# NGON modification proposals

## 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 performances.

# 1 Rationales for the modification proposals

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:

```
ElementConnectivity = Etype_1, Node1_1, Node2_1, ... NodeN_1,
                      Etype_2, Node1_2, Node2_2, ... NodeN_2,
                      ...
                      Etype_M, Node1_M, Node2_M, ... NodeN_M
```

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:

```
ElementDataSize =sum(n=[start, end], NPE[ElementType_n] + 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:

```
ElementConnectivity = Nnodes_1, Node1_1, Node2_1, ... NodeN_1,
                      Nnodes_2, Node1_2, Node2_2, ... NodeN_2,
                      ...
                      Nnodes_M, Node1_M, Node2_M, ... NodeN_M
```

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.

```
ElementConnectivity = Nfaces_1, Face1_1, Face2_1, ... FaceN_1,
                      Nfaces_2, Face1_2, Face2_2, ... FaceN_2,
                      ...
                      Nfaces_M, Face1_M, Face2_M, ... FaceN_M
```

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.

```
ElementDataSize =sum(n=[start, end],FPP[ElementType_n] + 1)
```

where FPP[$\text{ElementType}_n$] 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:

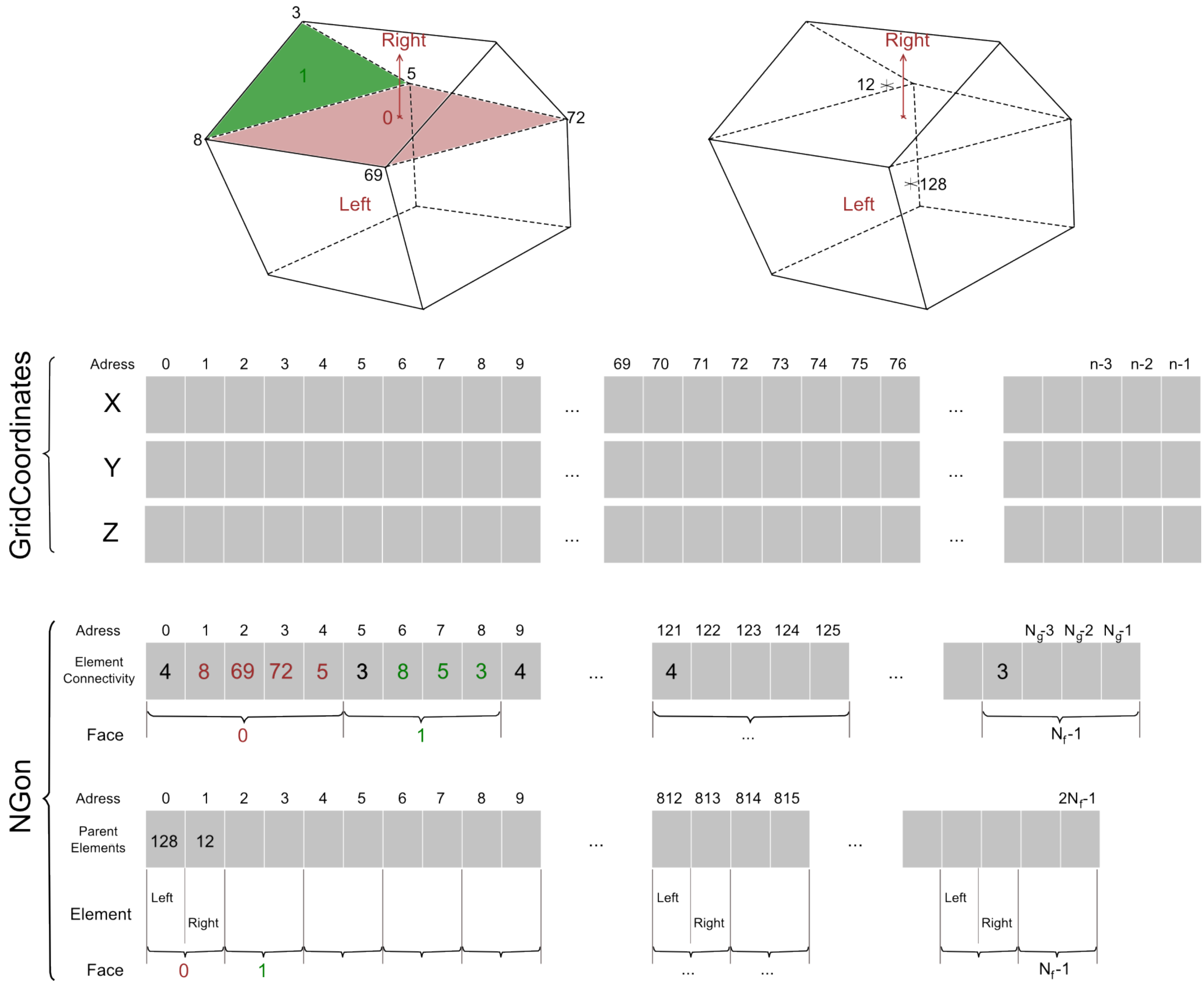


*Illustration 1: CGNS/NGON current face based representation.*

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 : `Nnodes1`
- a list of nodes for a face : `Node11, Node21, ... NodeN1`

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


*Illustration 2: ElementConnectivity interlaced data.*

In parallel, these interlaced data impose the load of the complete ElementConnectivity array on every processors. This has a significant impact on IO performances.

