*
(e - purple)	*1-bit-per-voxel seismic-zfp*

It is clear from Figure 3 that not all frequencies are perfectly reproduced by the compression algorithm (or the baseline we have chosen of 8-bit integer representation) and this is to be expected given its lossy nature. Much of the imperfection in recovery is in added noise in the highest frequencies. However, a key finding is that we can represent a 32-bit seismic volume almost entirely up to its Nyquist frequency using only 4 bits per voxel, a compression ratio of 8:1. Furthermore, if we are uninterested in the higher frequencies and are willing to apply a low-pass filter in the usage of our data, there is a distinct possibility to use 2-bit or 1-bit zfp representations. As high frequency energy is largely absorbed by even short distances of rock this is reasonable for everything but the shallowest of imaging targets and slashes our data storage and transfer costs by a factor of 16x or even 32x.

Conclusions

We have presented an open source seismic data compression library, built on top of a state-of-the-art floating-point compression algorithm. Fast arbitrary reading is achieved by using two key observations, namely that regularity may be preserved by using fixed-rate compression, and that storage hardware may be efficiently utilized by packing disk blocks with data which is frequently accessed together. Furthermore, we have demonstrated the quality of reproduction of the input data and claim that it is not only sufficient for Machine Learning, but also necessary for its efficient implementation.

Acknowledgements

We would like to acknowledge the contributors to the open source projects which seismic-zfp builds upon which do not have any publications which could be directly cited here, but are nonetheless vital to the functioning of the library:

- segyio: maintained by Jørgen Kvalsvik of Equinor ASA
- pyzfp: maintained by Navjot Kukreja of Imperial College, London

We are also grateful to Equinor ASA for enabling the open-source licensing of this work, in particular the Chief Engineer of IT, Knut Erik Hollund.

References

Lindstrom, Peter. (2014). *Fixed-Rate Compressed Floating-Point Arrays.* IEEE Transactions on Visualization and Computer Graphics. 20. 10.1109/TVCG.2014.2346458.

Wood, Lawrence C. (1974). *Seismic Data Compression Methods.* Geophysics Vol. 39, No 4, P. 499-525

Zhang, Yiming et al. (2017). *Massive 3D seismic data compression and inversion with hierarchical Tucker*, SEG Technical Program Expanded Abstracts.