Journal of Structural Biology 201 (2018) 252–257

Contents lists available at ScienceDirect

Journal of Structural Biology

journal homepage: www.elsevier.com/locate/yjsbi

Technical Note

MRCZ – A file format for cryo-TEM data with fast compression

Robert A. McLeod[a,*], Ricardo Diogo Righetto[a], Andy Stewart[b], Henning Stahlberg[a]

[a] Center for Cellular Imaging and NanoAnalytics (C-CINA), University of Basel, Basel, Switzerland
[b] Department of Physics, University of Limerick, Limerick, Ireland

ARTICLE INFO

Keywords:
Lossless compression
Data management
File archiving

ABSTRACT

The introduction of fast CMOS detectors is moving the field of transmission electron microscopy into the computer science field of big data. Automated data pipelines control the instrument and initial processing steps which imposes more onerous data transfer and archiving requirements. Here we conduct a technical demonstration whereby storage and read/write times are improved 10× at a dose rate of 1 e^-/pix/frame for data from a Gatan K2 direct-detection device by combination of integer decimation and lossless compression. The example project is hosted at *github.com/em-MRCZ* and released under the BSD license.

1. Introduction

The introduction of CMOS-based direct electron detectors for transmission electron microscopy greatly improved the duty cycle to nearly 100% compared to traditional slow-scan CCD detectors. The high duty-cycle allows for nearly continuous read-out, such that dose fractionation has become ubiquitous as a means to record many-frame micrograph stacks in-place of traditional 2D images. The addition of a time-dimension, plus the large pixel counts of CMOS detectors, greatly increases both archival and data transfer requirements and associated costs to a laboratory. Many laboratories have a 1 Gbit/s Ethernet connection from their microscope to their computing center, which implies a data transfer rate of around 60–90 MB/s under typical conditions. If the microscope is run with automated data collection, such as SerialEM (Mastronarde, 2005), then the so-called 'movie' may be 5–20 GB and may be saved every few minutes, or even faster. In such a case, it may not be possible to transfer the data fast enough to keep up with collection. Costs for storing data on spinning (hard disk) storage, for example through the use of Google Cloud (Google Cloud Storage Pricing, accessed 2017), is typically US$100–200/TB/year. A cryo-TEM laboratory producing 200 TB of data per year is potentially faced with an annual data storage cost on the same order of magnitude as a post-doctoral fellow salary.

One approach whereby considerable archival savings may be realized is by decimation of the data from floating-point format to integer-format. Nominally, the analog-to-digital converted signal from the detector is typically output as an integer. Due to data processing requirements, it is often necessary to convert the integer data to 32-bit floating point format. The most common initial step that results in decimal data is the application of a gain reference, where the bias of the detector white values is removed. In-addition, conversion to floating-point is often inevitable due to operations such as sub-pixel shifting in drift correction (Li et al., 2013a,b; Grant and Grigorieff, 2015; McLeod et al., 2016; Zheng et al. 2017), or image filtration. If instead the micrographs are stored as 8-bit integers, with the gain reference (and potentially other operations) stored in meta-data, then a 4× reduction in storage and transfer requirements is realized. In this case, the gain reference and other bias corrections must be performed at the computing center, rather than using the software provided by the direct electron detector vendor. Since vendor gain normalization techniques are often proprietary and secret, there is a need for open-source equivalent solutions (Afanasyev et al., 2015).

Further improvements in data reduction can be realized by modern high-speed lossless compression codes. Lossless compression methods operate on the basis of repeated patterns in the data. Nominally, purely-random numbers are incompressible. However counting electron data is Poisson distributed, such that its range of pixel histogram covers on only a limited range of values. In such a regime substantial compression ratios may be achieved. Therefore due to the repetition of intensity values, integer-format data can be compressed much more efficiently than gain-normalized floating-point data. Generally when comparing compression algorithms one is interested in the compression rate (in units of megabytes/s) and the compression ratio (in percent). Modern compression codecs such as Z-standard (github.com/facebook/zstd, accessed 03/2017) or LZ4 (github.com/lz4/lz4, accessed 03/2017) are designed for efficient multi-threaded operation on modern, parallel

Abbreviations: Blosc, blocked, shuffle, compress library; CMOS, complementary metal-oxide semiconductor; float32, floating-point 32-bit computer data, ~6 significant figures; FPGA, Field-Gate Programmable Arrays; GB, Gigabyte (2^{30} bytes); Gb, Gigabit, network (10^9 bits); MB, Megabyte (2^{20} bytes); uint8, unsigned 8-bit integer computer data, range 0255

* Corresponding author at: C-CINA, Biozentrum, University of Basel, Switzerland.
E-mail address: robbmcleod@gmail.com (R.A. McLeod).

https://doi.org/10.1016/j.jsb.2017.11.012
Received 24 March 2017; Received in revised form 20 November 2017; Accepted 22 November 2017
Available online 23 November 2017

CPUs and can compress on the order of 1–2 GB/s/core, such that the time for read/write/transfer plus compression operations is greatly faster than when operating on uncompressed data.

