PHASTA File-less Integration with Mesh Adaptation

Monday, September 19, 2011

Cameron Smith, Brian Orecchio, Onkar Sahni

A field data transfer mechanism between solver (phSolver) and adaptor (phParAdapt) was defined to increase interoperability between these software components. This mechanism relies on functional interfaces to enable automated inter-component field association, where field meta-data object is used to define, query and transfer the underlying simulation information needed. Field meta-data object contains a high-level description of a generic field and is composed of: (i) field-tag (e.g., name or alias of the field), (ii) field-dimension (e.g., degrees-of-freedom) and (iii) field-classification (e.g., the association with a sub-domain of the geometric model). A C++ class structure was used to implement these functional interfaces. Each simulation component provides the field meta-data objects associated to the fields they require from the other component (e.g., coordinates and connectivity fields as required by the flow solver component, and similarly, error indicator and solution fields as required by the adaptor component). Thus, in a multi-component adaptive simulation the produced fields of one component satisfy the prerequisites of another component. This process is depicted on lines 17 and 18 of Figure 1, which show association of fields produced by the solver to those required by the adaptor. After association of fields between components, for each field the field-data transfer mechanism iterates over the dimensions of the field and invokes accessors and modifiers on a given field, respectively (i.e., to retrieve the field-data from the supplier component and subsequently pass it to the consumer component). Each component implements the underlying accessors and modifiers functions in order to abstract the internal representations of the field data. Similarly, lines 20 through 24 in Figure 1 depict the transfer of the solver field-data to adaptor.

Additionally, solver code (phSolver) was refactored to support the necessary set of functional interfaces for procedure control and field access as described above. To do this, a phSolver class was implemented. Singleton class structure was used here in order to account for heavy reliance on global data due to legacy design choices. This class currently supports: (i) access to the single phSolver instance, (ii) reading solution, boundary condition and mesh data from files (writing/checkpointing will be provided in next release), and (iii) field data constructors, accessors and modifiers. Fortran 2003 iso_c_binding and modules were used to update the previous common block implementation of shared global data between procedures implemented in Fortran 77/90 and C. These mechanisms enable explicit coupling of variables and procedures, which reduces the complexity and frailty of the coupling, improves readability, and also results in less code to maintain since 'block data' initialization statements were removed in favor of initialization within the module declaring the variables.

In the next period SCOREC researchers will resume implementation of the phParAdapt functional interfaces and complete the phSolver functional interfaces such that an adaptive loop initiated from phSolver restart data files is supported.

```
int main ()
{
  //Initialize phSolver and phParAdapt
  phSolver* phS = new phSolver(<configuration file name>)
  phParAdapt* phA = new phParAdapt(<configuration file name>)

  phS->readMeshAndSolution_fromFiles(<LIST OF FILE NAMES e.g. geombc.*.dat, restart.*.*> )
  phA->readMeshDatabaseAndGeometry_fromFiles(<mesh database file e.g. FMDB or Simmetrix Mesh>)
  phA->readAttributes_fromFile(<attributes file>)

  //Time Stepping - Adaptation Loop
  while( 1 )
  {
    int returnCode = phS->solve()
    if( returnCode == <Adaptation Required>) {
       // transfer data from phSolver to phParAdapt
       for reqField in phA->requiredFields() {
          satField = phS->getField(reqField)
          // transfer data
          for unitIdx in satField.getNumUnits() {
             for varIdx in satField.getNumVar() {
                val = phS->getValueDbl(satField, unitIdx, varIdx)
                phA->setValueDbl(reqField, unitIdx, varIdx, val)
             }
          }
       }
       phA->adapt()
       phA->preProcess()
       // transfer data from phParAdapt to phSolver
       for reqField in phS->requiredFields() {
          satField = phA->getField(reqField)
          // resize phSolver fields
          phS->resize(reqField, satField.getNumUnits(), satField.getNumVar())
          // transfer data
          for unitIdx in satField.getNumUnits() {
             for varIdx in satField.getNumVar() {
                val = phA->getValueDbl(satField, unitIdx, varIdx)
                phS->setValueDbl(reqField, unitIdx, varIdx, val)
             }
          }
       }
    }
    else if ( returnCode == <Error> ) {
       exitWithError()
    }
    else if ( returnCode == <MaxTimeStepsReached> ) {
       exit()
    }
  } // end while
} // end main
```

Figure 1. Pseudo code for file-less solve-adapt loop.