

# RFC: Nagg Option for Compressing Datasets

Larry Knox  
Elena Pourmal

This document describes and makes implementation recommendations for a new feature to enable compression when creating aggregations with **nagg**.

The HDF Group developers solicit requirements, comments, and suggestions from **nagg** users before starting implementation to ensure that the feature will address users’ needs.

We assume that the reader is familiar with **nagg** and the JPSS data products.

## Table of Contents

1	Introduction.....	2
2	Compressing JPSS files with h5repack.....	3
3	Compression methods in nagg.....	7
4	Changing storage layout with nagg.....	9
5	Command line flags to support compression and chunking.....	12
6	Testing.....	15
7	Summary and Recommendations.....	16
8	Revision History.....	17
9	Appendix A-1 Effect of compression on datasets in a VIIRS-M7-SDR file.....	18
10	Appendix A-2 Effect of compression on datasets in an OMPS-NP-SDR file.....	20

## 1 Introduction

**Nagg** ([http://www.hdfgroup.org/projects/npoess/nagg\\_index.html](http://www.hdfgroup.org/projects/npoess/nagg_index.html)) is a tool for aggregating JPSS data products from existing files into new files with a different number of granules per file or different combinations of compatible products than in the original files.

The data files for many of these products are quite large. Since **nagg** writes new files, the amount of space to store both original JPSS files and newly aggregated ones is twice the original space. One can reduce the size of a JPSS file by compressing the file with one of the freely available compression utilities such as **gzip**. While this approach reduces the file size in most cases, it is not desired because one has to uncompress the entire file before reading data with the HDF5 library or tools. Instead, one can apply HDF5 compression to individual datasets stored in the file. We refer readers not familiar with HDF5 compression filters to <http://www.hdfgroup.org/HDF5/Tutor/compress.html>.

The proposed compression feature in **nagg** will reduce amount of disk space needed when the tool is used on JPSS files and will improve I/O performance during aggregation due to a smaller amount of data being written. Since **nagg** has knowledge of the JPSS file structure and the properties of the stored datasets, compression may be applied in a “smart” fashion avoiding work to compress data that cannot benefit from compression due to its size or properties (such as data type).

IDPS is planning to add compression to NPP files provided for download beginning in January of 2014. In certain cases it will be beneficial for users to change the compression method or the storage layout. This can be done with **h5repack**, but at the cost of reading and writing the file a second time. The proposed changes to **nagg** will combine the capability to change compression or storage layout with the aggregation and packaging capabilities.

This document is organized as follows. Section 2 discusses benefits of compression for JPSS data shown by using the **h5repack** utility; it also discusses disadvantages of using **h5repack**, thus justifying compression implementation in **nagg**. Section 3 considers tradeoffs between different compression methods available to **nagg**. Section 4 proposes a capability to modify chunk sizes during the aggregation. Section 5 introduces the command line flags to enable compression and new chunking properties of aggregated data. Section 6 outlines testing to be added for the new features. Section 7 summarizes the findings and recommendations of this RFC.

## 2 Compressing JPSS files with h5repack

One can use the **h5repack** <http://www.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Repack> utility to rewrite an HDF5 file applying compression and changing storage layouts for one or more datasets. **h5repack** supports all compression methods available in HDF5. Dynamically loaded compression methods will be available for **h5repack** in the HDF5 release 1.8.12 and for **nagg** in the 1.5.2 release. For further information on dynamically loaded filters, see <http://www.hdfgroup.org/HDF5/doc/Advanced/DynamicallyLoadedFilters/HDF5DynamicallyLoadedFilters.pdf>.

We used **h5repack** with sample files for six NPP products packaged with their GEO product. The files were repacked with the SZIP compression and with the Shuffle filter followed by the GZIP compression to see how effective some common compression methods were for reducing the sizes of NPP files. With one exception, the output files were smaller than the input files. The exception was the GCRIO-REDRO file, which increased in size by 2%. Other products decreased in size by a range of 14% to 69% **Table 1**.

Compression method:	SZIP	SHUF + GZIP
<i>Products in file:</i>		
<i>GCRSO-SCRIS</i>	66	56
<i>GATMO-SATMS</i>	84	79
<i>GMODO-SVM07</i>	42	35
<i>GDNBO-SVDNB</i>	40	31
<i>GONPO-SOMPS</i>	60	56
<i>GCRIO-REDRO</i>	88	74

**Table 1:** Output file size as % of original size for compressed NPP files

In order to obtain specific evidence of the effects of compression on various datasets in some NPP product files, **h5repack** applied GZIP level 7 alone and GZIP level 7 preceded by the SHUFFLE filter to 4 granule GMODO-SVM07 and GONPO-SOMPS aggregated files. The tables below show the results of repacking the GMODO-SVM07 and GONPO-SOMPS files with **h5repack**.