We demonstrate here combining decimation to 8-bit integer with lossless compression. We utilize an extension of the venerable MRC format (Cheng et al. 2015; CCP-EM), where meta-*compression* is implemented which implies the combination of lossless compression and lossless filtering to improve compressibility as well as execution with efficient blocked and multi-threaded processing. We propose using common serialization tools to embed metadata in the MRC2014 extended header, and compare JSON (ECMA-404, 2013) and Message Pack (msgpack.org, accessed 03/2017).

2. The MRCZ format

The MRC format was introduced by Crowther et al. (1996) as an extension of the CCP4 format. It features a 1024-byte metadata header, followed by binary image data with provisions for 3-dimensions. The supported data types are byte (int8), short (int16), or single-precision floating-point (*float32*). The simplicity of the MRC format, and its ease of implementation, is a likely reason contributing to its popularity. However, the MRC format suffers from some drawbacks. There is no one standard format for MRC, in-spite of many efforts to define one (Cheng et al., 2015). Furthermore, it cannot compress the data, so it is inefficient from an archival and transmission/distributed computing perspective.

An alternative public domain archival format for electron microscopy is HDF5. However, HDF5 is a “heavyweight” library consisting of ~350′000 of lines of code and 150-pages of specification (HDF Group, accessed 2017), which makes integration in existing projects difficult. HDF5 has previously demonstrated compression filters including *blosc* and an additional LZ4-based filter funded by Dectris (Baden, CH) (Nexus format, accessed 09/2017).

Here we introduce an evolution of the MRC format, MRCZ, with additional functionalities that have become needed in the era of ‘Big Data’ in electron microscopy. We provide sample libraries for MRCZ in C/99 and also Python 2.7/3.5, as well as a command-line utility that may be used to compress/decompress MRC files so that legacy software can read the output. To facilitate the introduction of MRCZ into other software packages, we have kept the implementations as small as possible (currently *c-mrcz* is < 1000 lines of code).

The MRCZ library package leverages an open-source, meta-compression library, *blosc* (*blo*cking, *s*huffle, *c*ompression), principally written by Francesc Alted and Valentin Haenel (Haenel, 2014, and Alted, accessed 03/2017). *Blosc* combines multi-threaded compression (currently six different codecs are available) with blocking, such that each operation fits in CPU cache (typically optimized to level 2 cache), and filter operations (namely *shuffle* and *bitshuffle*). In testing on cryo-TEM data *blosc* achieved $>10 GB/s$ compression rates on a modern CPU, and furthermore achieves superior compression ratios to codecs such as LZW (Welch, 1984) implemented in TIFF. The performance gain is sufficient such that loading a compressed image stack from disk and applying post-processing gain normalization and outlier pixel filtering to it is faster than loading an uncompressed but pre-processed floating-point result. The Python version of the library also supports asynchronous file writing and reading, where the file is read or written in a background thread, freeing the interpreter for other tasks.

Here results for three compressors are compared for operation on cryo-EM data. Other codecs were tested, including the new Lizard codec (released in March 2017), but not found to have performance advantages for the test data:

1. lz4 is the fastest compressor, with the worst compression ratio, making it ideal for live situations where distributing the data from a master computer is the priority.
2. zStandard (zstd): achieves the highest compression ratio and has the fastest decompression, making it the best choice for archiving. On the lowest compression level it maintains good compression rates.
3. Zlib is a very common library that has been accelerated by blosc. Zlib provides a valuable baseline for comparison, although blosc compression rate with zlib exceeds that of tools such as pigz.

2.1. Blocked compression

Roughly around 2005, further increases in CPU clock-frequencies were slowed due to heat generation limitations. Further performance improvements where then realized by packing parallel arithmetic and logic cores per chip. Most common compression algorithms were designed before the era of parallel processing.

Operations in image processing are often relatively simple and executed on the full-frame consisting of many million elements. With the larger number of cores available on modern CPU, often program execution rate is limited not by processing power but the amount of memory bandwidth available to feed data to the cores. Typically fetching data from random-access memory (RAM) is an order of magnitude slower than the cache found on the CPU die. Therefore if the data can be cut into blocks that fit into the lower-level caches large speed improvements are often observed. Parallel algorithms can be made to work efficiently in the case where a computational task can be cut into blocks, and each block can be dispatched to an individual core, and run through an algorithm to completion, as illustrated in Fig. 1a. Parallel algorithms should also avoid branching instructions (e.g. conditional *if* statements), as modern processors request instructions from memory in-advance, and a wrong guess can leave the process idle waiting for memory. For example, *zStandard* also makes use an of a faster and more effective method for evaluating entropy, known as Asymmetric Numeral System (ANS), than classic compression algorithms. ANS significantly improves compression ratio in data with large degrees of randomness (Duda, 2013; Duda et al., 2015).

