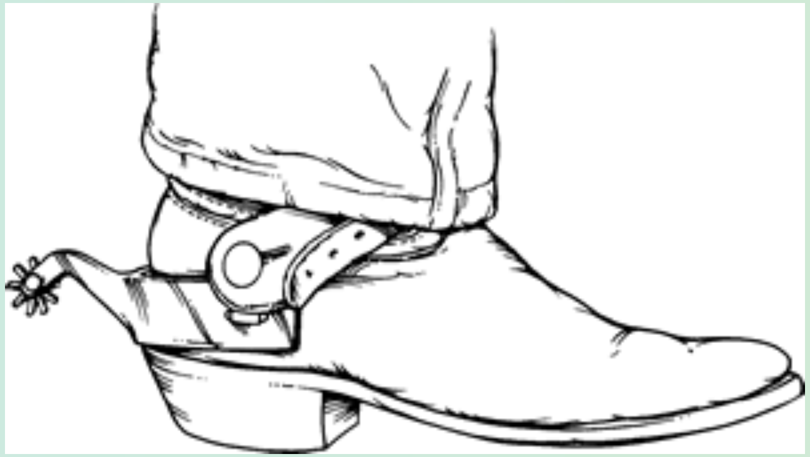
Scientific Data Compression with SPERR

(github.com/NCAR/SPERR)

Presenter: Samuel Li

What makes SPERR unique?



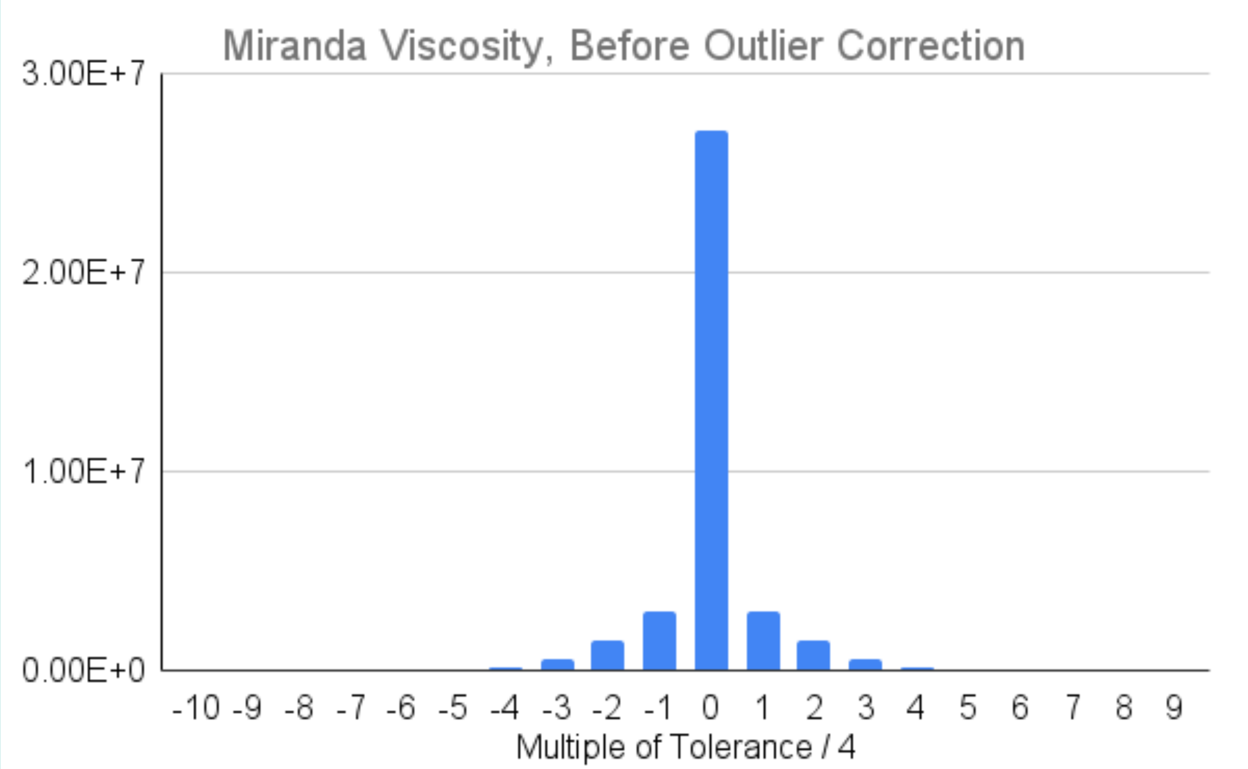
- It’s got a weird name
  - Pronounced like *spur*
- Based on wavelet transforms
  - Excellent on decorrelation
  - Fixed-rate compression
- Natural support for “flexible-rate decoding”
  - A prefix (subset from the beginning) of the compressed bitstream is still valid for decompression, though less accurate.

