Astronomical Data Analysis Software and Systems XXXI
ASP Conference Series, Vol. 535
B. V. Hugo, R. Van Rooyen, and O. M. Smirnov, eds.

HDF5 Parallelization for Hierarchical Semi-Sparse Data Cubes

Jiří Nádvorník,[1] Petr Škoda,[2,1] and Pavel Tvrdík[1]

[1]*Faculty of Information Technology, Czech Technical University in Prague, Czechia; nadvoji1@fit.cvut.cz*

[2]*Astronomical Institute of the Czech Academy of Sciences, Ondřejov, Czechia*

Abstract. Big Data is not only about big volumes but also a higher number of dimensions of the data. For every observed astronomical object, we usually have multiple observations in different times, wavelengths, polarization, or even created by different instrument types. Intuitively, taking all of the relevant information into account will produce higher quality results for classification or clustering algorithms, rather than just focusing on a single aspect of the object. Most often we are talking about spectroscopic and photometric observations which can be combined into data cubes. With the Hierarchical Semi-Sparse data cubes (HiSS cubes) engine we combine spectral and imaging data within the HDF5 format for efficient use of machine learning algorithms and visualization. The HiSS cube ensures this efficiency by implementing an indexing mechanism within the HDF5 that also takes advantage of the native chunking feature. Preprocessing that rescales the spectral and photometry measurements, in order to be directly comparable, takes significant time. Therefore, it needs to be parallelized, and this parallelization also takes advantage of the native HDF5 parallel I/O feature. This contribution focuses on the parallel performance of the Python version `h5py` of the HDF5-based solution in the construction of the HiSS cube.

1. HiSS cube

The HiSS cube (Nádvorník et al. 2021) is a software framework that has two purposes: 1) Providing fast API for machine learning algorithms that require the spectral cubes as a stream of data. 2) Enabling fast visualization of multi-resolution spectra and images and their combined spectral cubes, mainly for purposes of validating the machine learning results. The motivation is to show that combined multi-modal data (such as spectra and images) can be processed as input for machine learning and will provide higher quality results than to run the machine learning separately on images and spectra. For the overall architecture of the framework, see Fig. 1. It describes the data flow stages comprising of *Preprocessing, HiSS-Cube, Sparse Cube VOTable* and *Contiguous data stream*.

The HDF5 [1] storage format is designed for use as a self-sustaining portable file, which can be viewed as a container file system, thereby making it easier to distribute and maintain the data. HDF5 uses Groups as alternatives to folders, Datasets as alternatives to files.

[1]https://www.hdfgroup.org/solutions/hdf5/

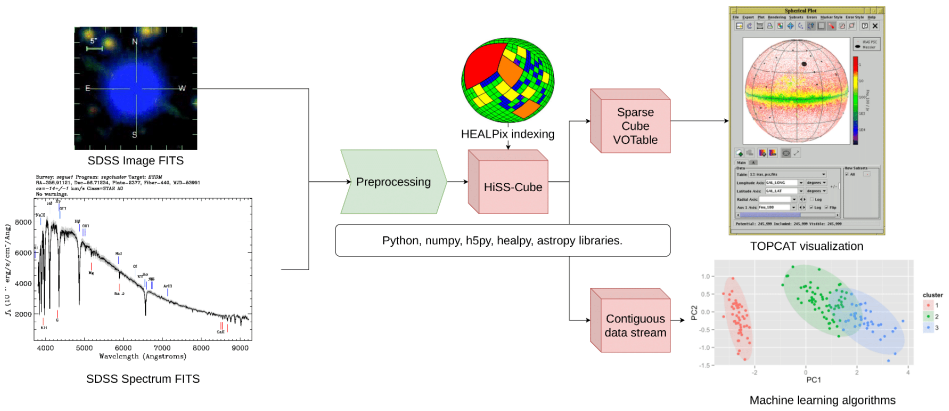

Figure 1. Data flow pipeline for HiSS-Cube. The data are combined in a single HiSS-Cube HDF5 file, which is indexed via HEALPix (Gorski et al. 2005) to allow efficient queries. For data export, we use either combination of VOTables and FITS files to ensure compatibility with existing Virtual Observatory tools or a contiguous `NumPy` array that can be utilized by a machine learning library.

