mrcz

Release 0.5.6

Mar 25, 2020

Contents

Contents:

CHAPTER 1

Introduction

MRCZ is a union of the MRC file format with `blosc` meta-compression. `blosc` is not a compression algorithm, rather it is a standard that supports most popular compression algorithms. It can also apply lossless filters that improve compression performance, such as the `bitshuffle` filter. It achieves high-performance through the use of multi-threading the supported compression codecs. Generally you should expect MRCZ to result in faster file read/write rates, as the compression is faster than hard drive read/write rates, as well as near entropy-limited compression ratios. So you get something for nothing.

Typical usage patterns are:

```
imageData, imageMeta = mrcz.readMRC('my_filename.mrcz')
```

where `imageData` is a `numpy.ndarray` and `imageMeta` is a Python `dict` containing metadata. After some manipulation, you may want to then save to disk so the file can be passed into a third-party application, such as a CTF estimation tool. Here for maximum compatibility we will save it uncompressed (which is the default keyword argument for `compressor`):

```
mrcz.writeMRC( imageData, 'passed_file.mrc', compressor=None )
```

Alternatively you may want to save an archival compression version of your data in the background using the asynchronous feature. In this case, the exact time when the write finishes is typically not a concern (although see the the function documentation for finer control):

```
mrcz.asyncWriteMRC( imageData, 'my_newfile.mrcz', meta=newMeta, compressor='zstd',␣
↪clevel=1 )
```

See the API reference docs for detailed information on usage. The recommended compression codecs and levels are:

- `compressor='zstd'` and `clevel=1` for general archival use.
- `compressor='lz4'` and `clevel=9` for speed-critical applications.

The `bitshuffle` filter is always used in MRCZ compressed files as it was found to improve both compression rate and ratio with representative electron microscopy data.

CHAPTER 2

MRCZ Specification

In general MRCZ follows the CCPEM MRC2014 standard as outlined here:

http://www.ccpem.ac.uk/mrc_format/mrc2014.php

Please note we count bytes starting from `0`. CCP-EM counts bytes starting from `1`.

2.1 Required deviations from CCPEM MRC2014 standard

1. **Word 4 (@ byte 16)**: The **MODE** parameter is now the sum of the MRC2014 **MODE** plus the `blosc` compression used * 1000.

   The compressor enumeration is:

   ```
   { 0:None, 1:'blosclz', 2:'lz4', 3:'lz4hc', 4:'snappy', 5:'zlib', 6:'zstd' }
   ```

   Unpacking is generally performed as follows:

   ```
   mrcMode = numpy.mod(mrczMode, 1000)
   compressor = numpy.floor_divide(mrczMode, 1000)
   ```

   In practice any **MODE** > 1000 indicates the use of a compression codec. `blosc` will discover the actual codec used itself.

2. In the case where `compressor != None`, starting at byte 1024 (or 1024 + **EXTRA** if the extended header is used) a `c-blosc` header is found. The `c-blosc` header format specification may be found here:

   https://github.com/Blosc/c-blosc/blob/master/README_HEADER.rst

   `blosc` is limited to `2**31` bytes per chunk. Chunking for compression is accomplished by compressing each slice/frame in the z-axis with a separate call to `blosc.compress()`. Therefore the data section consists of **NZ** structs of `c-blosc` headers followed by the packed bytes for the associated slice/frame.

2.2 Optional deviations from CCPEM MRC2014 standard

1. **Word 33 (@ byte 132)**: Accelerating voltage in keV, float-32 format. **Deprecated**.
2. **Word 34 (@ byte 136)**: Spherical aberration in mm, float-32 format. **Deprecated**.
3. **Word 35 (@ byte 140)**: Detector gain in e^-/DN, defaults to 1.0. **Deprecated**.
4. **Word 36-37 (@ byte 14)**: Size of compressed data in bytes stored as a 64-bit integer, including `blosc` headers. Present for convenience only.
5. **Word 57 (@ byte 224)**: The ascii-encoded identifier label 'MRCZ<version>'. For example, ''b'MRCZ0.3.1'.

Failure to include any of these variables will not result in an exception.

2.2.1 JSON extended meta-data

When the keyword argument `meta` is used with `writeMRC` and `asyncWriteMRC` the passed dictionary will be converted to UTF-8 encoded JSON and written into the extended header. This is indicated by the ascii-encoded bytes `'json'` written into the **EXTTYP** variable of the MRC2014 header. The length of the encoded JSON metadata is stored in the **EXTRA** variable of the MRC2014 header.

