High-Throughput Processing of Diviner Data from the Planetary Data System (PDS) using various Compression Utilities

Norbert Schorghofer (norbert@psi.edu)
Planetary Science Institute, Honolulu, Hawaii

December 17, 2021

Abstract

As the data volume in NASA's Planetary Data System (PDS) increases, techniques for high-throughput data processing become increasingly important. The Diviner instrument onboard Lunar Reconnaissance Orbiter, a radiometer that measures surface temperature, has returned many Terabytes of data over the so far twelve years it has been in lunar orbit. Here, several approaches to decompression and parallel filtering for this dataset are explored and benchmarked.

Lossless compression with `lzma`, `brotli`, and `zstd` produces files that are less than half the size of the (already compressed) `zip` files stored on the PDS. The fastest decompression is achieved with `lz4` and `zstd`, although passing the data through a simple filter of the data columns limits the throughput. Modern compression utilities can outperform `zip` in terms of compression ratio and decompression time simultaneously. The throughput reaches the physical bandwidth of the system with simple but thoughtfully written shell scripts for data analysis. In brief, download times and throughput for numerical data can both be improved simply by switching to a modern compression file format.

When data are read from a single solid state drive, multi-threaded (parallel) data processing increases the throughput significantly, whereas only a moderate scaling is observed for a hard disk drive.

1 Introduction

The Planetary Data System (PDS) is the long-term archive of digital data products returned from NASA's planetary missions. As of 2020, the PDS holds more than 1.8 Petabyte of data from over 70 missions (McClanahan, 2020). It becomes increasingly important to develop Big Data techniques that enable planetary scientists to quickly process large volumes of data. Here, the performance of compression utilities is evaluated, using the Diviner dataset from the Lunar Reconnaissance Orbiter as case study. The conclusions will also be relevant for many other data analysis tasks of numerical tables that are read-intensive (I/O limited).

Table 1 lists the lossless compression utilities and compression formats evaluated in this study. They include well-known formats (`gzip`, `zip`, `bz2`, and `rar`), and more recently developed formats, such as `brotli`, `lz4`, `lzma`, and `zstd` (Zstandard). Numerous compression benchmarks can be found online; the benchmarks here are specifically for numerical data.

Diviner on the Lunar Reconnaissance Orbiter (LRO) provides high-resolution surface temperature (Paige et al., 2010a,b) and has continuously collected data since 2009. The Diviner Reduced Data Record (RDR), the Level 1b data set, has currently about 24 TB, and continues

Compression utility	File extension	level default (range)
brotli 1.0.7	br	11 (1–11)
bzip2 1.0.8	bz2	9 (1–9)
gzip 1.10	gz	6 (1–9)
lz4 1.9.2	lz4	1 (1–12)
lzma 5.2.4	lzma	6 (0–9)†
rar 5.50	rar	3 (0–5)
rzip 2.1	rz	6 (0–9)
zip 3.0	zip	6 (0–9)
zstd 1.4.4	zst	3 (1–19)

Table 1: List of lossless compression utilities evaluated in this study. †Each level also has an "e" option that may lead to stronger compression.

to grow by 170 GB per month. The RDR files (`https://pds-geosciences.wustl.edu/lro/lro-l-dlre-4-rdr-v1/lrodlr_1001/`) are archived on the PDS in `zip` format, compressed at the default compression level 6. These files have about 40 MB in compressed format each, and 289 MB in uncompressed ASCII format (file extension `.TAB`). Each file represents 10 minutes of collected data, and 144 data tables are produced each day. A data table consists of a few header lines and around 886,000 lines of mostly numerical entries. Each line contains 33 comma-separated entries, including date and time strings, but mostly integers and floating-point numbers (without exponentials). In total, PDS volume LRODLR_1001 contains 391,437 such files for data collected from Sep. 7, 2009 to Dec. 31, 2016. Another volume was started after this date. Nowadays tens of Terabytes of compressed data is not an unwieldy large data set to store, but detailed data analysis requires it to be read many times over.

