LEAPS-INNOV – WP7

Data reduction and compression

Evaluation of Lossless Compression Algorithms on Tomography Data obtained at SOLEIL

Synchrotron SOLEIL

May 3, 2022

Definition of data compression[1]

Lossless vs lossy

- Data compression consists in **reducing the number of bits** needed to **store** or **transmit** some data;
- Compression can either lossless or lossy;
- Losslessly compressed data can be decompressed to exactly its original value:

$$\mathcal{D}(\mathcal{C}(s)) = s$$

- Lossy compression consists of a transform to separate important (s_1) from unimportant (s_2) data[2], followed by lossless compression of the important part and discarding the rest:

$$s = s_1 + s_2 \text{ and } \mathcal{D}(\mathcal{C}(s_1)) = s_1 \simeq s$$

In this talk, we'll be focusing on lossless data compression[3].

[1] Adapted from Matt Mahoney's book: *Data Compression Explained*.

[2] Unimportant data are those that falls outside human perception.

[3] Lossless data compression is much more important than lossy data compression. In accelerator-based photon source facilities, scientists are generally very reluctant to lose and/or throw away data, since all of them are potentially useful, even decisive, at some point for a reason or another. Besides, it might seem odd to develop more and more powerful detectors (with more pixels, higher sampling rate and/or extra dimension) and do lossy compression right after... Lossy data compression is useful when critical results are needed quickly in a workflow or to facilitate analysis of scientific data once the data has been acquired.

Characterizing compression

Compactness and speed

The compression ratio, R, is defined as the ratio between the number of bits between the uncompressed data size and compressed data size. It depends on the:

- strategy used to compress (e.g., sliding window, dictionary, etc.);
- understanding of the data (spatial or temporal structure, noise, etc.);
- memory size.

The compression speed, V_c, is the number of bits that are processed per unit of time during compression. It depends on the:

- hardware[1] (i.e., CPU core, bus and memory access speeds; # cores);
- algorithm implementation details (data structures used in the program, algorithmic complexity of the involved operations, etc.);
- bandwidth.

Storage targets large R, but **what about time-related aspects?**

[1] Therefore, the values of V_c are highly dependent on the computer being used and, potentially, what is happening on it during the performance testing.

Introducing some variables

N	# bits in the raw data
N_c	# bits in the compressed data
$R \equiv N/N_c$	**Compression ratio**[1]
V_c	Data compression speed
V_d	Data decompression speed
V_t	Data transfer speed
V_r, V_w	Speed for reading/writing data from/to a disk
V_f	Speed to transform/rearrange data
$\sigma_{c/\bullet} \equiv V_c/V_\bullet$	**Normalized compression speed**
$\sigma_{d/\bullet} \equiv V_d/V_\bullet$	**Normalized decompression speed**
$T_c(N)$	Time for compressing N bits
$T_d(N)$	Time for decompressing N bits
$T_t(N)$	Time for transferring N bits
$T_r(N), T_w(N)$	Time for reading/writing N bits from/to a disk
$T_f(N)$	Time for transforming some data of size N bits

Table: Definition of some variables used later.

[1] (pre)-caution: some authors use the opposite convention...

Time consideration: transfer over the network

Is compression really necessary?

A compression/decompression cycle is interesting in terms of time if:

$$\underbrace{T_c(N)}_{\substack{\text{time to}\\ \text{compress } N \text{ bits}}} + \underbrace{T_t(N_c)}_{\substack{\text{time to}\\ \text{transfer } N_c \text{ bits}}} + \underbrace{T_d(N_c)}_{\substack{\text{time to}\\ \text{uncompress } N_c \text{ bits}}} < \underbrace{T_t(N)}_{\substack{\text{time to}\\ \text{transfer } N \text{ bits}}} \quad (1)$$

That is:

$$\frac{N}{V_c} + \frac{N/R}{V_t} + \frac{N/R}{V_d} < \frac{N}{V_t}$$

which leads to the condition (using $\sigma_{c/t} \equiv V_c/V_t$ and $\sigma_{d/t} \equiv V_d/V_t$):

$$\boxed{\frac{1}{\sigma_{c/t}} + \frac{1}{R} + \frac{1}{R\,\sigma_{d/t}} < 1} \quad (2)$$

It suffices that one of these terms is ≥ 1 to break the above inequality.
In particular, we want $\sigma_{c/t} > 1$ which implies that V_c is a limiting factor.