In blocking strategies, data is conceptually separated into chunks and blocks, with chunks being senior to blocks. In the MRCZ format, each chunk is a single image frame (~16 million pixels for 4 k detectors, or ~64 million for 8 k), and each chunk is broken into numerous blocks, with a default block size of 1 MB. Such a block size provides a balanced trade-off between compression rate and the ratio between compressed and uncompressed data. For image stacks, the highest compression ratio would likely be in the time-axis, but this is the least convenient axis for chunking, as it would make retrieving individual frames or slices of frames impossible.

2.2. Bit-decimation by shuffling

Direct electron detectors may be operated in counting mode whereby the ratio of dose rate to detector cycle rate is low enough that two electrons landing in the same pixel on a rapidly cycling detector is statistically rare. The order of magnitude of the usable dose rate before mis-counting is on the order of magnitude of $1 e^{-}/pix/frame$ per $100 Hz$ cycle rate of the detector. Typically drift correction is performed on the time scale of a second, so for the K2 Summit (Gatan, Pleasanton, CA) operating at 400 Hz the expected dose per pixel in an integrated frame is $\widetilde{1}-8$. This implies that even a single byte (*uint8*) to store each pixel is too large of a data container, as it can hold data values up to 255. David Mastronarde implemented in *SerialEM* and *IMOD* (Mastronarde, 2005) a new data type for MRC that incorporates a decimation step where each pixel is packed into 4-bits, leading to maximum per-pixel values of 16 before clipping occurs, thereby providing an effective compression ratio of 2.0 compared to *uint8*. The disadvantage of 4-bit packed data is that it is not a hardware data type, such that two pixels are actually packed into an 8-bit integer. Whenever the data is loaded into memory for processing, it must be unpacked with bit-shifting operations, which is computationally not free. There is also the risk of intensity-value clipping.