For our experiments, we have used SDSS spectra and images. For example result of *Sparse Cube VOTable* visualized in TOPCAT client (Taylor 2005), see Fig. 2.

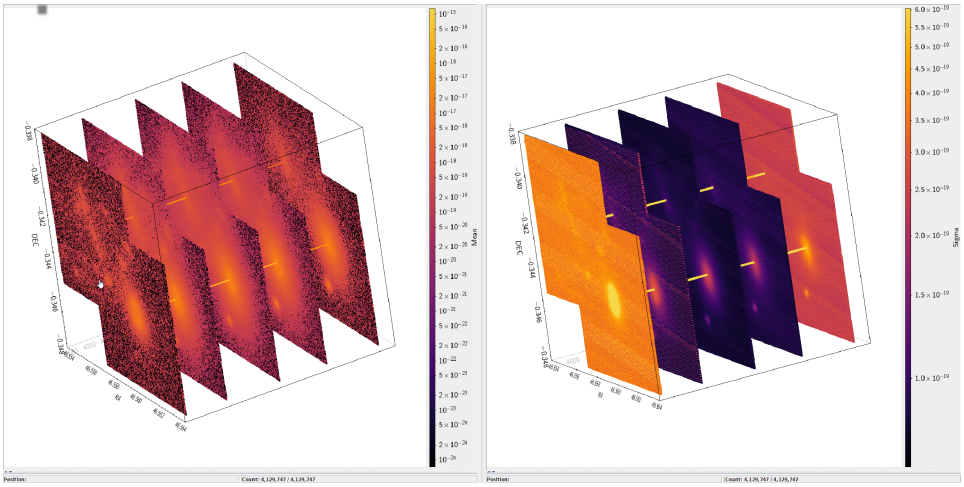

Figure 2. Screenshot of two randomly selected galaxies exported to VOTable and visualized in TOPCAT with uncertainties. The data cube contains images in five filters and spectra of both galaxies.

2. Parallelization

The preprocessing requires significant processing time, as shown on Fig. 3. Unfortunately, the *HDF5 - add cutout links* for Spectra preprocessing is not paralllelizable because of limitations of HDF5. The other phases, such as *HDF5 - write metadata* and

HDF5 - write index + data are fast, so the parallelization of preprocessing focuses only on *FITS - read+preprocess* part.

For the parallel architecture see Fig. 4. The *write groups + allocate datasets + write attributes* actions are done by a single writer in a serial access mode to the HDF5 file. `h5py` can write to the HDF5 file in collective mode where all of the processes can write simultaneously. Because of the nature of these writes, however, it is faster and more efficient if a single writer is used. Multiple writers do the parallel writes to preallocated datasets for images, and then, after synchronizing on a barrier, for spectra. After these are written, the *Master* process reopens the file again in serial mode, and writes the *links to spectra and images* as HDF5 region references.

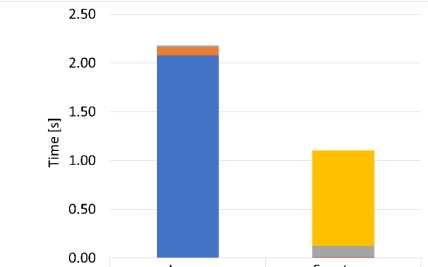

	Image	Spectrum
HDF5 - add cutout links	0.00	0.98
HDF5 - write metadata	0.02	0.11
HDF5 - write index+data	0.08	0.01
FITS - read +preprocess	2.08	0.01

Figure 3. Preprocessing sequential times in seconds.