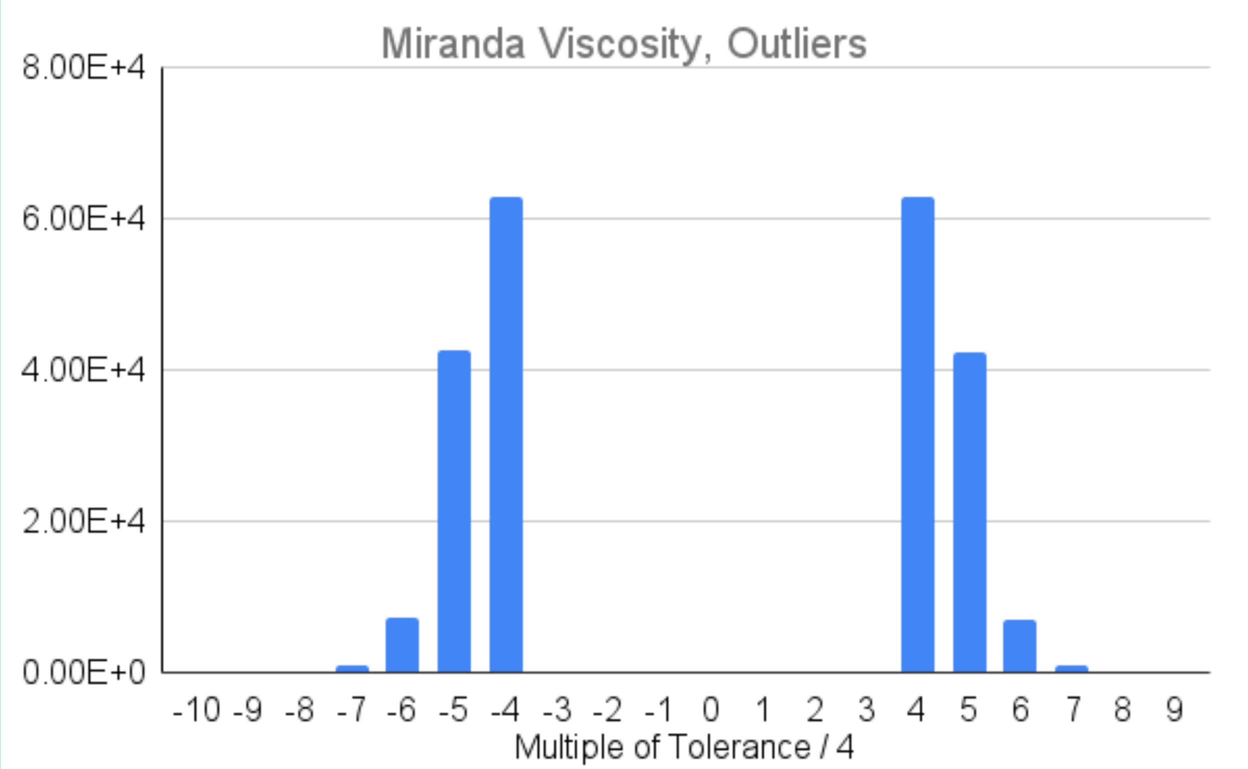
Design Consideration / Motivation

- Wavelets naturally supports *fixed-size compression*.
- *Error-bounded compression* is a must have to be useful on scientific data compression.
    - Maximum point-wise error (PWE) to be specific.
- Observation: error distribution is approximately a bell curve
    - Very few data points have large errors.
    - Viable to explicit encode data points violating a PWE tolerance: *outliers*.

