Lossless compression

Compression methods supported by HDF5

HDF5 supports

- gzip (deflate), gzip - CLI, zlib - API
- SZIP, n-bit, scale-offset, and shuffling (with deflate) compression filters. SZIP Compression in HDF
- Users defined compression filters Third-party compression filters
  - LZF

gzip

file to file compression

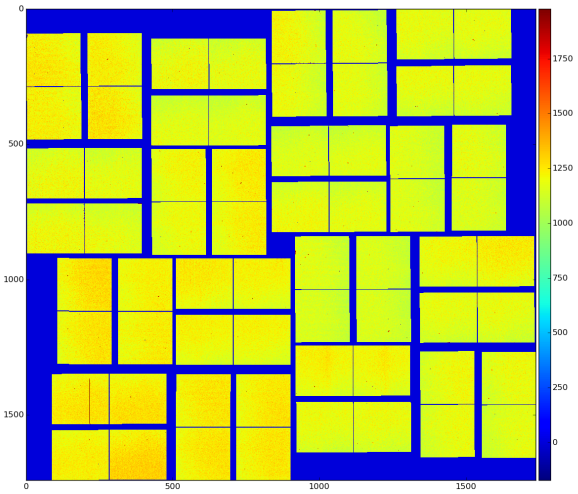
```
gzip -c test.xtc > test.xtc.gz
-rw-r--r-- 1 dubrovin br 168126152 Jan 23 14:48 test.xtc
-rw-r--r-- 1 dubrovin br  88829288 Jan 23 14:51 test.xtc.gz

compression factor = 1.89, time 30sec
```

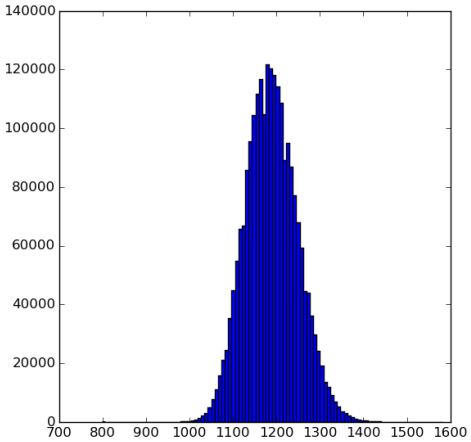
zlib in mempry compress/decompress single cspad image

exp=cxitut13:run=10 event 11:

Array entropy is evaluated using formula from Entropy (information_theory).

