Accessing Structured Grid Data with Rind Planes

Author: Stephen Guzik, Colorado State University
Contact: Stephen.Guzik@colostate.edu

This document only concerns structured grid data with rind planes. Applications that read or write data using unstructured grids or structured grids without rind planes should not be affected by the observed issues or proposed changes.

1 Summary

The SIDS mentions that physical data (the *core* grid information) in a zone begins at index [1, 1, 1]. For grid coordinates, the SIDS states[1]:

> Core vertices in a zone are assumed to begin at [1, 1, 1] (for a structured zone in 3-D) and end at `VertexSize`. If `Rind` is present, it will provide information on the number of "rind" points in addition to the core points that are contained in the `DataArray_t` structures.

For the flow solution, the SIDS does not say anything specific about where the core grid starts.[2] When discussing `Rind_t`[3], the SIDS specifically states that the range of indices for the grid is

$$\begin{aligned} i &: (1 - a, \texttt{II} + b) \\ j &: (1 - c, \texttt{JJ} + d) \\ k &: (1 - e, \texttt{KK} + f)\,. \end{aligned}$$

This is illustrated graphically at `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/conv.html#rind_struct`.

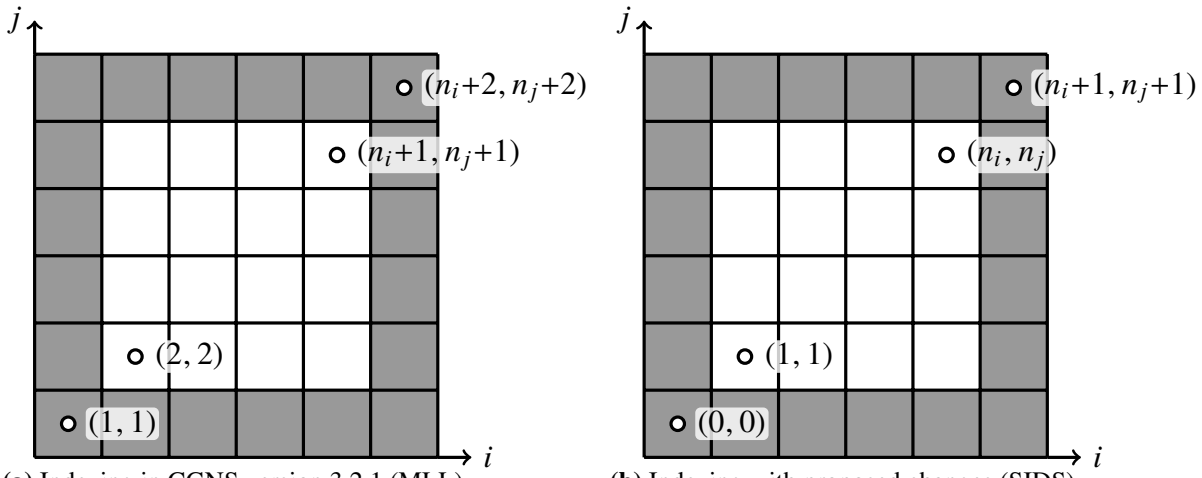
1.1 Problem

In structured grids, when rind (or ghost) cells are present, reading data at `rmin` = [1, 1, 1] yields information in the rind planes, as illustrated in Fig. 1a. This behavior is different from that suggested by the SIDS and illustrated in Fig. 1b. When reading or writing a CGNS file, it is important to be able to include or omit rind plane information as required. The fix to the observed discrepancy between the MLL and SIDS and a proposed extension address this concern.

1.2 Proposed Changes

When reading grid or solution information, index [1, 1, 1] should always denote the start of core grid information. Therefore, the range `rmin` = [1, 1, 1] to `rmax` = $[n_i, n_j, n_k]$ can always be used to read the full core-grid information, where $[n_i, n_j, n_k]$ = [`NCellI`, `NCellJ`, `NCellK`] or $[n_i, n_j, n_k]$ = [`NVertexI`, `NVertexJ`, `NVertexK`]. To read rind plane information, in addition to

[1] `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/gridflow.html#GridCoordinates`
[2] `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/gridflow.html#FlowSolution`
[3] `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/build.html#Rind`



(a) Indexing in CGNS version 3.2.1 (MLL). **(b)** Indexing with proposed changes (SIDS).

