

RFC: Support for multiple products in nagg

Larry Knox
Albert Cheng

Previous nagg prototypes aggregated only one product plus an associated geoproduct. The next version will support multiple compatible input and output products. This document presents details of the operation of nagg and changes to be made for aggregating multiple products.

1 Purpose

This describes the addition of the feature of support multiple sensor data products to the Nagg tool. The output files can be in the Packaged and Unpackaged formats. Section 2 describes the Functionality specification of the multiple data products features and the two output formats. Section 3 describes the Implementation design. Section 4 describes the Implementation details.

2 Functionality Specification

Two new functions are added to the Nagg tool, the multiple sensor data products and the two output file formats. We describe the output formats first since the multiple products build on top of the two output formats.

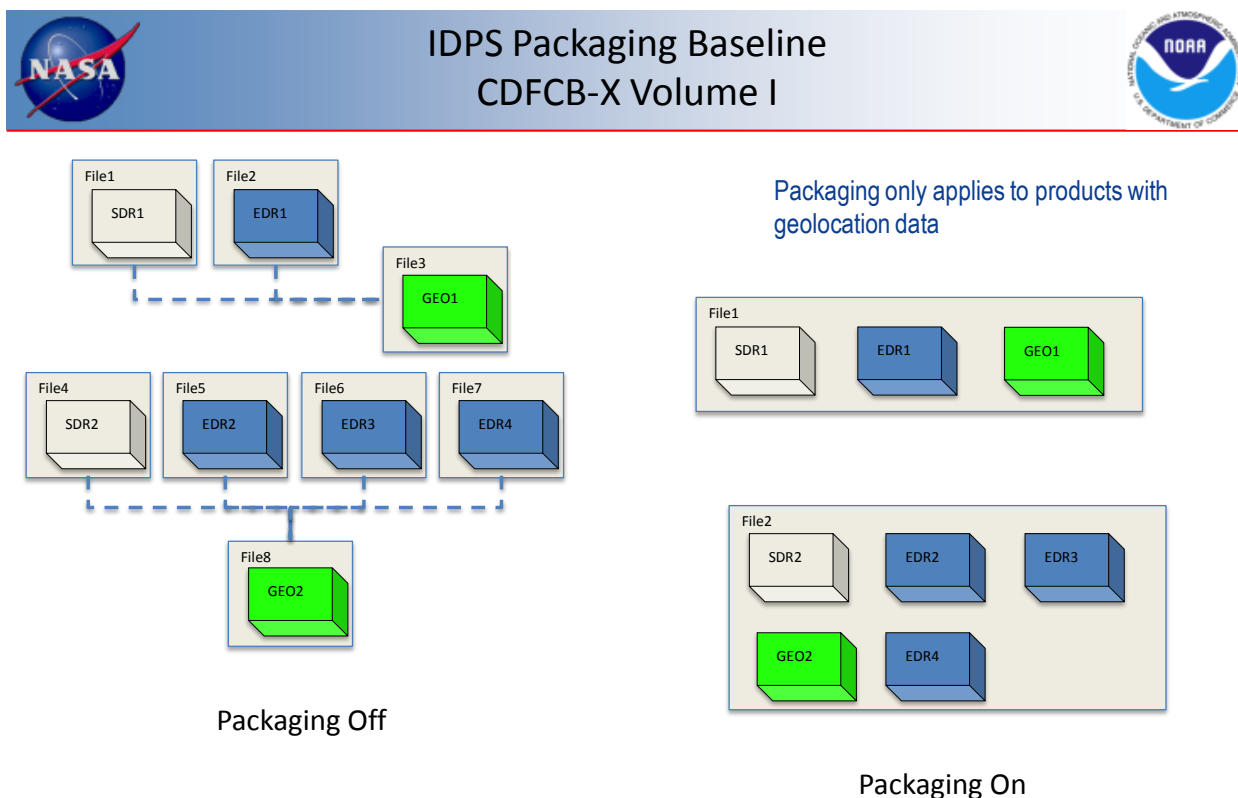
2.1 Packaged and Unpackaged file Formats

There are two data product file formats according to section “**3.5.7 Geolocation Packaging (From CDFCB Vol1.)**”.