GMODO-SVM07\_npp\_d20121002\_t2357443\_e0003247\_...\_noaa\_ops.h5 was first compressed with “h5repack -f GZIP=7”. The file size decreased by 30% from 393458600 bytes to 276523430 bytes. The same file was compressed with “h5repack -f SHUF -f GZIP=7”. The resulting file was 65% smaller, 137765191 bytes. Table 2 shows the types and sizes of the corresponding datasets in the output files. A table showing effect of compression for each dataset in this file can be found in Appendix A-1.

<b>Datasets in /All_Data/VIIRS-M7-SDR_All group:</b>	<b>Original size in bytes (% of total file size)</b>	<b>Size and compression ratio with -f GZIP=7</b>	<b>Size and compression ratio with -f SHUF -f GZIP=7</b>
Small Datasets (12)	6,080 (.0015%)	576 (10.55:1)	588 (10.34:1)
Large Datasets (3)	68,812,800 (17.5%)	46,574,399 (1.477:1)	40,743,531 (1.688:1)
<b>Datasets in /All_Data/VIIRS-MOD-GEO_All:</b>			
Small Datasets (12)	11,936 (.0029%)	10,526 (1.124:1)	8,396 (1.422:1)
Large Datasets (9)	324,403,200 (82.4%)	229,707,941 (1.412:1)	97,183,519 (3.338:1)

**Table 2:** Summary of dataset changes with -f GZIP=7 and -f SHUF -f GZIP=7 in GMODO-SVM07 product file.

The size changes of the Large Datasets in the Geolocation group dominate the overall effect because they make up 82% of the total file size. The large datasets (> 10K bytes) in both groups in the file account for 99.9% of the file size, while the 24 small datasets account for less than 0.1%. While compressing the small datasets in many cases further reduces the file size, some of them actually increase in size, and the contribution to the overall reduction of the file size is minimal. The shuffle filter followed by the GZIP filter more than doubles the effective compression for this and other VIIRS products.

The same procedure was repeated with an OMPS-NP-SDR file GONPO-SOMPS\_npp\_d20120508\_t0333549\_e0336247\_b02735\_c20130809150538592361\_XXXX\_XXX.h5. This file was smaller than the VIIRS file with much smaller datasets, but the size of the output file with Shuffle + GZIP was still about 45% smaller than the original file. Table 3 below shows the types and sizes of the corresponding datasets in the output files. A table showing effect of compression for each dataset can be found in Appendix A-2.

Datasets in /All_Data/OMPS-NP-SDR_All:	Original size in Bytes (% of total file size)	Size and Compression ratio with -f GZIP=7	Size and Compression ratio with -f SHUF -f GZIP=7
Small Datasets (15)	888 (0.47%)	216 (4.1:1)	254 (3.5:1)
Large Datasets (6)	179,200 (95.09%)	5909 (30.3:1)	5333 (33.6:1)
Datasets in /All_Data/OMPS-NP-GEO_All:			
Small Datasets (21)	8356 (4.43%)	949 (8.8:1)	1470 (5.68:1)
Large Datasets (0)			

**Table 3:** Summary of dataset changes with -f GZIP=7 and -f SHUF -f GZIP=7 in GONPO-SOMPS product file.

The files for the OMPS-NP-SDR and OMPS-NP-GEO products are much smaller than the VIIRS files. The large datasets (>10K bytes) are also much smaller than those in the VIIRS files. Compression has a less pronounced effect on the files because the size of the metadata is similar for both products and both GEO products. As a result, compression for the OMPS product is applied to at most 50% of the file, where for the VIIRS products a slightly larger amount of metadata accounts for only .05% of the file size.

IDPS is planning to use a dataset size threshold for applying compression, currently 1K, but 100K is being considered. Compression will reduce file size, but if IDPS goes to 100K as the threshold for compression it is possible none of the datasets in this file will be big enough to be compressed. Compressing all of the datasets in the GONPO-SOMPS file results in an output file 55.5% as big as the original file. Compressing only the large datasets results in 57.6% of the original size.

Compression of an Ancillary file was attempted with GZIP and SZIP, but effectively produced no change. There was only one sizeable dataset, which used nearly all of the possible values for the U8BE type, more or less randomly distributed.

While **h5repack** provides the needed features (compression and chunking) there are two obvious disadvantages in using **h5repack** vs. directly writing compressed datasets with **nagg**:

1. First, **nagg** reads data from the original files and writes aggregation files, and then **h5repack** reads data again and writes it applying compression. The I/O operations will be doubled. If both **nagg** and **h5repack** are run on a collection of the JPSS files the disadvantage of this approach is obvious.
2. Not all data in a JPSS file will benefit from compression. For example, datasets with the region references stored under the /Data\_Products group do not benefit from compression at all because data of the HDF5 reference type cannot be compressed. If **h5repack** is applied to the whole file, performance of the tool will suffer. A user will need to specifically exclude the datasets with the references making the whole process more complex and cumbersome.

The optimal compression method for a dataset may vary according to the dataset's datatype and the distribution patterns of values in the dataset. It is also possible that users will want to use compression software unknown to **nagg**. **Nagg** can provide users with the flexibility to specify different compression methods for various datasets, and should avoid compression for very small datasets. Furthermore, while it will be appropriate for IDPS to choose one combination of dataset storage layouts and compression methods that will maximize efficiency for delivering NPP data files, users and applications may need or want to convert their files to use different layouts and methods.