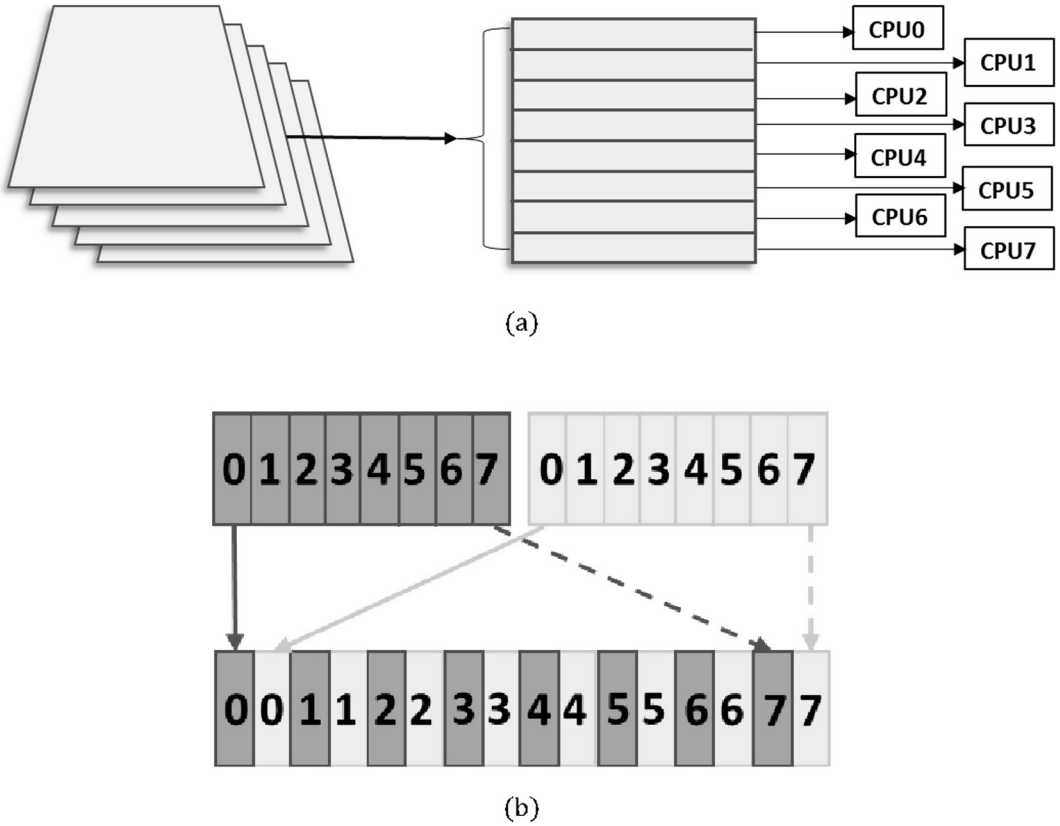

Fig. 1. (a) Each MRC volume is chunked, such that each z-axis slice is compressed separately. Then in *blosc* each chunk is further sliced into blocks, which are then dispatched to individual CPU cores for compression. Decompression works in reverse. (b) Normally pixel values are stored in memory contiguously (top row). With bit-shuffling (on little endian systems) the most significant bits (**7** index) are stored adjacently, and similarly for the least-significant bits (**0** index). This improves compressibility and as a result both compression ratio and compression rate are improved.

Blosc optionally makes use of a filter step, of which there are two currently implemented, *shuffle* and *bitshuffle*. *Shuffle* re-arranges each pixel by its most significant byte to least, whereas *bitshuffle* performs the same task on a bit-level, illustrated in Fig. 1b. The shuffle-style filters are highly efficient when the underlying data has a narrow histogram, such that the most-significant digit in a pixel has more commonality with other pixels' most significant digit than its own least-significant digit. For example, if an image saved as *uint8* type contains mostly zeros in its most significant digits, they will be bit-shuffled into a long-series of zeros, which is trivially compressible. As such, *bitshuffle* effectively performs optimized data decimation without any risk of clipping values. Shuffling is also effective for floating-point compression, as the sign bit and the exponent are compressible whereas the mantissa usually does not contain repeated values and therefore it is not especially compressible. The mantissa can be made more compressible by rounding to some significant bits, for example the nearest 0.001 of an electron, but this generates round-off error.

2.3. Benchmarks

Benchmarks for synthetic random Poisson data were conducted for images covering a range of electron dose levels consisting of [0.1, 0.25, 0.5, 1.0, 1.5, 2.0, 4.0] electron counts/pixel. The free parameters examined consist of: compression codec, block size, threads, and compression level were all evaluated. Here the term 'compression level' refers to the degree of computing effort the algorithm will use to achieve higher compression ratios. The machine specification for benchmark results is as follows:

Two Intel® Xeon® E5-2680 v3 CPUs operating with Hyperthreading® and Turboboost®:

- No. of physical cores: 2 × 12
- Average clock rate: 2.9 GHz (spec: 2.4 GHz)
- L1 cache size: 32 KB per core
- L2 cache size: 256 KB per core
- L3 cache size: 30,720 KB per processor

The size of the L2 cache generally has a large impact on the compression rate as a function of the blocksize used by *blosc*. For file I/O the RAID0 hard drive used was benchmarked to have a read/write rate of ~300 MB/s, which is comparable to parallel-file systems in general use in cluster environments.

Example benchmarks on cryo-TEM image stacks are shown in Table 1 for a variety of *blosc* libraries as well as external compression tools. *Uint4* refers to the *SerialEM* practice of interlaced packing of two pixels into a single-byte. JPEG2000 and *uint4* were not multi-threaded; all other operations used 48 threads. *Pigz*, *lbzip2* and *pxz* are command-line utilities and hence include a read and write. Indicated times are averages over 20 read/writes. To achieve repeatable result, the disk was flushed between each operation, with the Linux command:

$$\text{echo3|sudotee/proc/sys/vm/drop}_{\text{caches}}$$

The gains in compression ratio by using more expensive algorithm such as Burrows-Wheeler (bzip2) or LZMA2 (xz) are quite minimal with cryo-TEM data, likely due to the high degree of underlying randomness (or entropy). *Lbzip2* is the clear winner among command-line compression tools, as it still is faster than reading or writing uncompressed data and achieves the second-best compression ratio. *Blosc* accelerates the read/write by a factor of 3–6x over that of the uncompressed data.

Figures for benchmark results are shown in Fig. 2. Best compression ratio as a function of dose rate is shown in Fig. 2a. An important consequence of compressing Poisson-like data is that compression ratios

Table 1
Comparison of read/write times for 60 × 3838 × 3710 cryo-TEM image stacks.