The performance benchmarks were carried out on a desktop workstation with a 6-core (12 thread) Intel Xeon CPU E5-1650 v4, 3.6 GHz running Ubuntu Linux 20.04LTS. The level-3 cache has 15 MB. The magnetic disk spins nominally at 7200 rpm and has a buffer 23 MB in size. According to the output of the `hdparm` command, the magnetic disk drive has a cached read speed of 12 GB/s and a buffered read speed of 0.18 GB/s. The buffered read speed indicates how fast the drive can sustain sequential data reads, whereas the higher cached read speed is the maximum throughput of the processor, cache, and memory combined. The same workstation also has a solid state drive, where these numbers are 12 GB/s and 0.40 GB/s, respectively. Both drives use the `ext4` filesystem. All times were measured on an otherwise inactive system. The shell scripts used are available at `https://github.com/nschorgh/PDS-Throughput`

2 Compression ratios

The compression ratio is the ratio of compressed to uncompressed file size. Table 2 shows a subset of the results. The best compression ratio is achieved by `lzma`, `brotli`, and `zstd`. They produce files that are less than half the size of the `zip` format used on the PDS archive. The size of the lzma file is 18 MB, as opposed to 41 MB for the `zip` file. **Downloads would be more than twice as fast if these data files were archived on the PDS in lzma instead of zip format.** Brotli and Zstandard at high compression levels produce similarly small files as `lzma`.

In a situation where files are created only once but downloaded and analyzed many times, the compression time is irrelevant. Nevertheless, compression times were measured for several of the formats (using the `time` utility, averaged over three runs, and without enabling multi-threaded compression). Figure 1 shows compression times and file sizes. Note that `zstd` provides both faster compression and smaller files than `zip`.

File format	compression level	File size (MB)	Compression ratio
raw	-	289	1.0
lz4	1	67	4.3
lz4	*9*	46	6.3
lz4	*12*	45	6.4
zip	6	41	7.0
gzip	6	41	7.0
zip	*9*	40	7.2
zstd	3	38	7.6
brotli	*4*	38	7.7
rar	3	34	8.5
brotli	*9*	34	8.5
zstd	*15*	33	8.8
bzip2	9	28	10
rzip	6	26	11
zstd	*16*	22	13
zstd	*19*	19.9	15
brotli	*10*	19.4	15
brotli	11	18.3	16
lzma	6	18.0	16
lzma	*9*	17.9	16
lzma	*9e*	17.2	17

Table 2: The size of compressed Diviner RDR data files, sorted by file size. Compression levels in italics indicate non-default values. The average size for files `200908010000_RDR.TAB` and `201601010000_RDR.TAB` is reported, which have very similar compression ratios.

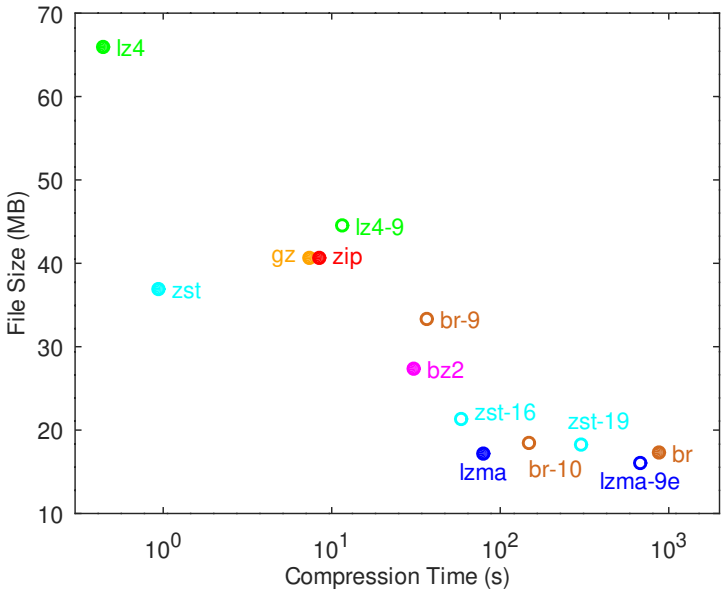

Figure 1: Performance of various compression utilities when applied to a file which has 289 MB in uncompressed (plain text) format. Filled symbols correspond to the default compression level and empty symbols to non-default levels. The time axis is logarithmic.

File format	Compression level	Time (s)	
		decompress to file	decompress & filter
raw	-	1.7	1.8
brotli	11	0.52	1.5
brotli	*9*	0.59	-
bzip2	9	5.3	-
gzip	6	1.3	1.3
lz4	1	0.43	1.4
lz4	*9*	0.35	1.4
lz4	*12*	0.31	-
lzma	*0*	1.9	-
lzma	6	1.4	1.5
lzma	*9*	1.4	1.5
rar	3	2.0	-
rzip	6	7.2	-
zip	6	1.3	1.4
zstd	3	0.42	1.0
zstd	*19*	0.34	1.0

Table 3: Results from decompression study for Diviner RDR data file `200908010000_RDR.TAB`. Compression levels in italics indicate non-default values. The time shown is the elapsed time for decompression plus writing output back to the same disk. The last column is for decompression to `stdout` plus `awk` filtering.