### 3 Compression methods in nagg

The experiments in section 2 considered the effects of GZIP and Shuffle + GZIP on the size of the files and individual datasets. A set of tests was run with results in the table below with those same filters and with the SZIP filter. The VIIRS-DNB-SDR product was chosen to avoid fill granules, which typically have datasets where every value is identical and for which no data may be written, reducing the time to write the dataset. Files with 4, 16, and 72 granule aggregations were tested with **h5repack** and custom versions of **nagg** built to apply the same specific filters.

**Nagg** and **h5repack** with the same filters and parameters produce the same size files. **Nagg** takes longer on small files because it constructs a table of granules in the input files, while **h5repack** takes longer for the larger files. **Nagg** is significantly faster than **h5repack** for the largest aggregation. The reason for the longer **h5repack** time to process larger files is under investigation.

Shuffle doubles the effectiveness of the gzip compression, and at the same time reduces the processing times for both **nagg** and **h5repack**. It is not effective and appears to be detrimental in combination with szip. Szip wins for fastest processing time; shuffle + gzip for effective compression. Possibly the latter is due to the narrow range of values in geolocation file datasets relative to the range of numbers for the datatypes.

IDPS is planning to apply compression to datasets larger than some threshold (100K is being discussed). When granules are aggregated, datasets that were uncompressed may cross the threshold. **Nagg** could calculate the size of the output dataset, and compress any datasets that cross the threshold due to aggregation. It could also apply compression according to a user threshold supplied on the command line

The 2x16 granule file was also processed with an experimental version of **nagg** using the HDF5 example bzip2 dynamically loaded filter. The resulting file was 66% of the original size with a runtime of 6:41. The compression ratio is similar to GZIP. While the run time will not generate much excitement, the test demonstrates that **nagg** can write output datasets using a dynamically loaded filter.

The discussion above is summarized in Table 4.

	No compression	gzip level 7	shuffle + gzip level 7	-f SZIP=32,NN	-f SHUF -f SZIP=32,NN	-f SHUF -f SZIP= 32,EC
2x4 granules: file size	.574	.415 (72%)	.183 (25%)	.230 (40%)	.269 (47%)	.516 (90%)
h5repack run time	0:02	0:44	0:21	0:07	0:08	0:07
Nagg run time	0:04	0:44	0:24	0:06	0:07	Not tested
2x16 granules: file size	2.298	1.649 (72%)	.718 (31%)	.914 (40%)	1.064 (46%)	2.060 (90%)
h5repack run time	0:18	3:03	1:30	0:25	0:30	0:25
nagg run time	0:33	3:03	1:41	0:26	0:31	Not Tested
2x72 granules: file size	10.340	7.496 (72%)	3.333 (32%)	4.199 (41%)	4.876 (47%)	9.313 (90%)
h5repack run time	1:56	20:24	13:44	9:49	9:58	9:59
nagg run time	2:38	13:47	7:43	2:53	Error <sup>1</sup>	Not tested

**Table 4:** Processing times for compression on GDNBO-SVDNB files of various sizes with **nagg** and **h5repack** on a 64-bit Linux machine (platypus). Sizes are in GB, times in minutes:seconds, and % of original size is displayed.

<sup>1</sup> HDF5 returns an error indicating that Szip decompression failed.



## 4 Changing storage layout with nagg

The HDF5 datasets that describe NPP product granule use a special storage layout called chunked layout or chunk storage (see “HDF5 Datasets” chapter in the HDF5 User’s Guide [http://www.hdfgroup.org/HDF5/doc/UG/UG\\_frame10Datasets.html](http://www.hdfgroup.org/HDF5/doc/UG/UG_frame10Datasets.html)). When chunk storage is used, a dataset is logically divided into chunks or tiles (in 2-D case), and each chunk is stored as a contiguous sequence of bytes in an HDF5 file.

Chunk storage is required if more data will be added to a dataset and/or to store compressed data. Chunking is also used to improve performance when reading and writing a subset of data stored in an HDF5 file, especially if data is compressed. For more information about HDF5 chunking and how chunking parameters such as chunk size and size of the chunk cache affect I/O performance we refer the reader to <http://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/index.html>. In this section we just outline why the current chunk sizes may affect sub-setting of aggregated datasets.