Codec/data type/ compression level	Compressed Size (MB)	Compression Ratio	Compression-Write Time (s)	Decompression-Read Time (s)
None/int8	854	1.00	3.40	3.21
uint4	427	0.50	2.14	6.05
blosc-lz4/ int8/9	340	0.40	0.50	0.96
blosc-zstd/ int8/1	320	0.37	0.76	1.10
blosc-zstd// int8/5	319	0.37	0.86	1.09
JPEG2000/ uint8	317	0.37	106.8	N/A
pigz/int8/1	367	0.43	0.86	4.74
lbzip2/int8/9	314	0.37	3.17	2.23
pxz/int8/6	305	0.36	47.5	31.8

increase substantially with sparseness. I.e. compressed size scales sub-linearly with decreasing dose fractions. For example, a cryo-tomography projection of 10 frames of 4 k × 4 k data recorded at a dose rate of $0.1e^{-}/pix/frame$ would have a compressed size of 13 MB, compared to 670 MB for its uncompressed, gain-normalized image stack. Such compression therefore enables finer-dose fractionation for advanced drift correction algorithms without imposing onerous storage requirements. (See Fig. 3)

With regards to compression level, shown in Fig. 2b, which is a reflection on the effort level of the compressor, generally *zlib* saturates at 4–5, whereas *zstd* saturates at 2–3, and *lz4* sees little disadvantage to running at its highest compression level. Good compromises for processing are compression level 1 for *zstd* and *zlib* and for archiving 3 for *zstd* and *5* for *zlib*. *Lz4* can operate with compression levels of 9 for real-time applications but it is not as suitable for archiving due to the lower compression ratios. The *bitshuffle* filter is important in this situation and contributes heavily to the quickest compression level of 1 being the best compromise between rate and ratio for *zstd*, in that it uses *a priori* knowledge about the structure of the pixel values to pre-align the data into its most compressible order.

In *blosc* the scaling with threads is roughly $1/0.7N_{threads}$ for $N_{threads} \geqslant 2$, up to the number of physical cores. When hyper-threading is enabled an oversubscription of approximately $N_{threads} \approx 1.5N_{cores}$ gives the highest absolute compression rate.

Cache sizes are important in that they impose thresholds on data sizes, shown in Fig. 1d. *blosc* chops the data into blocks, and *MRCZ* cuts a volume into single z-axis slices called *chunks*. For example a 4 k × 4 k