Figure 1 Current and proposed indexing for reading/writing to CGNS files. Rind planes are shown in gray, core grid is shown in white. The SIDS describes the indexing shown in (b) while the MLL (version 3.2.1) implements the indexing shown in (a).

core-grid information, one needs to provide the offset from the core-grid. E.g., to read one layer of rind planes, use `rmin` = $[0, 0, 0]$ and `rmax` = $[n_i + 1, n_j + 1, n_k + 1]$. This approach provides a consistent interface to accessing core-grid data, in contrast to the current implementation (version 3.2.1) where one first has to probe the sizes of rind-plane information to find the offsets required to read only core-grid information. This modification is considered a bug fix since the MLL does not produce the behavior described in the SIDS.

A second proposed modification (an extension to the MLL) allows for reading data into an array in memory that is not the same size as the data being read. For example, one may have an array in memory that is sized to hold both the core grid and rind planes. Into this array, one may wish to read only core-grid information from the CGNS file. There is currently no method in the MLL that supports this need. Yet it is easily achieved with HDF5 hyperslabs and only small changes are required to the CGNS library. Additional routines are proposed, e.g., `cg_field_read_to_shaped_array` (or alternatively `cg_field_read_to_array` or `cg_field_general_read`) and `cg_field_partial_write_from_shaped_array` (or alternatively `cg_field_partial_write_from_array` or `cg_field_general_write`). In addition to `rmin` and `rmax` which describe the hyperslab to read from the file, the user must specify `mem_numdim`, `mem_dim`, `mem_rmin`, `mem_rmax`, to fully specify the shape of the hyperslab in memory. The only restriction is that the number of points in both hyperslabs (file and memory) must be the same[4]. However, a second restriction which is imposed, but can be removed, is that the number of dimensions in memory must be less than or equal to the number of dimensions in the file.

[4]See `http://www.hdfgroup.org/HDF5/doc/UG/UG_frame04ProgModel.html` for more information on hyperslabs in HDF5.

2 Compatibility

Any code reading information with rind planes will be broken by the proposed changes. It is possible to modify the library so that when the user asks for the full range of data, irrespective of the actual indices, the full range of data is correctly returned. E.g., with one rind plane, `rmin` = $[1, 1, 1]$ to `rmax` = $[n_i + 2, n_j + 2, n_k + 2]$ would work the same as `rmin` = $[0, 0, 0]$ to `rmax` = $[n_i + 1, n_j + 1, n_k + 1]$ A disadvantage of allowing this behavior is that it does not enforce consistent indexing and can lead to confusion. Any code not using rind planes (where rind planes are by default set to zero) should not be affected.

2.1 Support for ADF

The proposed changes are easily supported with the HDF5 database since hyperslabs are available. ADF has not been investigated but is assumed to work as well since the routines in `cgns_io.c` do not need to be modified to fix the discrepancy. The extension is only fully supported in HDF5 because it relies on *in-situ* type conversion provided by the HDF5 library.

2.2 Support for Fortran

Fortran user code has not yet been tested. The library changes proposed herein are expected to be compatible with Fortran user code.

3 Changes

3.1 SIDS

Few changes are required to the SIDS. At `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/gridflow.html#GridCoordinates`, the comment that "Core vertices in a zone are assumed to begin at [1, 1, 1] (for a structured zone in 3-D) . . ." should be strengthened to state the following:

> Core vertices in a zone begin at [1, 1, 1] (for a structured zone in 3-D) and end at `VertexSize`. If `Rind` is present, it will provide information on the number of "rind" points in addition to the core points that are contained in the `DataArray_t` structures. Indices in `DataArray_t` structures have the range $[1 - a, 1 - c, 1 - e]$ to $[\mathtt{II} + b, \mathtt{JJ} + d, \mathtt{KK} + f]$ where `VertexSize` = [`II`, `JJ`, . . .] and `RindPlanes` = $[a, b, \ldots]$ (see the `Rind_t` structure for the definition of `RindPlanes`).