For each HDF5 dataset that is used to describe an NPP granule, the chunk size for that dataset is equal to the size of the dataset. For example, this output occurs for the Radiance dataset when running “h5dump -p -H

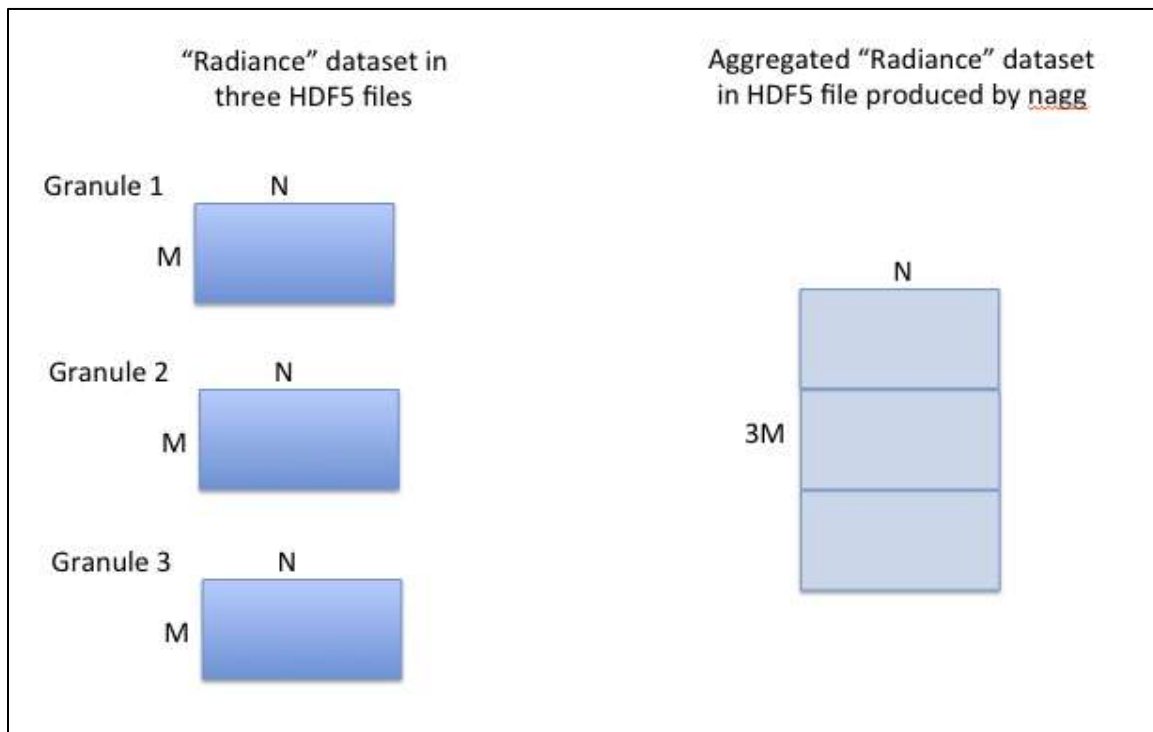
SVDNB\_npp\_d20130727\_t0000538\_e0002179\_b09046\_c20130727063722389520\_noaa\_ops.h5”:

```

DATASET "Radiance" {
  DATATYPE H5T_IEEE_F32BE
  DATASPACE SIMPLE { ( 768, 4064 ) / ( H5S_UNLIMITED, H5S_UNLIMITED ) }
  STORAGE_LAYOUT {
    CHUNKED ( 768, 4064 )
    SIZE 12484608
  }
  ...
}
```

Note that the current dimensions of the dataspace are the same as the dimensions of the chunk.

When **nagg** aggregates NPP granules it doesn’t currently change the chunk sizes. This is illustrated in Figure 1.



**Figure 1:** **nagg** aggregates data using default chunk size. Aggregated dataset is stored in three chunks.

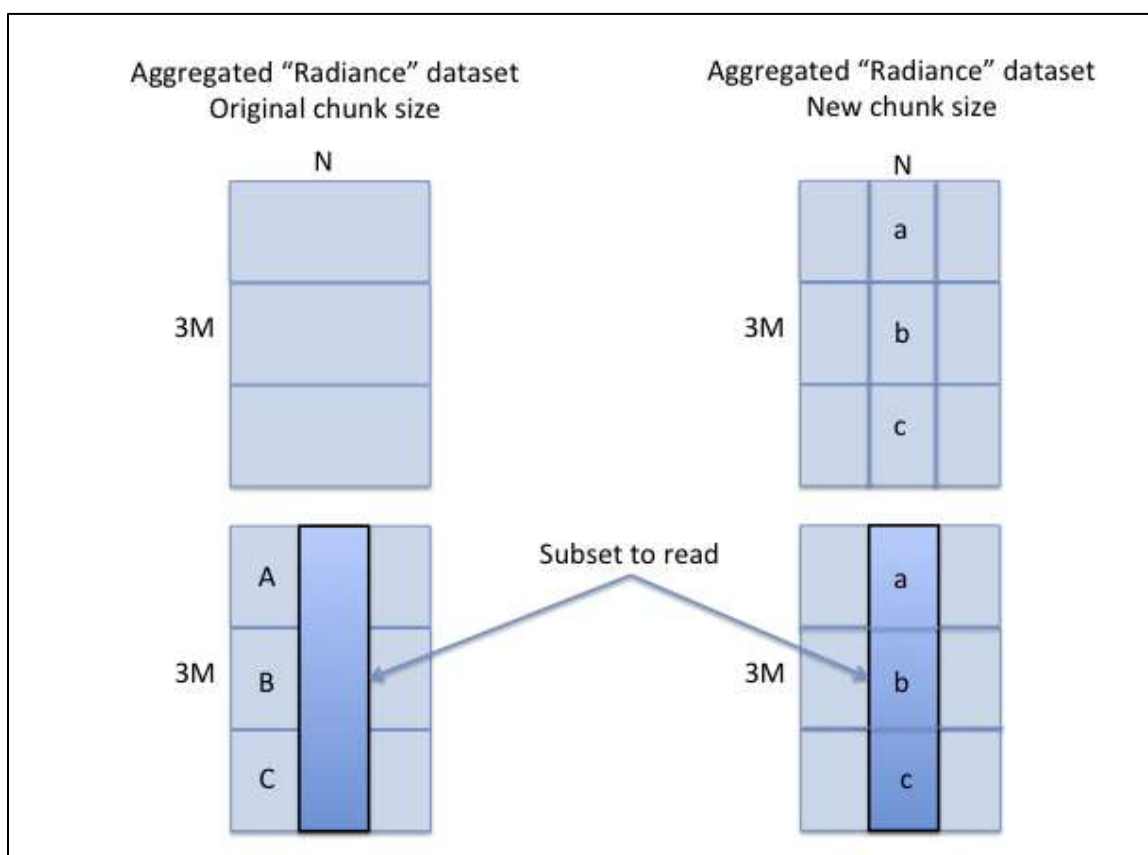
The chunk size for "Radiance" dataset is  $M \times N$  and corresponds to the size of the dataset in a granule file. When **nagg** aggregates granules, it creates a "Radiance" dataset in a new HDF5 file. The dimensions of the dataset will be  $3M \times N$  and chunk size will still be  $M \times N$  as in the original dataset.