Note: `python-rapidjson` is preferred but the standard library `json` module is used as a fallback.

CHAPTER 3

Frequently Asked Questions

Can I access individual slices in a compressed MRCZ file?

Currently not, as `blosc` does not record the indices of its individual compressed blocks. The new feature of 'super-chunks', equivalent to slices or frames in the context of microscopy, is expected to be implemented in `c-blosc2`, currently under development at https://github.com/Blosc/c-blosc2

CHAPTER 4

API Reference

mrcz.**readMRC**(*MRCfilename*, *idx=None*, *endian='le'*, *pixelunits='\AA'*, *fileConvention='ccpem'*, *useMemmap=False*, *n_threads=None*, *slices=None*)

Imports an MRC/Z file as a NumPy array and a meta-data dict.

Parameters

image: numpy.ndarray a 1-3 dimension `numpy.ndarray` with one of the supported data types in `mrcz.REVERSE_CCPEM_ENUM`

meta: dict a `dict` with various fields relating to the MRC header information. Can also hold arbitrary meta-data, but the use of large numerical data is not recommended as it is encoded as text via JSON.

idx: Tuple[int] Index tuple `(first, last)` where first (inclusive) and last (not inclusive) indices of images to be read from the stack. Index of first image is 0. Negative indices can be used to count backwards. A singleton integer can be provided to read only one image. If omitted, will read whole file. Compression is currently not supported with this option.

pixelunits: str can be `'AA' (Angstoms), 'nm', '\mum', or 'pm'`. Internally pixel sizes are always encoded in Angstroms in the MRC file.

fileConvention: str can be `'ccpem'` (equivalent to IMOD) or `'eman2'`, which is only partially supported at present.

endian: str can be big-endian as `'be'` or little-endian as `'le'`. Defaults to *'le'* as the vast majority of modern computers are little-endian.

n_threads: int is the number of threads to use for decompression, defaults to use all virtual cores.

useMemmap: bool = True returns a `numpy.memmap` instead of a `numpy.ndarray`. Not recommended as it will not work with compression.

slices: Optional[int] = None Reflects the number of slices per frame. For example, in time-series with multi-channel STEM, would be `4` for a 4-quadrant detector. Data is always written contiguously in MRC, but will be returned as a list of `[slices, *shape]`-shaped

arrays. The default option `None` will check for a `'slices'` field in the meta-data and use that, otherwise it defaults to `0` which is one 3D array.

Returns

image: Union[list[numpy.ndarray], numpy.ndarray] If `slices == 0` then a monolithic array is returned, else a `list` of `[slices, *shape]`-shaped arrays.

meta: dict the stored meta-data in a dictionary. Note that arrays are generally returned as lists due to the JSON serialization.

mrcz.**writeMRC**(*input_image, MRCfilename, meta=None, endian='le', dtype=None, pixelsize=[0.1, 0.1, 0.1], pixelunits='\AA', shape=None, voltage=0.0, C3=0.0, gain=1.0, compressor=None, clevel=1, n_threads=None, quickStats=True, idx=None*)

Write a conventional MRC file, or a compressed MRCZ file to disk. If compressor is `None`, then backwards compatibility with other MRC libraries should be preserved. Other libraries will not, however, recognize the JSON extended meta-data.

Parameters

input_image: Union[numpy.ndarray, list[numpy.ndarray]] The image data to write, should be a 1-3 dimension `numpy.ndarray` or a list of 2-dimensional ‘‘numpy.ndarray‘‘s.

meta: dict will be serialized by JSON and written into the extended header. Note that `rapidjson` (the default) or `json` (the fallback) cannot serialize all Python objects, so sanitizing `meta` to remove non-standard library data structures is advisable, including `numpy.ndarray` values.

dtype: Union[numpy.dtype, str] will cast the data before writing it.

pixelsize: Tuple[x,y,z] is [z,y,x] pixel size (singleton values are ok for square/cubic pixels)

pixelunits: str = u'AA' one of - `'\AA'` for Angstroms - `'pm'` for picometers - `'\mum'` for micrometers - `'nm'` for nanometers. MRC standard is always Angstroms, so pixelsize is converted internally from nm to Angstroms as needed.