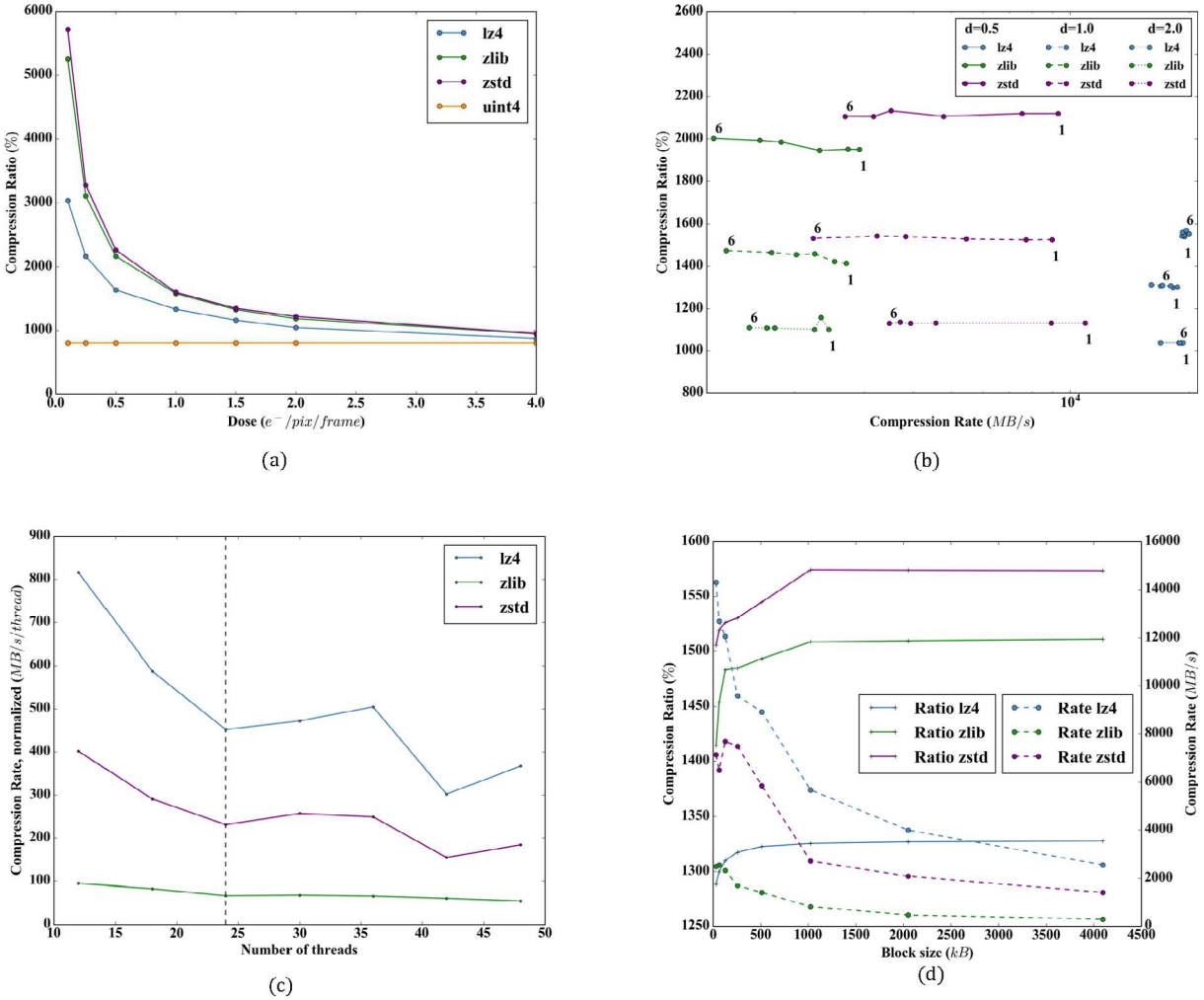

Fig. 2. Performance for various compression codecs found in *blosc*. (a) The dependence of compression ratio varies strongly with the dose. Here *zstd* has the best compression ratio. (b) The dependence of the compression level on the compression ratio is mild, such that for *zstd* and *zlib* typically 1 is used. (c) Scaling on the compression rate with the number of parallel computing threads employed. The machine used has 2 × 12 physical cores, indicated with the dashed line. The area to the right of the dashed line indicates the region in which Intel Hyperthreading® is active. (d) Dependence of the compression ratio and rate on the blocksize used, which is the most critical parameter examined. Typically a blocksize scaled to fit into L2 cache ($256kB$) is optimal for speed, but the compression ratio benefits from a larger blocksize ($\geqslant 512kB$).

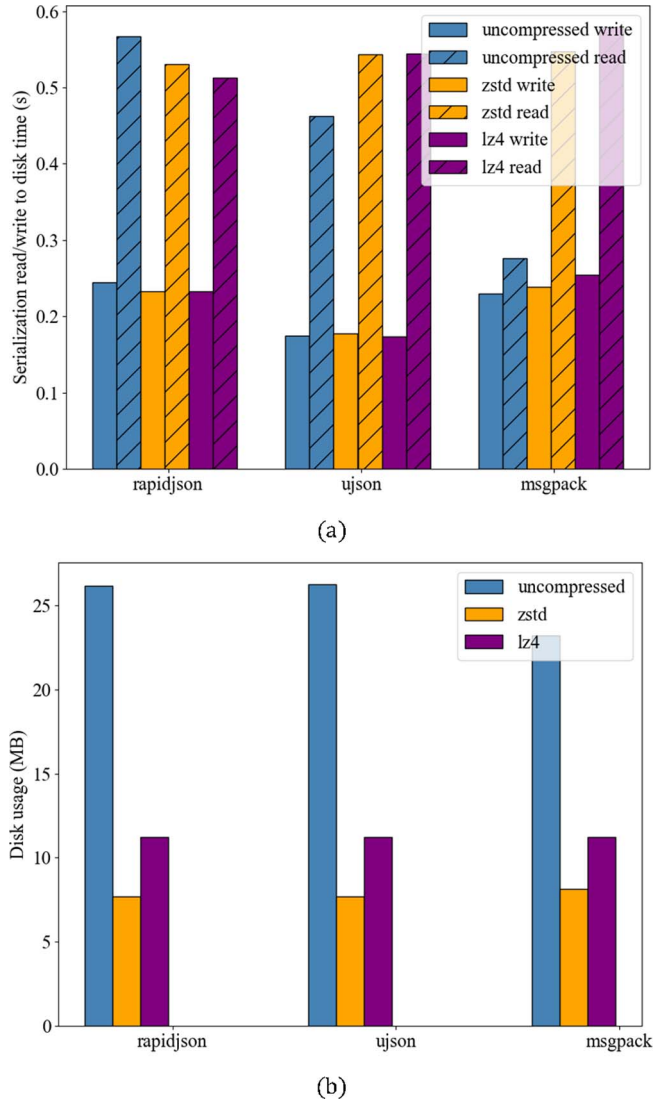

Fig. 3. Performance evaluation of different serialization methods for meta-data paired with compression. (a) Read and write times for the profiled serialization methods on a sample of 25 MB of JSON-like text metadata data, when used with and without blosc compression. (b) Size on disk of the metadata.

image chunk may be cut into 64 separate 256 kB blocks. If the block size fits into the L2 cache ($\leqslant 256kB$) then compression rate advantage is expected, and this is evident. However, testing on simulated Poisson data shows that larger blocks (which result in larger dictionaries in the compression algorithm) achieve a higher compression ratio. Similarly for chunking, if the chunk size is less than the L3 cache ($\leqslant 30MB$) then only one memory call is sufficient for the entire chunk.