There are two options for receiving Geolocation data for NPP/NPOESS Data Products5:

1. Packaging Off – For all data products with the same geolocation data, deliver only one geolocation HDF5 file and reference the geolocation HDF5 file from each corresponding data product HDF5 file per request. Each data product requested is also delivered in a separate HDF5 file
2. Packaging On – Package all data products sharing the same geolocation data in a single HDF5 file and include their corresponding geolocation data in the HDF5 file with the data products per request.

The figure below illustrates the two file formats where “Packages on” is equivalent to Packaged files and “Packaging off” is to Packaged off files.



11/4/11

PROPOSED nagg utility - DRAFT FOR
DISCUSSION

12

Figure 1: Packaged and Unpackaged File Formats

2.2 The Simple (-S) Option of the Nagg Tool

The Nagg tool has the default to produce output files in Packaged format, that is granules of sensor products and their corresponding Geolocation product of the same time duration are aggregated into one file.

When the Simple (-S) option is specified, the Nagg tool produce in Unpackaged format, that is one file each for the granules of each sensor product and their corresponding Geolocation product.

Examples

For simplicity, the following assume all input granules are aggregated into 1 bucket time period. If there are more granules, more sets of files of the same layout, are produced. Other Nagg options, such as the required -n and input files are not shown here.

```
% nagg -t EDR4 ...
```

Produces 1 file containing EDR4 and GEO2 granules.

```
% nagg -S -t EDR4 ...
```

Produces 1 file containing EDR4 granules and 1 file of GEO2 granules.

2.3 Multiple Products List

The `-t` (product) option may take an argument of a list of multiple sensor product IDs, separated by commas. The list of products must be compatible, that is, they all use the same Geolocation product. The Nagg tool will aggregate the granules of all given sensor products and the Geolocation product into the output files, in either Packaged or Unpackaged file formats.

A Nagg Tool Extension

If for some reasons that the users do not want the Geolocation product in the output files or the Geolocation product granules are missing from the input files, they may use the “-g no” option to tell the Nagg tool not to look for nor produce Geolocation granules.

Geolocation Product Only Output

If the users want to aggregate only Geolocation granules and do not care for any sensor products, they may specify the Geolocation product ID via `-g` and do not use the `-t` option at all. For example,

```
% nagg -g GEO2 ...
```

Produces 1 file containing GEO2 granules.

2.4 Formal Description of the two options

`-t list`

list specifies a comma separated list of NPP sensor record type mnemonics. Unless `-S` is specified the granule types will be packaged together. Types must be compatible to be packaged together. (Use `-h` to list valid package groupings). If only the Geolocation granules are aggregated, this option should not be used and the `-g` option should specify the geo-product-ID.

`-g criterion`

criterion is the criterion for searching the Geolocation granules

no | 0: aggregate product files without Geolocation input or output

yes | 1: allow approximate matching of Geolocation input filenames (default)

strict | 2: require exact matching of geolocation input filenames

geo-product-ID: only the Geolocation granules of *geo-product-ID* are aggregated. When this is specified, `-t` should not be used.

`-S`

Simple aggregates are produced. Each type is packaged separately. Default is not set, that is, all types are packaged together.

2.5 Examples Using the Three Options

```
% nagg -t SDR1,EDR1 SDR1*.h5 EDR1*.h5
```

One output file containing SDR1, EDR1 and GEO1

```
% nagg -t SDR1,EDR1 -S SDR1*.h5 EDR1*.h5
```

Three output files containing SDR1, EDR1 and GEO1 each in separate files

```
% nagg -t SDR1,EDR1 -g no SDR1*.h5 EDR1*.h5
```

One output file containing SDR1 and EDR1

```
% nagg -t SDR1,EDR1 -g no -S SDR1*.h5 EDR1*.h5
```

Two output files containing SDR1 and EDR1 each in separate files

```
% nagg -g GEO1 GEO1*.h5
```

One output file containing GEO1

The following examples incur errors for different reasons:

```
% nagg -t SDR1,SDR2 SDR1*.h5 SDR2*.h5
```

ERROR: Incompatible Geolocation products because SDR1 uses GEO1 while SDR2 uses GEO2.

```
% nagg -t SDR1,EDR1 SDR1*.h5
```

ERROR: Missing EDR1 product granules.

```
% nagg -g GEO1 SDR1*.h5
```

ERROR: Missing GEO1 product granules. Note that even though nagg can have found the GEO1 granules via the SDR1 Geolocation reference, the option tells the Nagg tool to look for GEO1 granules directly.

```
% nagg -g GEO1 SDR1_GEO1*.h5
```

This succeeds because SDR1_GEO1*.h5 are packaged files containing both SDR1 and GEO1 granules.