The note in parenthesis can possible be removed from the subsequent section titled "FUNCTION `DataSize[]`:" if deemed redundant.

A similar comment should be added to `http://www.grc.nasa.gov/WWW/cgns/CGNS_docs_current/sids/gridflow.html#FlowSolution` where `VertexSize` is replaced by `GridSize` (or alternatively `DataSize`) which is dependent on `GridLocation`.

3.2 Library

Only changes to `FlowSolution_t` node are discussed. The changes to the `GridCoordinates_t` node are identical or at least very similar.

Read routines take `rmin` and `rmax` as arguments and write them to arrays `s_start` and `s_end`. The required change is to simply adjust the value when writing to `s_start` and `s_end`. For `cg_field_read`:

```
4134            s_start[n]  = rmin[n] + sol->rind_planes[2*n];
4135            s_end[n]    = rmax[n] + sol->rind_planes[2*n];
```

Line numbers refer to the a patched library. There are other changes related to checking the input arguments.

Routine `cg_field_write` returns the entire array and needs no modifications. Routine `cg_field_partial_write` takes `rmin` and `rmax` as arguments, similar to read, and the changes are identical:

```
4533        s_start[n] = rmin[n] + sol->rind_planes[2*n];
4534        s_end[n]   = rmax[n] + sol->rind_planes[2*n];
```

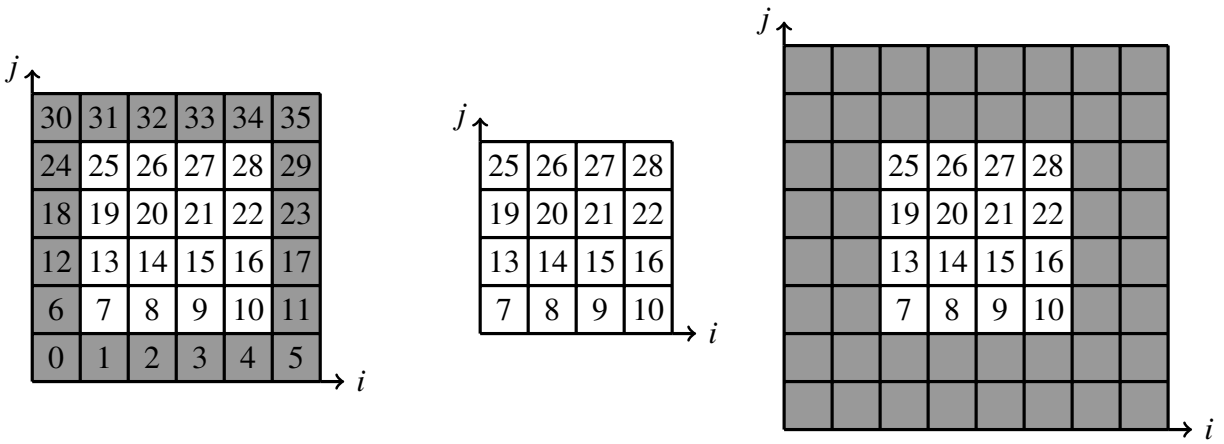
Input checking is modified and the routine is restructured to specify the shape of the array in file space and the shape in memory space in advance. These shapes can then be used whether a `DataArray_t` node is overwritten or a new one is created. Some of the restructuring makes more sense when considering the extension proposed in Section 4 where routine `cgi_new_node_partial` is modified to be more general.

3.3 UserGuideCode

Code in the user guide will be updated to also show the writing of rind planes for grid coordinates in a new file, `write_gridrind_str.c`. File `read_flowcentrind_str.c` will be modified to correct indices for reading full range for density, and to illustrate reading only core-grid data for pressure.

