NGON modification proposal

Unstructured grid connectivity

Author: Pierre-Jacques Legay, ONERA
Contact: pierre-jacques.legay@onera.fr

Index table

This CPEX focuses on the NGON representation. The rationale for requiring an extension to CGNS/SIDS for unstructured grid connectivity is detailed in the first part of this document. The second part details the proposal which includes a solution to the problem and an impact analysis of that solution on the SIDS and on the performance.

1 Rationale for the modification proposal

The first section is a reminder of the current SIDS description for unstructured grid connectivity. The following section details the problems induced by this description.

1.1 Current SIDS description for unstructured grid connectivity

The unstructured grid connectivity is stored in the Elements_t of the CGNS/SIDS. As described hereafter, its storage depends on the elements type:

For all element types except MIXED, NGON_n, and NFACE_n, ElementConnectivity contains the list of nodes for each element. If the elements are sorted, then it must first list the connectivity of the boundary elements, then that of the interior elements.

```
ElementConnectivity = Node1_1, Node2_1, ... NodeN_1,
                      Node1_2, Node2_2, ... NodeN_2,
                      ...
                      Node1_M, Node2_M, ... NodeN_M
```

where M is the total number of elements (i.e., ElementSize), and N is the number of nodes per element.

ElementDataSize indicates the total size (number of integers) of the array ElementConnectivity. For all element types except MIXED, NGON_n, and NFACE_n, the ElementDataSize is given by:

```
ElementDataSize = ElementSize * NPE[ElementType]
```

where NPE[ElementType] is a function returning the number of nodes for the given ElementType. For example, NPE[HEXA_8]=8.

When the section ElementType is MIXED, the data array ElementConnectivity contains one extra integer per element, to hold each individual element type:

$$\begin{aligned}\texttt{ElementConnectivity} = {} & \texttt{Etype}_1,\ \texttt{Node1}_1,\ \texttt{Node2}_1,\ \ldots\ \texttt{NodeN}_1, \\ & \texttt{Etype}_2,\ \texttt{Node1}_2,\ \texttt{Node2}_2,\ \ldots\ \texttt{NodeN}_2, \\ & \ldots \\ & \texttt{Etype}_M,\ \texttt{Node1}_M,\ \texttt{Node2}_M,\ \ldots\ \texttt{NodeN}_M\end{aligned}$$

where again M is the total number of elements, and N_i is the number of nodes in element i. In the case of MIXED element section, ElementDataSize is given by:

$$\texttt{ElementDataSize} = \texttt{sum}(_{n=[start,\ end]},\ \texttt{NPE[ElementType}_n\texttt{]} + 1)$$

Arbitrary polyhedral elements may be defined using the NGON_n and NFACE_n element types. The NGON_n element type is used to specify all the faces in the grid, and the NFACE_n element type is then used to define the polyhedral elements as a collection of these faces.

I.e., for NGON_n, the data array ElementConnectivity contains a list of nodes making up each face in the grid, with the first value for each face defining the number of nodes making up that face:

$$\begin{aligned}\texttt{ElementConnectivity} = {} & \texttt{Nnodes}_1,\ \texttt{Node1}_1,\ \texttt{Node2}_1,\ \ldots\ \texttt{NodeN}_1, \\ & \texttt{Nnodes}_2,\ \texttt{Node1}_2,\ \texttt{Node2}_2,\ \ldots\ \texttt{NodeN}_2, \\ & \ldots \\ & \texttt{Nnodes}_M,\ \texttt{Node1}_M,\ \texttt{Node2}_M,\ \ldots\ \texttt{NodeN}_M\end{aligned}$$

where here M is the total number of faces, and N_i is the number of nodes in face i. The ElementDataSize is the total number of nodes defining all the faces, plus one value per face specifying the number of nodes making up that face.