**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 adressed in dedicated CPEX.**

# 2 NGON modification proposals

This proposal offers to alter the NGON SIDS representation in order to allow an efficient parallele IO access and to optimize the data representation.

### 2.1.1 Solution description

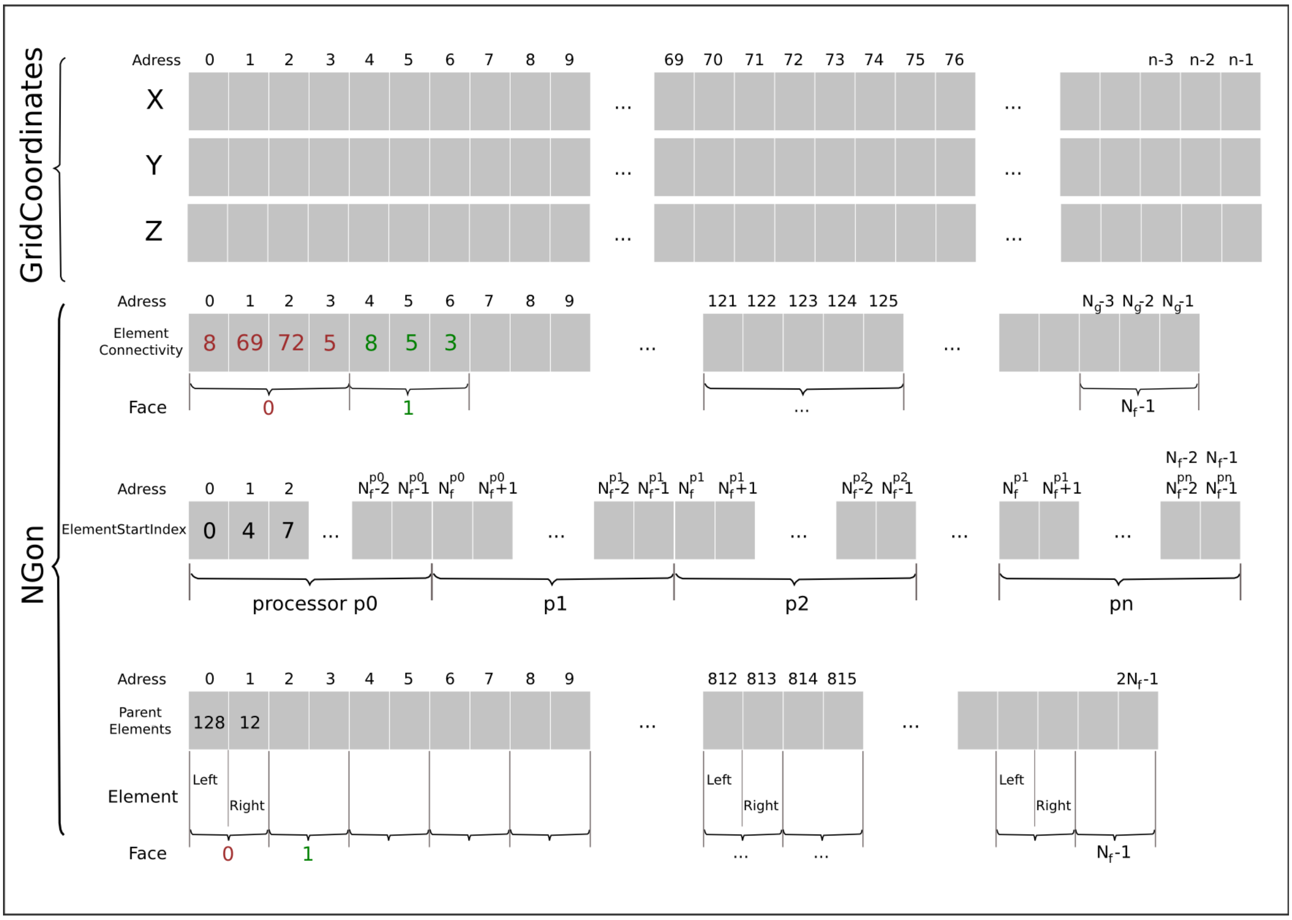


*Illustration 3: NGON with new ElementConnectivity array and ElementStartIndex position array.*

In this representation the ElementConnectivity array is un-interlaced. The following pictures compares the current standard versus the new un-interlaced ElementConnectivity array:

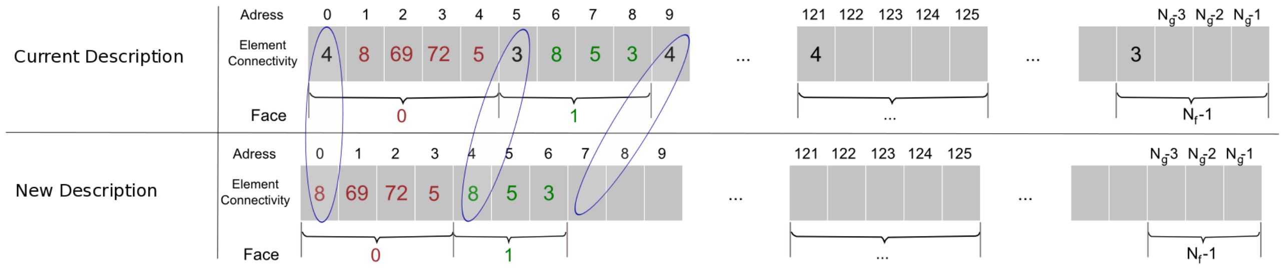


*Illustration 4: Current ElementConnectivity vs new ElementConnectivity.*

The number of face vertex have been removed from the connectivity. As a direct consequence the new array includes a unique data type.

Adresses needed to read or analyze the ElementConnectivity array must be obtained from the new FaceConnectivityPosition array.

