zfp compression on a CMS NanoAOD

Erik Wallin

IRIS-HEP Fellowship (Based at Lund Uni., Sweden)
with Oksana Shadura and Brian Bockelman

"zfp is a BSD licensed open source library for compressed floating-point arrays that support **high throughput read and write random access**. [...]
zfp was designed to achieve high compression ratios and therefore uses **lossy** but optionally **error-bounded compression**. Bit-for-bit lossless compression of integer and floating-point arrays is also supported. zfp is often more accurate and faster than other lossy compressors, especially in its **OpenMP and CUDA multithreaded modes.**"

From: **https://computing.llnl.gov/projects/floating-point-compression**

In short:

- Lossy (optionally lossless)
- High throughput read and write random access
- `https://github.com/LLNL/ZFP`
- Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. IEEE Transactions on Visualization and Computer Graphics, 20(12):2674-2683, December 2014. doi:10.1109/TVCG.2014.2346458

Compression in CMS NanoAODs today:

1. Float mantissas are first truncated, to a user specified precision

   E.g. 1.1101011 $\rightarrow$ 1.11010000

   This is done in libminifloat.h in CMSSW (Abbreviated **LMF** on plots)
   `https://github.com/cms-sw/cmssw/blob/master/DataFormats/Math/interface/libminifloat.h`

2. Followed by the usual lossless compression methods available (which improves due to the float truncation)

Let's see how zfp compares to: LZMA, ZLIB, lz4 and Zstd

Super naive implementation:

- Pure Python (3.6) implementation using external libraries: pyzfp, Zstd and lz4. As well as LZMA and ZLIB from the standard library.
- Flattening all arrays (no jagged arrays)

We compare:

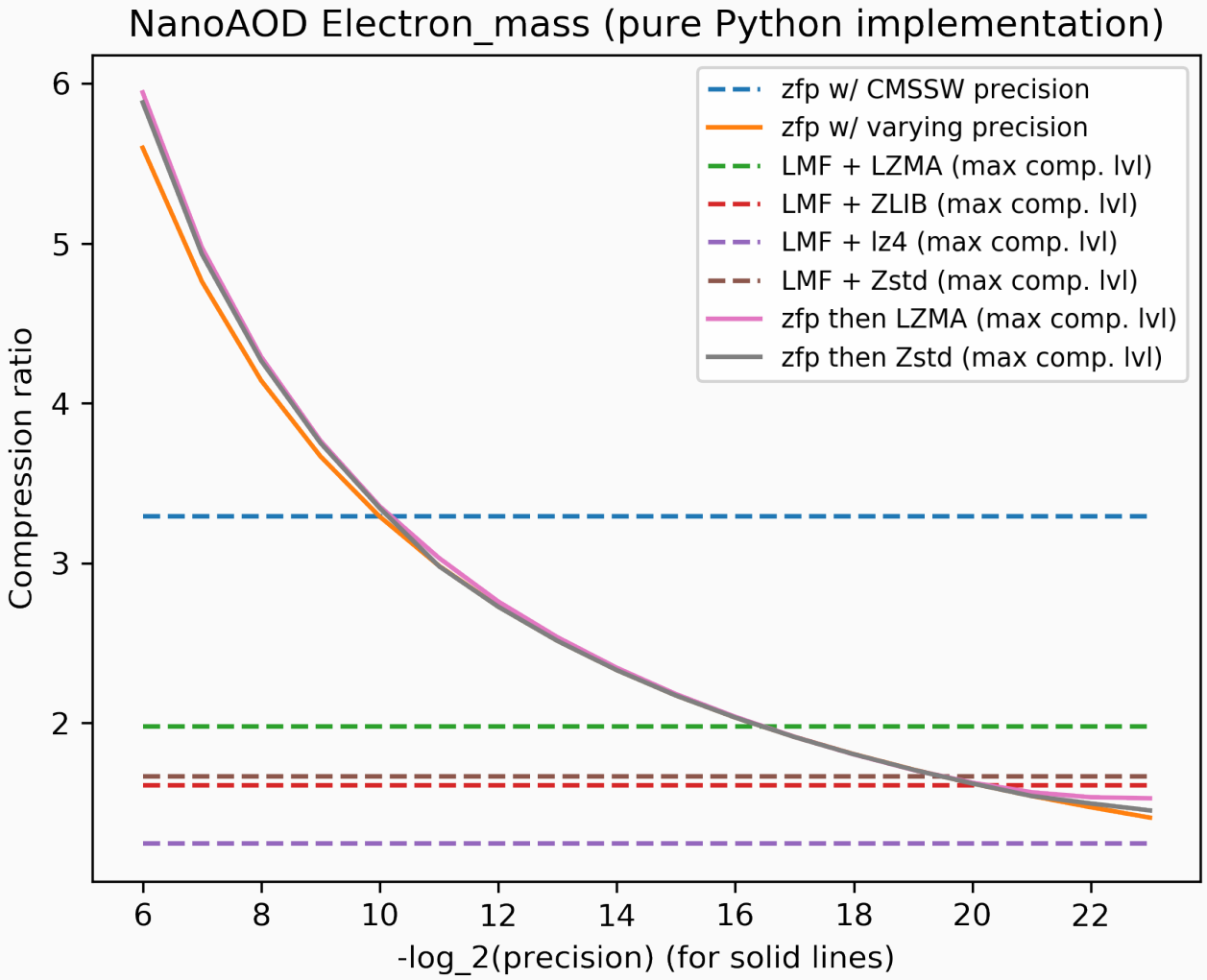
- zfp on data that **is not** compressed by libminifloat
- LZMA, lz4, ZLIB, Zstd on data that **is** already compressed by libminifloat