shape: Optional[Tuple[int]] is only used if you want to later append to the file, such as merging together Relion particles for Frealign. Not recommended and only present for legacy reasons.

voltage: float = 300.0 accelerating potential in keV

C3: float = 2.7 spherical aberration in mm

gain: float = 1.0 detector gain in units (counts/primary electron)

compressor: str = None is a choice of `None`, `'lz4'`, `'zlib'`, `'zstd'`, plus `'blosclz'`, `'lz4hc'` - `'lz4'` is generally the fastest. - `'zstd'` generally gives the best compression performance, and is still almost

as fast as ‘lz4’ with `clevel == 1`.

clevel: int = 1 the compression level, 1 is fastest, 9 is slowest. The compression ratio will rise slowly with clevel (but not as fast as the write time slows down).

n_threads: int = None number of threads to use for blosc compression. Defaults to number of virtual cores if `== None`.

quickStats: bool = True estimates the image mean, min, max from the first frame only, which saves computational time for image stacks. Generally strongly advised to be `True`.

idx can be used to write an image or set of images starting at a specific position in the MRC file (which may already exist). Index of first image is 0. A negative index can be used to

count backwards. If omitted, will write whole stack to file. If writing to an existing file, compression or extended MRC2014 headers are currently not supported with this option.

Returns

None

mrcz.**asyncReadMRC**(**args*, ***kwargs*)
Calls *readMRC* in a separate thread and executes it in the background.

Parameters

Valid arguments are as for 'readMRC()'.

Returns

future A `concurrent.futures.Future()` object. Calling `future.result()` will halt until the read is finished and returns the image and meta-data as per a normal call to *readMRC*.

mrcz.**asyncWriteMRC**(**args*, ***kwargs*)
Calls *writeMRC* in a seperate thread and executes it in the background.

Parameters

Valid arguments are as for 'writeMRC()'.

Returns

future A `concurrent.futures.Future` object. If needed, you can call `future.result()` to wait for the write to finish, or check with `future.done()`. Most of the time you can ignore the return and let the system write unmonitored. An exception would be if you need to pass in the output to a subprocess.

class mrcz.**readDM4**(*filename*, *verbose=False*)
A fast DM4 file reader to strip the data out of the large movie-mode files generated by K2 detectors along with important tags. Due to the emphasis on speed it's not a general file parser with dynamic allocation, so if Gatan changes the format a lot it will break the script.

Parameters

filename: str the

verbose: bool whether to output debugging info or not.

discardTag(*self*)
Quickly parse to the end of tag that we don't care about its information

parseTag(*self*, *parent*)
Parse a tag at the given file handle location

parseTagDir(*self*, *parent*)
Parse a tag directory at the given file handle location

retrieveTagData(*self*, *tag_ninfo*)
Get the actual data from the tag, which is then stored in one of the dicts or imageData

mrcz.**test**(*verbosity=2*)
Run `unittest` suite for `mrcz` package.

mrcz.**__version__**
The version of Python MRCZ.

CHAPTER 5

Release Notes

5.1 0.5.6

- In 0.5.5 a for-loop was omitted which lead to every frame being the zeroth frame in the stack. For this reason, upgrading from 0.5.5 is _strongly_ recommended.

5.2 0.5.5

- Meta-data with keys that match those used in the header could accidently overwrite critical values, such as 'dimensions'. Any keys in the JSON meta-dictionary that overlap with the standard values are now ignored.
- Integration tests are now performed for Python 3.8.

5.3 0.5.4

- Added support for serialization of Python Enum objects in JSON serialization of meta-data.

5.4 0.5.3

- Ricardo Righetto added the means to append frames to a stack.
- Support for Python 3.4 was dropped as it is past end-of-life by Python.org.

5.5 0.5.2

- Improved on serialization of non-standard (i.e. NumPy) types in JSON-ized meta-data by making use of the *default* callable in *json.dumps*. In particular deeply nested NumPy types should now serialize without erroring.

Note that there is no support for complex numbers in JSON meta-data, as JSON itself does not support it by default.

5.6 0.5.1

- Versions of MRCZ <= 0.4.1 were improperly writing the dimensions into the (Mx, My, Mz) volume fields. Added a check for the MRCZ version tag, and if an older file is found, it defaults to `slices == 1`, i.e. one 2D frame per element in the returned list. - In order to suppress the warning message, files can be read into memory and

  re-saved. A utility script for batch processing is provided in `utils\update_mrcz_0.5.0.py`.

