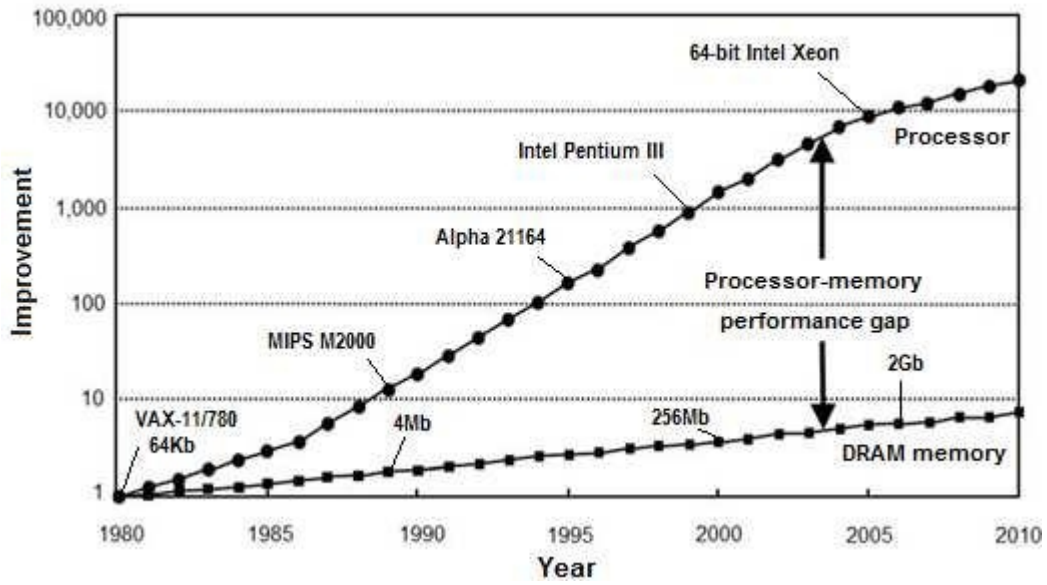
Data Compression and Scientific Applications

Fabian Knorr

University of Innsbruck, Austria

Why is Data Compression relevant today?

A widening gap between compute performance and memory, I/O bandwidth allows investing more resources into data compression



Development of processor speeds (SPECint benchmark execution times) compared to memory speeds (RAS-CAS access latency) over 30 years (Efnusheva et al., 2017)

What makes Data Compression Interesting in Theory?

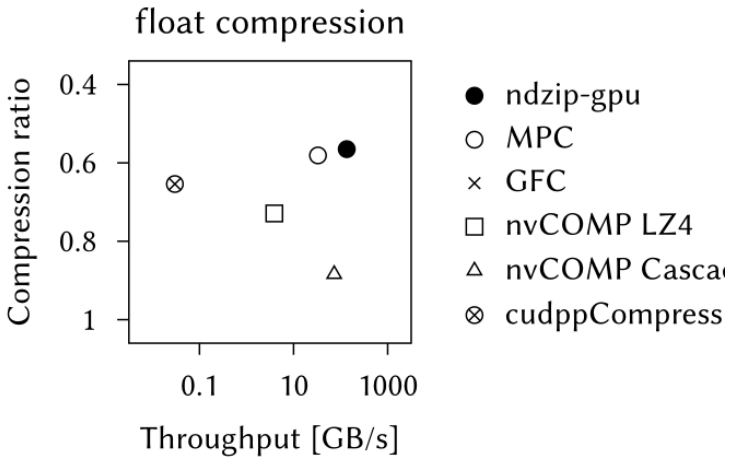
Data compression is related to AGI research: Compressing data well requires *understanding* its contents

500'000€ Prize for Compressing Human Knowledge
(widely known as the Hutter Prize)

Compress the 1GB file enwik9 to less than the current record of about 115MB

Hutter Prize for advances in Data Compression,
http://prize.hutter1.net

Well-defined, progress is easily quantified in each domain



Throughput vs. data reduction tradeoff for lossless floating-point compressors (Knorr et al., 2021)