3 Implementation Design

This section describes the interfaces of functions in the Nagg tool that is involved in the support of the two new features.

3.1 parse_options

```
parse_options(int argc, char * const argv[])
```

Parameters:

argc IN: number of elements in argv
argv IN: the list of command options argument

Return values:

0 if successful, call leave(EXIT_FAILURE) if it encounters irrecoverable errors such as illegal options or bad option values.

Description:

The `parse_options()` function uses the standard `getopt()` function to parse the command options. It will set up the values of the following global variables during its execution.

Option	Global variables	Description
-n	<code>ngranulesperfile</code>	The number of granules per product in each output file. Default is 1.
-t	<code>products_arg</code>	A linked list of products requested.
	<code>nproducts</code>	Number of products specified in <code>-t</code> flag.
	<code>geoproduct</code>	The Geolocation product ID used by the products. The Geolocation product should not be specified in this list since it is determined according to a pre-defined Products Table and assigned to the variable <i>geoproduct</i> . If the user wants to aggregate the Geolocation product only, he should specify it via the <code>-g</code> option and not use <code>-t</code> at all.
-g	<code>geofiles_arg</code>	An enum variable representing different geolocation granules selection criterion of “no”(0), “yes” (1), “strict”(2), and “geoproduct(3)”. If the criterion is actually a Geolocation product ID (and <code>-t</code> is not specified), the product ID is assigned to the variable <i>geoproduct</i> . (See <code>-t</code> option above.)
-d	<code>outDir</code>	Directory name in which output files are generated. Default is NULL (generate files in the current directory).
-O	<code>origin_arg</code>	Origin identifier of 4 characters. Default is “XXXX”.
-D	<code>domain_arg</code>	Domain identifier of 3 characters. Default is “XXX”.
-S	<code>outfile_format</code>	An enum variable representing the output file format of PACKAGED and UNPACKAGED for the Packaged and Unpackaged formats respectively.
<input_files> ...		
	<code>inputfiles</code>	A link list of input files.
	<code>ninputfiles</code>	Number of elements in <i>inputfiles</i> .

The parse options module checks that all requested products are compatible, as determined by all of them use the same Geolocation product. It is an error if the requested products are not compatible (using different Geolocation products).

3.2 get_geo_id_by_product_id

`get_geo_id_by_product_id(char *prod_id)`

Parameter:

`prod_id` IN: 5 character product DPID of a sensor product.

Return values:

DPID of the Geolocation product that the sensor product uses.

NULL otherwise.

3.3 set_granule_pattern

set_granule_pattern(char *product_id, granule_p_t granule)

Parameters:

product_id IN: DPID of product or GEO product
 granule_p_t IN: pointer to granule structure to be used to get information to be saved for creating attributes and datasets for product in output file.

Return values:

0 if successful, -1 otherwise

Input/output scenarios

Inputs: product DPID not already in array, valid granule
 Results: new gran_pattern_p_t structure for product in array.
 Return value: 0

Inputs: product DPID already in array, valid granule
 Return value: -1

Inputs: malformed DPID or invalid or incomplete granule
 Return value: -1

3.4 has_granule_pattern

has_granule_pattern(char *product_id)

Parameters:

product_id IN: DPID of product or GEO product

Return values:

1 if pattern structure exists for product
 0 if pattern structure for product does not exist
 -1 if product_id is malformed (not 5 characters)

Input/output scenarios

Input: valid product DPID
 Condition: granule in array for product DPID

Return value: 1

Input: valid product DPID
 Condition: no granule in array for product DPID

Return value: 0

Input: invalid product DPID

Return value: -1

3.5 get_granule_pattern

get_granule_pattern(char *product_id)

Parameters:

product_id IN: DPID of product or GEO product

Returns:

gran_pattern_p_t structure if successful, NULL otherwise

Input/output scenarios

Input: valid product DPID

Condition: granule in array for product DPID

Returns: gran_pattern_p_t structure.