Since granules can be pretty big (The VIIRS-IMG-GTM-EDR-GEO product has seven 50MB datasets, each stored as a single  $1541 \times 8241$  chunk) and compressed (in the future) I/O performance may suffer when sub-setting is performed on the read. The following discussion outlines the issue.

HDF5 performs I/O operations on the whole chunk. This means that when a subset of a dataset is read, all chunks that store data from that subset have to be read. Figure 2 illustrates the idea.

On the left, the "Radiance" dataset is stored in three chunks A, B, and C. When the highlighted subset is read, all three chunks A, B, and C must be read separately. The HDF5 library reads a chunk, selects requested data from each chunk and places it in the application buffer. I.e., in this particular case the HDF5 library has to read all data in order to read the requested subset.

On the right, the same dataset is stored using nine chunks. The subset is stored in the chunks a, b, and c. When library reads the subset, three times less data is read. If "Radiance" dataset is compressed, then clearly less CPU time is required to uncompress the data.



**Figure 2:** Reading a subset from the dataset with different chunk sizes. Reading a highlighted subset from the dataset stored in three chunks will require reading all data (on the left), while reading the same subset from the dataset stored in nine chunks will require reading only the third of total data.

We recommend adding a capability to `nagg` to specify chunk size when aggregating data. This will allow a user to tune HDF5 storage for better I/O for sub-setting data from the aggregated datasets.

## 5 Command line flags to support compression and chunking

Command options to enable compression and new storage layout properties of aggregated data will be “borrowed” to match the long form of the **h5repack** utility options. The single character **h5repack** options will not be used since one of them (-l) is already in use for the **nagg** utility. If no compression or storage layout property is specified, the properties of the datasets in the input files will be preserved.

Examples of the **nagg** command line for changing layout and adding compression:

1. `nagg -n4 -tSVDBN --minimum=10240 --filter=SHUF --filter=GZIP=7  
SVDNB_npp_d20130727_t0*.h5`

This command will create packaged 4 granule aggregations of VIIRS-DNB-SDR and VIIRS-DNB-GEO products from files matching the SVDNB\_npp\_d20130727\_t000\*.h5 pattern. All datasets of at least 10240 bytes size will be compressed with the Shuffle and GZIP filters.

2. `nagg -n4 -tSVDBN --layout=/All_Data/VIIRS-DNB-GEO_All/Height,/All_Data/VIIRS-DNB-GEO_All/Latitude,/All_Data/VIIRS-DNB-GEO_All/Longitude,/All_Data/VIIRS-DNB-GEO_All/LunarAzimuthAngle,/All_Data/VIIRS-DNB-GEO_All/LunarZenithAngle,/All_Data/VIIRS-DNB-GEO_All/QF2_VIIRSSDRGEO,/All_Data/VIIRS-DNB-GEO_All/SCAttitude,/All_Data/VIIRS-DNB-GEO_All/SCPosition,/All_Data/VIIRS-DNB-GEO_All/SCVelocity,/All_Data/VIIRS-DNB-GEO_All/SatelliteAzimuthAngle,/All_Data/VIIRS-DNB-GEO_All/SatelliteRange,/All_Data/VIIRS-DNB-GEO_All/SatelliteZenithAngle,/All_Data/VIIRS-DNB-GEO_All/SolarAzimuthAngle,/All_Data/VIIRS-DNB-GEO_All/SolarZenithAngle,/All_Data/VIIRS-DNB-SDR_All/QF1_VIIRSDNBSDR,/All_Data/VIIRS-DNB-SDR_All/Radiance:CHUNK=768x1016 --minimum=10240 --filter=SZIP=32,NN SVDNB_npp_d20130727_t0*.h5`

Command 2 will create the same packaged 4 granule aggregations as command 1. This command also changes the storage layout for improved performance when subsetting only a part of the swath, and applies SZIP compression instead of Shuffle and GZIP. The datasets to which the layout change is to be applied are specified by the list of datasets in --layout=<list of datasets>: CHUNK=768x1016.