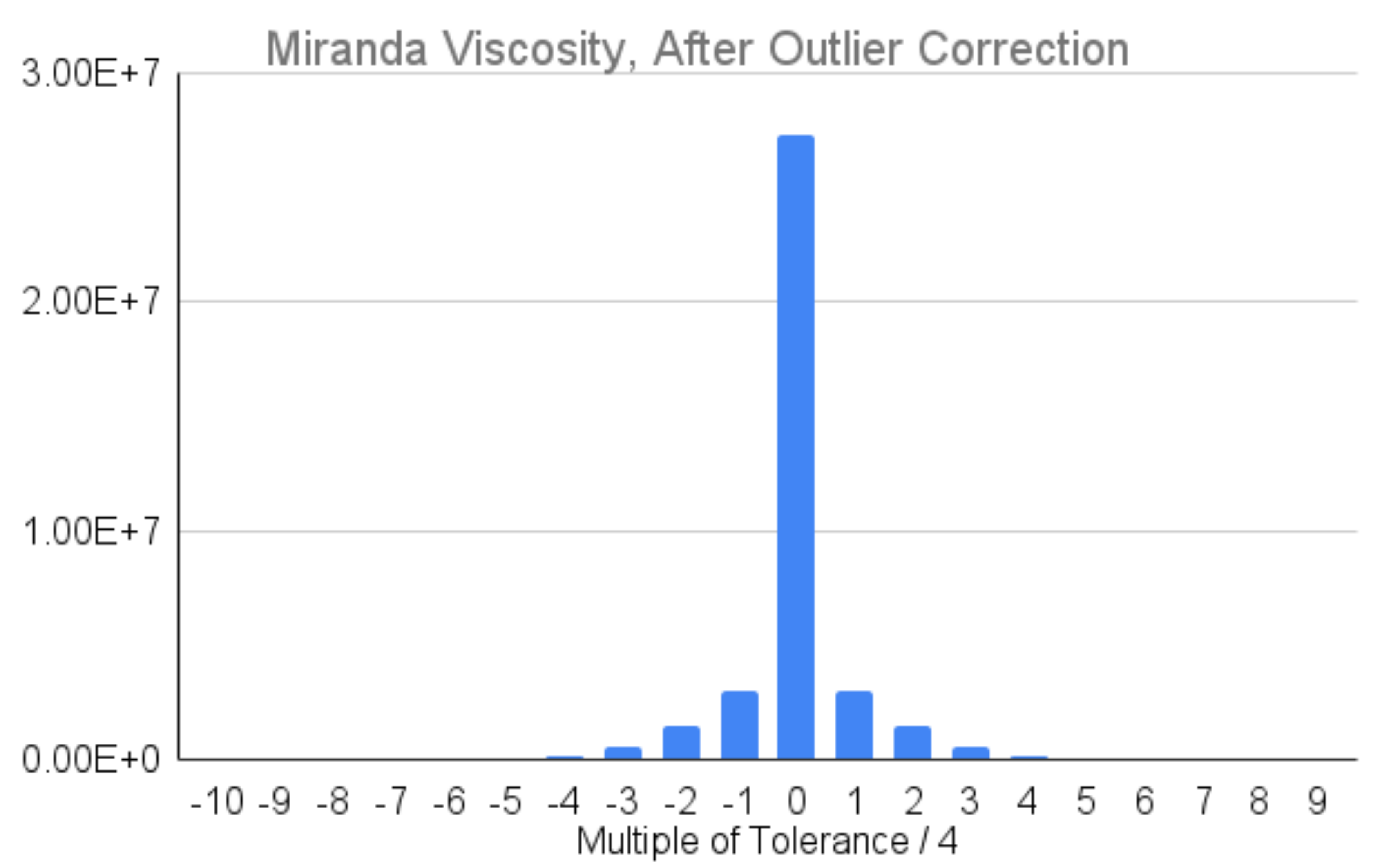
Error Distribution and Outlier Correction

- Example: Miranda Viscosity field: ~37M data points. Tolerance = 1×10^{-8}





Error Distribution and Outlier Correction



Compression Pipeline

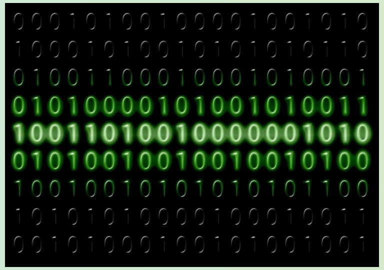
- Two-step process: wavelet compression + outlier correction

(Step 1)
Data Volume

Wavelet XForm

Wavelet Coeff.