Input: valid product DPID

Condition: no granule in array for product DPID

Returns: NULL

Input: invalid product DPID

Returns: NULL

3.6 nagg_get_granules

```
nagg_get_granules(char **file_list, int number_of_files,
char **products_list, int nproducts, geolocation_t geofiles_arg,
char *geoproduct, granule_p_t *granule_info_p[], int *number_of_granules_p)
```

Parameters:

file_list	IN: list of files containing granules to be added to the granule table.
number_of_files	IN: number of file names in the list.
products_list	IN: list of product types for which granules will be written to a file.
nproducts	IN: number of products types in the list.
geofiles_arg	IN: enum value from -g command option (default GEOFILE_YES).
geoproduct	IN: the DPID of the geolocation product.
*granule_info_p[]	OUT: address of the granule table to be populated.
*number_of_granules_p	OUT: address of variable for number of granules put in the table.

Return values:

0 if successful, -1 otherwise

Input/output scenarios

Inputs: file_list SDR1*.h5 EDR1*.h5
 products_list SDR1 EDR1
 geoproduct GEO1

Returns: 0
 granule_info_p pointer to table of SDR1, EDR1, and GEO1 granules found
 in files SDR1*.h5 EDR1*.h5 and files referenced in those
 files' N_GEO_Ref attributes.
 number_of_granules_p number of granules in granule_info_p table.
 gran_pattern_p array of structures containing sufficient information to
 create attributes and datasets for SDR1, EDR1, and GEO1
 products.

Inputs: file_list SDR1*.h5 EDR1*.h5
 products_list SDR1 EDR1
 geoproduct NULL

Returns: 0
 granule_info_p pointer to table of SDR1 and EDR1 granules found in files
 SDR1*.h5 EDR1*.h5.
 number_of_granules_p number of granules in granule_info_p table.
 gran_pattern_p array of structures containing sufficient information to
 create attributes and datasets for SDR1 and EDR1
 products.

Inputs: file_list GEO1*.h5
 products_list NULL
 geoproduct GEO1

Returns: 0
 granule_info_p pointer to table of GEO1 granules found in files
 GEO1*.h5.
 number_of_granules_p number of granules in granule_info_p table.
 gran_pattern_p array of structures containing sufficient information to
 create attributes and datasets for GEO1 product.

Inputs: file_list EDR1*.h5
 products_list SDR1 EDR1
 geoproduct NULL

Returns: -1

granule_info_p	undefined
number_of_granules_p	undefined
gran_pattern_p	undefined

Inputs: file_list SDR1*.h5 EDR1*.h5
 products_list NULL
 geoproduct GEO1
 condition: external GEO files available with GEO1 granules

Returns: -1
 granule_info_p undefined
 number_of_granules_p undefined
 gran_pattern_p undefined

Inputs: file_list SDR1*.h5 EDR1*.h5
 products_list SDR1 EDR1
 geoproduct GEO1
 condition: no external files available with GEO1 granules

Returns: -1
 granule_info_p undefined
 number_of_granules_p undefined
 gran_pattern_p undefined

3.7 select_granules

```
select_granules(granule_p_t granule_info[], int *_gindex, char **products_list,
               int nproducts, int total_nproducts, char *geoproduct, granule_p_t
               granules_selected[], int ngranulesperfile, int *_granules_remain, int
               *_total_granules_file)
```

Parameters:

granule_info	IN: table of granules for selection.
*_gindex	INOUT: index of the next available granule in the granule_info for selection. It reaches the end of the table if _granules_remain is equal to 0.
**products_list	IN: the list of products to match.
nproducts	IN: number of elements in products_list.
total_nproducts	IN: number of products and the geolocation product if wanted.
*geoproduct	IN: geolocation product (NULL if not wanted.)
granules_selected	INOUT: a table of selected granules for output. It is expected that sufficient space has been allocated for granules_selected to store all granules selected.
ngranulesperfile	IN: number of granules of each product per output file.
*_granules_remain	INOUT: number of granules in the granule_info table

available for selection.
 *_total_granules_file OUT: number of granules in the granules_selected table.