Theory: Modelling

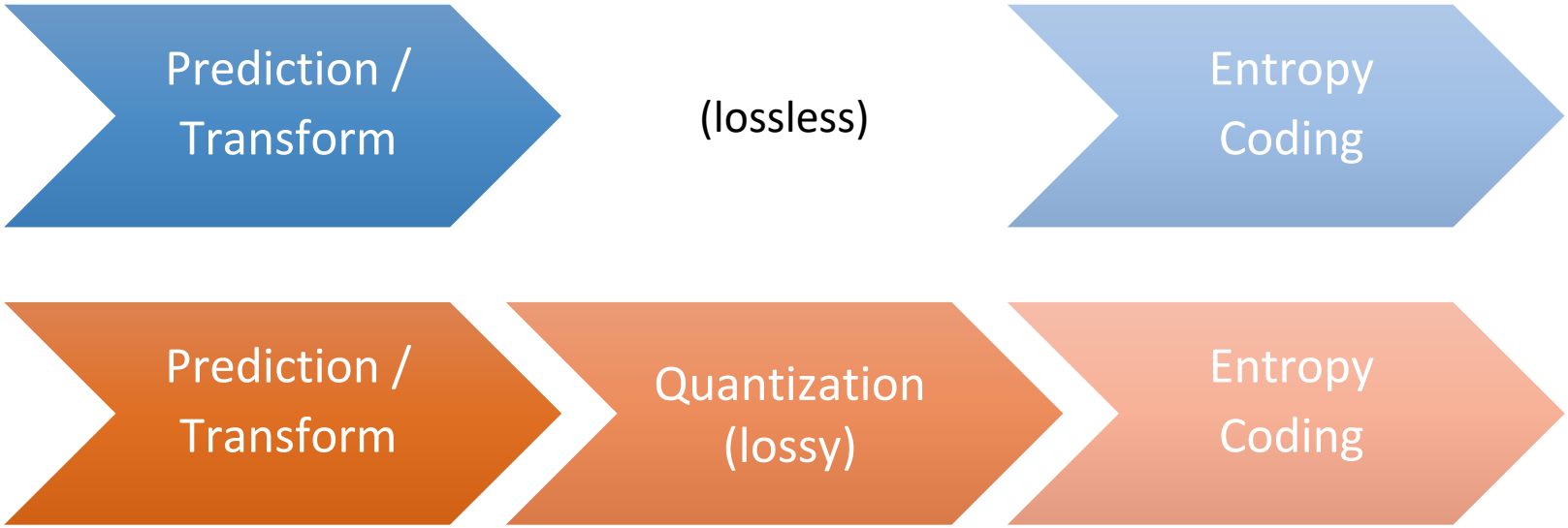
Given any message M and language L, what is the shortest program in L that reproduces M?

Provably not computable!

But with knowledge of the data domain, we can approximate:

- General Purpose: Find recurring byte strings (RLE, LZ77)
- Images: Expect low-energy high-frequency components
- Audio: Expect linear predictability

Basics: Building Blocks of a Compressor



Decorrelation:
Remove reduncancy between the next and already seen symbols

Reduction:
Discard irrelevant coefficients
Domain-specific

Representation:
Map symbols to bit strings according to their probability

Basics: Decorrelation via Prediction or Transform

A practical simplification of modelling!

Prediction
Predict the next symbol in a stream from previously seen sequences

Examples:

- Deflate: LZ77 sliding-window dictionary coder
- SZ: Polynomial prediction

Transform
Decorrelate a bounded sequence of symbols

Examples:

- JPEG: 8x8 Discrete Cosine Transform
- ndzip: 4096-element Integer Lorenzo Transform

Basics: Lossy Compression and Quantization

Idea: Truncate or reduce precision on low-impact coefficients

Inherently **specialized**, only applicable to **some domains**

Error Model:

- Absolute or relative point-wise error
- Fixed-rate with arbitrary permitted error
- Perceptual (partially subjective)

Basics: Coding

Shannon's coding theorem: From a set of symbols X with probabilities P(x), an optimal encoding for $x \in X$ uses

$$-\log_2 P(x) \text{ bits}$$

This is (almost) perfectly achievable using **Arithmetic Coding** or **Asymmetric Numeral Systems**

but

Estimating probabilities for **all** $x \in X$ must be reasonable!