5.7 0.5.0

- Added support for lists of 3D *numpy.ndarray* objects. This is largely intended to support multi-channel STEM time series. Stores the number of channels per frame in the *MZ* value of the MRC2014 header, which must be uniform for every ndarray in the list. Any MRCZ archive that has a 'strides' key in the JSON metadata will be returned as a list of arrays. - See http://www.ccpem.ac.uk/mrc_format/mrc2014.php for header details - *asList* keyword arguments have been removed.
- Fixed a bug in casting from float64/complex128 that was not actually casting.
- Cleaned up the code to be more PEP8 compliant.

5.8 0.4.1

- Improved docstrings in *ioDM4.py*.
- Added *asyncReadDM4* function, analogous to *asyncReadMRC*.

5.9 0.4.0

- Fix a minor bug with casting for lists of arrays
- Improved uncompressed write times by not using list comphrension
- Add scaling block size for small format images (e.g. Medipix) to scale to the number of threads.
- If the passed arrays are C_CONTIGUOUS and ALIGNED, *writeMRC* will use *blosc.compress_ptr* instead of coverting the array to a *bytes* object which is a significant speedup.

5.10 0.3.8

- Auto-casts *np.float64* -> *np.float32* and *np.complex128* -> *np.complex64* but logs a warning to the user.

5.11 0.3.7

- Updated MANIFEST.in and *setup.py* to make Conda-forge happy.

5.12 0.3.6

- *mrcz.ReliablePy* must be imported explicitely now, as it has requirements that the base *mrcz* package does not. This file may be removed in the future if no users are using it.

5.13 0.3.5

- If `blosc` is not installed and the user attempts to operate with compression on an `ImportError` is raised.
- Documentation now using Numpy docstrings.

5.14 0.3.4

- Add (temporarily) MRC types for *uint32* and *int32* to support 24-bit detectors. May break in the future, as the CCP-EM committee should make the final decision on such enumerations.
- Added handling of NumPy scalars (i.e. *np.float32(1.0)*) in metadata so that JSON serialization does not generate errors. Values will be case to Python *int* or *float* as appropriate.

5.15 0.3.3

- Removed use of star-expansion of args as it breaks Python 2.7/3.4.

5.16 0.3.2

- Made *blosc* an optional dependency due to difficulties involved in building wheels for PyPi.
- Implemented reading/writing of *list* of equally-shaped 2D *ndarray's instead of a single 3D 'ndarray*, where the *list* represents the Z-axis. This approach can be helpful for larger arrays that do not have to be continuous as the operating system can more easily interleave them into memory.

5.17 0.3.1

- Added ascii identifier label 'MRCZ' + <__version__> to the labels. I.e. at byte 224 in the header will appear b'MRCZ0.3.1'

5.18 0.3.0

- Documentation now available at http://python-mrcz.readthedocs.io/
- Added continuous integration testing with Appveyor and TravisCI, which was previously handled by *c-mrcz*.
- Added handling for *dask.array.core.Array* objects.
- *numpy.ndarrays* inside *meta* dictionaries will be converted to *list* objects to facilitate serialization.
- Updated license to BSD-3-clause from BSD-2-clause.

- Various bug fixes.

5.19 0.2.1-4

- Various bug fixes to incorporate into Hyperspy.

5.20 0.2.0

- Added support for asynchronous reading and writing.

5.21 0.1.4a1

- Fixed a bug with the machine-stamp not being converted to bytes properly.

5.22 0.1.4a0

- Fixed a bug in import of mrcz from ReliablePy

5.23 0.1.3a2

- Added ReliablePy, an interface for Relion .star and Frealign .par files.
- Fixes to maintain cross-compatibility with *c-mrcz*. Main functions are readMRC and writeMRC. readMRC always returns a header now.
- Added mrcz_test suite, which also tests *c-mrcz* if it's found in the path.
- Fixed bugs related to *mrcz_test.py*

5.24 0.1.1a1

- Renamed 'cLevel' to 'clevel' to maintain consistency with *blosc* naming convention.
- Updated license from MIT to BSD 2-clause.

5.25 0.1.0dev0

Initial commit

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

D

M

P

R

T

W