Raw data



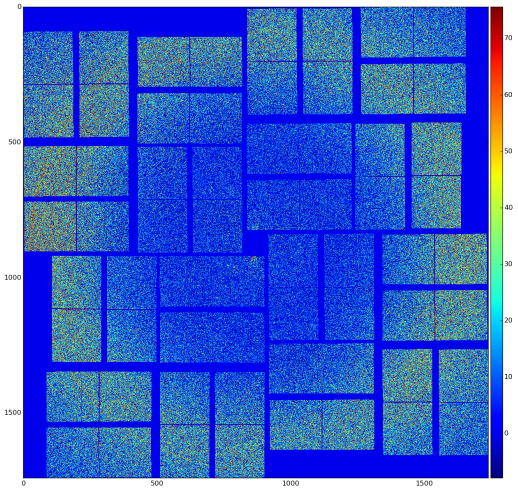


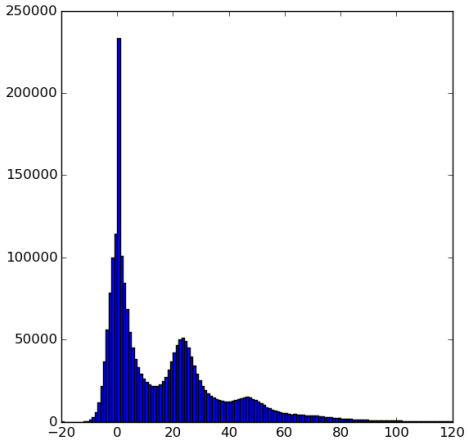
Results for raw CSPAD data

```
Load data from file nda-cxitut13-r0010-e000011-raw.npy
raw data:  shape:(32, 185, 388)  size:2296960  dtype:int16 [1028 1082 1101 1072 1131]...

Array entropy H(16-bit) = 7.947  H(8-bit) = 5.080  H(cpo) = 7.947

zlib level=0:  data size (bytes) in/out = 4593957/4594663 = 1.000  time(sec)=0.025749  t(decomp)=0.005665
zlib level=1:  data size (bytes) in/out = 4593957/2922633 = 1.572  time(sec)=0.108629  t(decomp)=0.026618
zlib level=2:  data size (bytes) in/out = 4593957/2908156 = 1.580  time(sec)=0.125363  t(decomp)=0.029112
zlib level=3:  data size (bytes) in/out = 4593957/2884917 = 1.592  time(sec)=0.170814  t(decomp)=0.027699
zlib level=4:  data size (bytes) in/out = 4593957/2886850 = 1.591  time(sec)=0.158719  t(decomp)=0.029466
zlib level=5:  data size (bytes) in/out = 4593957/2885665 = 1.592  time(sec)=0.261296  t(decomp)=0.030550
zlib level=6:  data size (bytes) in/out = 4593957/2834066 = 1.621  time(sec)=0.597133  t(decomp)=0.027355
zlib level=7:  data size (bytes) in/out = 4593957/2828951 = 1.624  time(sec)=0.609569  t(decomp)=0.026842
zlib level=8:  data size (bytes) in/out = 4593957/2828951 = 1.624  time(sec)=0.636173  t(decomp)=0.027226
zlib level=9:  data size (bytes) in/out = 4593957/2828951 = 1.624  time(sec)=0.611562  t(decomp)=0.027042
```

Calibrated data

calibrated data were obtained using det.calib(...) method, which essentially subtracts pedestals and apply common mode correction to raw data