3. `nagg -n4 -tSVDBN --layout=CHUNK=768x1016 --minimum=10240 --filter=SZIP=32,NN GDNBO-SVDNB_npp_d20130727_t0035017_e0057470_b09046_c20130731205748151135_XXXX_XXX.h5.`

Command 3 will have the same effect as command 2, but with a potential for unexpected consequences. The new layout is exactly ¼ of the original chunk and granule size for all the datasets listed in command 2. However, there are several other datasets with a different rank, and the h5repack behavior in that case is to issue a warning and create the dataset with the original properties of the dataset. Another unexpected consequence encountered is that in the case where the --layout chunk rank matches a dataset but has one or more dimensions that are larger than the original dataset, the larger chunk size will be used and extra storage space allocated, increasing the file size. Currently it is expected that nagg will follow this h5repack behavior. For these files the result was a .01% increase in the size of the output file using command 3 instead of command 2, but users should be aware of the possible consequences.

The following flags are proposed for **nagg**:

**--minimum=*size***

Apply filter(s) only to objects whose size in bytes is equal to or greater than *size*.  
*size* must be an integer greater than one ( 1 ).

*Default:* If no size is specified, a threshold of 1024 bytes is assumed.

**--filter=*filter***

Filter type

*filter* is a string of the following format:

*list\_of\_objects* : *name\_of\_filter*[=*filter\_parameters*]

*list\_of\_objects* is a comma separated list of object names meaning apply the filter(s) only to those objects. If no object names are specified, the filter is applied to all objects.

*name\_of\_filter* can be one of the following:

GZIP, to apply the HDF5 GZIP filter (GZIP compression)  
 SZIP, to apply the HDF5 SZIP filter (SZIP compression)  
 SHUF, to apply the HDF5 shuffle filter  
 FLET, to apply the HDF5 checksum filter  
 NBIT, to apply the HDF5 N-bit filter  
 SOFF, to apply the HDF5 scale/offset filter  
 NONE, to remove any filter(s)

*filter\_parameters* conveys optional compression information:

GZIP=*deflation\_level* from 1-9  
 SZIP=*pixels\_per\_block*,*coding\_method*  
     *pixels\_per\_block* is a even number in the range 2-32.  
     *coding\_method* is EC or NN.  
 SHUF (no parameter)  
 FLET (no parameter)  
 NBIT (no parameter)  
 SOFF=*scale\_factor*,*scale\_type*  
     *scale\_factor* is an integer.  
     *scale\_type* is either IN or DS.  
 NONE (no parameter)

**--layout=*layout***

Layout type

*layout* is a string of the following format:

*list\_of\_objects* : *layout\_type*[=*layout\_parameters*]

*list\_of\_objects* is a comma separated list of object names, meaning that layout information is supplied for those objects. If no object names are specified, the layout is applied to all objects.

*layout\_type* can be one of the following:

CHUNK, to apply chunking layout

COMPA, to apply compact layout

CONTI, to apply contiguous layout

*layout\_parameters* is present only in the `CHUNK` case and specifies the chunk size of each dimension in the following format with no intervening spaces:

*dim\_1* × *dim\_2* × ... *dim\_n*

## 6 Testing

Tests should be run with a variety of NPP products to verify the following:

1. Output files are valid HDF5 files and can be opened with HDF5 tools. Note that displaying data with h5dump for a dynamically loaded filter will require setting the environment variables `HDF5_PLUGIN_PATH` to the directory containing the filter plugin and `LD_LIBRARY_PATH` to the directory containing the filter library file.
2. Filter and layout changes were applied to output files.
3. Output file size is in line with a generally expected range for the filter and the product. Results that differ significantly from expectations based on previous results should be noted, investigated and potentially documented as results will vary with different combinations of products and filters.
4. Processing times similarly meet or modify expectations.
5. Compression with a filter can be removed as well as added.
6. A performance test processing compressed and uncompressed files created by nagg should be devised.