SPECK Coding

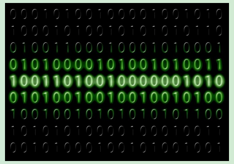


(Step 2)
Quantized
Coefficients

Inverse Wavelet

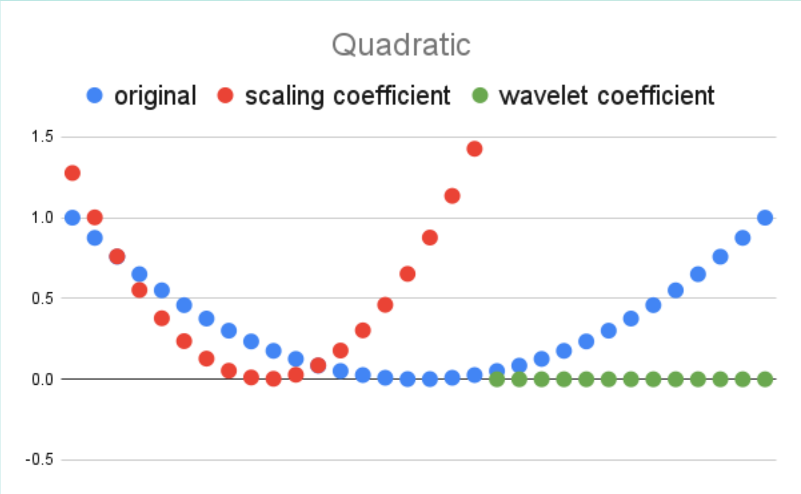
Outliers

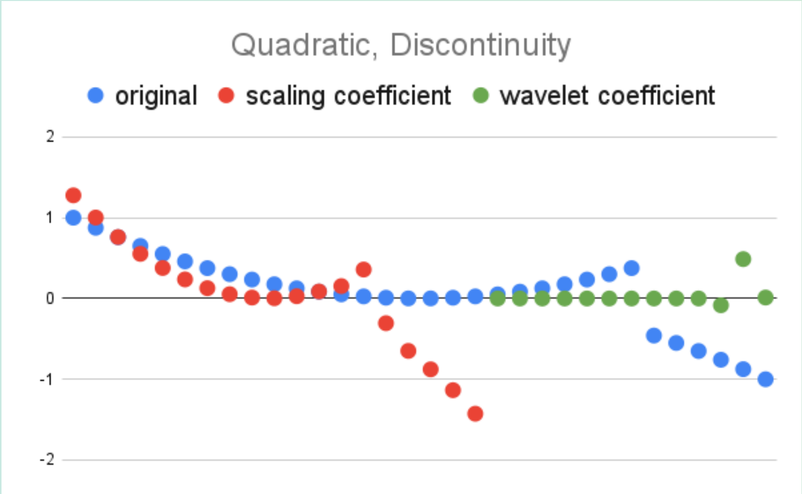
Outlier Coding



Decorrelation—Wavelet Transforms

- A specific type of wavelets, CDF 9/7, is very good at decorrelation, thus favored in compression applications.
- Wavelets take the input and produce the same number of coefficients: **reversible**!
- There are two flavors: *scaling coeff*. and *wavelet coeff*., serving different purposes:

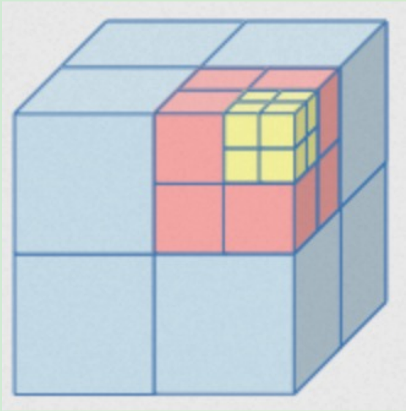




The better decorrelation, the smaller wavelet coefficients!