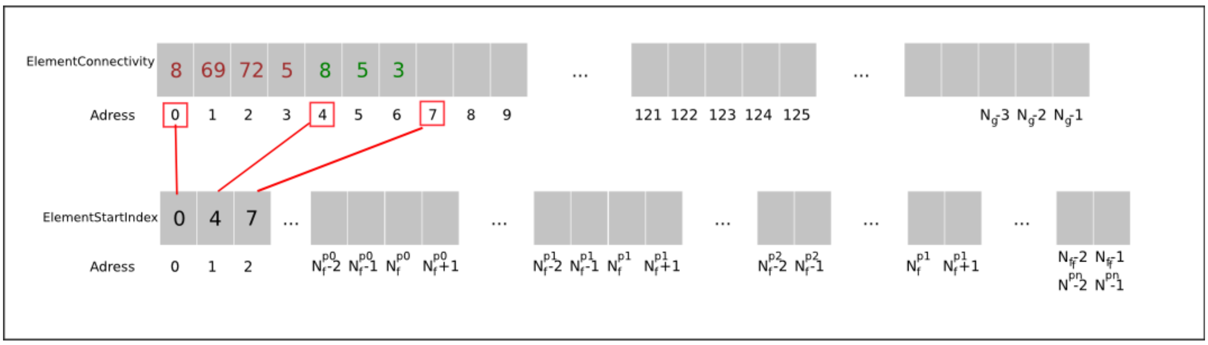


*Illustration 5: ElementStartIndex lists the face position in the ElementConnectivity*

This array lists the position in the ElementConnectivity of the first vertex for each face. This CGNS node is of type DataArray_t. Its data are consistent and its read can be distributed between N processors. Its size is N with N being the number of face of the ElementConnectivity face. The distributed read of this array allows to load partially the ElementConnectivity array.

NB: The user should use the ElementConnectivity array size to correctly access the last vertex of the last face of the ElementConnectivity array.

### 2.1.2 Solution analysis

1. This solution modifies the CGNS standard.
   It strongly modifies the representation of the ElementConnectivity node and it inserts the new array ElementStartIndex.
   These modifications could be challenging for the current CFD database.

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

3. Optmal data size
   This solution do not duplicate data thus the global data size is unchanged and stays optimal.

4. ElementStartIndex incremental index