4 Extensions

Mechanisms exist in the MLL to read or write to only part of the file-space array, and the changes outlined in Section 3 make it more intuitive to read/write the core-grid data. When using rind planes, users will likely need a similar capability for memory-space arrays. E.g., they may allocate memory for core grid and rind planes but wish to read only core-grid points from the CGNS file (see Fig. 2c). There is no mechanism in the MLL to allow this. Users must instead allocate contiguous memory, read into that memory (as shown in Fig. 2b), and then manually copy into the exact shape they require for their algorithm. The proposed extension is for two new interfaces, `cg_field_read_to_shaped_array` and `cg_field_partial_write_from_shaped_array`, in the MLL to permit reading/writing to an arbitrary array shape in memory space. As the above names are quite long, alternatives include `cg_field_read_to_array` or `cg_field_general_read` and similar for writing. New interfaces will also be included for specifying structured



(a) Layout in file space (CGNS).

(b) Layout of core-grid data in contiguous memory space.

(c) Layout of core-grid data in non-contiguous memory space.

Figure 2 Layouts of data in file (CGNS) and memory (user code) space. To read only core-grid data, the memory space must be contiguous as shown in (b). The extension supports reading and writing to a general layout in memory space as shown in (c). In the illustrated example, the user can then allocate an array with 2 layers of rind planes and read only core-grid data.

grid coordinates. This document focuses on the `FlowSolution_t` node. The changes to the `GridCoordinates_t` node are very similar.

4.1 Changes to Library

Changes are described first for low-level library code and then working towards the high-level MLL interface. Full details of the changes can be viewed in the patch file.

4.1.1 ADFH.c

The interface to routine `ADFH_Read_Data` was modified to add the parameter `const char* m_data_type` which is the ADF data type for the memory read. If this parameter is set to `NULL`, the data type is the same as that in the file (previous behavior). If this parameter is set to an ADF data type, HDF5 will do the conversion. This change allows for the HDF5 library to perform type conversion instead of the CGNS library (and avoids allocating the extra memory required by `cgi_convert_data`).

4.1.2 cgns_io.c

The IO routines already support an array in memory with a different shape from that in the file. Function parameters prefixed with `s_` describe the shape in the file and parameters prefixed with `m_` describe the shape in memory. To allow for type conversion by the HDF5 library, a new routine named `cgio_read_data_type` is created. This routine is identical to `cgio_read_data` except the interface includes an additional parameter specifying the ADF data type for memory space. This data type is propagated to `ADFH_Read_Data` in `ADFH.c`. An important difference is that the ADF file type is not supported. A user attempting type conversion when reading to a shaped array and using the ADF database will encounter a `CGIO_ERR_NOT_HDF5` error.

4.1.3 cgns_internals.c

Routine `cgi_new_node_partial` is modified so that the shape of memory is specified through parameters rather than hard-coded. This means that the memory shape must be specified or determined in the caller in `cgnslib.c`.

4.1.4 cgnslib.c

To accommodate the changes in `cgns_internals.c`, the memory shape is specified in `cg_field_partial_write` and passed as arguments to `cgi_new_node_partial`. Note that read routines do not require a similar change as they directly call functions in `cgns_io.c`. Routines `cg_field_read_to_shaped_array` and `cg_field_partial_write_from_shaped_array` are implemented that allow the user to specify the shape of memory from their applications.

4.1.5 Remarks

There is now a lot of duplicate code in `cgnslib.c` which will make maintenance more difficult. The code should be structured so that only a general path, e.g. `cg_field_partial_write_from_shaped_array`, is maintained. Less general interfaces, e.g., `cg_field_write` and `cg_field_partial_write`, will supply the additional parameters and call the general path.

5 Observations

1. Inconsistent use of 3, 12, or `CGIO_MAX_DIMENSIONS` when declaring indices for structured arrays.