The optimal block size for compression ratio is expected to be when each block holds one significant bit each. So for $4\,k \times 4\,k \times 8$-bit images the ideal block size would be $16/8 = 2MB$, whereas the saturation in compression ratio actually appears at 1 MB.

2.4. Enabling electron counting in remote computers with image compression

Direct electron detectors can generate data at very high rates that are difficult to continuously write to storage. Some versions of the K2, such as the In-Situ (IS) model, permit access to the full data rate of at $3838 \times 3710 \times 400$ Hz $\times$ uint8, or 5.3 GB/s. Such systems require a large quantity of random-access memory to record the data in bursts. The K2 Summit commonly used in cryo-TEM counts at 400 Hz through the use of Field-Gate Programmable Arrays (FPGA) but the maximum rate available to the user is 40 Hz, in large part due to the data rate. This is unfortunate as it does not permit experimentation with alternative subpixel detection algorithms that might better localize the impact of the primary electron in the detector layer.

However the multi-threaded meta-compression discussed above may be able to alleviate the data flow problem. A master node could, using *zstd* (or failing that, *lz4*) as a compression codec, compress the raw data from a K2 IS on-the-fly and dispatch it to worker computers for counting, thus enabling counting without a FPGA or similar hardware counting solution. The compression ratio achievable would depend heavily on thresholding of low intensity values but could be in the range of 50:1.

3. Extended metadata in MRCZ

File formats require complex, nested metadata to be encoded into a stream of bytes. The conversion of metadata to bytes is called serialization. In order to achieve a high level of portability in the future for the metadata, we advise use of a well-established serialization standard. Here two serialization standards are compared, JSON (JavaScript Object Notation), which is the most ubiquitous serialization method in the world, and Message Pack (msgpack.org and pypi.python.org/pypi/msgpack-python), a binary serialization tool with a similar language structure to JSON. Libraries are available for both for many different programming languages, with the exception of Matlab and Fortran for Message Pack. Here two high-speed JSON encoders available for C and Python are profiled, RapidJSON (rapidjson.org and pypi.python.org/pypi/pyrapidjson) and UltraJSON (github.com/esnme/ujson4c and pypi.python.org/pypi/ujson).

The three serialization methods were tested on a sample of 25 MB of complicated JSON data. All three methods produce more-or-less similar results in terms of read/write times to disk, as shown in Fig. 2a. Compression does not speed nor slow read/write times, except for Message Pack read rates. *Lz4* reduces the data size on disk to roughly one-half, and *zstd* to roughly one-third, of the uncompressed size. Message Pack was tested with Unicode-encoding enabled to make it equivalent to the JSON encoders, which slows its read/write time by ~20%.

4. Conclusion

The introduction of fast, large-pixel count direct electron detectors has moved the field of electron microscopy inside the domain of “Big Data” in terms of data processing and storage requirements. Here an extension of the MRC file format is demonstrated that permits on-the-fly data compression to lessen both transmission and storage requirements, using the meta-compression library *blosc*. *Blosc* is well suited as a library for Big Data purposes because it does not explicitly endorse any particular algorithm and intends to support new compression methods as they are developed. Development of a *blosc2* standard, with additional features, is currently underway. Major new expected features include principally: first, a super-chunk header, that records the position of individual image chunks in the file, and second, buffered output so the file can be written to disk as it is compressed, lessening memory consumption. Also *blosc* is especially targeted towards high-speed compression codecs. With high-speed compression using the *zStandard* codec, file input-output rates are accelerated and archival storage requirements are reduced.

With the use of compression, sparse data can be compressed to very high ratios. Hence, data can be recorded in smaller dose fractions with a less-than-linear increase in data size. For very small dose fraction applications, such as cryo-electron tomography, electron crystallography, or software electron counting schemes, compression can reduce data transmission and storage requirements by 10–50x.

Acknowledgements

The authors would like to thank the authors of the *blosc* library, Francesc Alted and Valentin Haenel, for their work on the library and feedback during our testing of MRCZ. This work was supported by the Swiss National Science Foundation (grant 205320_166164), and the NCCR TransCure program.

References

Afanasyev, P., Ravelli, R.B.G., Matadeen, R., De Carlo, S., van Duinen, G., Alewijnse, B., Peters, P.J., Abrahams, J.-P., Portugal, R.V., Schatz, M., van Heel, M., 2015. A posteriori correction of camera characteristics from large image data sets. Sci. Rep. 5.