Return values:

Returns SUCCEED (0) if success; FAIL (-1) otherwise.

If return values is FAIL, the values of the OUT or INOUT parameters are undefined.

3.8 compose_output_fname <<this section needs more work>>

/* Compose the output file name.

* Parameters:

* granule_info_p: IN: The table of all granules.

* number_of_granules: IN: number of granules in granule_info_p.

* products_list: IN: The list of all products requested.

* nproducts: IN: Number of products in the list.

* ngranulesperfile IN: Number of granules to be writtern to the output file.

* createtime: OUT: creation time string is returned via it.

* output_fname OUT: Composed new output file name is returned via it.

* geo_fname OUT: Composed new Geo-file name is returned via it.

* Return code:

* positive: output filename composed.

* 0: all filled granules; no output file composed.

* negative: error encountered.

*/

/* Algorithm

* file name convension

* <DPID>-...-<DPID>_<spacecraft>_d<Start_date>_t<Start_time>_e<Stop_time> \

* _b<Orbit_number>_c<Creation_date>_<Origin>_<Domain>.h5

*/

int

compose_output_fname(granule_p_t granule_info_p[], int number_of_granules,
 char **products_list, int nproducts, int ngranulesperfile,
 char *creationdate, char **output_fname, char **geo_fname)

```
select_granules(granule_p_t granule_info[], int *_gindex, char **products_list,
               int nproducts, int total_nproducts, char *geoproduct, granule_p_t
               granules_selected[], int ngranulesperfile, int *_granules_remain, int
               *_total_granules_file)
```

Parameters:

granule_info	IN: table of granules for selection.
*_gindex	INOUT: index of the next available granule in the granule_info for selection. It reaches the end of the table if _granules_remain is equal to 0.
**products_list	IN: the list of products to match.
nproducts	IN: number of elements in products_list.
total_nproducts	IN: number of products and the geolocation product if wanted.
*geoproduct	IN: geolocation product (NULL if not wanted.)
granules_selected	INOUT: a table of selected granules for output. It is expected that sufficient space has been allocated for granules_selected to store all granules selected.
ngranulesperfile	IN: number of granules of each product per output file.
*_granules_remain	INOUT: number of granules in the granule_info table available for selection.
*_total_granules_file	OUT: number of granules in the granules_selected table.

Return values:

Returns SUCCEED (0) if success; FAIL (-1) otherwise.

If return values is FAIL, the values of the OUT or INOUT parameters are undefined.

=====

3.9 start_write

```
start_write(const char **outfiles, int noutfiles, const char *outgeofile,
            char *geoproduct, char **products_list, int nproducts,
            const char *creationdate, const char *creationtime, int ngranulesperfile,
            gran_pattern_p_t *gran_pattern_p[])
```

Parameters:

outfiles	IN: list of file names to be created for writing an output aggregation
noutfiles	IN: number of names in the outfiles list.
outgeofile	IN: name of the corresponding geo-location file, or NULL.
geoproduct	IN: DPID of the corresponding geoproduct, or NULL.
products_list	IN: list of DPIDs, one for each product. Only one product is supported for this version.
nproduct	IN: number of DPIDs in the products_list argument.
creationdate	IN: date of creation of the output files (for writing to the N_HDF_Creation_Date attribute)

creationtime IN: time of creation of the output files (for writing to N_HDF_Creation_Time attribute).
 ngranulesperfile IN: number of granules in each aggregation.
 gran_pattern_p[] IN: address of variable for product granule pattern information.

Return values:

0 if successful, -1 otherwise

Input/output

noutfiles/nproducts:	0/0	1/n	n/n
outgeofile && geoproduct in outgeofile	Geoproduct in outgeofile	Error (possible but option not provided)	1 Product in each outfile Geoproduct in outgeofile
NULL && geoproduct	Error	N products and geoproduct in 1 outfile	Error (option not provided)
NULL && NULL	Error	N products in 1 outfile	N products in n outfiles
Outgeofile && NULL	Error	Error	Error

4 Implementation Details

This section describes the interfaces of functions in the Nagg tool that is involved in the support of the two new features.

