Lossy compression algorithms for floating-point data

JSM 2017
Numerical data is challenging to compress losslessly.
Large improvements in compression possible by allowing even small errors
- Least significant floating-point bits are effectively random noise
- Most compressors support relative or absolute error tolerances

Compressors must be cognizant of how compression errors propagate in data analysis
- **Biased error** (ideally zero mean)
- **Correlation** of error with function (ideally independent)
- **Autocorrelation** of error (ideally uncorrelated)
- **Spectral properties** of error (ideally white noise)
- **Distribution** of error (e.g. uniform, normal, Laplace, ...)
- Impact on **statistical quantities** like extrema, mean/median, moments, ...
- Impact on **differential quantities** like spatial & temporal derivatives

This talk will examine error distributions for several compressors
Numerical data compression usually involves three steps

1. **Decorrelate** data to make it more compressible
   - E.g. prediction, fitting, transformation, decomposition, ...
   - Make the data sparse in some alternative representation
   - Small values, repeated patterns are easier to compress

2. **Approximate** (for lossy compression)
   - E.g. scalar/vector quantization, truncation, thresholding, ...
   - Discard unimportant information to avoid encoding it

3. **Encode** remaining information losslessly
   - E.g. Huffman, arithmetic, universal, run-length, dictionary, ...
Case study: 8 lossy floating-point compressors in 8 minutes!

1. **SQ**: adaptive scalar quantization [Iverson et al. 2013]
2. **HVQ**: hierarchical vector quantization [Schneider & Westermann 2003]
3. **SZ**: error-bounded polynomial prediction [Di & Cappello 2016]
4. **fpzip**: lossless/lossy predictive coding [Lindstrom & Isenburg 2006]
5. **zfp**: block transform with embedded coding [Lindstrom 2014]
6. **VAPOR**: wavelet transform & thresholding [Clyne et al. 2007]
7. **Tucker**: tensor decomposition & thresholding [Ballester & Pajarola 2016]
8. **ISABELA**: sorting and spline fitting [Lakshminarasimhan et al. 2013]

**Challenge**: Compress 3D scalar field, f(x, y, z), defined on uniform Cartesian grid
SQ: Adaptive, error-bounded scalar quantization

- \([SQ]\) algorithm partitions data into \(\varepsilon\)-sized ranges
  - Sort data on function value
  - Greedily grow set \(S_i\) as long as \(\max S_i - \min S_i \leq \varepsilon\)
  - Use as prototype \(p_i = \text{mean } S_i\)
    - Minimizes RMS error
  - Replace values assigned to set \(S_i\) with index \(i\)
  - LZMA compress codebook \(\{p_i\}\) and indices \(\{i\}\)
SQ error distribution is nearly uniform but overly conservative

\[\begin{align*}
+2 \leq f < +4 \\
+1 \leq f < +2 \\
+0.5 \leq f < +1 \\
-1 < f \leq -0.5 \\
-2 < f \leq -1 \\
-4 < f \leq -2
\end{align*}\]
HVQ: Hierarchical Vector Quantization

- Similar to scalar quantization, but applied to multi-component vectors
  - E.g. vector/tensor fields, multiple correlated fields, blocks of values, ...
  - Can be done non-uniformly in both domain and range

- Hierarchical VQ [HVQ] uses different codebook on each level
  - Vectors formed by $4 \times 4 \times 4$ blocks of values
    - Next level given by block averages
  - Codebook is generated using Lloyd relaxation
    - Randomly select initial prototypes
    - Partition data by closest prototype
    - Replace prototype with mean/medoid/Voronoi centroid

- Most effective for low-precision data like 8-bit RGB
  - Codebook size, compute time become prohibitive for higher precision
VQ errors are difficult to bound due to difficulty of creating good codebook.
**SZ: Polynomial prediction extrapolates from past data points**

- Polynomial of degree $n - 1$ predicts next value from last $n$ transmitted values
  - Use best of three predictors: constant, linear, quadratic
  - “Mispredictions” outside of tolerance $\pm \varepsilon$ are corrected

\[ f(x) \]
SZ error distribution is approximately uniform and spans full tolerance

\[ -\varepsilon \leq f \leq -\varepsilon/2 \]
\[ +\varepsilon/2 \leq f \leq +\varepsilon \]

\[ +2 \leq f < +4 \]
\[ +1 \leq f < +2 \]
\[ +0.5 \leq f < +1 \]
\[ -1 < f \leq -0.5 \]
\[ -2 < f \leq -1 \]
\[ -4 < f \leq -2 \]
fpzip: Lossless mode combines multi-dimensional prediction with entropy coding

- Input stream
- Linearization
  - Reinterpret floats as sign-magnitude integers
- Prediction
  - Predict next data element from already coded ones
- Linearization
  - Compute integer residual
- Linearization
  - Update stats on sign/width distribution
  - Alphabet partitioning
  - Split residual into sign/width, value
  - Assign short code-words to frequent sign/width pairs
- Probability modeling
- Entropy coding
  - Recombine entropy codes, value bits
- Output stream

\[ r = +10010110 \]
fpzip: Lossy mode truncates (zeros) least significant bits, then compresses losslessly.
fpzip error distribution is dependent on function value $f$ and is highly biased

$$-\varepsilon < f < -\varepsilon/2 \leq f < 0 \leq f < +\varepsilon/2 \leq f < +2 \leq f < +4$$
fpzip systematic rounding toward zero leads to occasional issues in climate data analysis

#31, #33 compressed ensemble members
ZFP: Compressed floating-point arrays that support random access and error tolerances

- Align values in a 4\textsuperscript{th} block to a common largest exponent
- Transmit exponent verbatim