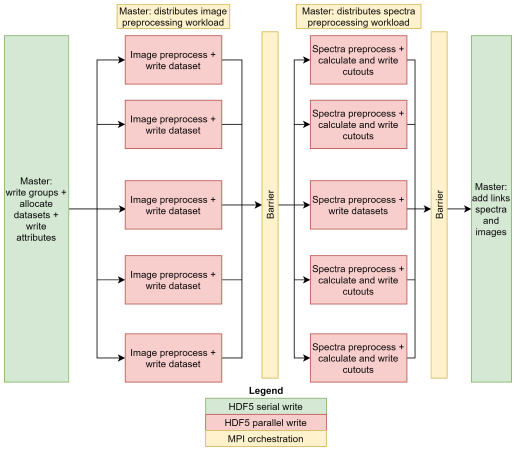

Figure 4. Parallel Master+Slave data flow architecture.

3. Results

Fig. 5 shows the activity of MPI (MPIF 1994) processes in time. The tool used is `MPI Parallel Environment`.[2] The black background means there is no communication between the processes, i.e. the processes are fully occupied by the preprocessing. The first line is the master process 0 that mostly waits for preprocessing tasks to be completed (*MPI_Mprobe*). After metadata are written (*MPI_barrier* for processes 1-8), all eight slaves perform preprocessing in parallel. The HDF5 writes (*MPI_File_write_at*) show that the HiSS-Cube is using efficiently the I/O capabilities of HDF5 with room for further improvement in the I/O Bandwidth, see Fig. 6.

4. Summary

In our experiment, the efficiency is heavily limited because we simulated parallel I/O on a single disk field. Fig. 6 shows that we reach the disk field bandwidth limit with a

[2]https://www.mcs.anl.gov/research/projects/perfvis/software/MPE/

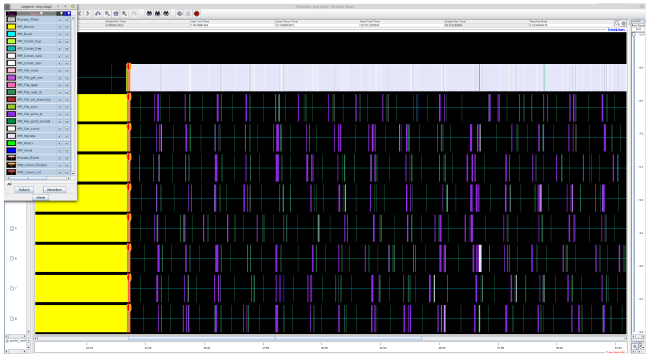

Figure 5. Post-mortem visualization of execution times of one master and eight MPI slave processes.

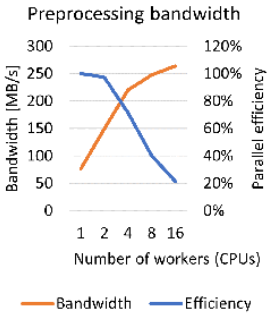

Figure 6. Preprocessing bandwidth.

reasonable number of cores per I/O node, with the FITS reading and parsing being the limiting factor. If this would be run on a true parallel I/O disk storage, the efficiency would remain above 80%, as can be seen in Fig. 6 for 2 or 3 cores. Our next steps will be optimizing the HDF5 parameters, such as the overall number of chunks used or the number of groups which will further increase the write bandwidth by reducing the number of write requests per second.

Overall, the performance of HDF5 parallel writes surpassed our expectations for write performance but other obstacles will be hopefully removed by future work, mainly with metadata operations that are performed in serial mode and do not scale well for 10TB+ HiSS cube files.

Acknowledgments. This research is supported by the project OP VVV, Research Center for Informatics, CZ.02.1.01/0.0/0.0/16_019/0000765 of the Czech Ministry of Education, Youth and Sports Astronomical Institute of the Czech Academy of Sciences is supported by the project RVO:67985815.

References

Gorski, K. M., et al. 2005, The Astrophysical Journal, 622, 759

MPIF 1994, MPI: A Message-Passing Interface Standard, Tech. rep., USA

Nádvorník, J., et al. 2021, Astronomy and Computing, 36, 100463

Taylor, M. B. 2005, in Astronomical Data Analysis Software and Systems XIV, edited by P. Shopbell, M. Britton, & R. Ebert, vol. 347 of Astronomical Society of the Pacific Conference Series, 29