Notes on SAMRAI parallel merge tree computation:

These notes are with regards to accessing the data of a SAMRAI application for the purposes of in-situ analytics.

On the high level SAMRAI treats an application in two separate components:

1. The geometry of the problem (grid or mesh)
   - BoxLevel, Box, etc.
2. The data stored on the geometry
   - PatchHierarchy, PatchLevel, Patch, PatchData, etc.

So, in the rest of this document, we would make use of a "patch" while referring to the data and "box" while referring the geometry.

The major requirements of the merge tree algorithm are:

1. Access to the decomposed data - the actual buffer that has the data
2. The shared boundary information with neighboring data i.e. ghost regions
3. The neighborhood information - the neighborhood of boxes to determine the join hierarchy

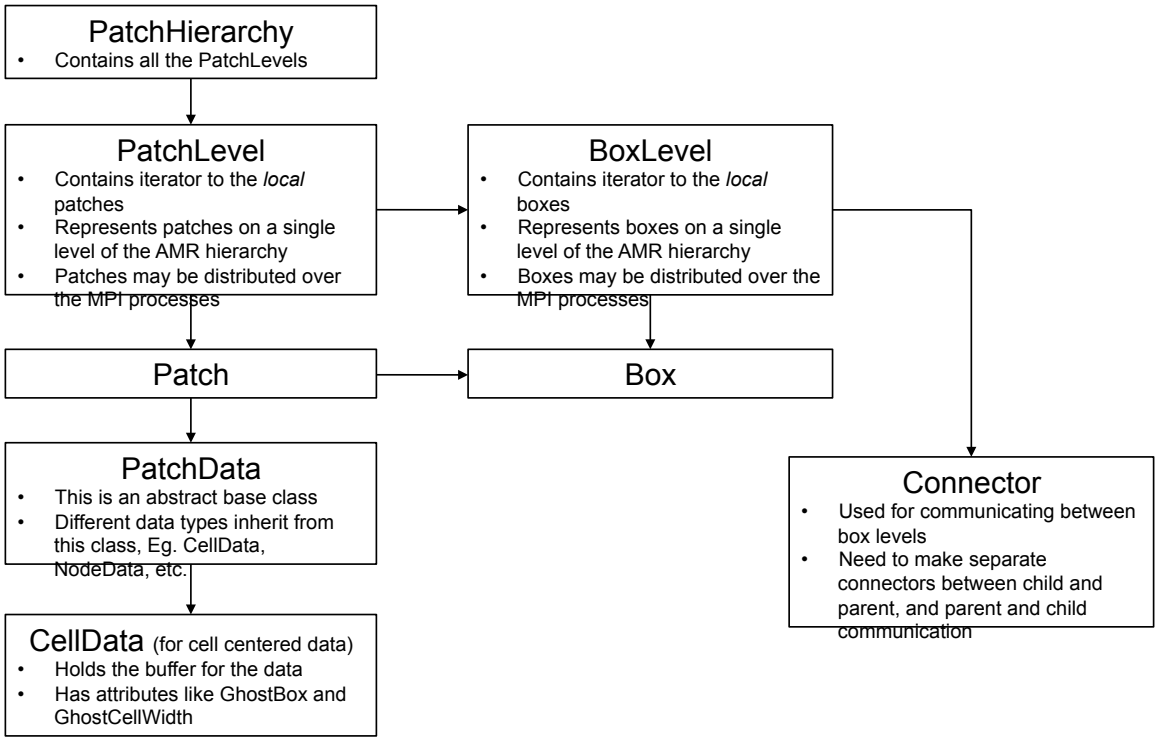
With regards to the above-mentioned points, we have made some investigation into SAMRAI classes to get access to the required information. We shall now describe how SAMRAI holds data and ways in which we can access it at run time.

The following diagram gives the relationship between the various SAMRAI classes. The data buffer is in the CellData class. But that can be accessed from the every Patch in the PatchLevel. The PatchLevel has an iterator to every patch that allows to access the CellData.

We should invoke the parallel merge tree computation for every patch. Since, local patches are all present in the PatchLevel and we would *join* the trees at every patch level first in the *join* hierarchy, it makes sense to have an API that takes in a SAMRAI::hier::PatchLevel along with the *Variable name* and *VariableContext name*.

SAMRAI stores the various data fields in a class called a Variable. Each variable has an associated context which determines the state of the variable. As an example, a variable may have a previous value (earlier timestep t_i) which is modified to get the current value (at t_{i+1}), which is typical in a differential equation.

All the variables' metadata is stored in a static class called VariableDatabase. We can query this class with the variable name and context to obtain the handles to the Varaible class and the VariableContext. The code snippet presents the API signature.



Function to compute merge tree:

```
void mergeTreeCompute(boost::shared_ptr< hier::PatchLevel > patch_level,
                      const std::string &variable_name,
                      const std::string &context_name);
```

The attached code example shows the usage of this API in a SAMRAI application and the implementation of the API.

Observation: We have used the MblkLinAdv example to extract the data. In the code, we extract the “vol” field with its “CURRENT” context. For this field, there is no ghost data present. Depending on the variable, the application may chose to include a ghost region. We still have not figured out a way to query or extract the ghost information from the CellData. The CellData does inherit the member functions called getGhost and getGhostWidth from PatchData, but these return an empty ghost at least for the “vol” and “uval” field in this example. We should explore more examples to see the behavior of this methods and if they provide us with the required ghost information.