- Encode one bit plane at a time from MSB using group testing
- Each bit increases quality—can truncate stream anywhere

- Lifted, separable transform using integer adds and shifts
- Similar to but faster and more effective than JPEG DCT
ZFP decorrelates $d$-dimensional block of $4^d$ values using an orthogonal transform

\[
\begin{pmatrix}
\hat{f}_1 \\
\hat{f}_2 \\
\hat{f}_3 \\
\hat{f}_4
\end{pmatrix}
= \frac{1}{2}
\begin{pmatrix}
1 & 1 & 1 & 1 \\
c & s & -s & -c \\
1 & -1 & -1 & 1 \\
s & -c & c & -s
\end{pmatrix}
\begin{pmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4
\end{pmatrix}
\]

- **Basis functions**
  - For $t = 1/4$
  - Increasing frequency (polynomial degree)

\[
s = \sqrt{2} \sin \frac{\pi}{2} t \quad c = \sqrt{2} \cos \frac{\pi}{2} t
\]
ZFP’s integer transform is efficient, effective, and well-suited for h/w implementation.

```
x += w; x >>= 1; w -= x;
z += y; z >>= 1; y -= z;
x += z; x >>= 1; z -= x;
w += y; w >>= 1; y -= w;
w += y >> 1; y -= w >> 1;
```
ZFP error distribution is normal due to linear transform of iid. errors (central limit theorem)
VAPOR: Discrete wavelet transform with coefficient thresholding

- Basis functions are given by translations and dilations of single mother wavelet
VAPOR wavelet errors are difficult to bound due to cascading effects.
Tucker: Generalization of SVD using Tucker tensor decomposition, core tensor truncation

- 2D structured grid data can be approximated via truncated SVD

\[
\text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) = \Sigma = U^T AV = \text{unvec}((V \otimes U)^T \text{vec}(A))
\]

- Singular value matrix, \( \Sigma \), is **diagonal** but singular vectors, \( U \) and \( V \), are **data-dependent**
  - \( U \) and \( V \) are expensive to encode for 2D data
  - \( A \) can be optimally approximated in the \( L_2 \) sense by discarding smallest singular values

- \( d \)-dimensional structured grid data can be approximated via tensor decomposition

\[
S = A \times_1 U \times_2 V \times_3 W = \text{unvec}((W \otimes V \otimes U)^T \text{vec}(A))
\]

- Unlike in SVD, core tensor, \( S \), is not diagonal, but large values appear in “hot corner”
  - \( U, V, \) and \( W \) matrices are relatively cheap to encode for 3D data
As in SVD, truncated core tensor & factor matrices yield “best” low-rank approximation

\[ S = \mathcal{A} \times_1 U_1 \times_2 U_2 \times_3 U_3 \]
Like wavelets, Tucker tensor decomposition errors are difficult to bound tightly
ISABELA: Sorting and spline fitting enables compression of even the noisiest data sets

- Most compression techniques fail miserably on noisy/unstructured data
- *ISABELA*: Sort noisy data, encode permutation, fit smooth sorted signal
Like fpzip, ISABELA bounds relative errors, but without bias.
ZFP and SZ decorrelate error with function

correlation of function with error

rate (bits/value)
Some compressors yield autocorrelated errors

- **ZFP**: 0.34 bits/value, $\|R\| = 2.8 \times 10^{-4}$
- **HVQ**: 5.00 bits/value, $\|R\| = 3.1 \times 10^{-4}$
- **Tucker**: 0.53 bits/value, $\|R\| = 4.6 \times 10^{-4}$
- **VAPOR**: 2.94 bits/value, $\|R\| = 5.3 \times 10^{-4}$
- **SZ**: 0.33 bits/value, $\|R\| = 4.6 \times 10^{-3}$
- **SQ**: 0.42 bits/value, $\|R\| = 4.7 \times 10^{-3}$
- **fpzip**: 9.61 bits/value, $\|R\| = 2.2 \times 10^{-2}$
- **LZ4A**: 0.79 bits/value, $\|R\| = 1.4 \times 10^{-1}$

Input data autocorrelation
Compressors other than ZFP show artifacts in derivative computations (velocity divergence)
Compressors other than ZFP show artifacts in derivative computations (velocity divergence)

- **HVQ**: 2.33 bits/value
  - Compression Ratio: 27:1
  - LLB: 23.3 dB
  - Compression Speed: 1 MB/s

- **VAPOR**: 2.18 bits/value
  - Compression Ratio: 29:1
  - LLB: 41.6 dB
  - Compression Speed: 40 MB/s

- **ZFP**: 1.96 bits/value
  - Compression Ratio: 33:1
  - LLB: 66.8 dB
  - Compression Speed: 514 MB/s

- **Uncompressed**: 64 bits/value
Compressors other than ZFP show artifacts in derivative computations (velocity divergence)
Conclusions

- Lossy data compression can be viable in scientific computing workflows
  - ~100x compression acceptable for visualization
  - ~10x compression acceptable for quantitative data analysis
  - ~4x compression of simulation state with <0.1% error in final quantity of interest

- Little effort has focused on metrics for evaluating compression errors
  - Error distributions can vary greatly between compressors but are rarely considered
    • Difficult to prescribe desired shape of error distribution
  - Z-checker tool, developed by Cappello and others at Argonne, is a good first step

- HPC community needs to provide analysis code with simulation results
  - How else can we quantify impact of lossy compression?
  - Need collection of “standard” data sets for evaluating & comparing compressors

- What statistical metrics and properties should we be concerned with?
References


[ISABELA] Lakshminarasimhan et al., “ISABELA for effective in situ compression of scientific data,” 2013


[SQ] Iverson et al., “Fast and effective lossy compression algorithms for scientific datasets,” 2012