3 Decompression time of single file

This benchmark measures the time it takes to decompress the file and write the uncompressed file back to the harddrive. Decompression times are measured with the Unix `time` utility, and represent the total time elapsed. Averages are taken over three runs, for one file (`200908010000_RDR.TAB`). To measure the throughput appropriately, the disk buffer has to be cleared beforehand. The Linux kernel prefers to keep files in unused memory, but for a data stream that exceeds the size of the unused memory the advantage of a disk buffer disappears.

Table 3 shows a subset of the decompression benchmark measurements. Plain text throughput is close to the specified unbuffered disk read speed (0.2 GB/s). The `lz4`, `zstd`, `brotli` formats are several times faster to decompress than the `zip` format currently used on the PDS.

Figure 2 illustrates the throughput combined with the compression ratio. The decompression time for `lzma`, the format with the best compression ratio, is almost the same as for the `zip` file. Higher compression levels lead to smaller files and the decompression time becomes shorter. This is observed for `lz4`, `zstd`, `brotli`, and `lzma`. Hence, it is economical to create `lz4` and `zstd` files with higher than default level, at the cost of increased compression time. Zstandard at the maximum compression level of 19 almost achieves the extreme compression ratio of `lzma` and the extreme decompression speed of `lz4`, simultaneously. It is therefore the best dual purpose compression format. **Modern compression formats can outcompete classic formats, such as zip, both in compression ratio and decompression rate simultaneously.** For example, `zstd` at default compression level compresses more quickly, creates smaller files, and decompresses faster than `zip`. By adjusting the compression level, one or more of these advantages can be accentuated.

Many of the utilities offer multi-thread support for compression, but not for decompression. Among the compression utilities tested, only `rar` uses two threads during decompression by default. Decompression times were also measured when, instead of writing the output directly to a file, the output was sent to `stdout` and then redirected to a file. The times were almost identical.

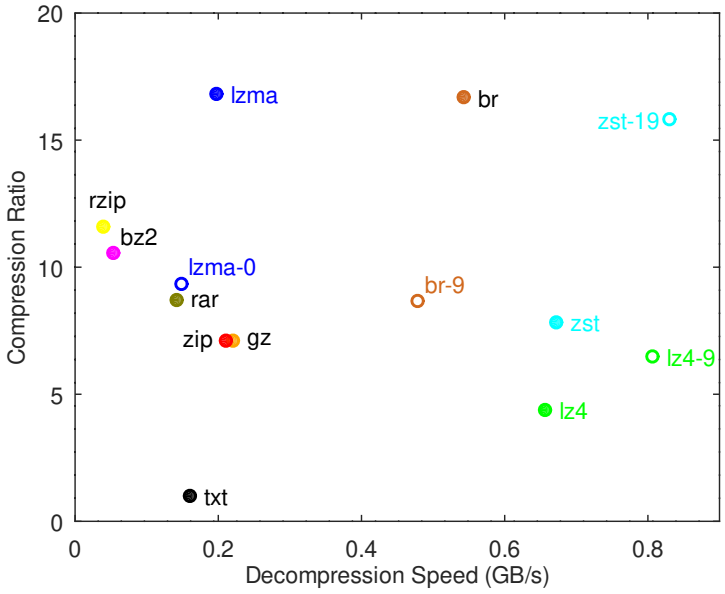

Figure 2: Performance of various compression utilities when applied to a Diviner RDR file, which has 289 MB in uncompressed (plain text) format. Filled symbols correspond to the default compression level, and empty symbols to selected non-default levels.

4 Time for decompression and filtering

Next, a file is decompressed and the output filtered with `awk`, so that only a small fraction of the output is written to harddrive. For this purpose, the output is limited to geographic longitudes between 120.0° and 120.1°. The longitude is among the columns in the data file. The decompressed data is written to `stdout` and piped to `awk` (gawk 5.0.1), which performs the filtering. (This particular task could be accelerated with indexing, but the approach here is more general.)