Coefficient Coding

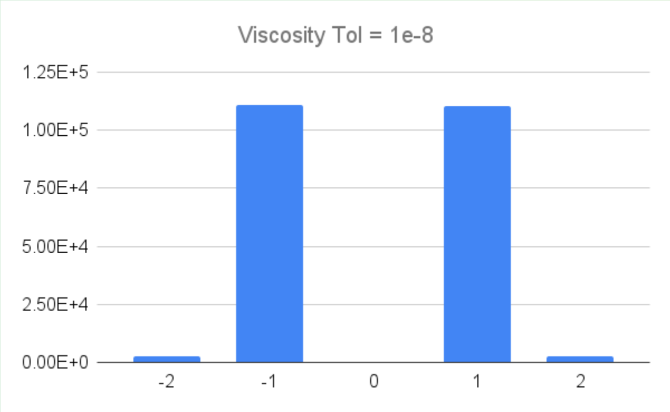
- The SPECK [Pearlman et al. 2004] algorithm encodes coefficients from most to least significant bits.
  - Step 1: locate “significant coefficients” w.r.t. the current bitplane.
  - Step 2: quantize all located “significant coefficients” w.r.t. the current bitplane.
  - Iterate these two steps with the next bitplane, which is less significant.



- Step 1:
  - Divide the volume (octree style), perform significance test on each subtree, repeat until finding individual significant points. **Significance test results are saved.**
  - <u>The more spatially clustered the significant coefficients, the more they share the cost of saving significance test results, i.e., higher coding efficiency.</u>
- Step 2:
  - Significant coefficients are quantized to a fixed precision q. **It produces 1 bit per coefficient**.
  - <u>Insignificant coefficients are treated as zero’s. The more zero’s, the higher coding efficiency.</u>

Outlier Coding

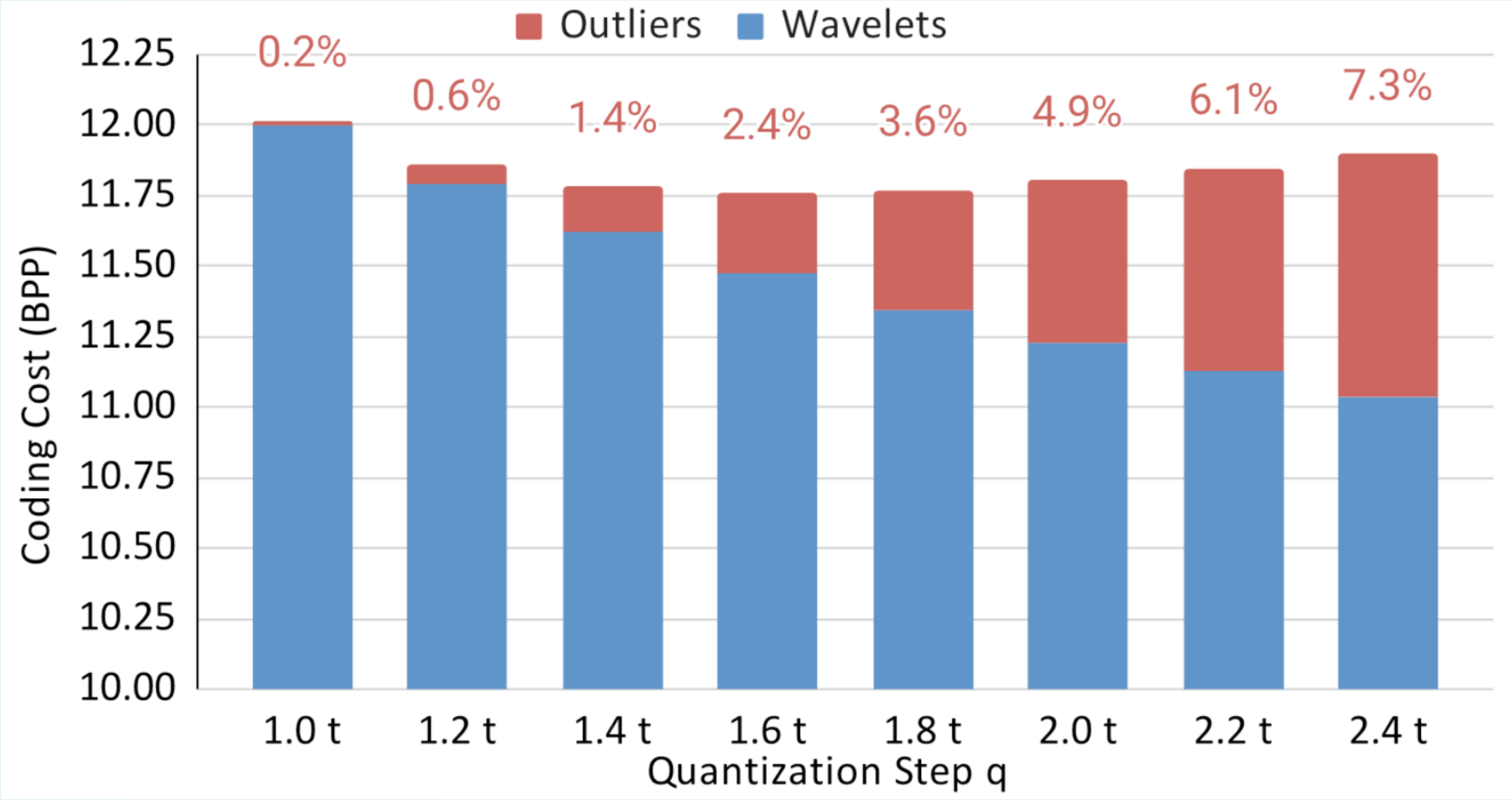
- Find the outliers: we perform *inverse wavelet transform* using the quantized coefficients, and compare the reconstruction to the original input.
  - Outliers: PWE > tolerance.
- Encode these outliers, using a modified SPECK algorithm.
  - Outliers are synonym of "significant coefficients"
  - Outliers are quantized as integer multiples of the tolerance: *correctors*.
  - Inliers are zeros.
- Observation: errors are mostly within 1 or 2 units of the tolerance.