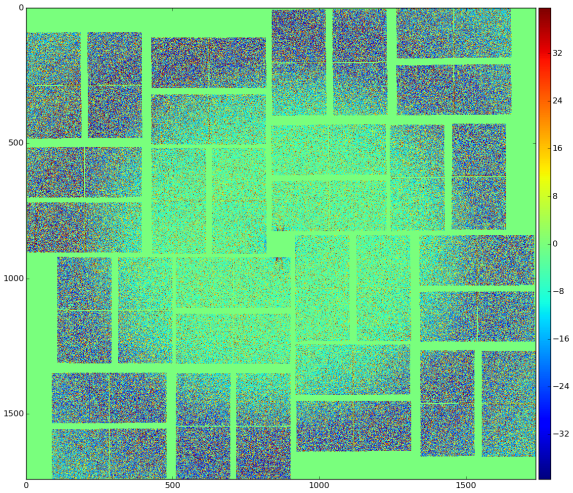


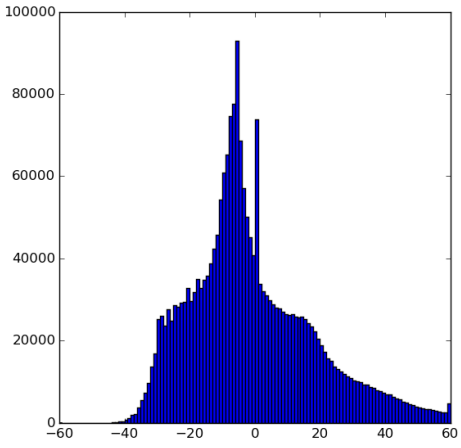


Results for pedestal subtracted cspad data

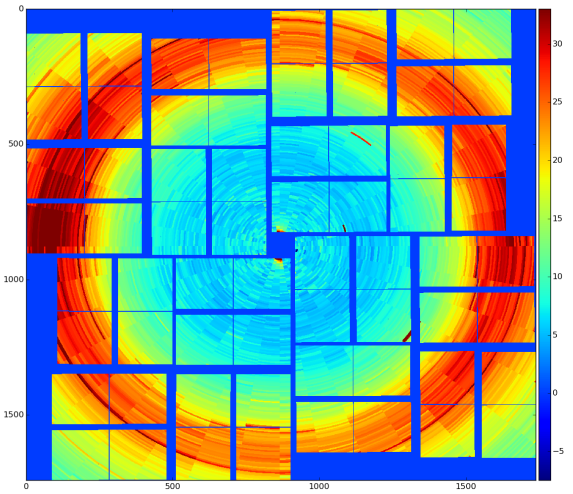
```
Array entropy H(16-bit) = 5.844  H(8-bit) = 3.951  H(cpo) = 5.844

zlib level=0:  data size (bytes) in/out = 4593957/4594663 = 1.000  time(sec)=0.249886  t(decomp)=0.176141
zlib level=1:  data size (bytes) in/out = 4593957/2261808 = 2.031  time(sec)=0.085962  t(decomp)=0.020319
zlib level=2:  data size (bytes) in/out = 4593957/2240202 = 2.051  time(sec)=0.101097  t(decomp)=0.019582
zlib level=3:  data size (bytes) in/out = 4593957/2187212 = 2.100  time(sec)=0.152278  t(decomp)=0.048836
zlib level=4:  data size (bytes) in/out = 4593957/2183572 = 2.104  time(sec)=0.142880  t(decomp)=0.049567
zlib level=5:  data size (bytes) in/out = 4593957/2242648 = 2.048  time(sec)=0.308234  t(decomp)=0.022228
zlib level=6:  data size (bytes) in/out = 4593957/2217193 = 2.072  time(sec)=0.677328  t(decomp)=0.020837
zlib level=7:  data size (bytes) in/out = 4593957/2205357 = 2.083  time(sec)=0.975548  t(decomp)=0.023253
zlib level=8:  data size (bytes) in/out = 4593957/2195009 = 2.093  time(sec)=1.581390  t(decomp)=0.023262
zlib level=9:  data size (bytes) in/out = 4593957/2193889 = 2.094  time(sec)=1.802204  t(decomp)=0.020965
```

Calibrated and radial-background subtracted data





binned background shape:



Results for background-subtracted data

```
Array entropy H(16-bit) = 6.280  H(8-bit) = 4.487  H(cpo) = 6.280

zlib level=0:  data size (bytes) in/out = 4593957/4594663 = 1.000  time(sec)=0.035164  t(decomp)=0.007174
zlib level=1:  data size (bytes) in/out = 4593957/2322746 = 1.978  time(sec)=0.137170  t(decomp)=0.019560
zlib level=2:  data size (bytes) in/out = 4593957/2310816 = 1.988  time(sec)=0.090709  t(decomp)=0.019657
zlib level=3:  data size (bytes) in/out = 4593957/2270123 = 2.024  time(sec)=0.137816  t(decomp)=0.023169
zlib level=4:  data size (bytes) in/out = 4593957/2257567 = 2.035  time(sec)=0.113220  t(decomp)=0.027111
zlib level=5:  data size (bytes) in/out = 4593957/2323615 = 1.977  time(sec)=0.345213  t(decomp)=0.022773
zlib level=6:  data size (bytes) in/out = 4593957/2312382 = 1.987  time(sec)=0.708425  t(decomp)=0.022472
zlib level=7:  data size (bytes) in/out = 4593957/2307002 = 1.991  time(sec)=0.935245  t(decomp)=0.023992
zlib level=8:  data size (bytes) in/out = 4593957/2304653 = 1.993  time(sec)=1.201955  t(decomp)=0.022417
zlib level=9:  data size (bytes) in/out = 4593957/2304574 = 1.993  time(sec)=1.215707  t(decomp)=0.022277
```