In this proposal we choose to use the face position in the element connectivty 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:

- 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.

- 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 $P_M$ ($1^{st}$ element of $P_M$ till $1^{st}$ element of $P_{M+1}$-1)

**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.

- 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.

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.

# 3 Example of extension

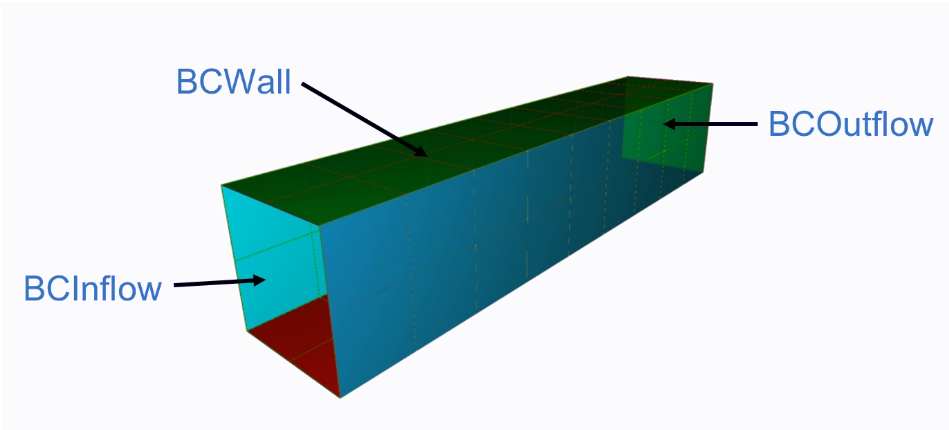


*Illustration 6: Simple configuration demonstrating the NGON proposal.*

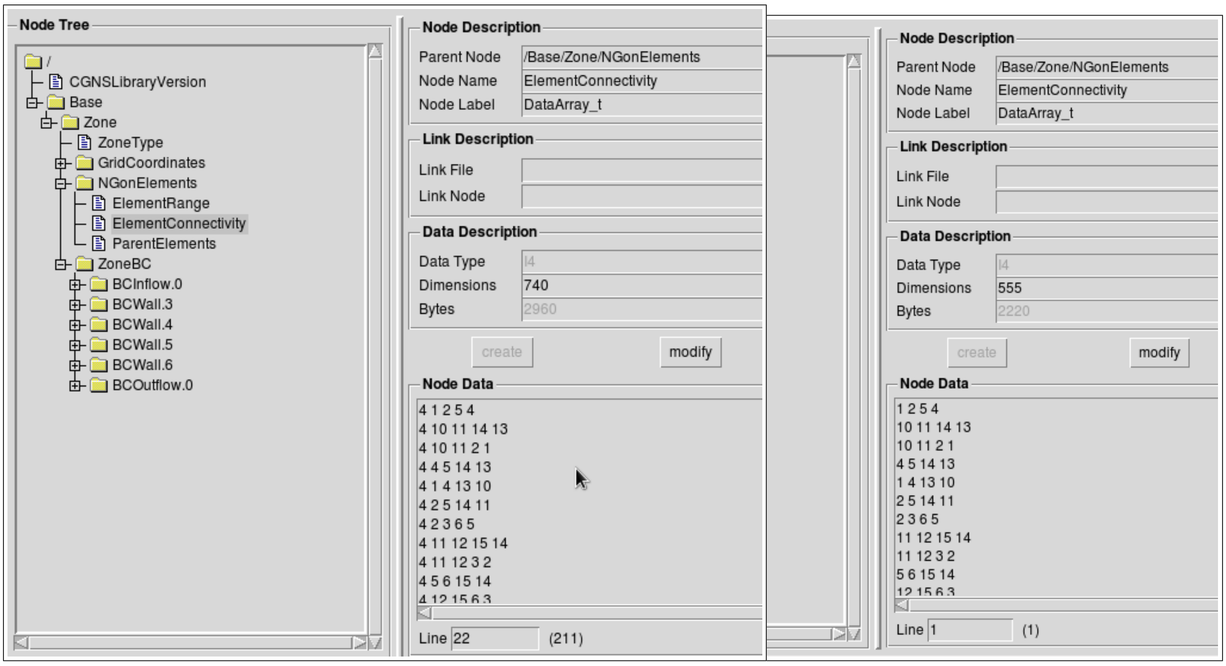


*Illustration 7: Previous vs new ElementConnectivity array*

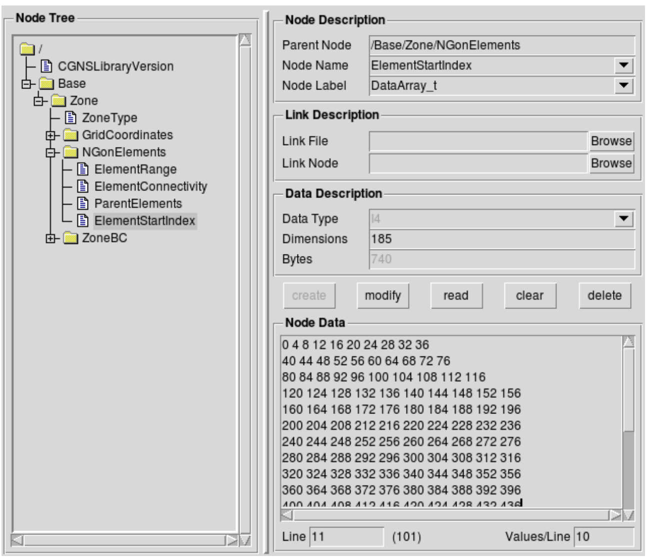


*Illustration 8: ElementStartIndex array*

# 4 Conclusions

This modification proposal for the NGON representation adresses the HPC issue due to the interlaced data representation of the unstructured connectivity description. The above solution optimizes the data representation for parallel IO and has a low impact on the SIDS to un-interlace 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 adressed in dedicated CPEX.

# 5 Annexe : 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