Then for NFACE_n, ElementConnectivity contains the list of face elements making up each polyhedral element, with the first value for each polyhedra defining the number of faces making up that polyhedral element.

$$\begin{aligned}\texttt{ElementConnectivity} = {} & \texttt{Nfaces}_1,\ \texttt{Face1}_1,\ \texttt{Face2}_1,\ \ldots\ \texttt{FaceN}_1, \\ & \texttt{Nfaces}_2,\ \texttt{Face1}_2,\ \texttt{Face2}_2,\ \ldots\ \texttt{FaceN}_2, \\ & \ldots \\ & \texttt{Nfaces}_M,\ \texttt{Face1}_M,\ \texttt{Face2}_M,\ \ldots\ \texttt{FaceN}_M\end{aligned}$$

where now M is the total number of polyhedral elements, and N_i is the number of faces in element i. The sign of the face number determines its orientation. If the face number is positive, the face normal is directed outward; if it’s negative, the face normal is directed inward.

$$\texttt{ElementDataSize} = \texttt{sum}(_{n=[start,\ end]},\texttt{FPP[ElementType}_n\texttt{]} + 1)$$

where FPP[ElementType$_n$] is a function returning the number of faces per polyhedra.

The following figure is set up to ease comprehension and comparison in the following sections:

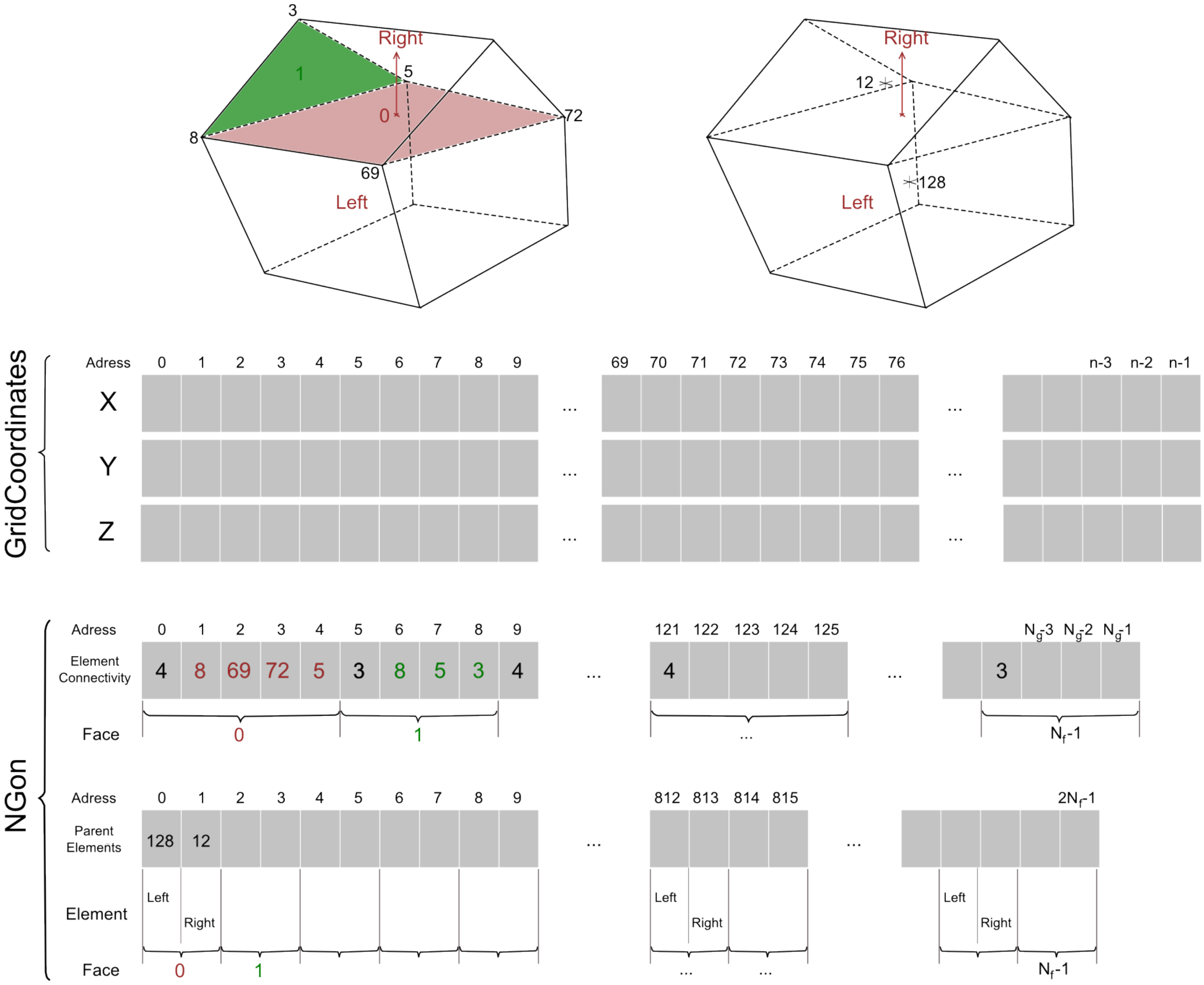

Illustration 1: CGNS/NGON current face based representation.