But regardless of any temporal consideration,
data compression is only useful if: $\boxed{R > 1}$!

Time consideration: transfer over the network

Which data compression algorithm to choose?

Given two compression techniques (indices 1 and 2), we compare the time taken by each of them to compress, transfer and decompress the data :

$$
\begin{aligned}
& T_{c,1}(N) + T_t(N_{c,1}) + T_d(N_{c,1}) \quad \text{vs} \quad T_{c,2}(N) + T_t(N_{c,2}) + T_d(N_{c,2}) \\
\Leftrightarrow \; & \frac{N}{V_{c,1}} + \frac{N/R_1}{V_t} + \frac{N/R_1}{V_{d,1}} \quad \text{vs} \quad \frac{N}{V_{c,2}} + \frac{N/R_2}{V_t} + \frac{N/R_2}{V_{d,2}} \\
\Leftrightarrow \; & \frac{1}{\sigma_{c/t,1}} + \frac{1}{R_1} + \frac{1}{R_1\,\sigma_{d/t,1}} \quad \text{vs} \quad \frac{1}{\sigma_{c/t,2}} + \frac{1}{R_2} + \frac{1}{R_2\,\sigma_{d/t,2}}
\end{aligned}
$$

In other words, we have to compute:

$$
\boxed{\theta_t \equiv \frac{1}{\sigma_{c/t}} + \frac{1}{R} + \frac{1}{R\,\sigma_{d/t}}} \tag{3}
$$

for each method and pick the one with the **smallest** value[1].

[1] If storage and fast transmission are considered part of the same problem, then the criterion is to choose a compression algorithm characterized by the largest R with a reasonably small θ.

Time consideration: transfer over the network

Application to the Silesia corpus

Numbers for R, V_c and V_d were taken from the LZ4 github[1].

Algorithm	R	$\frac{V_c}{\text{MB/s}}$	$\frac{V_d}{\text{MB/s}}$	$\frac{V_d}{V_c}$	θ_{100}	θ_{200}	θ_{300}
LZ4	2.101	780	4970	6.4	0.61	0.75	0.89
LZO	2.108	670	860	1.3	0.68	0.88	1.1
Snappy	2.091	565	1950	3.5	0.68	0.88	1.1
Zstd	2.883	515	1380	2.7	0.57	0.79	0.99
zlib deflate	2.730	100	415	4.2	1.5	2.5	3.6
LZ4-HC -9	2.721	41	4900	120	2.8	5.3	7.7

Table: Values of θ_t obtained for different transfer speeds: $V_t = 100$ MB/s (θ_{100}), $V_t = 200$ MB/s (θ_{200}) and $V_t = 300$ MB/s (θ_{300}).

[1] Benchmark values were obtained on a system with a Core i7-9700K CPU @ 4.9GHz (w/ turbo boost).

Time consideration: file system/HDF5/disk backup

Another very common situation where comp. and decomp. are considered as two separate phases

- Compression (usually done only once) is interesting if:

$$\underbrace{T_c(N)}_{\substack{\text{time to}\\ \text{compress } N \text{ bits}}} + \underbrace{T_w(N_c)}_{\substack{\text{time to}\\ \text{write } N_c \text{ bits}}} < \underbrace{T_w(N)}_{\substack{\text{time to}\\ \text{write } N \text{ bits}}} \Rightarrow \boxed{\theta_w \equiv \frac{1}{\sigma_{c/w}} + \frac{1}{R} < 1} \quad (4)$$

- Decompression (usually done many times[1]) is interesting if:

$$\underbrace{T_r(N_c)}_{\substack{\text{time to}\\ \text{read } N_c \text{ bits}}} + \underbrace{T_d(N_c)}_{\substack{\text{time to}\\ \text{decomp. } N_c \text{ bits}}} < \underbrace{T_r(N)}_{\substack{\text{time to}\\ \text{read } N \text{ bits}}} \Rightarrow \boxed{\theta_r \equiv \frac{1}{R} + \frac{1}{R\,\sigma_{d/r}} < 1} \quad (5)$$

But first and foremost, data compression is only useful if: $\boxed{R > 1}$![2]

In short: efficient data compression means taking up less storage and improving I/O throughput (at the cost of increased CPU use for the comp./decomp. ops).