The results are included in Table 3 (last column), and they reveal that the throughput is lower due to the added workload. Zstandard outcompetes all of the other formats for the combined task of decompression plus filtering. Using `mawk` (1.3.4) instead of `awk` did not improve the throughput. Using a dedicated C program instead of `awk`, with an `fscanf` command, reduced the throughput. The thread utilization of `awk` itself is never close to 100%, which suggests the bottleneck is not `awk` itself, but the processing data pipeline which passes on the data.

5 Parallel processing of multiple files on single disk

Next, multiple files are decompressed and filtered concurrently, taking advantage of multiple CPU cores. The GNU `parallel` command was used to queue the single-threaded jobs for parallel processing (Tange, 2011). The geographic region chosen is small enough so that the size of the output is much smaller than the input file. The orbit of LRO is two hours long, so many of the 10-minute files will result in no output data at all. For the specific parameters chosen, the total output is less than 1% of the total input file size. This benchmark is limited to `lz4` (at non-default level 9), `lzma` (at default level 6), `zstd` (at non-default level 19), and the classical `zip` and `gzip` formats (both at default level 6).

Figure 3 shows the throughput as a function of the number of processes that were executed in

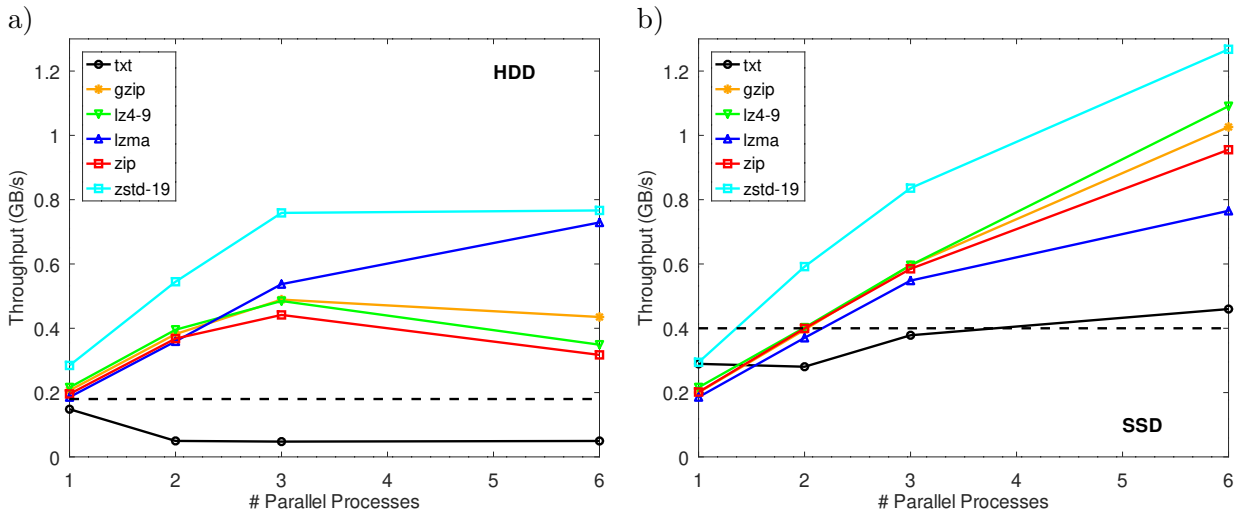

Figure 3: Throughput for various compression utilities when applied to 18 RDR files with a total uncompressed size of 5.08 GB for a) a hard disk drive (HDD), and b) a solid state drive (SSD). The processes executed concurrently on a multi-core CPU. The dash lines are the hardware limited bandwidth of the system and the throughput can be higher because it is measured relative to the uncompressed file size.

parallel, for a hard disk drive (HDD) and a solid state drive (SSD). For a HDD, parallel processing of the uncompressed raw data files always diminishes the throughput. For the compressed files, parallel processing leads to modest increases in throughput. The smallest compressed files (`lzma` and `zstd-19`) scale the best with the number of threads (Fig. 3a). They are small enough to fit into disk buffer memory, which is 23 MB (Table 2).

For the SSD, the scaling with the number of parallel processes is better (Fig. 3b). The highest throughput is achieved with `zstd-19`, on a single thread as well as with multiple threads. Several of the formats allow throughputs above 1 GB/s (when measured relative to the uncompressed file size) once six threads are utilized concurrently. Hence, parallel processing on a multi-core CPU that reads compressed files from a single drive yields only modest acceleration for an HDD, and a more considerable speedup for an SSD. For an HDD, the size of the disk internal memory compared to the size of the compressed files may play a role in the scaling behavior.