Small *Pregenerated* NanoAOD CMS NanoAOD:

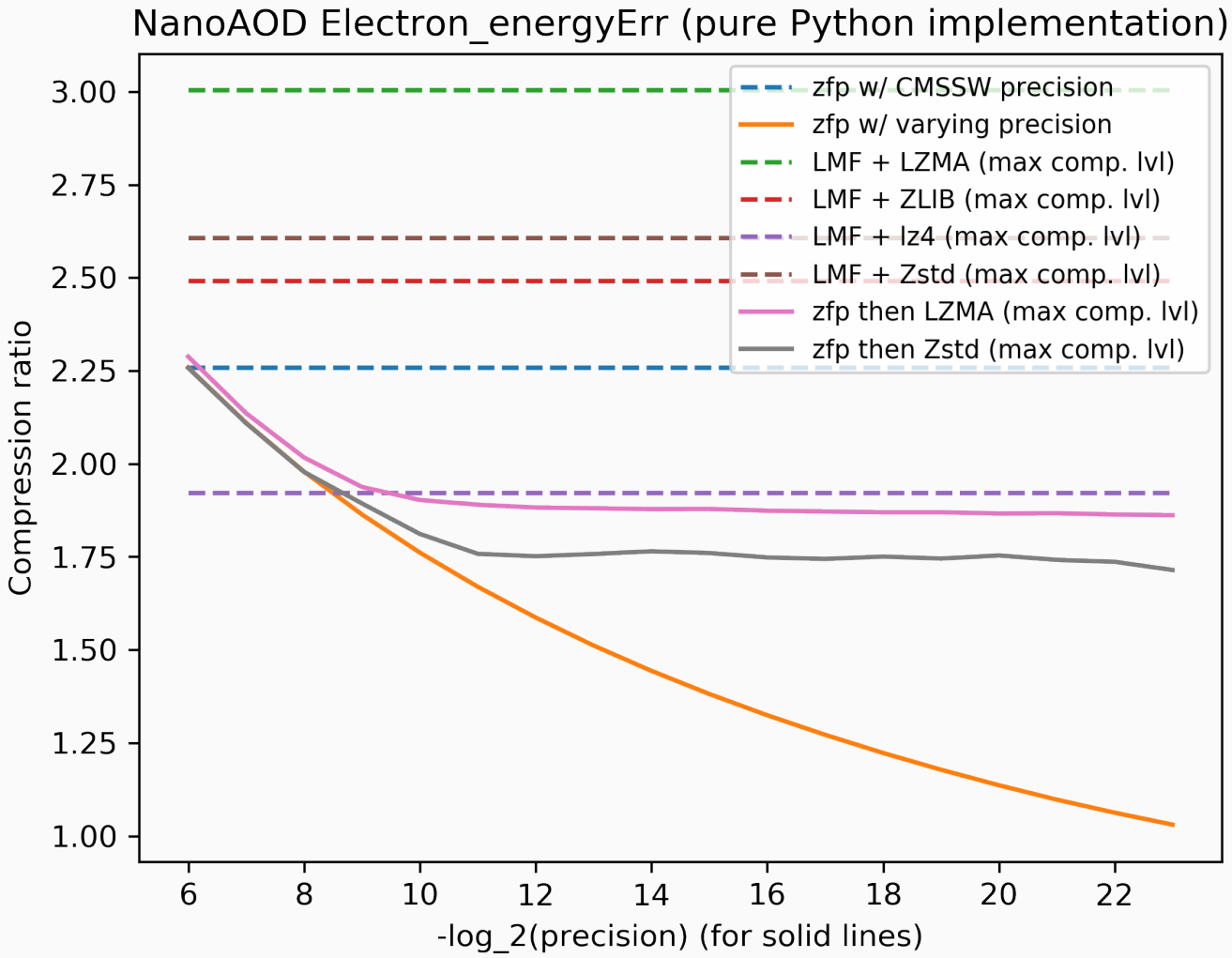
- 9600 events
- 21 MB in compressed size
- 15% of branches (in "Events") hold floats

Following are compression results on various branches:

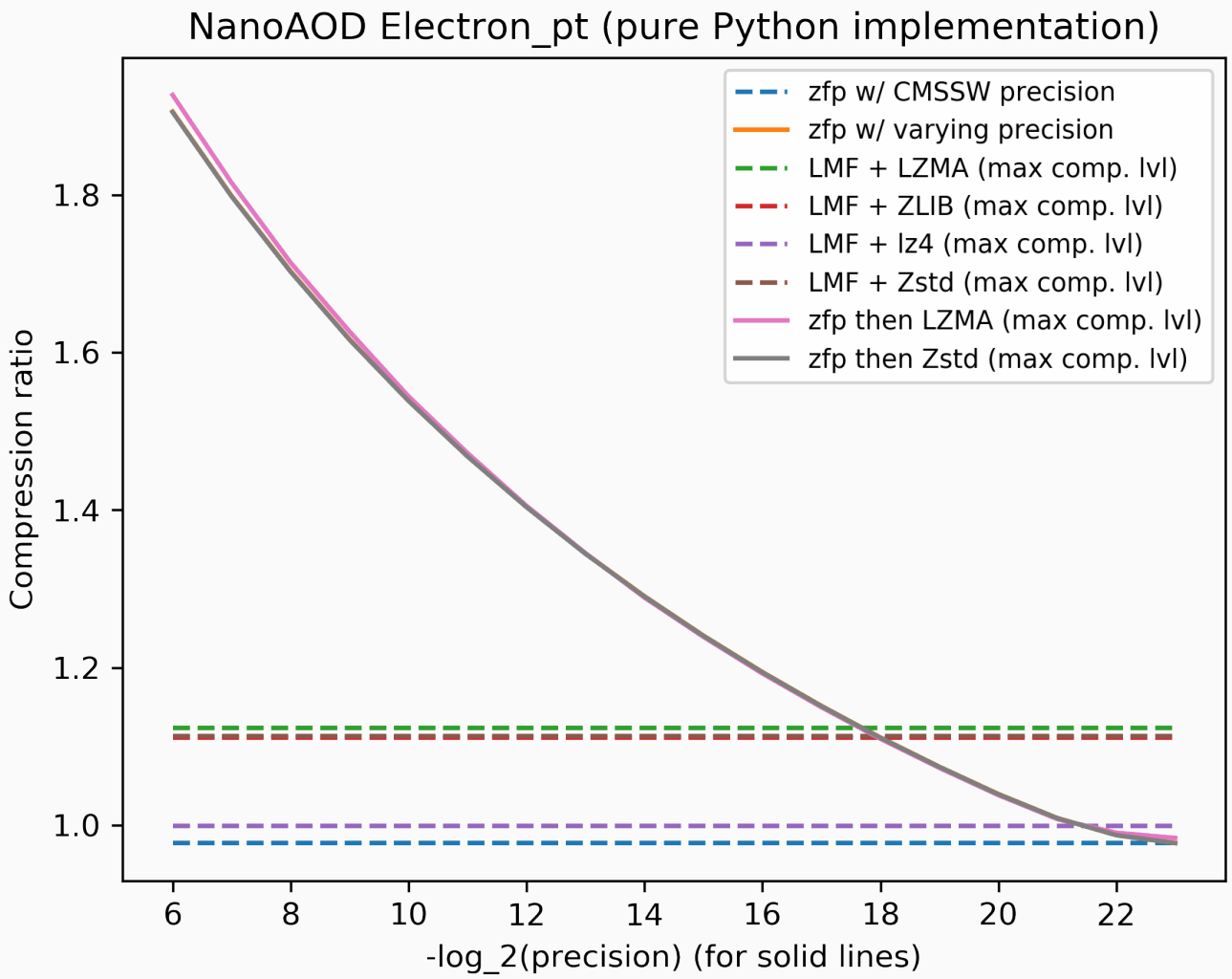
zfp Results


NanoAOD Electron_mass (pure Python implementation)
zfp w/ CMSSW precision
zfp w/ varying precision
LMF + LZMA (max comp. lvl)
LMF + ZLIB (max comp. lvl)
LMF + lz4 (max comp. lvl)
LMF + Zstd (max comp. lvl)
zfp then LZMA (max comp. lvl)
zfp then Zstd (max comp. lvl)
Compression ratio
2
3
4
5
6
6
8
10
12
14
16
18
20
22
-log_2(precision) (for solid lines)