[1] This is why it is much more important to have $V_d \gg V_c$.

[2] In ZFS, for instance, if the first portion of data being compressed is not smaller than the original, compression is disabled.

Ways of improvement to compress faster and/or more

"Memory I/O [is] the performance bottleneck" (F. Alted)

- Working directly in the RAM (e.g., `/dev/shm`) is much more efficient than working from an SSD or HDD disk and reduces the time for data compression (would this be compatible with your workflow?);
- Clever use of the characteristics of memory hierarchical model (i.e., L1, L2 and/or L3 CPU caches) to optimize memory I/O has proven to provide very good results (e.g., `LZ4` which uses L1 caches);
- Multi-threading (e.g., `Zstd`) can also reduce the time for data loading in memory and compression[1,2]. Data is split into chunks, chunks are compressed in parallel and merged at the end;
- Rearranging the internal data structure is very efficient for reaching higher R (e.g., `bitshuffle`). Of course, this introduces additional delays $T_f(N)$ and $T_{f^{-1}}(N)$ in the balance;
- Any combination of the above can be successful (e.g., `BloscLZ`).

[1] However, V_c will not increase linearly or indefinitely with the number of threads.

[2] A few algorithms, such as `pbzip2` and `pixz`, offer multi-threading capabilities for both compression and decompression, but in general at the cost of a very slow speed.

Constitution of a synchrotron data corpus

SOLEIL's contribution

Beamline (sorted by productivity)	Volume (TB) 2020	2021	Techniques
ANATOMIX*	194	281	Nano- and Micro-Tomography
PSICHE*	170	118	Tomography, Diffraction
PROXIMA-2A	82	150	Tomography
SWING[1]	23	29	Small/Wide Angle X-ray Scattering
NANOSCOPIUM[1]	21	14	Tomography, Diffraction, Spectroscopy
CRISTAL*	4	7	X-ray Diffraction
SAMBA	2	2	X-ray Absorption Spectroscopy

Table: List of the beamlines (out of the 29 in use at SOLEIL) selected for providing reference data sets for the LEAPS-INNOV WP7. All are in the hard X-ray regime ($E_\gamma \geq 5$ keV). The * indicates the beamlines for which we have obtained one or more data sets after presenting the project to the beamline officiers/scientists.

[1] SWING has produced more data than NANOSCOPIUM for two reasons. First, SWING has integrated a Nanoprobe system to do ptychography nano-imaging. Second, NANOSCOPIUM did not work at full capacity during these two years, not to mention that it does not only do tomography as indicated.

ANATOMIX

A beamline dedicated to tomography

Most productive beamline at SOLEIL: 194 TB (2020), 281 TB (2021).

Two kind of tomography:

- Transmission X-ray Microscopy (macro-tomography),
- Parallel Beam Imaging (micro-tomography).

Data acquisition:

- 1 second per scan (up to 50 ms per scan).

Outputs: reconstructed volumes (`.vol` file), obtained with PyHST2 using:

- projection radiographs (saved in a .nxs file or in many .edf files),
- calibration, flat-field, dark-field images.

Typical file sizes obtained in a few seconds[1] are about:

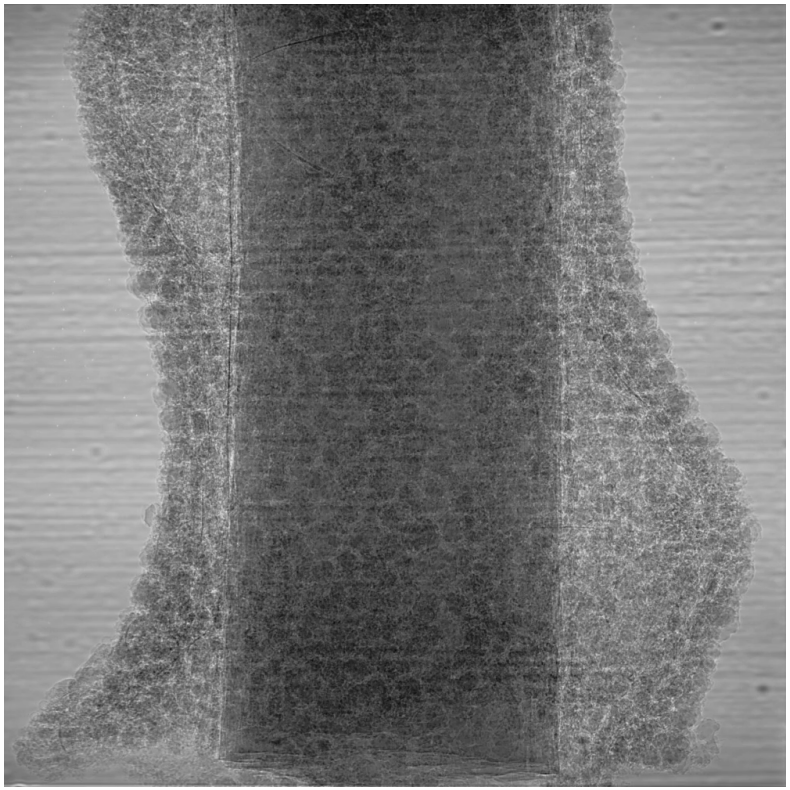
- 16 GB for a 2048x2048 pixels x 2 bytes/pixel x 2000 projections (`.nxs`)
- 32 GB for a 2048x2048x2048 voxels x 4 bytes/voxel (`.vol`)

[1] Using the Hamamatsu Orca Flash camera (20 fps, 168 MB/s recording speed).

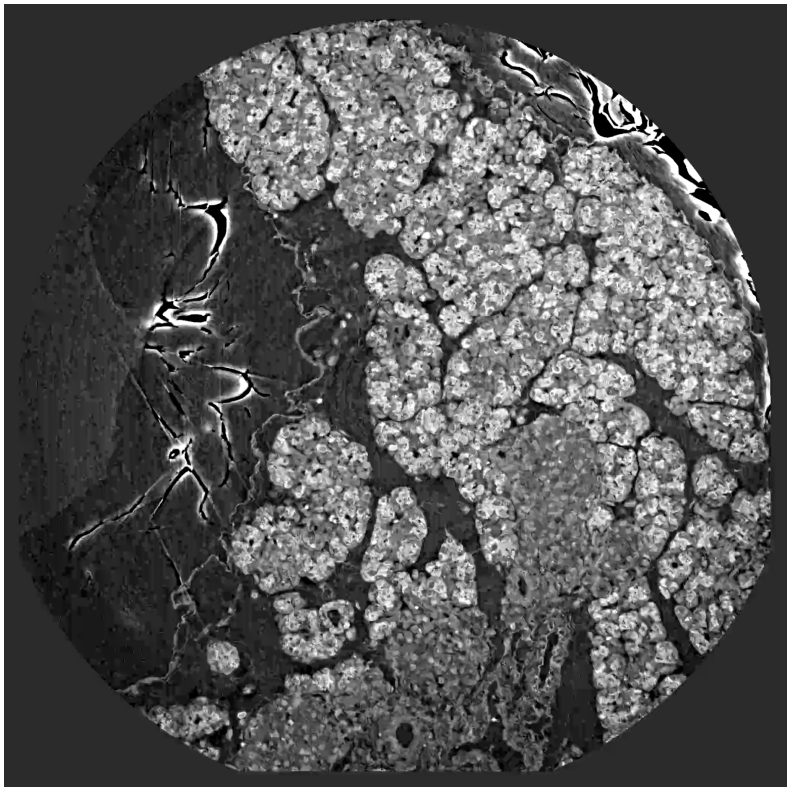
Data sets provided by ANATOMIX

Two examples (images)

Tropical coral (`.edf` file)



Pancreas (`.vol` file)



Data sets provided by ANATOMIX

Two examples (movies)

Tropical coral (`.edf` file)

