Fast Compression of Floating-Point Values with Exponent/Mantissa Shuffling

L. Oden, S. Litzinger, J. Keller

PARS Workshop 2024

Faculty of Mathematics and Computer Science
Parallelism and VLSI Group
Sebastian Litzinger

FernUniversität in Hagen

Motivation

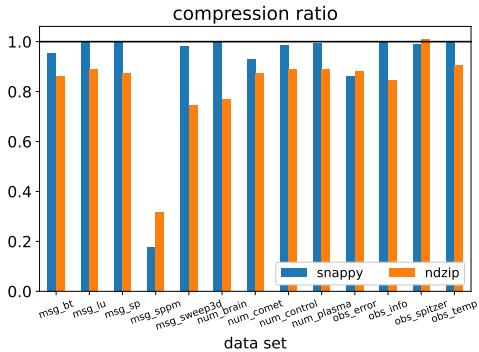
- performing computations on vectors and matrices with the help of accelerators incurs communication cost
- compression of data can aid in reducing communication overhead
- quality of compression must justify additional computational effort
- aim for high compression ratio and high throughput

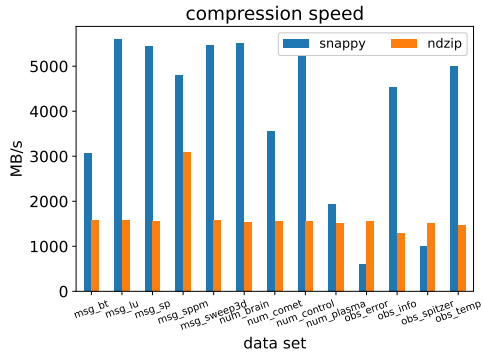
Floating-Point Compression

- established (lossless) compression algorithms, e.g.,
  - Google Snappy (https://github.com/google/snappy): "it aims for very high speeds and reasonable compression"
  - ndzip (Knorr et al. 2021): "competitive trade-off between compression effectiveness and throughput"

- floating-point data set collected by Burtscher et al.: https://userweb.cs.txstate.edu/ burtscher/research/datasets/FPsingle/
- 13 data sets, single-precision floating-point numbers
- observational data, numeric simulations, parallel messages

Floating-Point Compression





IEEE 754 Floating-Point Number Representations

- in *single precision*, we have a 32-bit value, concatenation of
  - sign bit s,
  - 8-bit exponent e,
  - 23-bit mantissa m.
- the number thus represented is $(-1)^s \cdot 2^{e-127} \cdot \langle 1.m \rangle, 1 \leq e \leq 254$
- $\langle m_0.m_{-1} \cdots m_{-23} \rangle = \sum_{i=-23}^{0} m_i \cdot 2^i$
- special cases for $e \in \{0, 255\}$

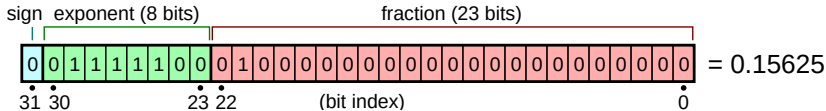
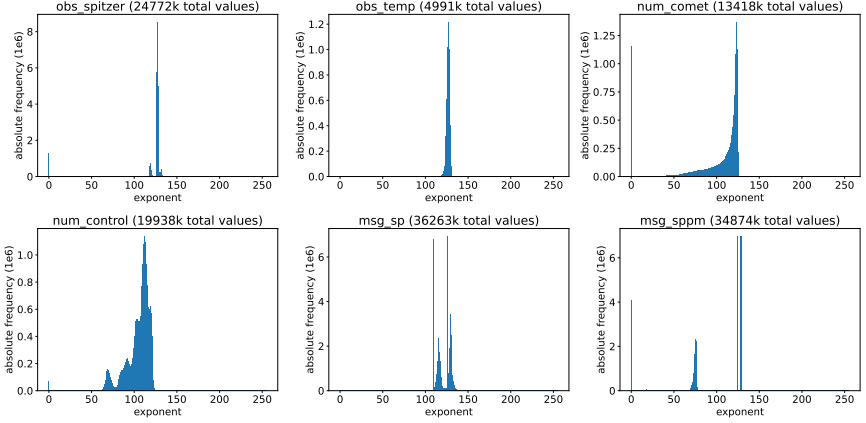
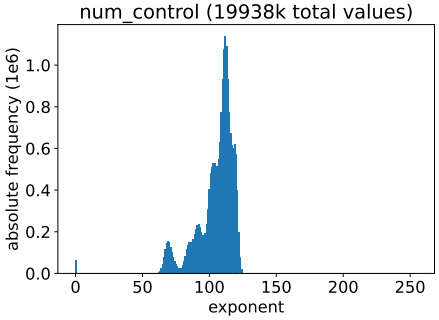