This figure shows an unstructured element using the NGON representation. It describes the physical construction of the CGNS related arrays GridCoordinates, ElementConnectivity, and ParentElements.

Notation:

- N_g: represents a vertex from the current face.
- N_f: represents a face of the current element.

1.2 Limitations for NGON_N and MIXED

The ElementConnectivity array can be very large for industrial CFD case. As a direct consequence we should be able to load this array fully or partially on multiple threads.

In the above description of the CGNS/NGON face based representation, the ElementConnectivity array mixes two data types which are interdependent :

- the number of nodes for each face : `Nnodes`$_1$
- a list of nodes for a face : `Node1`$_1$`, Node2`$_1$`, ... NodeN`$_1$

This implies that we cannot efficiently split this array to be read in parallel.

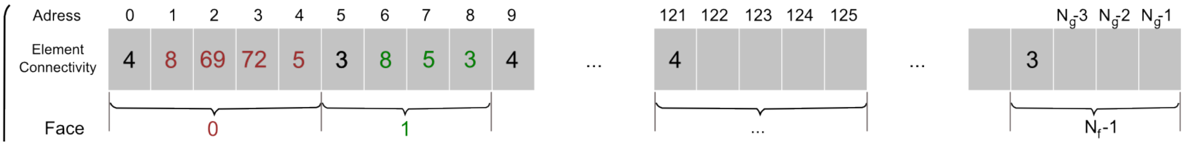

Illustration 2: ElementConnectivity interlaced data.

In parallel, the interlaced data requires that the entire ElementConnectivity array be read on every processor. This has a significant impact on IO performance.

NB: As described in the SIDS, the representation of MIXED and NFACE elements is identical to the NGON representation. As such, the parallel read suffers from the same performance issues. The issues of the MIXED and NFACE elements shall be addressed in a later CPEX.

2 NGON modification proposal

This proposal modifies the NGON SIDS representation to provide efficient parallel IO access and to optimize the data representation.