Pancreas (`.vol` file)

coral

pancreas

Evaluation of lossless compression algorithms[1] /1

Performances in the (V_c, R) plane

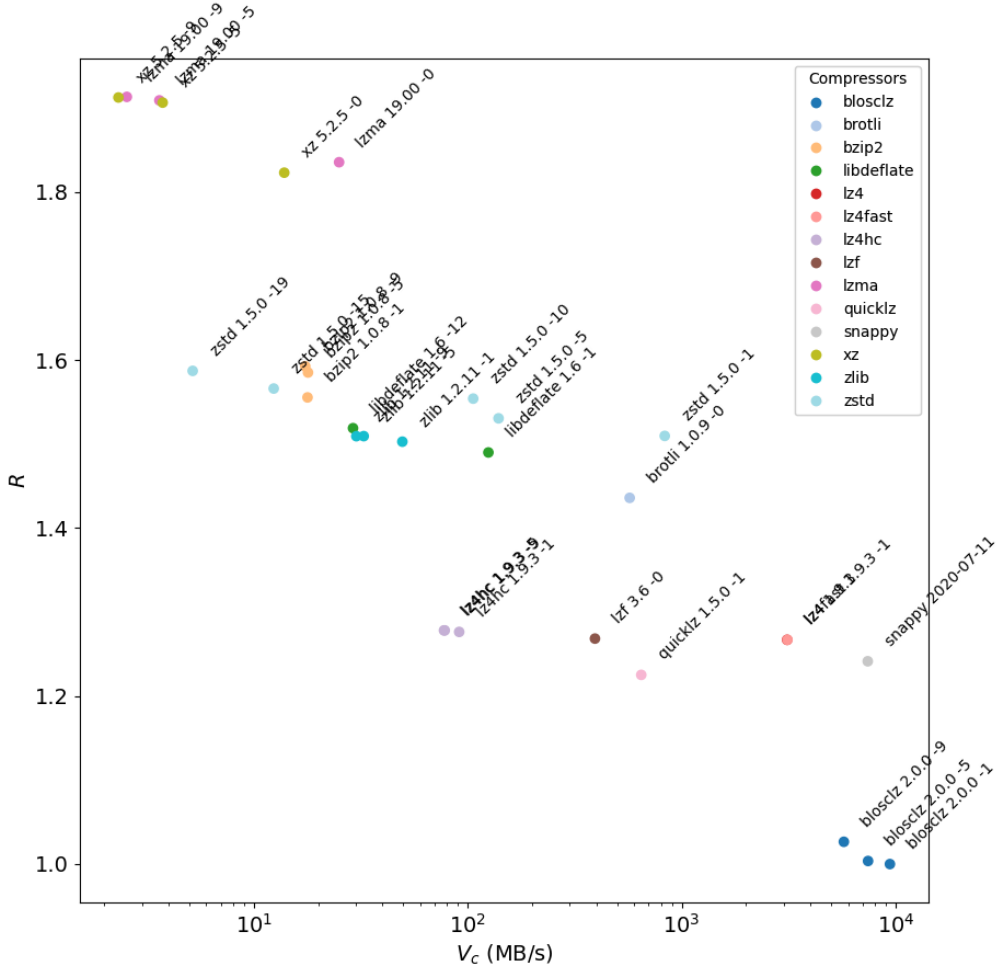


Figure: Comparison of several lossless compression algorithms on a tomography `.vol` file of size 32 GB (provided by the ANATOMIX beamline). The labels (e.g., `zstd 1.5.0 -19`) indicates the name of the compressor (`zstd`), its version (`1.5.0`) and often the compression level (`19`). Tests were done a system with an AMD Ryzen Threadripper 3990X 128-Core Processor @ 2.2 GHz (here and thereafter).