Entropy of low and high bytes

Results for CSPAD image with swapped bytes

```
load_nda_from_file:
Data from file nda-cxitut13-r0010-e000011-raw.npy:  shape:(32, 185, 388)  size:2296960  dtype:int16 [1028 1082
1101 1072 1131]...
H(8-bit) = 5.080
nda8 :  shape:(2296960, 2)  size:4593920  dtype:uint8 [  4   4  58   4  77   4  48   4 107   4]...

# split data array for two with even and odd bytes:
nda8L:  shape:(2296960,)    size:2296960  dtype:uint8 [  4  58  77  48 107  73  45 103  28  89]...
nda8H:  shape:(2296960,)    size:2296960  dtype:uint8 [4 4 4 4 4 4 4 4 4 4]...
H(low -byte) = 7.821
H(high-byte) = 0.376
```

Compression in HDF5

GZIP

"A number of compression filters are available in HDF5. By far the most commonly used is the GZIP filter. "

```
dset = f.create_dataset("BigDataset", shape=(32,185,388), dtype=np.int16, chunks=(1,185,388), compression="
gzip")
>>> dset.compression
'gzip'
>>> dset.compression_opts
9
```

GZIP features:

- Works with all HDF5 types
- Built into HDF5 and available everywhere
- Moderate to slow speed compression
- Performance can be improved by also using SHUFFLE

SZIP

"SZIP is a patented compression technology used extensively by NASA. Generally you only have to worry about this if you’re exchanging files with people who use satellite data. Because of patent licensing restrictions, many installations of HDF5 have the decompressor, but compressor is disabled."

```
dset= myfile.create_dataset("Dataset3", (1000,), compression="szip")
```

SZIP features:

- Integer (1, 2, 4, 8 byte; signed/unsigned) and floating-point (4/8 byte) types only
- Fast compression and decompression
- A decompressor that is almost always available

LZF

Lempel-Ziv dynamic dictionary compression

LibLZF by Marc Lehmann is designed to be a very small, very fast, very portable data compression library for the LZF compression algorithm.

"For files you'll only be using from Python, LZF is a good choice. It ships with h5py; C source code is available for third-party programs under the BSD license. It's optimized for very, very fast compression at the expense of a lower compression ratio compared to GZIP. The best use case for this is if your dataset has large numbers of redundant data points."

```
dset = myfile.create_dataset("Dataset4", shape=(32,185,388), dtype=np.int16, chunks=(1,185,388),compression="
lzf")
```

LZF features:

- Works with all HDF5 types
- Fast compression and decompression
- Is only available in Python (ships with h5py); C source available

Extra filters in HDF5

SHUFFLE

Treats low and high bytes separately

```
>>> dset = myfile.create_dataset("Data", shape=(32,185,388), dtype=np.int16, chunks=(1,185,388), compression="
gzip",
shuffle=True)
```

SHUFFLE features:

- Available with all HDF5 distributions
- Very fast (negligible compared to the compression time)
- Only useful in conjunction with filters like GZIP or LZF

FLETCHER32 Filter

Check-sum

```
dset = myfile.create_dataset("Data2", shape=(32,185,388), dtype=np.int16, chunks=(1,185,388), fletcher32=True,
...)
>>> dset.fletcher32
True
```

FLETCHER32 features:

- Available with all HDF5 distributions
- Very fast
- Compatible with all lossless filters