2.1 Solution description

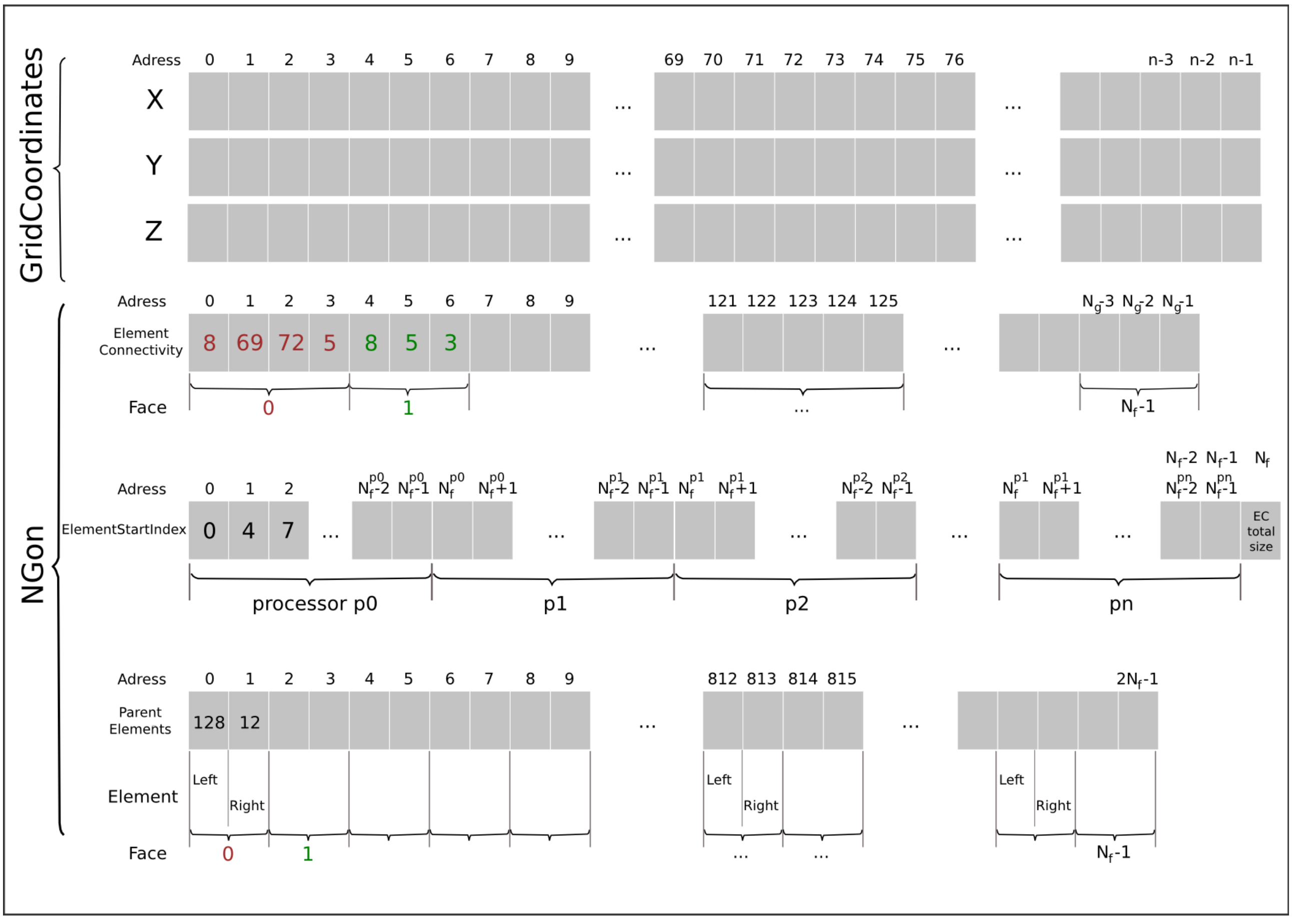

Illustration 3: NGON with new ElementConnectivity array and ElementStartIndex position array.

In this representation the ElementConnectivity array is deinterlaced. Illustration 4 compares the current standard versus the new deinterlaced ElementConnectivity array:

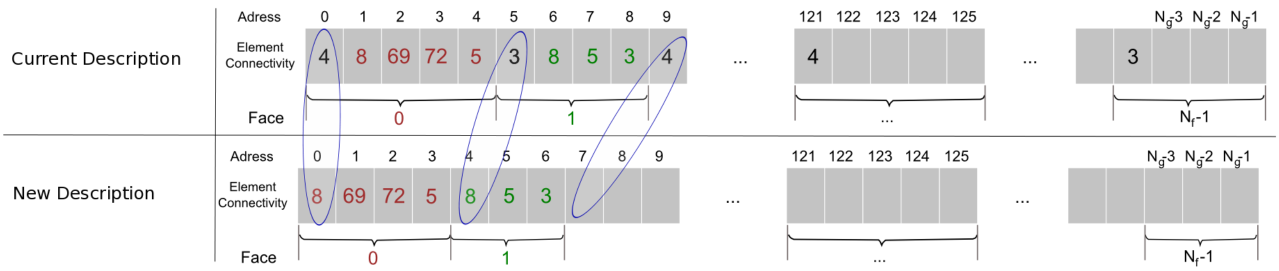

Illustration 4: Current ElementConnectivity vs. new ElementConnectivity.

The face vertex count has been removed from the connectivity. As a direct consequence the new array contains a unique data type.

Indices needed to read or analyze the ElementConnectivity array are stored in the new ElementStartIndex array.

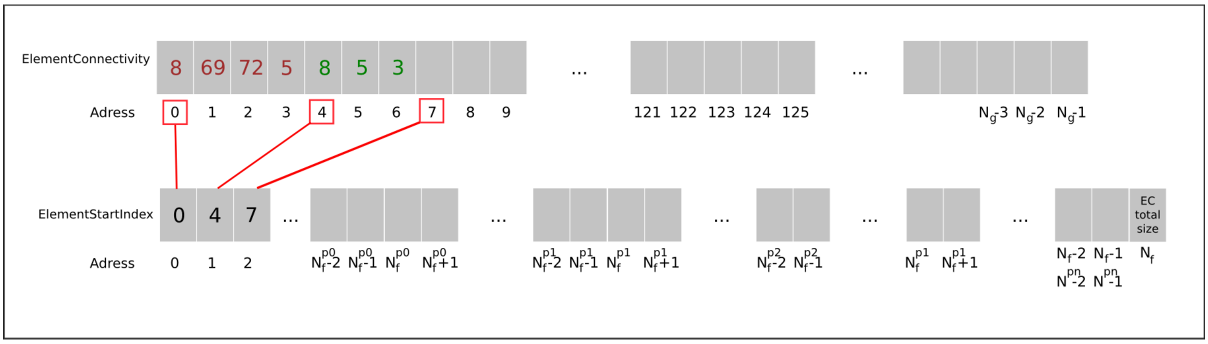

Illustration 5: ElementStartIndex lists the face position in the ElementConnectivity and it's last value indicates the ElementConnectivity total size.