Evaluation of lossless compression algorithms/2

Performances in the (V_c, R) plane

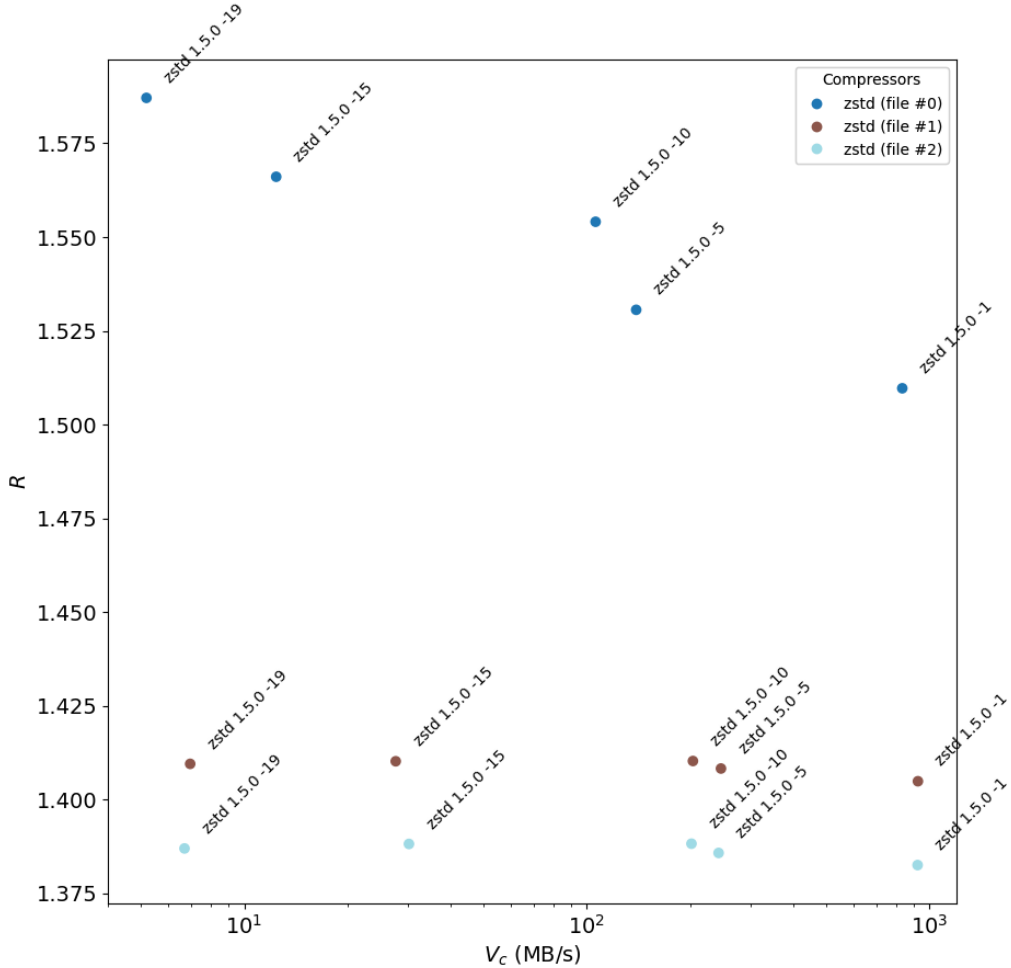


Figure: Comparing one compression algorithm (`Zstd`) on three different 32 GB tomography `.vol` files (provided by the ANATOMIX beamline). Low compression levels are close to real time, whereas higher compression levels take more CPU time (V_c decreases) and, as observed, do not necessarily bring significant improvement in the compression ratio, R. Furthermore, we can see that the different `.vol` files did not result in susbtantial variations in R.