Compressing Floating-Point Data

Preliminary thoughts...

- Symbols are not unrelated, but discretizations of a continuous field: exploit **smoothness** and **value proximity** rather than repetition
- Classic symbol probabilities and entropy coding not viable for lossless compression **- huge symbol sizes** (usually 32 or 64 bit)
- Unlike integers, knowledge of the data range alone does not allow predicting **leading-zero bits**

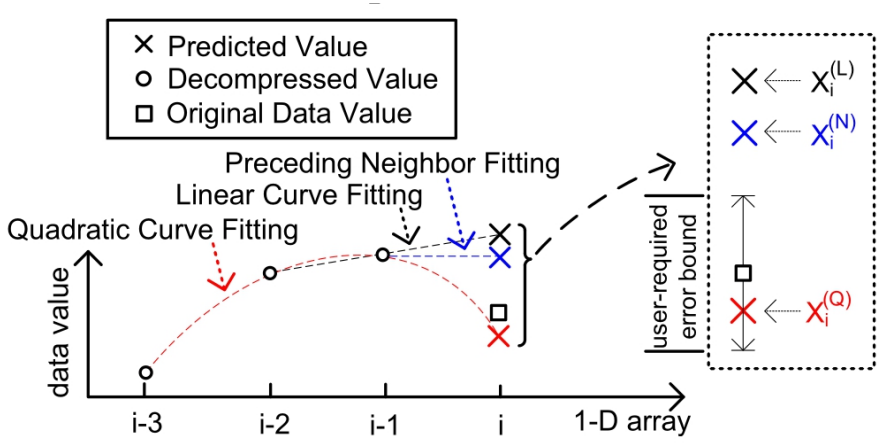
State of the Art Lossy FP Compression: SZ

Prediction-based, error-controlled compressor (Di et al., 2016)

Predictors:

- Neighbor,
- Linear,
- Quadratic

- If any prediction satisfies error bounds, encode predictor with 2 bits
- Otherwise quantize according to error bound, keep non-zero *bytes*



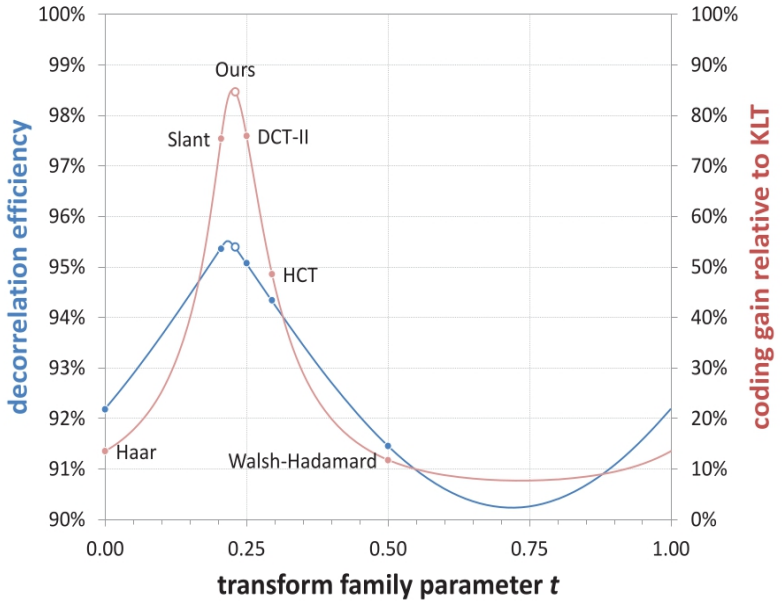
Predictors used in the SZ compressor (Di et al., 2016)

State of the Art Lossy FP Compression: ZFP

Transform-based, fixed-rate compressor (Lindstrom, 2014)

1. DCT/HWT-family **orthogonal transform** decorrelates values within a 4^d sized block
2. **Embedded coding** produces a bit stream that is **truncated** to fit the fixed, compressed block size