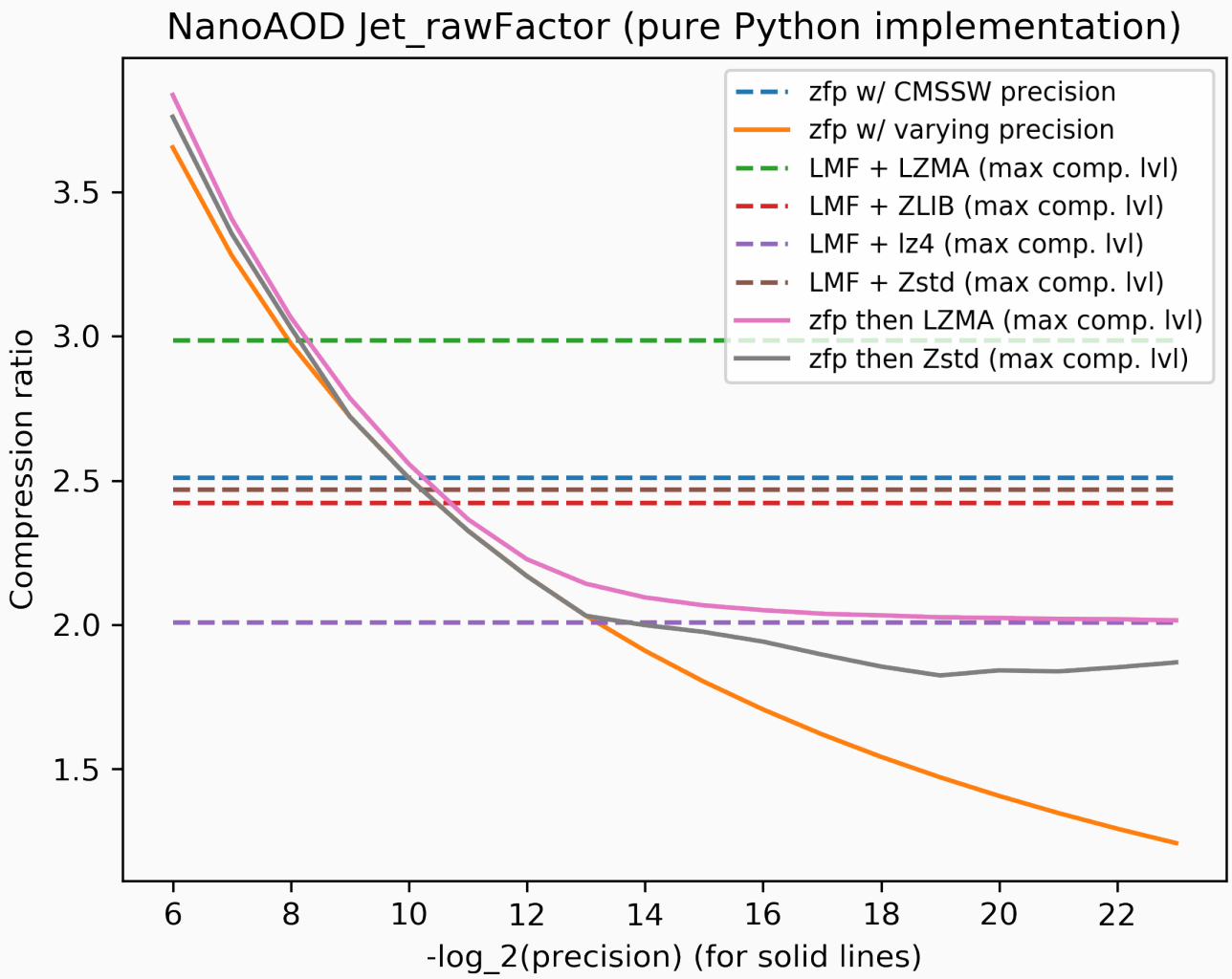
zfp Results


NanoAOD Electron_energyErr (pure Python implementation)
zfp w/ CMSSW precision
zfp w/ varying precision
LMF + LZMA (max comp. lvl)
LMF + ZLIB (max comp. lvl)
LMF + lz4 (max comp. lvl)
LMF + Zstd (max comp. lvl)
zfp then LZMA (max comp. lvl)
zfp then Zstd (max comp. lvl)
Compression ratio
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
6
8
10
12
14
16
18
20
22
-log_2(precision) (for solid lines)

zfp Results


NanoAOD Electron_pt (pure Python implementation)
zfp w/ CMSSW precision
zfp w/ varying precision
LMF + LZMA (max comp. lvl)
LMF + ZLIB (max comp. lvl)
LMF + lz4 (max comp. lvl)
LMF + Zstd (max comp. lvl)
zfp then LZMA (max comp. lvl)
zfp then Zstd (max comp. lvl)
Compression ratio
1.0
1.2
1.4
1.6
1.8
6
8
10
12
14
16
18
20
22
-log_2(precision) (for solid lines)

zfp Results


NanoAOD Jet_rawFactor (pure Python implementation)
zfp w/ CMSSW precision
zfp w/ varying precision
LMF + LZMA (max comp. lvl)
LMF + ZLIB (max comp. lvl)
LMF + lz4 (max comp. lvl)
LMF + Zstd (max comp. lvl)
zfp then LZMA (max comp. lvl)
zfp then Zstd (max comp. lvl)
Compression ratio
3.5
3.0
2.5
2.0
1.5
6
8
10
12
14
16
18
20
22
-log_2(precision) (for solid lines)

zfp Table of (some) branches

Branch name	Uncompressed (kB)	zfp* + LZMA (kB)	LMF + LZMA size (kB)
Electron_mass	44.6	13.3	22.5
Electron_energyErr	44.6	19.5	14.8
Electron_pt	44.6	45.3	39.7
Jet_rawFactor	372.8	145.8	125.3

*CMSSW precision

zfp Evaluation

zfp can outperform other compression methods, but not consistently.

It may yield *larger* sizes when working with near lossless precision, see Electron_pt.

In Electron_energyErr and Jet_rawFactor we see that poor results from zfp can be somewhat saved if followed by LZMA or Zstd.

zfp Estimating filesize

Let's estimate a whole filesize.

Uncompressed size of all (data) branches: 7.7 MB

1. Compress all float-branches with zfp and then LZMA
2. Compress other branches with LZMA

$\rightarrow$ 2.3 MB in total.

Normal LZMA compression on the same branches (with libminifloat enabled)

$\rightarrow$ 1.8 MB

So zfp + LZMA seems to perform slightly worse than LMF + LZMA for this dataset.

zfp + LZMA performed better than LMF + LZMA in 12% of branches.