## 7 Summary and Recommendations

- Compression (which requires a chunked storage layout) significantly reduces the size of many NPP product files. While GCRI-REDRO and Ancillary files do not compress well, size reduction for randomly selected NPP product files with SZIP/SHUFFLE + GZIP ranged from 16%/21% for GATMO-SATMS to 60%/69% for GDNBO-SVDNB.
- IDPS sets the dataset chunk sizes to the granule size. These can be up to about 50 MB. IDPS is also planning to add compression for most of its NPP files, probably with the same compression method for all products. Users with applications that regularly access subsets of granule data may want to change the storage layout or compression method to achieve better I/O performance.
- While compression and layout changes could be accomplished with **h5repack**, if aggregation or packaging changes are also desired, using both **h5repack** and **nagg** would require multiple commands and 2 new complete copies of the data. Adding the capability to **nagg** to change dataset compression and layout while also changing aggregation and packaging arrangements should provide benefit to users by reducing the necessary commands to run and the number of times the data must be read and written. We recommend adding these capabilities to the **nagg** tool as described herein.



## 8 Revision History

<i>August 13, 2013:</i>	Version 1 available for internal review.
<i>August 19, 2013</i>	Version 2 was edited and formatted; sent for internal review
<i>August 20, 2013</i>	Version 3; sent to JPSS
<i>October 21, 2013</i>	Version 4; checked into subversion repository.

## 9 Appendix A-1 Effect of compression on datasets in a VIIRS-M7-SDR file

This table shows the effect of h5repack with the GZIP filter and with the shuffle+GZIP filters on the size of all datasets in a packaged file with 4 VIIRS-M7-SDR granules and 4 VIIRS-MOD-GEO granules.

Total size of file: 393,461,512 bytes.

File metadata: 212,848 bytes

Raw data: 393,236,032 bytes

Using the shuffle filter before GZIP compression doubles the effective compression for this product. The primary gain is in the **VIIRS-MOD-GEO** datasets, probably because the values are in a narrow range and relatively small compared to the maximum possible values for the F32 data type. Adding the shuffle filter also significantly reduces the processing time required to add GZIP compression to the file.

The size of the largest of the “small” datasets is 3,072 bytes, and of the smallest of the “large” datasets is 9,830,400 bytes. Minimum size thresholds for compressing datasets from 3K to 9,000K bytes would result in compressing the larger datasets in this file. Compressing only the large files has no significant effect on the effective compression for the entire file for these products.

Datasets in /All_Data/VIIRS-M7-SDR_all	Original size (Bytes)	Size and Compression ratio with -f GZIP=7	Size and Compression ratio with -f SHUF -f GZIP=7
<b>Small Datasets:</b>			
ModeGran	4	36 (0.111:1)	36 (0.111:1)
ModeScan	192	48 (4.000:1)	48 (4.000:1)
NumberOfBadChecksums	768	48 (16.000:1)	48 (16.000:1)
NumberOfDiscardedPkts	768	48 (16.000:1)	48 (16.000:1)
NumberOfMissingPkts	768	48 (16.000:1)	48 (16.000:1)
NumberOfScans	16	48 (0.333:1)	48 (0.333:1)
PadByte1	12	44 (0.273:1)	44 (0.273:1)
QF2_SCAN_SDR	192	52 (3.692:1)	52 (3.692:1)
QF3_SCAN_RDR	192	48 (4.000:1)	48 (4.000:1)
QF4_SCAN_SDR	3072	60 (51.200:1)	60 (51.200:1)
QF5_GRAN_BADDETECTOR	64	44 (1.455:1)	44 (1.455:1)
ReflectanceFactors	32	52 (0.615:1)	64 (0.500:1)
<b>Large Datasets:</b>			
QF1_VIIRSMBANDSDR	9830400	170931 (57.511:1)	170931 (57.511:1)
Radiance	39321600	30003876 (1.311:1)	25837150 (1.522:1)
Reflectance	19660800	16399592 (1.199:1)	14735450 (1.334:1)

<b>Datasets in /All_Data/ VIIRS-MOD-GEO_All</b>			
<b>Small Datasets:</b>			
MidTime	1536	989 (1.553:1)	748 (2.053:1)
ModeGran	4	36 (0.111:1)	36 (0.111:1)
ModeScan	192	48 (4.000:1)	48 (4.000:1)
NumberOfScans	16	48 (0.333:1)	48 (0.333:1)
PadByte1	12	44 (0.273:1)	44 (0.273:1)
QF1_SCAN_VIIRSSDRGEO	192	52 (3.692:1)	52 (3.692:1)
SCAttitude	2304	2294 (1.004:1)	1983 (1.162:1)
SCPosition	2304	2132 (1.081:1)	1623 (1.420:1)
SCSolarAzimuthAngle	768	812 (0.946:1)	662 (1.160:1)
SCSolarZenithAngle	768	809 (0.949:1)	547 (1.404:1)
SCVelocity	2304	2271 (1.015:1)	1856 (1.241:1)
StartTime	1536	991 (1.550:1)	749 (2.051:1)
<b>Large Datasets:</b>			
Height	39321600	317350 (123.906:1)	256657 (153.207:1)
Latitude	39321600	30291485 (1.298:1)	12638737 (3.111:1)
Longitude	39321600	31961261 (1.230:1)	12111248 (3.247:1)
QF2_VIIRSSDRGEO	9830400	9616 (1022.296:1)	9616 (1022.296:1)
SatelliteAzimuthAngle	39321600	33272093 (1.182:1)	15279686 (2.573:1)
SatelliteRange	39321600	32856607 (1.197:1)	14091418 (2.790:1)
SatelliteZenithAngle	39321600	34629645 (1.135:1)	14965389 (2.628:1)
SolarAzimuthAngle	39321600	34312947 (1.146:1)	15279686 (2.573:1)
SolarZenithAngle	39321600	32056937 (1.227:1)	12551082 (3.133:1)