Not error-controlled, but well suited for **visualization** purposes

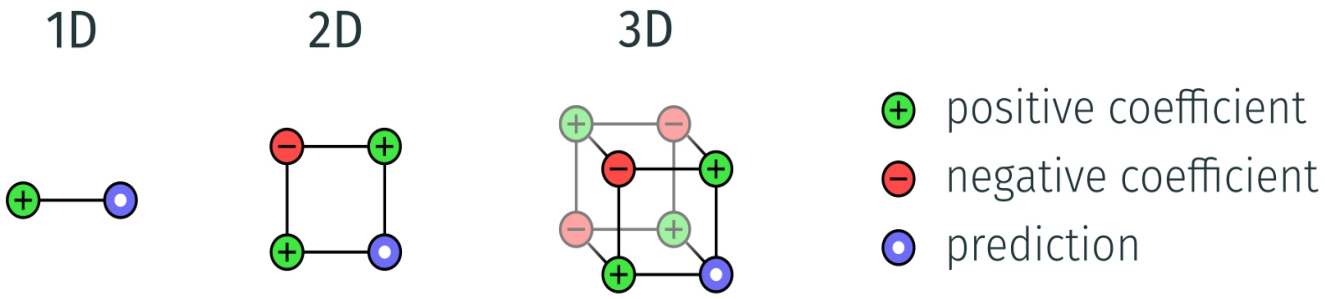


Parameter selection for orthogonal block transform in ZFP (Lindstrom, 2014)

State of the Art Lossless FP Compression: fpzip

Dimensionality-aware prediction, entropy coding (Lindstrom et al, 2006)

1- to 3-dimensional **Lorenzo predictor** for dense grids:

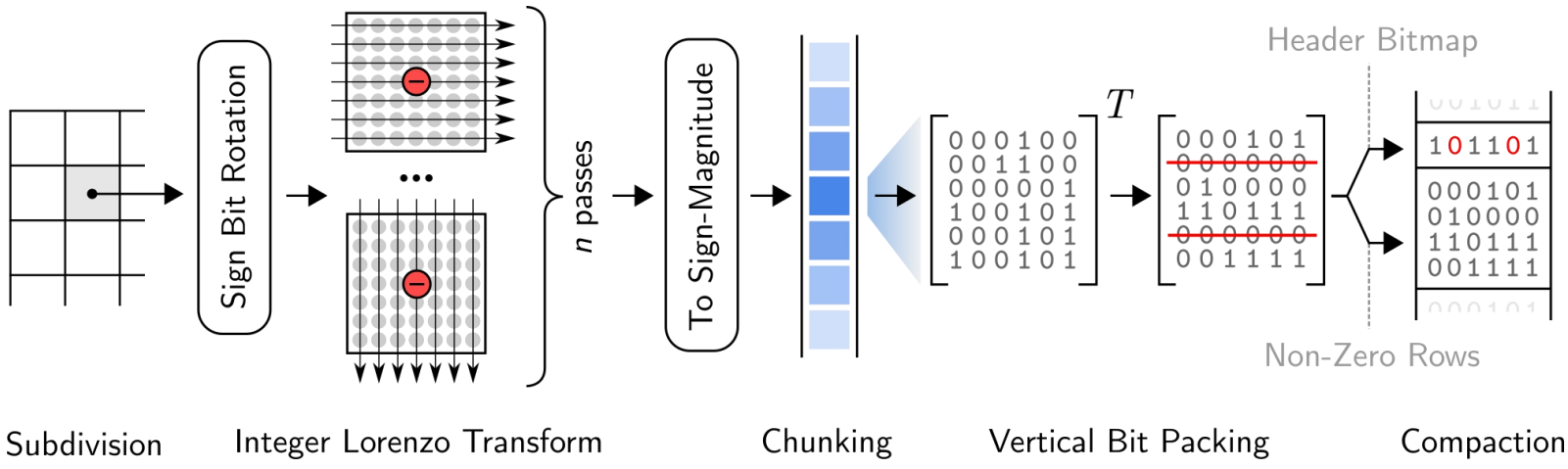


Stores **Arithmetic-Coded** residual-lengths; verbatim residual bits

Excellent prediction accuracy, limited throughput due to aritmetic coder

State of the Art Lossless FP Compressor: ndzip

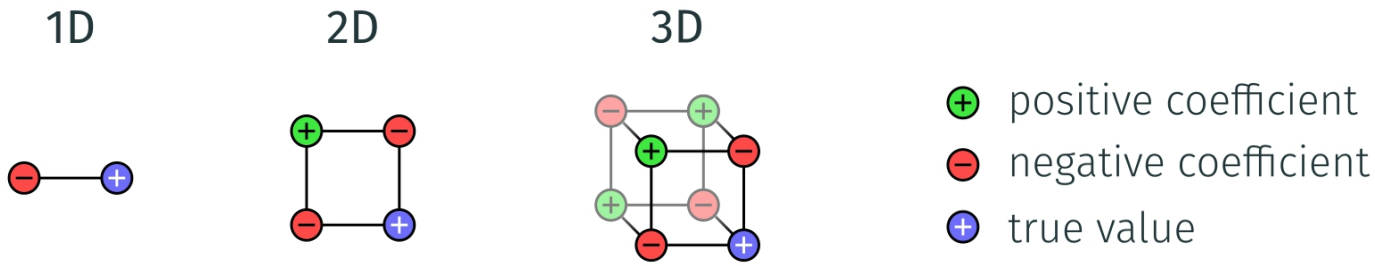
Transform-based, dimensionality-aware, bit-packing (Knorr et al., 2021)



Inherently parallel, designed for **efficient GPU implementation**

ndzip: Integer Lorenzo Transform (ILT)

1. Bijectively map floating-point bits to **integers**
2. Compute n-dimensional Lorenzo residuals **in-place**



- **Separable:** Perform 1D ILT along each axis to obtain *n*D ILT
- Forward transform is *n* **fully parallel steps**, inverse *n* **prefix sums**