Balance: Coefficient and Outlier Coding

- Total storage = Coefficient Storage + Outlier Storage.
  - Too much coefficient coding: reduce average error *unnecessarily* low.
  - Too much outlier coding: miss out the high efficiency of coefficient coding.
  - Goal: find a balance where the total storage is minimal.
- This balance is governed by **quantization step *q*** in coefficient coding:
  - Smaller ***q*** ⬄ more coefficient coding; bigger ***q*** ⬄ more outlier coding
  - ***q*** is on the same magnitude with the tolerance.

Balance: Coefficient and Outlier Coding



Empirical Formula: **q = 1.5 tol**

Flexible-rate Decoding

- Desirable property: the prefix is also the most efficient compressed form at that bitrate (in terms of average error). I.e., there’s **no storage overhead to** take advantage of flexible-rate decoding!
- Caveats:
  - No built-in error guarantee when using a portion of the bitstream.
  - When domain decomposition is used, flexible-rate decoding needs to be applied on each subdomain independently.
  - No incremental updates. I.e., a complete decompression operation is needed to incorporate incoming bits.

Characteristics, Performance

- SPERR is effective in a wide range of compression qualities:
  - Low quality, high compression ratio: visualization
  - High quality, low compression ratio: saving double-precision output at similar qualities as single-precision (or higher).
- Parallelization:
  - On CPUs: domain decomposition (256^3 by default). Each subdomain is processed independently.
  - On GPUs: noticeably, there isn't a GPU implementation yet.
- Given a prescribed PWE tolerance, SPERR likely produces the highest compression ratio.
- Given a prescribed PWE tolerance, SPERR takes a longer time (than ZFP and SZ) to compress.

Integrations/Applications

- I/O plugins: HDF5 and ZARR (work in progress)
- Applications: MURaM solar simulation
- Future: cloud-based data portals:
  - egress costs are high (9 cents per GB).
  - transmission may be slow.
- Future: tiered storage:
  - a fraction (of the compressed bitstream) on hot storage and the bulk on cold storage.