This array lists the position in the ElementConnectivity of the first vertex for each face and it's last value is the ElementConnectivity array size. This CGNS node is of type DataArray_t. Its read can be distributed between P processors. Its size is N_f+1 with N_f being the number of face in the ElementConnectivity array. The ElementStartIndex array makes it easy for a process or thread to read a portion of the ElementConnectivity array.

NB: The last value of the ElementStartIndex allows easy access to the last vertex of the last face of the ElementConnectivity array.

2.2 Solution analysis

2.2.1 CGNS standard modification

It strongly modifies the representation of the ElementConnectivity node and it inserts the new array ElementStartIndex. These modifications should be reflected in the current CFD database and CGNS related code to insure the continuity of the computational capability. See section 4 'Implementation note'.

2.2.2 Data consistency

The data type of both arrays ElementConnectivity and ElementStartIndex are consistent.

2.2.3 Optimal data size

This solution does not duplicate data thus the global data size is unchanged and stays optimal.

2.2.4 ElementStartIndex -- an incremental index

In this proposal we choose to use the face position in the element connectivity instead of the face number of nodes (ElementStartIndex=[0,4,7,11, ...] instead of [4, 3, 4, ...]). The rationales are all IO read/write oriented and detailed here after:

1. Processor inter-dependency
   To access the ElementConnectivity data we need to know the position of the face in the ElementConnectivity array.
   → "Face vertex number" solution
   In this configuration we would need to sum over the index array to obtain the face position in the ElementConnectivity array. This means either create a new table or perform the computation as many times as needed.
   Another problem is that we need the result of the sum of face vertex of processor P_0 to obtain the location of the first vertex of the first face assigned to processor P_1.
   This behavior generalizes with the need to sum the face vertex of processors $P_0,P_1,...,P_{M-1}$ to obtain the location of the first vertex of the first face assigned to processor P_M.
   This is not a complex operation but we do not like the idea of imposing a dependency on all previous processors computations.
   →"Face position" solution
   In this configuration the ElementStartIndex array can be split on P processors and each processor related part directly gives the location of the face vertex from the ElementConnectivity array.

2. Full vs. partial load
   As stated above we need to know the position of the face in the ElementConnectivity array.
   → "Face vertex number" solution
   If we want to access data in processor P_M, then we need to access data on all processors $P_0,P_1,..., P_{M-1}$. This means that we need to load the index array on every processor of inferior rank.
   → "Face position" solution
   To access the ElementConnectivity we need the boundary of locations for processor P_M.
   These boundaries can be partially loaded from the ElementStartIndex array by getting the first element of processor P_M part and the first element of processor P_{M+1} part. These two integers indicate the section of ElementConnectivity to be loaded on proc P_M (1st element of P_M till 1st element of P_{M+1}-1).
   NB: The last value of the ElementStartIndex allows to apply the same operator to the last section of the ElementConnectivity array. For the last section, the load is performed from P_M to P_{M+1}-1 with P_{M+1} being the ElementConnectivity size.

With this proposal, we need to load only two integers from the ElementStartIndex array on a processor to fully access the ElementConnectivity array. Partial load for CGNS/HDF5 is available in CHLone for example. CHLone (http://chlone.sourceforge.net) is a python module implementing the SIDS-to-Python mapping which allows a simple load/save of CGNS/HDF5 files.

3. direct access to a face connectivity
   For some specific unstructured software, it is interesting to have a simple access to a specific face in the ElementConnectivity array.
   This is only allowed with the “Face Position” solution.

4. Access to face position and face number of nodes
   Using the incremental index method, it is easy to calculate the “face number of nodes” for element ‘i’ as ElementStartIndex[i+1]-ElementStartIndex[i] is an O(1) calculation. So the selected method gives both data (face position and face number of nodes) in O(1) where the alternate method would give face number of nodes in O(1), but face position in O(N).