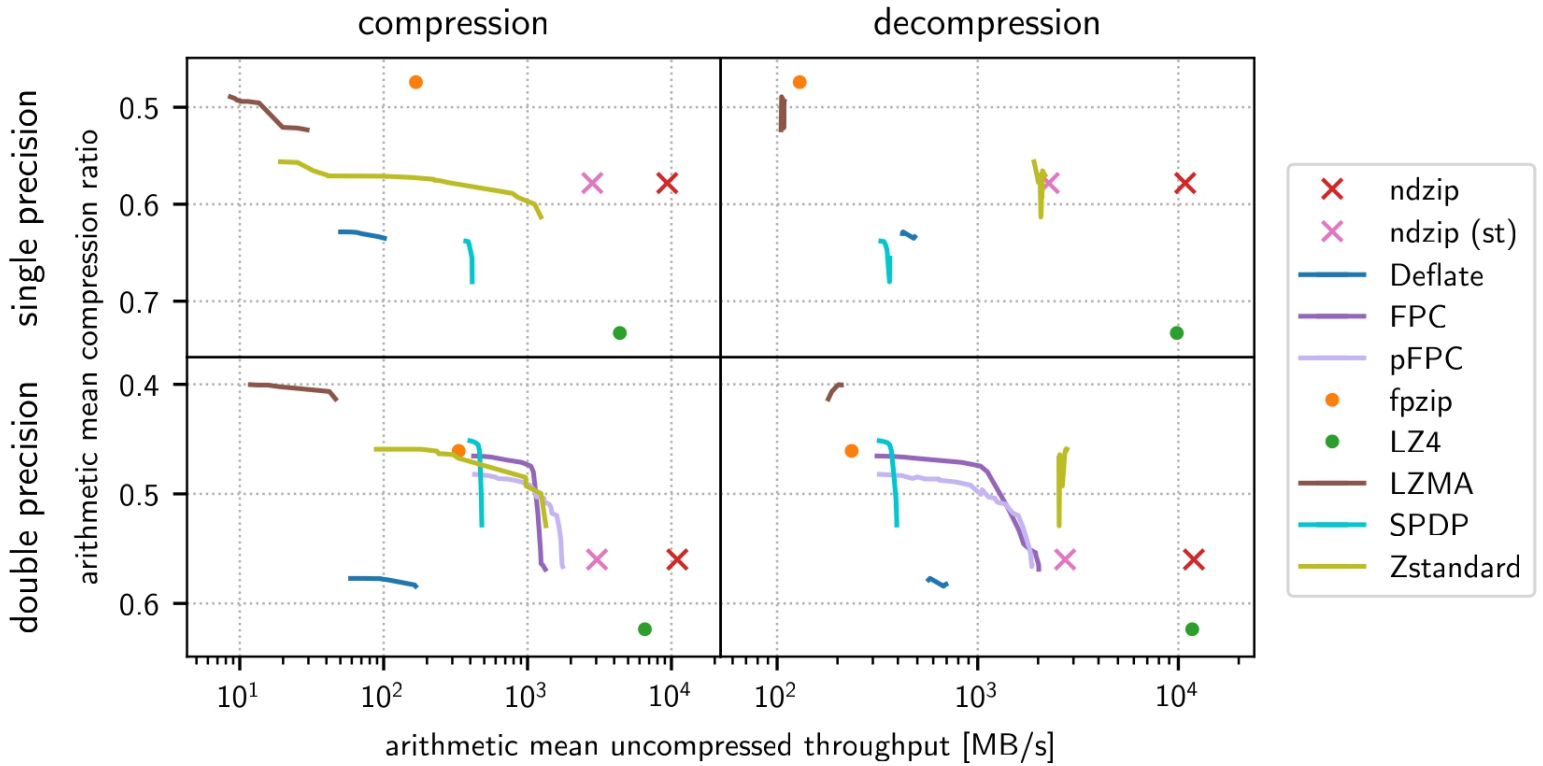
ndzip: Vertical Bit Packing

1. Map integer residuals to **sign-magnitude representation**
2. Chunk into 32×32 (64×64) bit blocks, **transpose**
3. **Strip all-zero** rows (≙ bit positions), indicate stripped positions in header

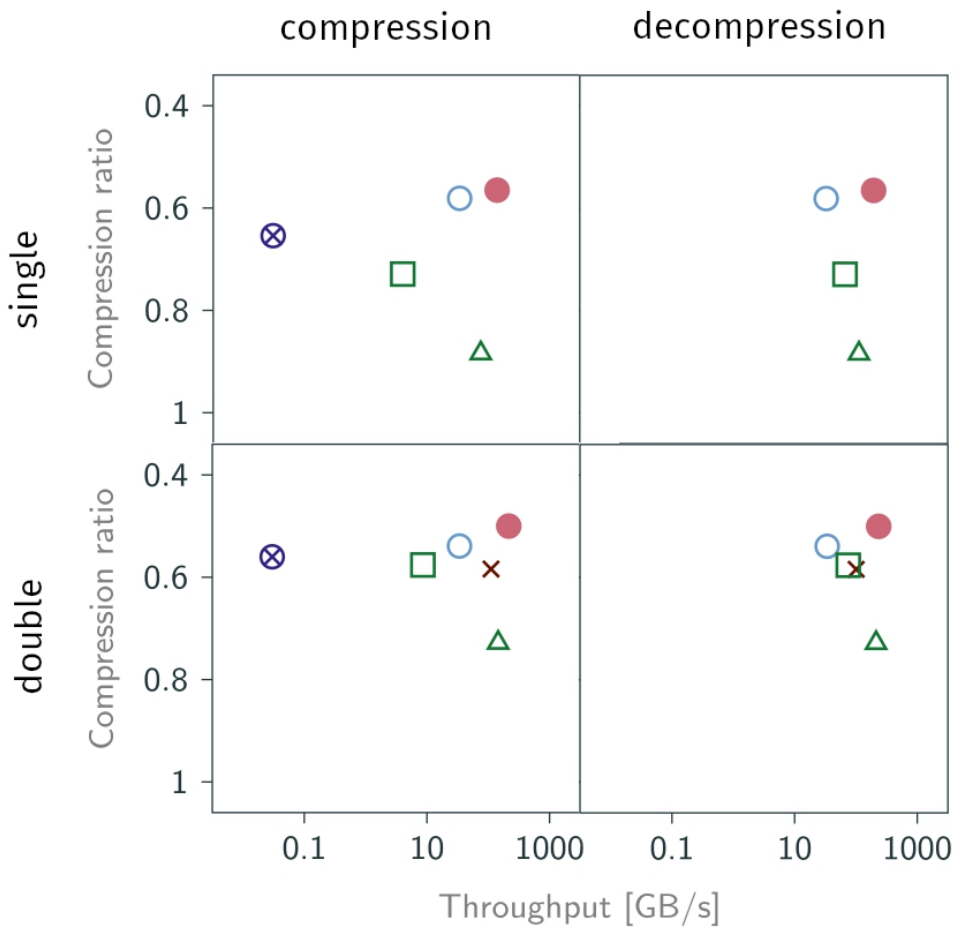
- Compresses sequences of **small-magnitude residuals** well
- **Parallel** in nature, does not require bit-level addressing