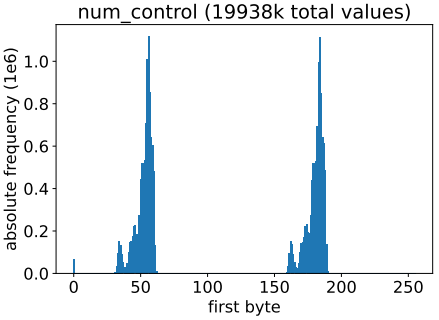
Figure: binary representation of a 32-bit floating-point number by Fresheneesz, English Wikipedia project

Exponent Distribution in Real Data Sets

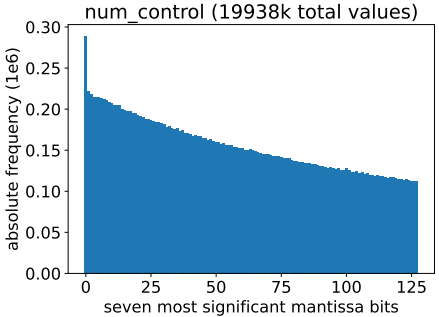


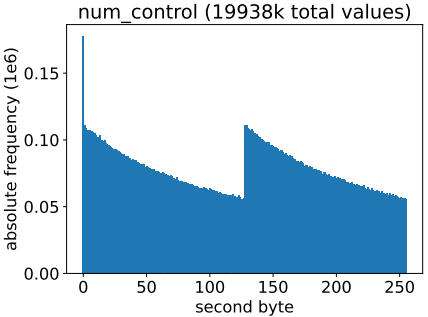
Value Distribution in Real Data Sets





Value Distribution in Real Data Sets





Lossless Compression

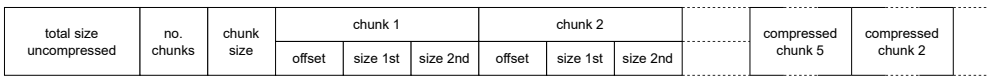
- redundancy in sequence of symbols: patterns can occur repeatedly
- remove pattern instance and refer to earlier occurrence
- (maximum) pattern length and (maximum) distance between pattern instances will impact compression ratio and computational effort
- additionally, encoding techniques or filters can be employed
- in HDF5, e. g., the SHUFFLE filter performs byte shuffling (i. e., sequence of multi-byte number representations is rearranged into sequence of all 1st bytes, followed by sequence of all 2nd bytes, etc.)

Exponent/Mantissa Shuffling

- idea: rearrange data into sequence of exponent values and sequence of mantissa values
- easier: rearrange data into sequence of first byte values and sequence of remaining bytes
- for floats $f_i, i = 0, \ldots, n-1$, we have bytes $m_{i,3}m_{i,2}m_{i,1}m_{i,0}$
- rearrange into the sequences $m_{0,3}m_{1,3}\cdots m_{n-1,3} \quad m_{0,2}m_{0,1}m_{0,0}\, m_{1,2}m_{1,1}m_{1,0}\cdots m_{n-1,2}m_{n-1,1}m_{n-1,0}$
- 0x38635FB7, 0x387380CF, 0x38813259, 0x3887FAA3 $\rightarrow$ 38383838 635FB773 80CF8132 5987FAA3
- blockwise application to the input enables parallelization

Parallelization

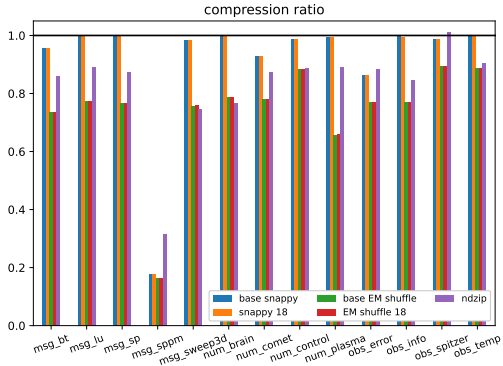
- split input into chunks
- for each chunk: shuffle, compress first bytes, compress remaining bytes (Snappy)
- threads can work on their respective chunks independently
- size of compressed chunks can vary: additional temporary buffer for each thread, shared variable for offset in output buffer
- order of compressed chunks in the output buffer may differ from order of umcompressed chunks in the input data
- if compression ratio for remaining bytes (“mantissa part”) is bad, store data uncompressed
- decompression can also be carried out in parallel