2.2.5 ElementStartIndex content

Using an incremental index --the face position in the ElementConnectivity array-- could lead to large numbers in the ElementStartIndex array. However, these numbers are limited to the size of the ElementConnectivity array as they give access to addresses of that array. We will probably reach the ElementConnectivity array limits first.

2.2.6 ElementStartIndex last value

The ElementStartIndex lists the first vertex position of each face in the ElementConnectivity. Then we added the ElementConnectivity size as last value. The rationale behind this is to improve data access when looping over the faces.

→ Positions without ElementConnectivity size

In this configuration an iteration loop over NGONs could be written as follows:

```
for (i=0; i < NGON; i++) {
    end = (I == NGON-1) ? ElementConnectivitySize :ElementStartIndex[i+1];
    for (j=ElementStartIndex[i]; j < end; j++) {
        entry = ElementConnectivity[j];
        // do something here
    }
}
```

For each NGON we have to check if we are reaching the last NGON to handle the access to the last vertex of the last face.

→ Positions with ElementConnectivity size (current proposal)

In this configuration an iteration loop over NGONs could be written as follow:

```
for (i=0; i < NGON; i++) {
    for (j=ElementStartIndex[i]; j < ElementStartIndex[i+1]; j++) {
        entry = ElementConnectivity[j];
        // do something here
    }
}
```

The last value of the ElementStartIndex gives a boundary to access the last vertex of the last NGON without the conditional assignment.

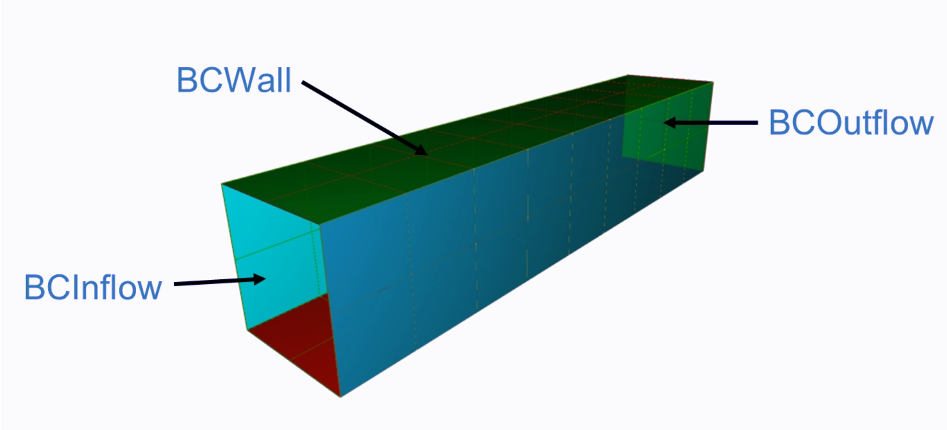

Illustration 6: Simple configuration demonstrating the NGON proposal.

