VLSV File Format

April 12, 2012

1 Quad multimesh

"Multimesh" here means that the simulation domain is written to file as separate mesh pieces. This is one of the mesh formats supported by Silo and VisIt. Furthermore, Silo plugin in VisIt supports parallel visualization when there are more than one mesh pieces present. In this Section instructions are given how to write a multimesh to vlsv file. It is assumed that every process writes out its own mesh piece (although this is not really required).

Multimesh type is enabled by using `VLSV::MESH_QUAD_MULTI` as the value of "type" attribute of array `MESH`.

```
map<string,string> attributes
attributes["name"]="SpatialGrid"
attributes["type"]=VLSV::MESH_QUAD_MULTI
<write array>
```

1.1 Bounding Box

This section describes the contents of array `MESH_BBOX`.

The mesh is assumed to be logically Cartesian in the sense that each node can be assigned a unique (i,j,k) index tuple. The bounding box name can be a bit misleading, in practice this array tells `vlsv2silo` how to convert (i,j,k) tuples to (x,y,z) coordinates.

index	value
0	$x_{\min}$
1	$y_{\min}$
2	$z_{\min}$
3	Δx
4	Δy
5	Δz

Table 1: Contents of `MESH_BBOX` array.

Only one process should write MESH_BBOX array. It should contain the values given in Table 1. The important part here is that vlsv2silo calculates cell coordinates in the following manner:

$$\begin{aligned} x &= x_{\min} + i \cdot \Delta x, \\ y &= y_{\min} + j \cdot \Delta y, \\ z &= z_{\min} + k \cdot \Delta z, \end{aligned}$$

where (i,j,k) are indices of each node. Array MESH_BBOX is written out using arraySize=6, vectorSize=1. The datatype must be floating point.

1.2 Node Indices

This section describes the contents of array MESH.

Each process should write its mesh piece in the following manner. Cells are written out as (i,j,k) index tuples, where the indices are the indices of the bottom lower left corner nodes of the cells. The eight nodes associated with a cell are (i,j,k), (i+1,j,k), (i,j+1,k), (i+1,j+1,k), (i,j,k+1), (i+1,j,k+1), (i,j+1,k+1), and (i+1,j+1,k+1). Thus, only one of the nodes is written out – (i,j,k) – the rest can be inferred. This is the logic used by vlsv2silo when it eliminates duplicate nodes from multimesh pieces.

Finally, each process must first write its local cells followed by all remote cells. Variables are written in the same order as cells except that processes do not write remote cell data. vlsv2silo fills missing ghost cell values during conversion (see below).

Array MESH is written out using vectorSize=3. The datatype must be signed or unsigned integer. Each process must write $\mathrm{N_{cells}}$ values (see below).

1.3 Mesh Piece Sizes

This section describes the contents of array MESH_ZONES.

Each process writes the total number of cells (local+remote) and the number of ghost (remote) cells, in its mesh piece. This information is written out into single array element in vlsv file, i.e. each process writes a vector of size two $[\mathrm{N_{cells}}, \mathrm{N_{ghosts}}]$. The arraySize of MESH_ZONES must be equal to number of mesh pieces.

vlsv2silo calculates offsets to other arrays based on the information stored here. For example, $\mathrm{N_{cells}}$ values are used to calculate offsets that tell where the data for each mesh piece is stored in array MESH. This array also defines mesh piece numbering which runs from $[0 \ldots \mathrm{N_{meshes}} - 1]$, where $\mathrm{N_{meshes}}$ is the same as array size of MESH_ZONES.

Array MESH_ZONES is written out using vectorSize=2. The datatype must be signed or unsigned integer.

1.4 Ghost Cell Domain Array

This section describes the contents of array MESH_GHOST_DOMAINS.

When vlsv files are converted to silo format (actually visit requires multimesh data in this format), each mesh piece must contain at least one layer of ghost cells that contain exactly same variable values as the same cells in neighbouring mesh pieces. Each process could write required ghost cell data directly to vlsv files (after synchronization of remote cell data), but this would increase the file size and slow down the data writing process. Instead, information on how to fill ghost cell variable data is written with two arrays.

Each process writes as many elements to array MESH_GHOST_DOMAINS as it has ghost cells in its mesh piece, i.e. N_{ghosts} elements. The elements are written out in the same order as the ghost cell indices in array MESH. Each element tells which mesh piece owns the cell. If each process writes its own mesh piece, each element contains the MPI rank of the process who owns the remote cell. This information tells `vlsv2silo` which mesh piece contains variable data for this ghost cell, i.e. valid values are in the range $[0, N_{\mathrm{meshes}} - 1]$.

Array MESH_GHOST_DOMAINS is written out using `vectorSize=1`. The datatype must be signed or unsigned integer. Each process must write out N_{ghosts} elements.

1.5 Ghost Cell Local ID Array

This section describes the contents of array MESH_GHOST_LOCALIDS.

This array is read by `vlsv2silo` when it fills ghost cell variable values during vlsv to silo conversion, as described in Section 1.4. Array MESH_GHOSTS_DOMAINS only tells which mesh piece owns the ghost cell, but not which cell in that mesh. That information is given here as indices (local IDs) to neighbouring mesh pieces' variable arrays.

For example, assume that MESH_GHOST_DOMAINS array has value "1" for a certain ghost cell in mesh piece #0. This tells `vlsv2silo` that variable values for that cell should be read from mesh piece #1. Assume further that the corresponding ghost cell has value "134" in array MESH_GHOST_LOCALIDS. This tells `vlsv2silo` that the ghost cell is 134^{th} cell in mesh piece #1 – first cell in every mesh piece is always "cell number 0".

Most likely processes need to exchange data with each others before the contents of this array can be written out. Easy way to do this is to define an integer value for each cell (let's call this variable `localID`). Each process then iterates over its local cells as demonstrated in the following pseudocode

```
int i=0
for each local cell
        localID[cell] = i
        ++i
```

Assuming that each process writes out its cells and variables in the same order as local cells are iterated above, it is sufficient that processes fetch the `localID` values defined above to their remote cells, and then write the obtained values to MESH_GHOST_LOCALIDS.

Array `MESH_GHOST_LOCALIDS` is written out using `vectorSize=1`. The datatype must be signed or unsigned integer. Each process must write out N_{ghosts} elements.

1.6 How to Skip Mesh Writing

Multimesh data only needs to be written out if cell partitioning has changed, i.e. after each load balance. However, each vlsv file must contain an array `MESH` but its array size can be zero. This adds an XML tag to the vlsv file which tells `vlsv2silo` that a) the vlsv file contains a multimesh, and b) mesh data should be read from another file. The array `MESH` must contain an attribute "file", whose value is the name of vlsv file where the mesh is written. Compare the example below with the one given in Section 1.

```
int* ptr = NULL
map<string,string> attribs
attribs["name"]="SpatialGrid"
attribs["type"]=VLSV::MESH_QUAD_MULTI
attribs["file"]="state00000.vlsv"
VLSVWriter.writeArray("MESH",attribs,0,1,ptr)
```

Note that `vectorSize` must always be non-zero. If one passes `NULL` pointer directly to `VLSVWriter` instead of `ptr`, compiler will throw an error because the pointer datatype is a template parameter, and `void` pointers do not work here.