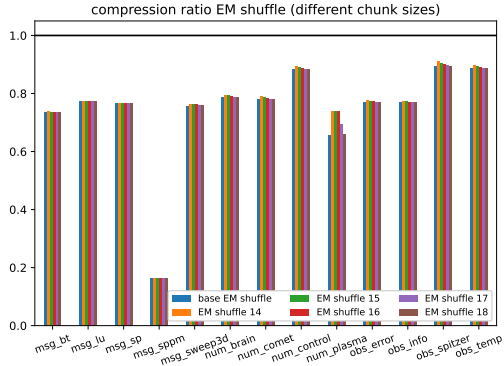


Experimental Setup

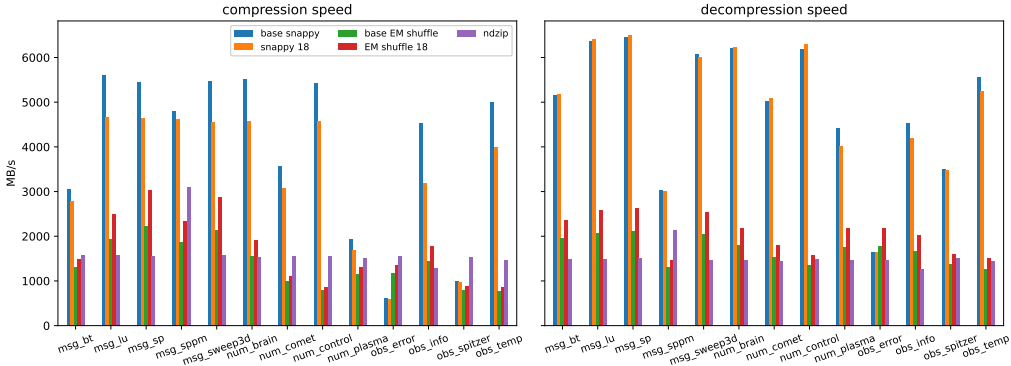
- experiments on system featuring 2x AMD EPYC 7742 (*Rome* series, Zen 2 microarchitecture), 2x 64 cores
- 13 single-precision floating-point data sets from Burtscher's collection
- Snappy
- ndzip (CPU)
- EM shuffling + Snappy
- EM shuffling + Snappy (256 kB chunks)
- Snappy (256 kB chunks)
- various chunk sizes for EM shuffling + Snappy

Experimental Results: Compression Ratio

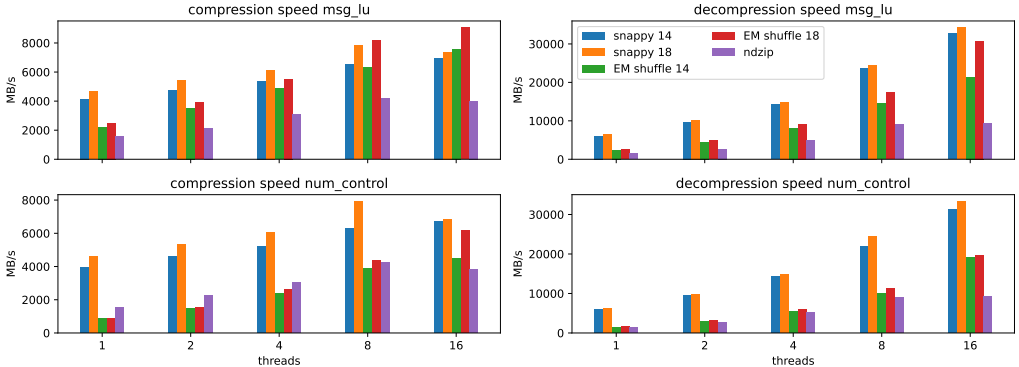




Experimental Results: Speed



Experimental Results: Multi-Threaded Performance



Summary

- lossless compression of floating-point data: compression ratio vs. performance
- in real-world data sets, oftentimes few exponent values occur
- rearrange data into sequence of first bytes and sequence of remaining bytes
- blockwise application for parallelization
- mostly slower than Snappy but better compression
- GPU-based algorithms (e.g., ndzip-GPU), half precision, double precision, parallel performance

Thank you!

sebastian.litzinger@fernuni-hagen.de

https://fernuni-hagen.de/pv

https://sglitzinger.github.io