4.1 parse_options

Description:

The parse_options() function uses the standard *getopt()* function to parse the command options. It will set up the values of the following global variables during its execution.

Option	Global variables	Description
-n	ngranulesperfile	The number of granules per product in each output file. Default is 1.
-t	products_arg	A linked list of products to store in each output file
	nproducts	Number of products specified in -t flag.
-d	outDir	Directory name in which output files are generated. Default is NULL (generate files in the current directory).
-O	origin_arg	Origin identifier of 4 characters. Default is "XXXX".
-D	domain_arg	Domain identifier of 3 characters. Default is "XXX".
-g	geofiles_arg	An enum variable representing different geolocation granules selection criterion of "no"(0), "yes" (1), "strict"(2), and "geoproduct(3)".
<input_files> ...		
	inputfiles	A link list of input files.
	ninputfiles	Number of elements in <i>inputfiles</i> .

The parse options module will now be checking that all requested products are compatible, as determined by all of them having one corresponding GEO product. Leave(EXIT_FAILURE) should be called if the requested products are not compatible (use more than one geo product).

Parse options will also need to handle multiple entries in the -t string.

4.2 get_geo_id_by_product_id

Description:

This function returns the DPID of the sensor data product's corresponding geo product. Geo products have no corresponding geo product, so the function should return NULL when called for a geo product. Geo product names can be obtained using get_product_sname_by_id(geo_product_id).

4.3 set_granule_pattern

gran_pattern_p_t structure

Will contain structures, which are to be allocated and populated while getting granules, with information for creating attributes and raw data datasets.

Description:

Function to create a gran_pattern_t structure for the input product and populate it with information from the input granule.

4.4 has_granule_pattern

Description:

Function to test whether or not there is a gran_pattern_t structure for the input product.

4.5 get_granule_pattern

Description:

Function to get the granule_pattern_t structure for the input product.

4.6 nagg_get_granules

Description:

The nagg_get_granules() function opens and reads the files in the list provided by the command parser, putting the values of attributes necessary for reaggregating the granules in the members of an instance of the granule_t structure which is added to the granule table. Unless the -g no option is specified or the file is a GEO file, the file specified by the file's N_GEO_Ref attribute will also be opened and read, and its granules added to the granule table.

Error messages will be returned if a file specified is not an HDF5 file, if the file does not exist or cannot be accessed due to insufficient file permissions, if the file cannot be opened due to an HDF5 failure. Error messages will also result if no granules are found for a requested product, for any of the input combinations resulting in error output in section 3.8, or if the geo product is not found except when "-g no" is not specified. The tool will not continue if any of these errors are encountered.

The attributes from which granule information is gathered are attributes of several different objects in the file. Some are attributes of the root group. Others are attributes of the product groups which are subgroups of the /Data_Products group. The function iterates through all subgroups of /Data_Products, collecting granule information from the groups and their aggregate and granule datasets.

There will now be a pointer to an array of granule_pattern_t structures, one for each product plus one for the geoproduct. The information from the first granule for each product will be saved in one of these structures and will be used in write granules to initialize files when the first granule is a fill granule. This may be extended to initialize all files from the pattern granule in the future, especially if compression is added.

The nagg_get_granules() function needs to add only granules in the product list or the corresponding geo granules from files indicated by granules in the product list to the granule table. It also needs to check for a geo group in the files with the product as well as checking for an N_Geo_Ref attribute with the name of a geo file.

4.7 select_granules

Description:

The select_granules function selects granules that will fit in the output file according to bucket alignment boundary. The following is a description of the algorithms used.

Nagg algorithm in the calculation of bucket alignment:

Let N be the number of granules requested by the nagg user to reaggregate the NPP product files.

Let T_g be the duration of the first selected granule. (This value is different for different products and is defined in the products table.)

Then $T_{bucket} = N * T_g$ seconds.

Let A_n be the n -th bucket since epoch.

Let A_{sn} and A_{en} be the starting and ending time of A_n .

Let G_s be the beginning time of the first selected granule.

