HIGH-THROUGHPUT PROCESSING OF DIVINER DATA FROM THE PDS.

Norbert Schorghofer, Planetary Science Institute, Tucson, AZ & Honolulu, HI, USA (norbert@psi.edu)

Introduction: As the data volume in NASA's Planetary Data System (PDS) increases, techniques for high-throughput data processing become increasingly important. Here, several approaches to decompression and filtering of PDS data are explored and benchmarked, using the Diviner dataset as a case study.

Diviner PDS data. The Diviner instrument onboard Lunar Reconnaissance Orbiter (LRO), a radiometer that measures surface temperature [1], has been in lunar orbit for more than 11 years. The Diviner Reduced Data Record (RDR)—the Level 1b data set— is archived on the PDS in `zip`-format, generated at the default compression level 6. These data tables have about 40 MB in compressed format each, and 289 MB in uncompressed ASCII format. Each file represents 10 minutes of collected data, and consists of a few header lines and around 886,000 lines of mostly numerical entries. At this time, $\gtrsim$21 TB of `zip` files are archived.

Benchmarks: Table 1 lists the compression utilities and compression formats investigated in this study. They include classic and widely-known formats (e.g., `gzip`, `zip`, `bz2`), as well as more recently developed formats, such as Brotli (`br`, Google), `lz4`, `lzma`, and Zstandard (`zst`, Facebook) [2]. Only lossless compression is considered in this study.

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 magnetic disk drive (HDD) spins nominally at 7200 rpm and has a 23 MB buffer. It has a buffered read speed of 0.19 GB/s, which indicates how fast the drive can sustain sequential data reads. The same workstation also has a solid state drive (SSD) with a buffered read speed of 0.37 GB/s.

Compression ratios. The first test simply measures the size of the compressed files. Figure 1 includes a subset of these results. The compression ratio is the ratio of compressed to uncompressed file size. The best compression ratios are 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 take less than half the time if the data files were archived on the PDS in `lzma` instead of `zip` format.

Decompression times. The second test measures the time it takes to decompress individual files and write the uncompressed files back to the hard drive. Averages are taken over three runs, for one file. The memory's disk buffer was cleared beforehand, because the Linux kernel often keeps files in unused memory.

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 compression utilities evaluated in this study.

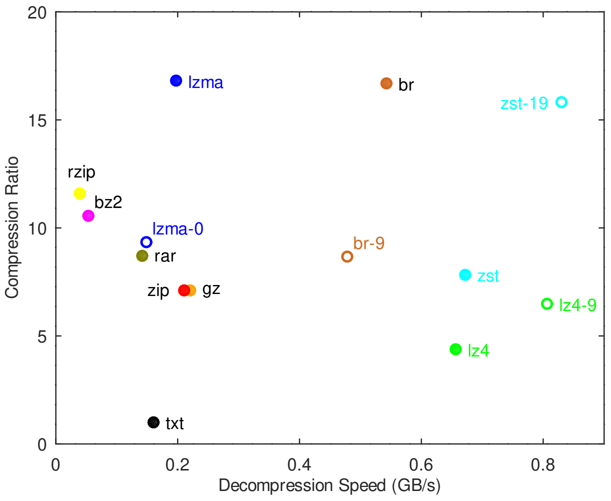

Figure 1: Performance of various compression utilities when applied to a level-1 RDR file, which has 289 MB in uncompressed (plain text, TAB) format. Filled symbols correspond to the default compression level, and empty symbols to selected non-default levels.

Table 2 shows a subset of the decompression benchmark measurements, and Figure 1 illustrates the throughput combined with the compression ratio. Plain text throughput is close to the buffered disk read speed (0.2 GB/s). The `lz4`, `zstd`, and `brotli` formats decompress several times faster than the `zip` format currently used on the PDS. Zstandard at the maximum compression level of 19 almost achieves the compression ratio of `lzma` and the decompression speed of `lz4`, simultaneously. It is the best dual purpose compression format.

File format	Compression level	Time (s) decompress & write	Time (s) decompress & filter
raw	-	1.7	1.8
brotli	11	0.52	1.5
gzip	6	1.3	1.3
lz4	1	0.43	1.4
lz4	*9*	0.35	1.4
lzma	6	1.4	1.5
lzma	*9*	1.4	1.5
zip	6	1.3	1.4
zstd	3	0.42	1.0
zstd	*19*	0.34	1.0

Table 2: Results from decompression study for Diviner RDR Level-1 data file 200908010000_RDR.TAB (289 MB). Compression levels in italics indicate non-default values. Third column: Elapsed time for decompression plus writing output back to the same disk. Fourth column: decompression plus `awk` filtering.

In a situation where files are compressed only once but downloaded and read many times, decompression time is far more important than compression time. It takes about ten times longer to create an `lzma` file than a `zip` file. Compression with `zstd` takes even longer when very high compression levels are chosen.

Decompression and filtering. Next, a file is decompressed and the output filtered, so that only a small fraction of the output is written to hard drive. For this purpose, we find and store data from a small geographic area. The longitude and latitude of the measurement are 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.

These results are included in Table 2 (last column), and they show that the throughput is slower than the decompression, although less output is written to disk. Most of the time is thus consumed either by `awk` or the mere process of parsing each line of the input file. The thread utilization of `awk` itself was never close to 100%, which suggests the bottleneck lies elsewhere. Using `mawk` (1.3.4) instead of `awk` did not improve the throughput. Using a dedicated C program with `fscanf` instead of `awk` reduced the throughput. Zstandard outcompetes all of the other formats for the combined task of decompression plus filtering.

Parallel processing of multiple files on single disk. Next, many files are decompressed and filtered in parallel. The GNU `parallel` command was used to queue the single-threaded jobs for parallel processing. For the specific parameters chosen, the total output is less than 1% of the total input file size. This analysis was limited to `lz4` (level 9), `lzma` (level 6), `zst` (level 19), and the classical `zip` and `gzip` formats (level 6). The throughput was measured as a function of the number of processes that were executed in parallel, for a hard disk drive (HDD) and a solid state drive (SSD).

For a HDD, the smallest compressed files (`lzma` and `zst` lev. 19) perform the best in terms of scaling with the number of threads. They are small enough to fit into internal disk memory, which is 23 MB. Parallel processing of the raw text files always reduced the throughput relative to sequential processing.

For the SSD, parallel processing increases throughput for all file formats. Zstandard level 19 offers the highest throughput, on a single thread as well as with multiple threads.

Conclusions: Modern compression formats result in file sizes far smaller than the Diviner `zip` files stored on the PDS. Compression with `lzma`, `brotli`, and `zstd` produces 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. 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`.

Passing the data through a simple filter based on the data columns often limits the throughput to less than the rate of decompression.

When data are read from a single HDD, multi-threaded (parallel) data processing increases the throughput only if the individual compressed files are smaller than the internal disk memory. When data are read from a SSD, mutli-threaded data processing increases the throughput significantly.

More detailed results are available at [3]. 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.

Acknowledgments: This material is based upon work supported by NASA under Grant No. 80NSSC19K1255 issued through the Lunar Data Analysis Program.

References

[1] D. A. Paige et al. The Lunar Reconnaissance Orbiter Diviner Lunar Radiometer Experiment. *Space Sci. Rev.*, 150: 125–160, 2010.

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

[3] N. Schorghofer. High-throughput processing of data from the PDS, 2020. URL `https://github.com/nschorgh/PDS-Throughput/`.