Evaluation of lossless compression algorithms/3

Evolution of V_c with increasing threads

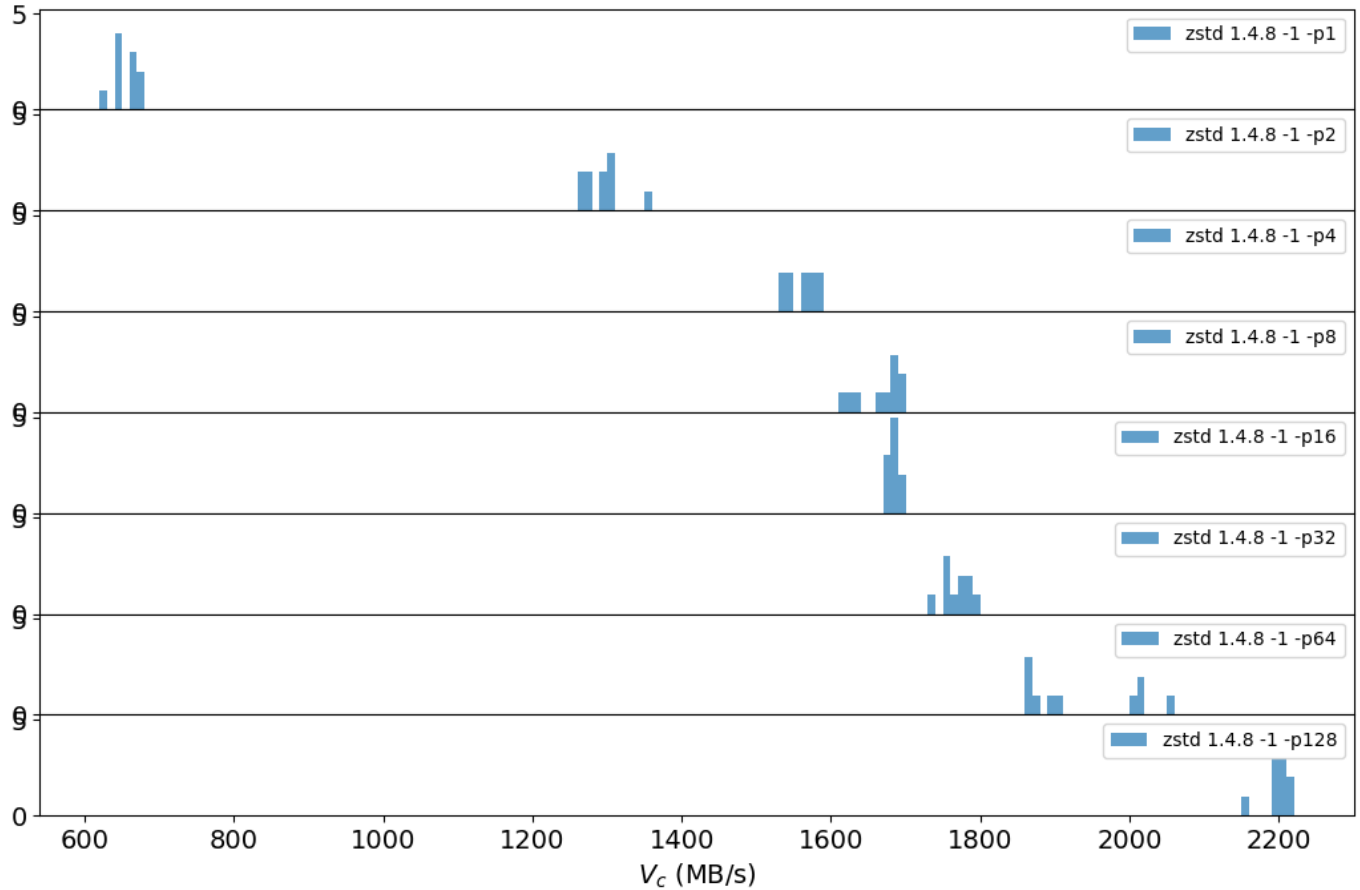


Figure: Effect of multi-threading on V_c using `Zstd` on a 33 GB `.vol` file (provided by the ANATOMIX beamline). The number of threads was progressively increased in a sequence of powers of two, going from 1 to 128. The performance tests were repeated 10 times, hence the histograms of V_c (the bin size was set to 10 MB/s).

Source codes developed for benchmarking

Available on GitLab

- lzbench-toolbox: bash scripts to use lzbench (an in-memory benchmark of open-source LZ77/LZSS/LZMA compressors);
- codec-bench: bash scripts to obtain the metrics associated with compression and decompression (i.e., R, V_c, V_d). It outputs CSV files similarly to lzbench, but it includes compressors with multi-threading capabilities. The compressors currently implemented are: `bzip2`, `gzip`, `lz4`, `pbzip2`, `pigz`, `pixz`, `xz`, `zstd`;
- codec-bench-analysis: Python scripts to plot, analyze and print the results from lzbench-toolbox and codec-bench.

Evaluating compression algorithms

A delicate excercise...

- **Possible disymmetry**: compression and decompression are usually tested on the same machine for convenience, but one and the other could happen in different places in real life;
- **Evaluation condition**: it would be desirable to work in an isolated and clean computing environment (e.g., clean memory caches) to obtain intrinsic and robust measures on V_c and V_d, but these would still remain overestimates compared to what would happen in real conditions of use;
- **Length**: compressing large volume of data (tens of GB) can be quite long (up to an hour or so), especially for the highest levels of compression (which, moreover, may not bring any significant improvement in terms of R) and/or for compressors developed solely for the purpose of having large R, regardless of time;
- **Statistics**: running the tests several times to obtain statistics on the V_c and V_d is questionable given the above points. Gaussian distributions should not be expected;
- **Ecological/economical footprint**: test less, test better[1].

[1] A very compressible sentence...