Performance Evaluation: Lossless FP Compression

CPU general-purpose and specialized lossless compressors
measured on a set of one- and multi-dimensional dense FP datasets



Performance Evaluation: Lossless FP Compression



GPU lossless compressors measured against the same collection of datasets

- ndzip-gpu
- MPC [6]
- GFC [4] (partial)
- nvCOMP [3] LZ4
- nvCOMP Cascaded
- cudppCompress [5]

How to Choose a Suitable FP Compressor?

- For **Human Consumption**
  - Error control optional, perceptual / fixed-rate compression viable
- For data with **known precision** (e.g. measurement data)
  - Absolute or relative error-controlled compression
- In the **general case**, and for data with **tight error bounds**
  - Lossless compression
  - Compression ratio / throughput trade-off (use-case dependent)

ndzip is Developed as Part of the Celerity Runtime

(Thoman et al., 2019)

Runtime system for GPU clusters

- Based on SYCL
- Purely declarative data flow
- Well-suited for multidimensional algorithms on dense arrays

Current development goals:

- Transfer latency optimization
- Fast automatic checkpointing

. . . all without user intervention!

Thank You!

... and feel free to reach out anytime:

Me	Fabian Knorr <fabian.knorr@uibk.ac.at>
ndzip	https://github.com/fknorr/ndzip
Celerity	https://github.com/celerity/celerity-runtime

Further Reading

Matt Mahoney on General Data Compression:

http://www.mattmahoney.net/dc/dce.html

A broad **overview over lossy floating-point compression**:

Cappello, Franck, et al. "Use cases of lossy compression for floating-point data in scientific data sets." The International Journal of High Performance Computing Applications 33.6 (2019): 1201-1220.

References

Efnusheva et al, 2017: Efnusheva, Danijela, Ana Cholakoska, and Aristotel Tentov. "A survey of different approaches for overcoming the processor-memory bottleneck." AIRCC's International Journal of Computer Science and Information Technology 9.2 (2017): 151-163.

Knorr et al., 2021: Knorr, Fabian, Peter Thoman, and Thomas Fahringer. "ndzip: A High-Throughput Parallel Lossless Compressor for Scientific Data." 2021 Data Compression Conference (DCC). IEEE, 2021.

Knorr et al. 2021: Knorr, Fabian, Peter Thoman, and Thomas Fahringer. "ndzip-gpu: efficient lossless compression of scientific floating-point data on GPUs." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021.

Di et al., 2016: Di, Sheng, and Franck Cappello. "Fast error-bounded lossy HPC data compression with SZ." 2016 ieee international parallel and distributed processing symposium (ipdps). IEEE, 2016.

Lindstrom, 2014: P. Lindstrom, "Fixed-Rate Compressed Floating-Point Arrays," in IEEE Transactions on Visualization and Computer Graphics, vol. 20, no. 12, pp. 2674-2683, 31 Dec. 2014, doi: 10.1109/TVCG.2014.2346458.

Lindstrom et al., 2006: P. Lindstrom and M. Isenburg, "Fast and Efficient Compression of Floating-Point Data," in IEEE Transactions on Visualization and Computer Graphics, vol. 12, no. 5, pp. 1245-1250, Sept.-Oct. 2006, doi: 10.1109/TVCG.2006.143.

Thoman et al., 2019: Thoman, Peter, et al. "Celerity: High-level c++ for accelerator clusters." European Conference on Parallel Processing. Springer, Cham, 2019.