6 Discussion

The fact that the maximum physical disk-CPU bandwidth has been achived implies that additional techniques (such as changing the block size of the drive) are not necessary and will not accelerate the processing any further. Once an efficient compression format is chosen, the bottleneck of the system overall appears to be the passing of the decompressed data to the next step of the data processing.

Another relevant property for long term archiving is data recovery from corrupted files. However, some of the compressed file formats used on the PDS, like `zip`, already do not have this property.

Whether multiple disks connected to a single CPU can lead to further speedup has not been measured in this study, but judged by Fig. 3a the CPU is not the bottleneck. Multiple CPU-disk pairs (nodes) would allow for dramatic parallelization. The output from large-volume data

processing is often only a tiny fraction of the input volume, so combining the results from each node will be fast, and the scaling with the number of nodes will be almost proportional. At least for HDDs, the throughput will scale better with the number of nodes than with the number of CPU cores. For example, consider a system with 100 nodes where each HDD holds compressed files and is connected to a four-core CPU. More than a hundred disks are already required to hold all of the PDS data. With a 0.2 GB/s physical bandwidth, the throughput for arithmetically light data processing could be expected to be 0.5 GB/s per node or about 50 GB/s (3 TB/min or 176 TB/hr) in total.

Finally, Brotli and Zstandard are still actively developed, so the benchmarks might be surpassed in the future.

7 Conclusions

The results for compression ratio and decompression rate are summarized in Figure 2. Modern compression formats result in file sizes far smaller than the Diviner `zip` files stored on the PDS. Lossless compression with `lzma`, `brotli`, and `zstd` can produce files that are less than half the size of the (already compressed) `zip` files. Using these formats would result in considerably shorter download times.

Formats `zstd`, `lz4`, and `brotli` can be decompressed far faster than `zip` files, or other older classic compression formats. The modern formats `zstd` and `brotli`, when created with high compression levels, simultaneously result in smaller file sizes and faster decompression than `zip` or `gzip`. Using these file formats can enable faster throughput for read-intensive (I/O limited) data processing. For example, Zstandard with a high compression level would be an efficient choice. Passing the data through a simple filter often limits the throughput to less than the rate of decompression, but the modern compression file formats still enable enhanced throughput.

When data are read from a single HDD, multi-threaded (parallel) data processing increases the throughput only moderately, at least for the data set benchmarked here, where the individual compressed files are larger than or comparable in size to the disk-internal memory buffer. When data are read from a SSD, mutli-threaded data processing increases the throughput significantly. Simple data processing can realistically proceed at 1 GB/s (a few TB/hr) on a standard workstation with a single multi-core CPU and a solid state drive, and at about half that rate from a single disk drive.

Whereas this study was limited to Diviner RDR data, it is apparent that modern compression utilities provide significant advantages that could be exploited for many other types of PDS data products as well and for numerical data in general.

Benchmark results and scripts are available at `https://github.com/nschorgh/PDS-Throughput/`

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. 80NSSC19K1255 issued through the Lunar Data Analysis Program.

References

Brotli compression format. URL `https://github.com/google/brotli`.

LZ4 - extremely fast compression. URL `https://github.com/lz4/lz4`.

Zstandard - fast real-time compression algorithm. URL https://github.com/facebook/zstd.

Y. Collet and M. Kucherawy. Zstandard compression and the application/zstd media type, 2018. URL https://tools.ietf.org/html/rfc8478.

T. McClanahan. Planetary Data System: Project Office Report to the Planetary Science Advisory Committee. Presentation, 2020. URL https://science.nasa.gov/science-red/s3fs-public/atoms/files/PDS_PAC_Presentation_20_03_McClanahan%20%20(1).pdf. March 9-10, 2020.

D. A. Paige et al. Diviner Lunar Radiometer observations of cold traps in the Moon's south polar region. *Science*, 330:479–482, 2010a. doi: 10.1126/science.1187726.

D. A. Paige et al. The Lunar Reconnaissance Orbiter Diviner Lunar Radiometer Experiment. *Space Sci. Rev.*, 150:125–160, 2010b. doi: 10.1007/s11214-009-9529-2.

O. Tange. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*, 36(1): 42–47, Feb 2011. doi: 10.5281/zenodo.16303. URL http://www.gnu.org/s/parallel.