gzip, szip, lzf compression results

```
gzip default compression_opts level=4
  raw:  gzip  t1(create)=0.003280(sec)  t2(+save)=0.216324(sec)  input size=4594000(byte)  ratio=1.583
shuffle=False  fletcher32=False
  raw:  gzip  t1(create)=0.003025(sec)  t2(+save)=0.146706(sec)  input size=4594000(byte)  ratio=1.958
shuffle=True   fletcher32=False

calib:  gzip  t1(create)=0.002738(sec)  t2(+save)=0.168040(sec)  input size=4594000(byte)  ratio=2.072
shuffle=False  fletcher32=False
calib:  gzip  t1(create)=0.002926(sec)  t2(+save)=0.178174(sec)  input size=4594000(byte)  ratio=2.188
shuffle=True   fletcher32=False
calib:  gzip  t1(create)=0.002579(sec)  t2(+save)=0.182965(sec)  input size=4594000(byte)  ratio=2.187
shuffle=True   fletcher32=True


calib:  lzf  t1(create)=0.003225(sec)  t2(+save)=0.100822(sec)  input size=4594000(byte)  ratio=1.351
shuffle=False  fletcher32=False
calib:  lzf  t1(create)=0.002815(sec)  t2(+save)=0.086916(sec)  input size=4594000(byte)  ratio=1.473  shuffle=
True  fletcher32=False
  raw:  lzf  t1(create)=0.003125(sec)  t2(+save)=0.108339(sec)  input size=4594000(byte)  ratio=1.045
shuffle=False  fletcher32=False
  raw:  lzf  t1(create)=0.003071(sec)  t2(+save)=0.075530(sec)  input size=4594000(byte)  ratio=1.698  shuffle=
True  fletcher32=False

Compression filter "szip" is unavailable
Compression filter "lzo" is unavailable
Compression filter "blosc" is unavailable
Compression filter "bzip2" is unavailable
```

Igor's compressor

https://pswww.slac.stanford.edu/svn-readonly/psdmrepo/

Compressor designated for LCLS detector uint16 data:

1. estimates dataset spread,
2. use 16-bit and 8-bit words to save data.

Features

- Optimized to work with 16-bit detector data only (not with xtc or hdf5 files containing metadata).
- By design Hist16 compression factor 2.
- Single array of data is split and processed in multi-threads (inside compression algorithm).
- Igor statement: up to ~two order of magnitude faster than gzip.
- Igor thinks that further specialization of data (separation of signal and background regions between threads) may improve compression factor.

Matt's Hist16 and HistN compressors

Available in external package `pdsdata/compress/`

1. Hist16 - the same as Igor's compressor, **but does not use multi-threading** - slow
2. HistN - developed by Matt, uses 16-bit and 8,7,6...-bit words, compression factor HistN upto ~2.

SZ compressor from Argonne

https://github.com/disheng222/SZ

-> Clone or download -> Download ZIP -> installed under ~/lib/sz/sz-1.4.9/

Run tests like:

~/lib/sz/sz-1.4.9/SZ-master/example]$ ./testfloat_compress sz.config testdata/x86/testfloat_8_8_128.dat 8 8 128

- works with float and double.
- int16 and uint16 not implemented

compression factors ~ 56, 110, and 49 for
- testfloat_8_8_128.dat,
- testdouble_8_8_128.dat, and
- testdouble_8_8_8_128.dat, respectively.

But for data with VERY NARROW SPECTRA:

```
 testfloat_8_8_128.txt           mean=1.000000  std=1.232407
testdouble_8_8_128.txt       mean=1.000000  std=1.254261
testdouble_8_8_8_128.txt  mean=1.300935   std=0.502083
```

References

- Using compression in HDF5
- Szip Compression in HDF
- Third-party compression filters
- HDF5 Tutorial
- HDF5 Software Documentation
- Using HDF5 filters
- HDF5 Data Compression Demystified
- gzip - CLI, zlib - API
- Entropy (information_theory)
- Dictionary compression
- Python and HDF5
- LibLZF
- SZ compressor (sz-1.4-user-guide.pdf) Authors: Sheng Di, Dingwen Tao, supervisor: Franck Cappello
- 2017-02-24-Report-of-SZ-Lossy-Compression-on-EXAFEL-Datasets-v5.pdf
- 2013-04-17-igor-pyana_xtc_decompression_status.pdf, psdmrepo
- 2017-03-21-Lossless-compression.pdf
- 2017-09-27-Lossless-compression.pdf - on Users mtg