Proposal for the support of hierarchical data structures in the CGNS unstructured meshes format

July 2000

Michel Delanaye, Etienne Robin, Alpesh Patel
michel.delanaye@numeca.be
NUMECA Int., Av. Franklin Roosevelt, 5, 1050 Brussels

It is proposed to modify the current `Elements_t` data structure to include hierarchical information between elements of a mesh. Hierarchical information is created when an unstructured mesh is refined using a cell subdivision procedure or inversely coarsened. The storage of the relation between parent and children elements is very useful for further adaptation of the mesh. We propose to add a new data array of type integer named `HierarchicalData` to store hierarchical data in the `Elements_t` data structure.

Furthermore, the current unstructured element data structure is essentially dedicated to nodal connectivity where each element is defined by listing its nodes. This is a too restrictive. As an example, the data structure of the NUMECA unstructured grid generation and flow solver code is based on cell->faces, face->edges, edge->nodes relationships. These cannot be properly stored within the current CGNS format which only allows for relationship types cell->nodes, face->nodes and edge->nodes.

We propose to enrich the `Elements_t` data structure with a new enumeration called `ConstituentType_t` that defines the type of constituent used in the connectivity data array, such as node, edge or face. In addition, we propose to add a new function to the API, to inquire for the number of data per element depending on `ElementType` and `ConstituentType`.

SIDS definition of the proposed modified `Elements_t` data structure

Here follows the SIDS definition of the modified `Elements_t` data structure with the additional hierarchical information and the definition of the constituent type:

```
Elements_t :=
  {
  List( Descriptor_t Descriptor1 ... DescriptorN ) ;              (o)
  IndexRange_t ElementRange ;                                     (r)
  int ElementSizeBoundary ;                                       (o)
  ElementType_t ElementType ;                                     (r)
  ConstituentType_t ConstituentType;                              (o)
  DataArray_t<int, 1, ElementDataSize> ElementConnectivity ;      (o)
  DataArray_t<int, 2, [ElementSize, 4]> ParentData ;              (o)
  DataArray_t<int, 2, [ElementSize, 2]> HierarchicalData ;        (o)
 }
```

HierarchicalData

The hierarchical information between parent and children elements are stored using two `int` for each element. We propose to store a pointer to the first child element and a pointer to the sibling element. This type of storage is compact and allows to easily reconstruct more usable data structures. A pointer equals to -1 means that there is no child or no sibling element.

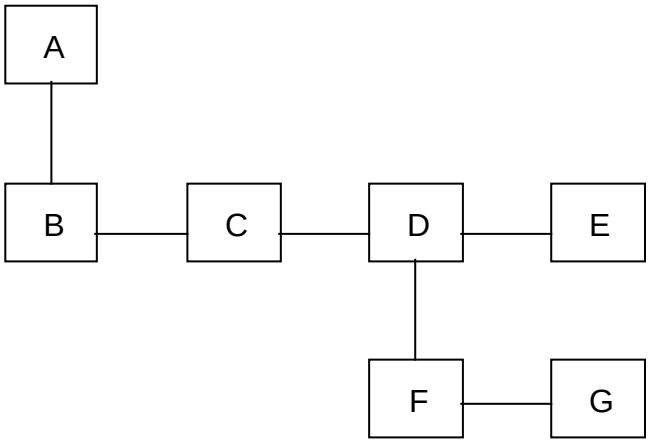

Figure 1: Sample hierarchical connectivity between elements

The situation is illustrated in Figure 1. Element A has 4 children, B, C, D, E, and element D has two children F and G. The `HierarchicalData` array for these 7 elements is given in Table 1.

Cell	Child pointer	Sibling pointer
A	B	-1
B	-1	C
C	-1	D
D	F	E
E	-1	-1
F	-1	G
G	-1	-1

Table 1: `HierarchicalData` array

ConstituentType_t enumeration

The new enumeration `ConstituentType_t` defines the type of constituent used to define the element connectivity and stored in the `DataArray_t ElementConnectivity`. We propose the following enumeration:

```
ConstituentType_t := Enumeration(
                        Null, UserDefined, Node, Edge, Face);
```

The number of information stored in the array `ElementConnectivity` depends on the element type (`ElementType_t`) and the constituent type (`ConstituentType_t`). By default, and in order to ensure backward compatibility, it is assumed that the connectivity is defined using nodal constituents. For example, if `ElementType_t` is set to `HEXA_8`, this implies by default that the 8 nodes hexahedral element is defined using nodal connectivity. However, if

ConstituentType_t is set to Edge or Face, then the HEXA_8 connectivity is defined with its 12 edges or its 6 faces respectively. Similarly, QUAD_4 alone defines by default a quadrangle face connected by 4 nodes. If in addition, ConstituentType_t is set to Edge, then the QUAD_4 connectivity represents the face 4 edges.

In the current SIDS, only nodal connectivity is supported. Therefore, the number of data per element defined in the ElementConnectivity array always equals the number of nodes per element.

The API currently provides a function which return the number of nodes per element (NPE) based on the element type.

e.g. `NPE[HEXA_8] = 8`

With the new proposal, the number of data per element depends not only on the type of element, but also on the type of constituent. So unless ConstituentType is set to Node, this function is no longer sufficient to determine the number of data per element in the ElementConnectivity array. It is therefore proposed to create a new function to inquire the number of data per element given the two parameters, ElementType and ConstituentType. This new function is a generalization of the NPE function. It could be called DPE, for data per element.

e.g.:
```
DPE[HEXA_8, Edge] = 12
DPE[HEXA_8, Face] = 6
DPE[HEXA_8, Node] = 8
NPE[HEXA_8] = 8      (as before)
```

Usage of the new ConstituentType_t generalizes the CGNS unstructured format to any kind of connectivity and lifts potential ambiguities related to the enumeration ElementType_t.