Then

$$A_n = \text{floor}(G_s / T_{bucket})$$

$$A_{sn} = A_n * (T_{bucket})$$

$$A_{en} = A_{sn} + T_{bucket}$$

How nagg adds fill granules to produced files:

First produced file

For the first file, if the starting time of the first selected granule is bigger than A_{sn} , no fill granules are added before copying existing granules to the new file. This will produce a partial file.

Second to $(n-1)$ -th files

N existing granules per product requested are copied to each of the new files, insert fill granules in place of any missing granules.

Last (n) -th file

Remaining granules per product requested are copied to the last file.

If the ending time of the last granule is less than the ending time of the last bucket, no fill granules are added. This will produce a partial file.

4.8 start_write

Description:

The `start_write()` function is the first function called when writing an aggregation of granules. For a single product with the corresponding geo granules in a separate file, `start_write()` creates the product and geo output files. When multiple products are supported in the future, for the `-S nagg` tool option, `start_write()` will create an output file for each product for each aggregation of granules, plus the geo file if geo granules are aggregated separately. When packaging is supported, `start_write()` will create one output file for all products in an aggregation.

All of the granules selected for an aggregation will be written to the output files before any granules are selected for the next aggregation. The granules within an aggregation may be written in any order, and typically will be written one to each output file in rotation. The write granules module

creates an array of `product_info_t` structures to keep track for each product of output filenames, input and output file handles, number of granules written, and a pointer to the previously written granule.

```
typedef struct {
    const char dpid[DPID_size+1];
    hid_t infile;
    hid_t outfile;
    const char * outfilename;
    int last_i_granule;
    int granules_written;
    granule_p_t prev_granule;
} product_info_t;
```

A `product_info_t` structure is created and populated for each product and the geofile by the `start_write()` function. The `write_granules()` function will then select the `product_info_t` for each granule that matches its DPID. The `product_info_t` for the separate geolocation file is created last so that its index will always be `nproducts`.

The `start_write()` function also writes 3 attributes to the root group of the files: `N_GEO_Ref`, for files except the geo file, `N_HDF_Creation_Date`, and `N_HDF_Creation_Time`. Values for these attributes are generated by `nagg` with the new geo file name and the current time.

`Start_write` will also determine from the parameters whether the output is to be packaged or unpackaged. If unpackaged, each product will be written to a separate file, including the geo product. If packaged, all products will be written to a single file, including the geo product.

4.9 write_granules

Description:

The `write_granules()` function is called for each granule selected to be written to an aggregation, and is responsible for writing most of the data and attributes to the new file, whether the values are from the original file or are generated by the `nagg` tool. The function does the following:

- Selects the `product_info_t` structure matching the granule's product ID (DPID) to find the correct output file.
- Opens the input file specified by `granule->file_in`.
- Initializes the output file when first called with a granule.
 - Copies root group attributes except those written by `start_write()` from the input file to the output file.
 - Creates group structure in the file, creating product groups in `/All_Data` and `/Data_Products`. Product groups in `/All_Data` are named `<productname>_All`; those in `/Data_Products` are named `<productname>`.
 - Copies datasets from the `/All_Data` group in the input file to the `/All_Data` group in the output file; resizes the datasets for the new aggregation size.
 - Copies attributes from the `/Data_Products/<productname>` group in the input file to the `/Data_Products/<productname>` group in the output file.

- Copies the /Data_Products/<productname>/<productname>_Gran_n dataset for the granule in the input file to the dataset for the granule in the output file. References and metadata that are specific to the new file will be overwritten in subsequent steps.
- Copies the granule's hyperslab for each dataset in /All_Data from the input file to the output file creating a region reference to the new location in the granules new file's /Data_Products/<productname>/<productname>_Gran_n dataset
- Creates the /Data_Products/<productname>/<productname>_Aggr dataset with object references to all the datasets in /All_Data/<productname> group. Copies attributes from the Aggregate dataset in the input file to the Aggregate dataset in the output file.
- Copies values for the Aggregate dataset's AggregateBeginningDate, AggregateBeginningGranuleID, AggregateBeginningOrbitNumber and AggregateBeginningTime from the first granule in the aggregation.

Increments the value of the variable that keeps track of the number of granules written.