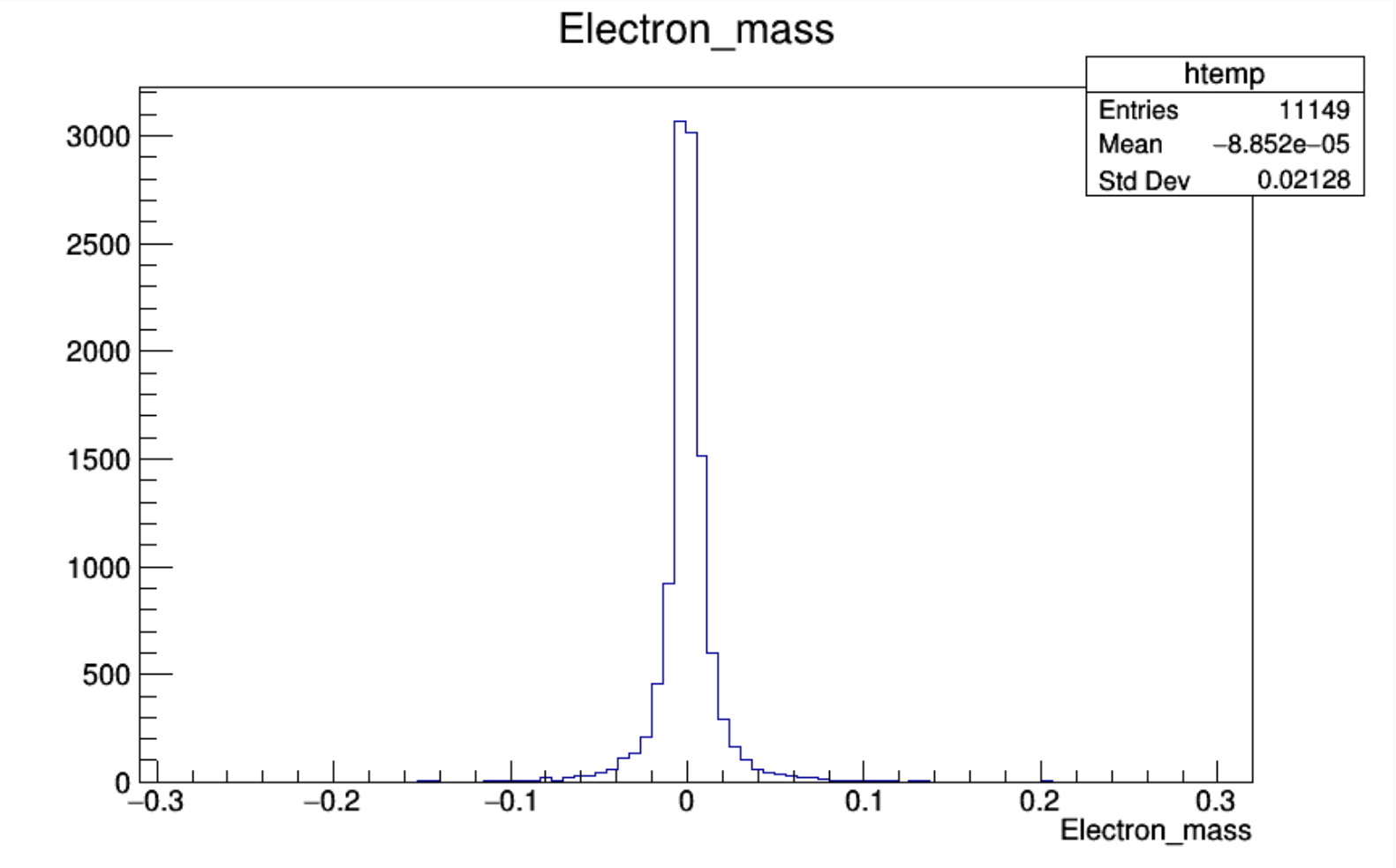
zfp Outlook

A ROOT implementation is being made right now, so that it can actually be tested properly.

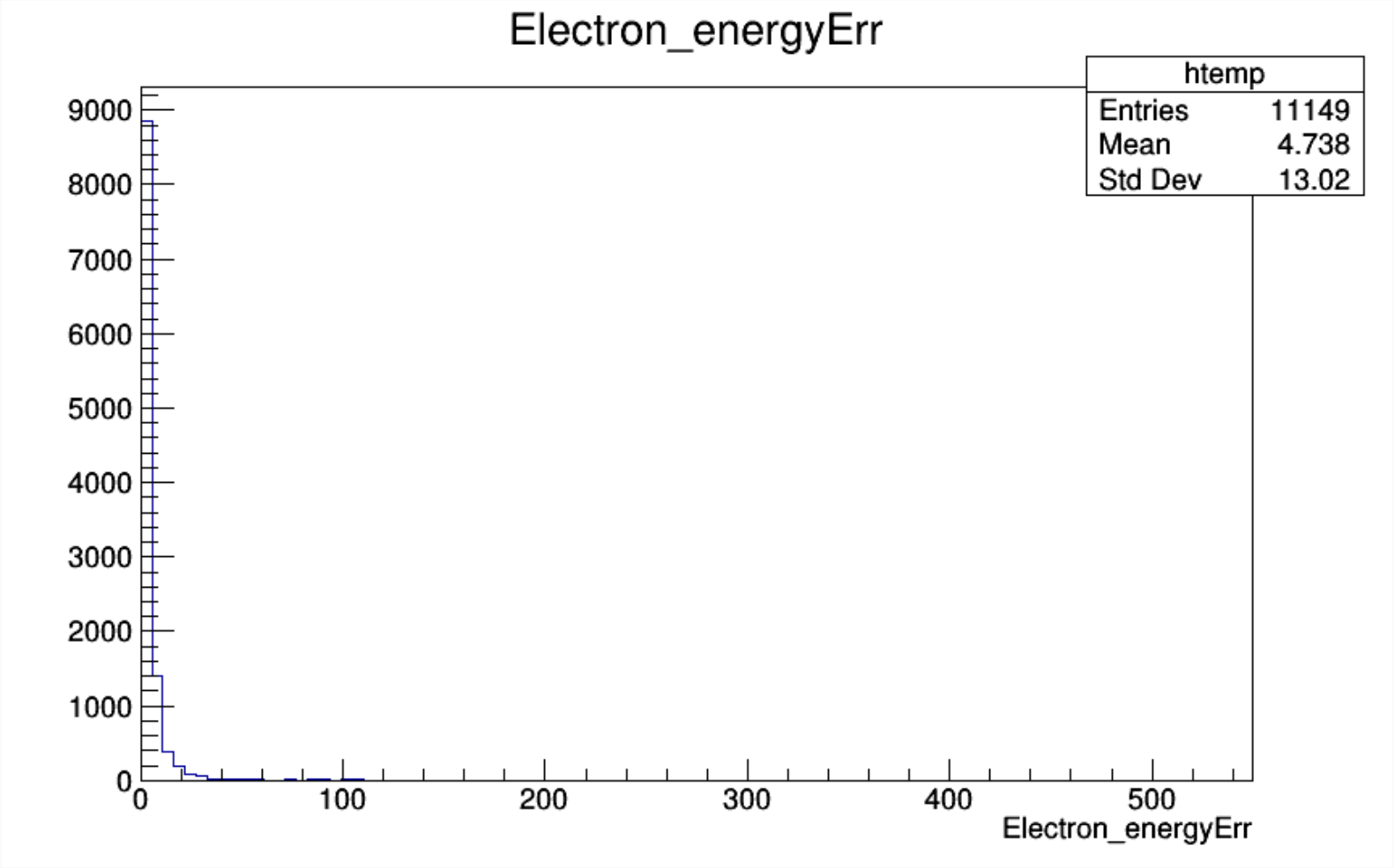
Chain compression methods, e.g. zfp followed by lossless ones. Maybe there is a good combination.