## 10 Appendix A-2 Effect of compression on datasets in an OMPS-NP-SDR file

This table shows the effect of h5repack with the GZIP filter and with the shuffle+GZIP filters on the size of all datasets in a packaged file with 4 OMPS-NP-SDR granules and 4 OMPS-NP-GEO granules.

Total size of file: 409,192 bytes.

File metadata: 204,280 bytes

Raw data: 190,796 bytes

Compression ratios are actually higher for this file than for the VIIRS file, but the higher % of metadata, which is not compressed, reduces the effect on the overall file size. The addition of the shuffle filter results in better compression, but the improvement is less dramatic than in the VIIRS file.

The size of the largest of the “small” datasets is 1,600 bytes, and of the smallest of the “large” datasets is 16,000 bytes. Minimum size thresholds for compressing datasets from 1.6K to 15K bytes would result in compressing the larger datasets in this file. Compressing only the large files reduces the effective compression by 2% for these products.

Datasets in /All_Data/ OMPS-NP -SDR_All	Original size (Bytes)	Size and Compression ratio with -f GZIP=7	Size and Compression ratio with -f SHUF -f GZIP=7
<b>Small Datasets:</b>			
BadCal	4	12 (0.333:1)	12 (0.333:1)
Bias1	16	14 (1.143:1)	19 (0.842:1)
DarkExposeEarth	32	19 (1.684:1)	30 (1.067:1)
GainTblVersion	16	14 (1.143:1)	16 (1.000:1)
LinearityTblVersion	16	14 (1.143:1)	16 (1.000:1)
NPLinearCorrection	20	14 (1.429:1)	14 (1.429:1)
NumberOfFOVs	8	12 (0.667:1)	14 (0.571:1)
NumberOfSpectralPixels	8	12 (0.667:1)	14 (0.571:1)
NumberOfSwaths	8	12 (0.667:1)	14 (0.571:1)
QualityEarth	40	16 (2.500:1)	19 (2.105:1)
RadFlag	400	23 (17.391:1)	30 (13.333:1)
SAA	20	14 (1.429:1)	14 (1.429:1)
SolarEclipse	100	15 (6.667:1)	15 (6.667:1)
SunGlint	100	12 (8.333:1)	12 (8.333:1)
WaveFlag	100	15 (6.667:1)	15 (6.667:1)
<b>Large Datasets:</b>			
Cal	16000	645 (24.806:1)	586 (27.304:1)
DarkCurrentEarth	19200	622 (30.868:1)	655 (29.313:1)
RadianceEarth	80000	2441 (32.773:1)	2155 (37.123:1)
SmearDataEarth	16000	798 (20.050:1)	656 (24.390:1)
SolarFlux	16000	657 (24.353:1)	616 (25.974:1)
Wavelengths	32000	746 (42.895:1)	665 (48.120:1)
<b>Datasets in /All_Data/ OMPS-NP-GEO_All</b>			
<b>Small Datasets:</b>			
Height	400	28 (14.286:1)	40 (10.000:1)

Latitude	400	37 (10.811:1)	55 (7.273:1)
LatitudeCorners	1600	75 (21.333:1)	104 (15.385:1)
Longitude	400	37 (10.811:1)	56 (7.143:1)
LongitudeCorners	1600	76 (21.053:1)	103 (15.534:1)
MidTime	160	47 (3.404:1)	65 (2.462:1)
MoonVector	240	69 (3.478:1)	79 (3.038:1)
NumberOfFOVs	8	12 (0.667:1)	14 (0.571:1)
NumberOfSwaths	8	12 (0.667:1)	14 (0.571:1)
QF1_OMPSNPGeo	20	15 (1.333:1)	15 (1.333:1)
RelativeAzimuthAngle	400	36 (11.111:1)	54 (7.407:1)
SCAttitude	240	69 (3.478:1)	79 (3.038:1)
SCPosition	240	71 (3.380:1)	83 (2.892:1)
SCVelocity	240	68 (3.529:1)	80 (3.000:1)
SatelliteAzimuthAngle	400	37 (10.811:1)	55 (7.273:1)
SatelliteRange	400	34 (11.765:1)	48 (8.333:1)
SatelliteZenithAngle	400	36 (11.111:1)	51 (7.843:1)
SolarAzimuthAngle	400	36 (11.111:1)	55 (7.273:1)
SolarZenithAngle	400	36 (11.111:1)	55 (7.273:1)
StartTime	160	47 (3.404:1)	69 (2.319:1)
SunVector	240	71 (3.380:1)	80 (3.000:1)