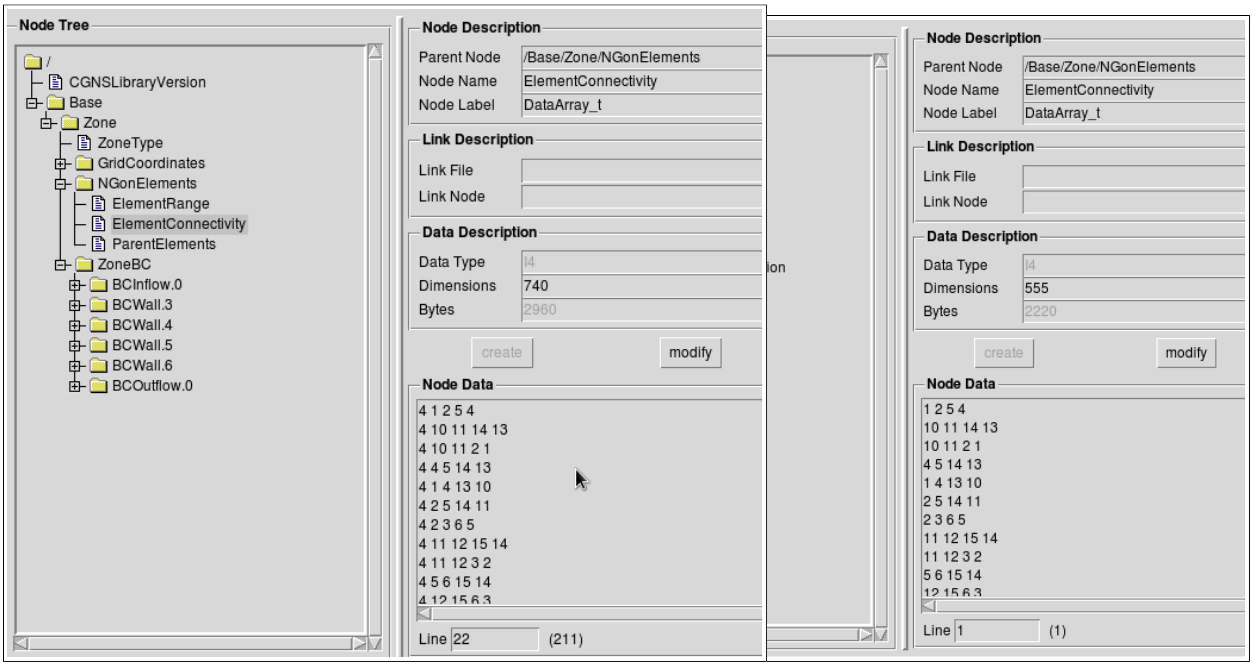

Illustration 7: Previous vs. new ElementConnectivity array

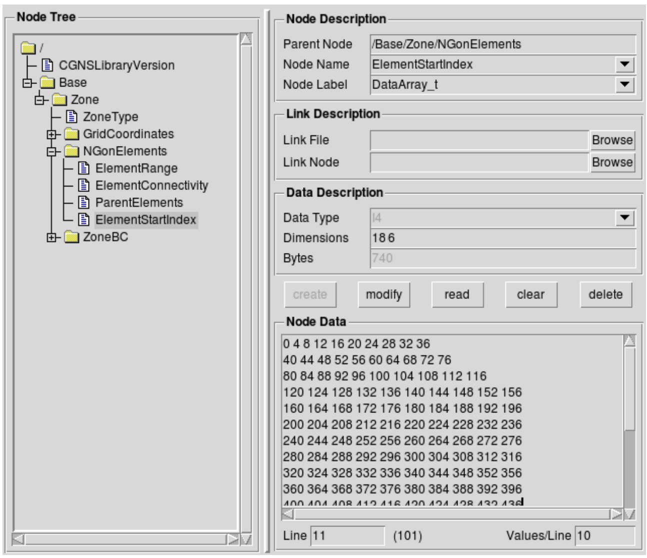

Illustration 8: ElementStartIndex array

4 Implementation note

This section aimes to reflect discussions during previous CGNS meetings related to the implementation impact of this modification proposal. As detailed above, this proposal strongly modifies the representation of the ElementConnectivity node. As a consequence we need to address the backward compatibility issue.

The following solutions were discussed in order to insure backward compatibility:

- CGNSLibraryVersion
  In software, the CGNSLibraryVersion could be used to determine whether the CGNS file uses the current or proposed NGON representation. This number attached to the standard version will allow parsing tools to use the correct NGON representation.
- Conversion utility
  A dedicated tool could be written to convert current CGNS files to the new NGON representation
-

These two solutions should allow a smooth transition for all users of the CGNS standard.

5 Conclusions

This proposal for the NGON representation addresses the HPC issue due to the interlaced data representation of the unstructured connectivity description. The solution optimizes the data representation for parallel IO and has a low impact on the SIDS to deinterlace data.

As shown in the first part, MIXED and NFACES elements are equally concerned by the parallel IO access. As such their SIDS representation should be updated accordingly to the solution chosen for NGON. This issue should be addressed in a later CPEX.

6 Appendix: Document modification list

1. Following Marc Poinot remarks:
   - Rename array from FaceConnectivityPosition to ElementStartIndex (text and figures)
   - Reformulate a sentence describing the ElementStartIndex at line 101
   - Remove remark concerning array size limitation in section 2.1.2.5

2. Following Robert Bush suggestion:
   - Added section “Implementation note” to reflect CGNS committee discussions.

3. Remove multiple typo.

4. Following Gregory Sjaardema feed back:
   - Modify the ElementStartIndex definition to improve the ability to loop on the data array. The new size is N+1 and last index represent the ElementConnectivity total size. This modification is propagated through sections 2 and 3.
   - Correct multiple grammar, typographical or formatting issues.