Backup slides

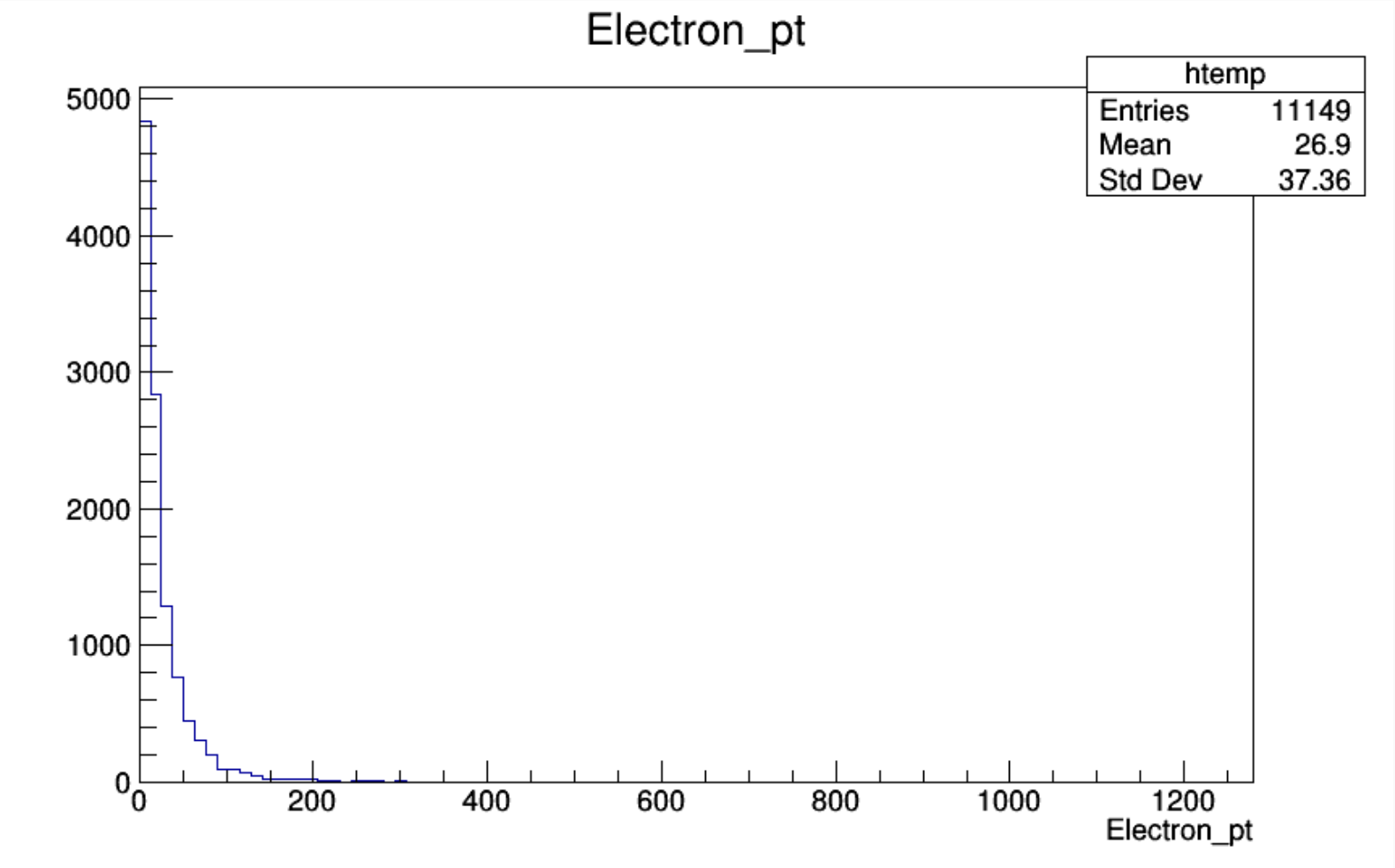
zfp Distributions



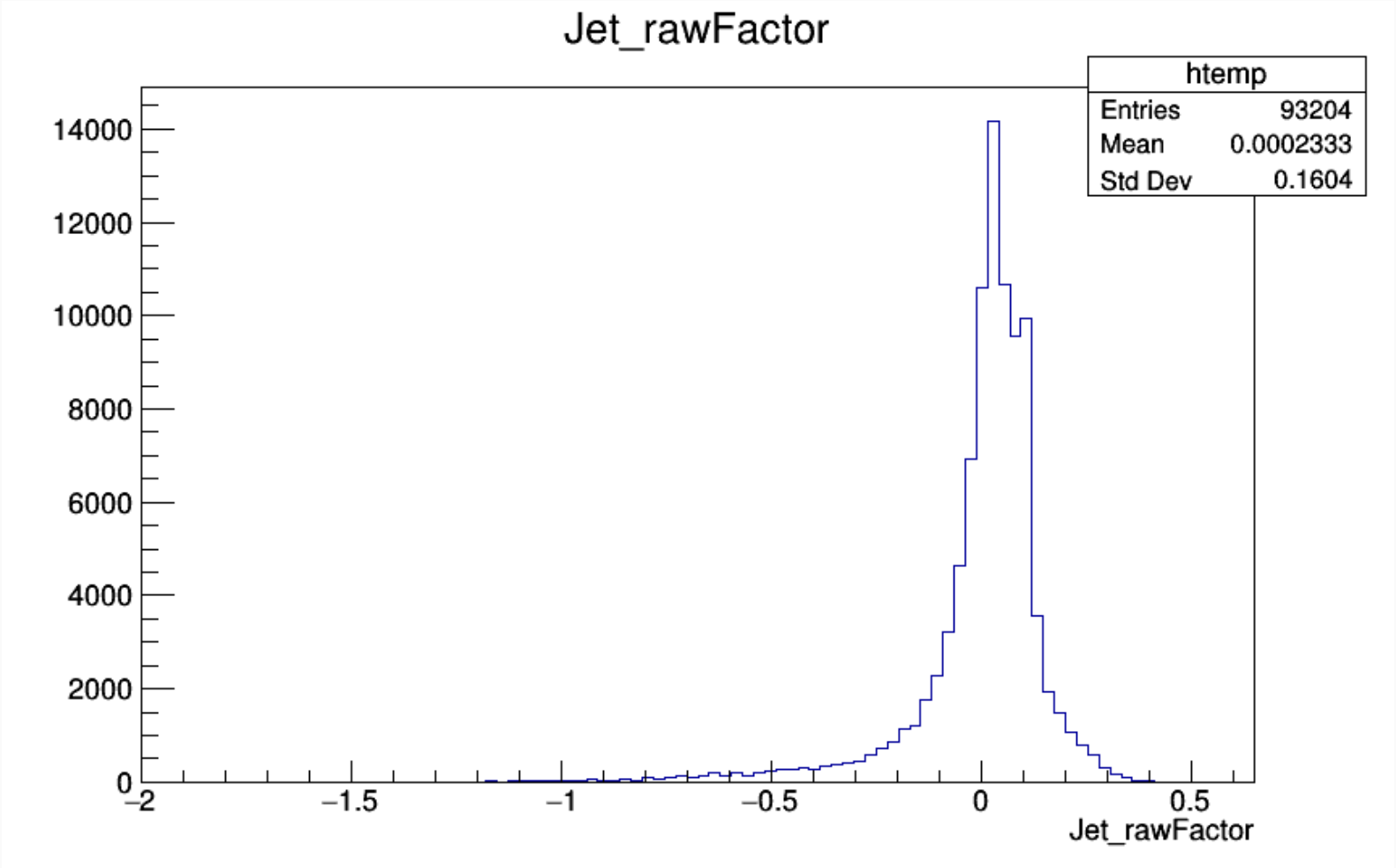