Alted, K. 2014. Blosc, an extremely fast, multi-threaded, meta-compressor library [WWW Document]. Blosc Main Page. URL http://www.blosc.org/index.html (accessed 3.13.17).

MRC/CCP4 file format for images and volume [WWW Document]. n.d. CCP-EM, URL http://www.ccpem.ac.uk/mrc_format/mrc_format.php (accessed 7.27.17).

Cheng, A., Henderson, R., Mastronarde, D., Ludtke, S.J., Schoenmakers, R.H.M., Short, J., Marabini, R., Dallakyan, S., Agard, D., Winn, M., 2015. MRC2014: extensions to the MRC format header for electron cryo-microscopy and tomography. J. Struct. Biol. Rec. Adv. Detector Technol. Appl. Mol. TEM 192, 146–150. http://dx.doi.org/10.1016/j.jsb.2015.04.002.

Crowther, R.A., Henderson, R., Smith, J.M., 1996. MRC image processing programs. J. Struct. Biol. 116, 9–16. http://dx.doi.org/10.1006/jsbi.1996.0003.

Duda, J., 2013. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. arXiv:1311.2540 [cs, math].

Duda, J., Tahboub, K., Gadgil, N.J., Delp, E.J., 2015. The use of asymmetric numeral systems as an accurate replacement for Huffman coding. In: 2015 Picture Coding Symposium (PCS). Presented at the 2015 Picture Coding Symposium (PCS), pp. 65–69. doi:10.1109/PCS.2015.7170048.

Standard ECMA-404 [WWW Document], n.d. URL http://www.ecma-international.org/publications/standards/Ecma-404.htm (accessed 3.11.17).

Grant, T., Grigorieff, N., 2015. Measuring the optimal exposure for single particle cryo-EM using a 2.6 Å reconstruction of rotavirus VP6. eLife Sciences e06980. doi:10.7554/eLife.06980.

Google Cloud Storage Pricing | Cloud Storage Documentation [WWW Document], n.d. Google Cloud Platform. URL https://cloud.google.com/storage/pricing (accessed 3.11.17).

Haenel, V., 2014. Bloscpack: a compressed lightweight serialization format for numerical data. arXiv:1404.6383 [cs].

HDF5 File Format Specification Version 3.0 [WWW Document], n.d. URL https://support.hdfgroup.org/HDF5/doc/H5.format.html (accessed 1.3.17).

Li, X., Mooney, P., Zheng, S., Booth, C.R., Braunfeld, M.B., Gubbens, S., Agard, D.A., Cheng, Y., 2013a. Electron counting and beam-induced motion correction enable near-atomic-resolution single-particle cryo-EM. Nat. Meth. 10, 584–590. http://dx.doi.org/10.1038/nmeth.2472.

Li, X., Zheng, S.Q., Egami, K., Agard, D.A., Cheng, Y., 2013b. Influence of electron dose rate on electron counting images recorded with the K2 camera. J. Struct. Biol. 184, 251–260. http://dx.doi.org/10.1016/j.jsb.2013.08.005.

lz4/lz4 [WWW Document], n.d. GitHub. URL https://github.com/lz4/lz4 (accessed 3.13.17).

Mastronarde, D.N., 2005. Automated electron microscope tomography using robust prediction of specimen movements. J. Struct. Biol. 152, 36–51. http://dx.doi.org/10.1016/j.jsb.2005.07.007.

McLeod, R.A., Kowal, J., Ringler, P., Stahlberg, H., 2016. Robust image alignment for cryogenic transmission electron microscopy. J. Struct. Biol. http://dx.doi.org/10.1016/j.jsb.2016.12.006.

MessagePack: It's like JSON. but fast and small. [WWW Document], n.d. URL http://msgpack.org/index.html (accessed 3.11.17).

Nexus Format/HDF5-External-Filter-Plugins [WWW Document], n.d. URL https://github.com/nexusformat/HDF5-External-Filter-Plugins (accessed 13.9.17).

Welch, T.A., 1984. A Technique for high-performance data compression. Computer 17, 8–19. http://dx.doi.org/10.1109/MC.1984.1659158.

Zheng, S.Q., Palovcak, E., Armache, J.-P., Verba, K.A., Cheng, Y., Agard, D.A., 2017. MotionCor2: anisotropic correction of beam-induced motion for improved cryo-electron microscopy. Nat. Meth. Adv. Online Publ. http://dx.doi.org/10.1038/nmeth.4193.

facebook/zstd [WWW Document], n.d. GitHub. URL https://github.com/facebook/zstd (accessed 3.13.17).