zfp Distributions



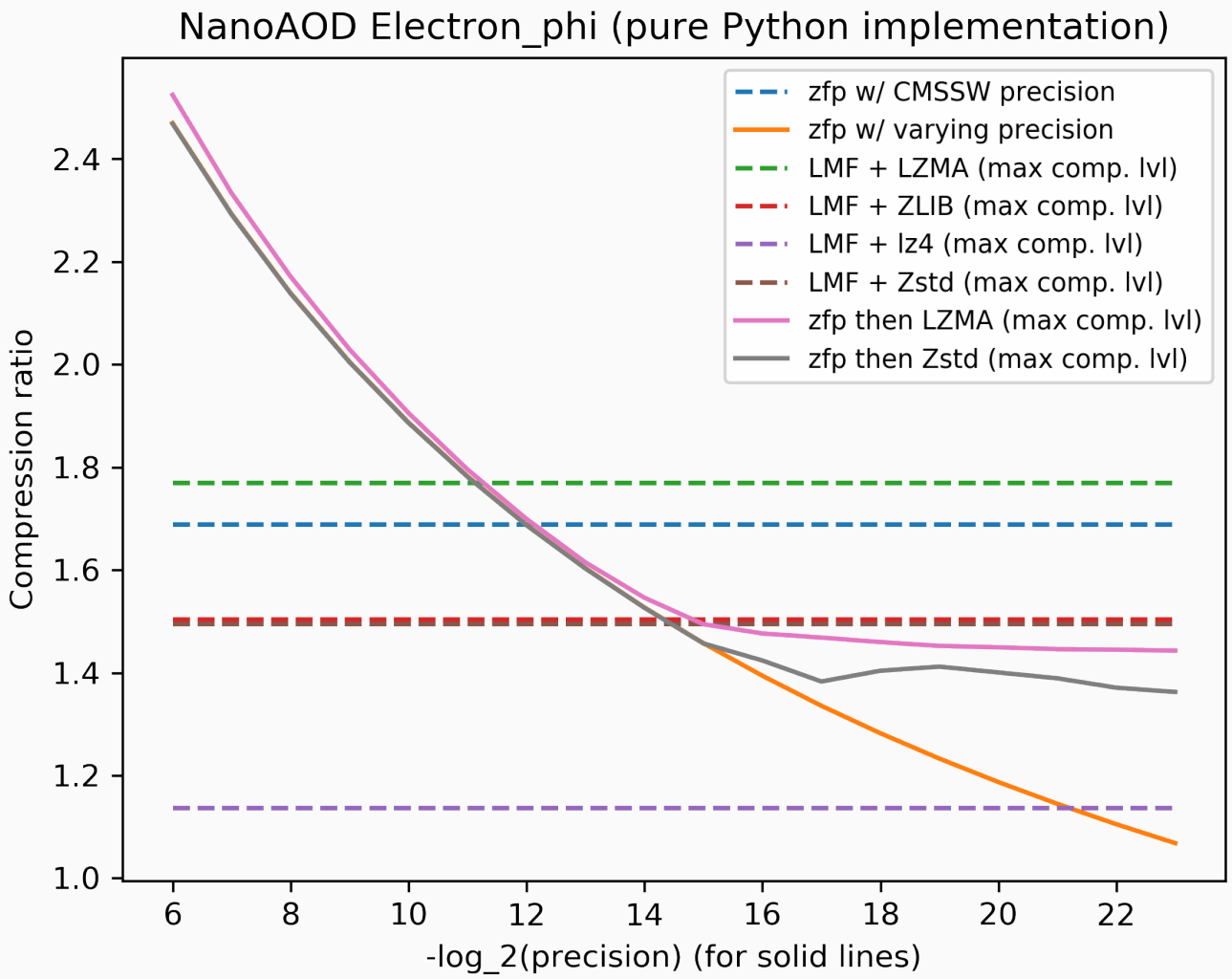
zfp Distributions



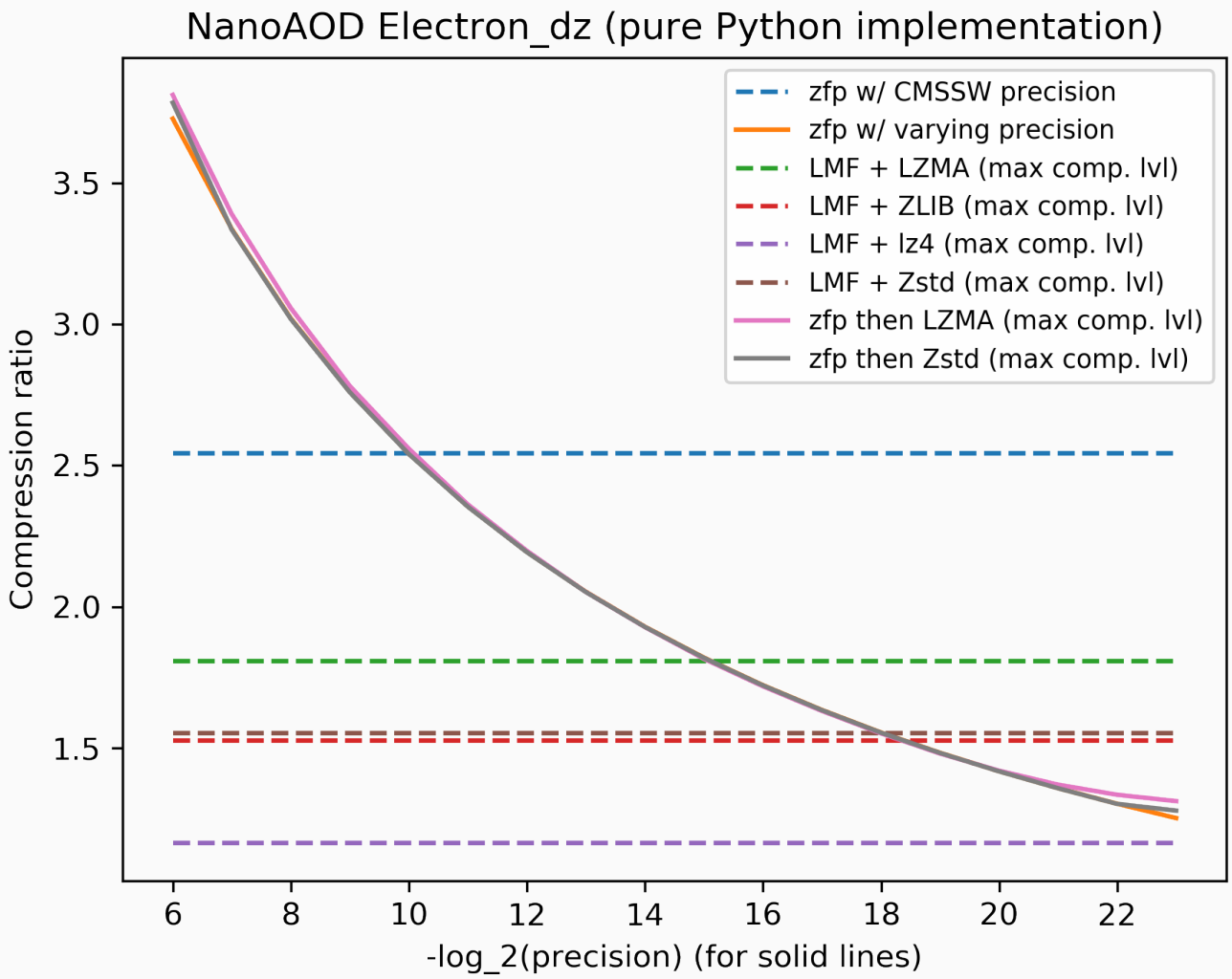
zfp Distributions



zfp Additional results



zfp Additional results



zfp Additional results