Chapter 1

Parallel solution of the FSI channel with leaflet problem: Distributing problems with algebraic node updates

This document provides an overview of how to distribute fluid-structure interaction problems in which algebraic node update methods are used to deform the fluid mesh in response to changes in the shape of the domain boundary. It is part of a series of tutorials that discuss how to modify existing serial driver codes so that the Problem object can be distributed across multiple processors.

As discussed in the general MPI tutorial, the parallel implementation of algebraic node update methods for specific meshes is greatly facilitated if the GeomObject that describes the motion of the moving domain boundary is available on all processors. In FSI problems, the moving boundary is typically represented by a MeshAsGeomObject – a compound GeomObject formed from the lower-dimensional mesh of SolidElements that define the moving boundary of the fluid domain. To ensure that the MeshAsGeomObject remains available on all processors when the underlying mesh is distributed, we use the function Mesh::keep_all_elements_as_halos() to indicate that all elements should be retained on all processors.

Note

The procedure described here is appropriate **only** if the solid mesh used to create the MeshAsGeomObject is **not adapted** during the solution of the problem. We refer to another tutorial for instructions on how to deal with the case when the solid mesh is itself adapted.

1.1 Revisiting the FSI channel with leaflet problem

We demonstrate the methodology for the problem of flow past an elastic leaflet in a 2D channel. Most of driver code is identical to its serial counterpart and we only discuss the changes required to distribute the problem. Please refer to another tutorial for a more detailed discussion of the problem and its (serial) implementation.

1.1.1 The main function

As with other parallel driver codes, the changes from the serial driver code are extremely modest, essentially including the initialisation and shutdown of MPI. Once the problem is set up, we call Problem

::distribute(...), using the boolean flag report_stats to indicate that the statistics of the distribution should be reported on screen.

```
  // Distribute problem using METIS to determine the partitioning
  problem.distribute(report_stats);
```

1.1.2 The problem class

The only additional function is actions_after_distribute(), described later in this tutorial.

1.1.3 The problem constructor

The only difference from the serial counterpart is that we insist that all beam elements are kept as halos when the mesh is distributed.

```
// Discretise leaflet
//-------------------

// Geometric object that represents the undeformed leaflet
UndeformedLeaflet* undeformed_wall_pt=new UndeformedLeaflet(x_0);

//Create the "wall" mesh with FSI Hermite beam elements
unsigned n_wall_el=5;
Wall_mesh_pt = new OneDLagrangianMesh<FSIHermiteBeamElement>
 (n_wall_el,hleaflet,undeformed_wall_pt,wall_time_stepper_pt);

// Flag to tell the wall mesh that all its elements should be halo
Wall_mesh_pt->set_keep_all_elements_as_halos();
```

1.1.4 Actions after adaptation

The actions_after_adapt() function requires only a minor change from the serial version — a simple re-ordering of the sequence in which the various steps are performed. In serial it does not matter in which order the (re-)assignment of the auxiliary node update functions and (re-)setup of the fluid-structure interaction are performed. In the parallel version, however, the assignment of the auxiliary node update functions must take place **after** the (re-)setup of the fluid-structure interaction. This is because the function FSI_functions::setup_fluid↩_load_info_for_solid_elements(...) creates halo copies of the fluid elements (and their nodes!) if the required fluid element is not present on the current processor. Any newly-created halo fluid nodes on the FSI boundary are accessible via the usual boundary lookup schemes and must be told about the auxiliary node update function which applies the no-slip condition.

```
//==== start_of_actions_after_adapt=================================
/// Actions_after_adapt()
//==================================================================
template<class ELEMENT>
void FSIChannelWithLeafletProblem<ELEMENT>::actions_after_adapt()
{
 // Unpin all pressure dofs
 RefineableNavierStokesEquations<2>::
  unpin_all_pressure_dofs(Fluid_mesh_pt->element_pt());

 // Pin redundant pressure dofs
 RefineableNavierStokesEquations<2>::
  pin_redundant_nodal_pressures(Fluid_mesh_pt->element_pt());


 // Re-setup FSI
 //-------------

 // Work out which fluid dofs affect the residuals of the wall elements:
 // We pass the boundary between the fluid and solid meshes and
```

```
 // pointers to the meshes. The interaction boundary is boundary 4 and 5
 // of the 2D fluid mesh.

 // Front of leaflet: Set face=0 (which is also the default so this argument
 // could be omitted)
 unsigned face=0;
 FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
  (this,4,Fluid_mesh_pt,Wall_mesh_pt,face);

 // Back of leaflet: face 1, needs to be specified explicitly
 face=1;
 FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
  (this,5,Fluid_mesh_pt,Wall_mesh_pt,face);

 // (Re-)apply the no slip condition on the moving wall
 //----------------------------------------------------

 // The velocity of the fluid nodes on the wall (fluid mesh boundary 4,5)
 // is set by the wall motion -- hence the no-slip condition must be
 // re-applied whenever a node update is performed for these nodes.
 // Such tasks may be performed automatically by the auxiliary node update
 // function specified by a function pointer:
 for(unsigned ibound=4;ibound<6;ibound++ )
  {
   unsigned num_nod= Fluid_mesh_pt->nboundary_node(ibound);
   for (unsigned inod=0;inod<num_nod;inod++)
    {
     Fluid_mesh_pt->boundary_node_pt(ibound, inod)->
      set_auxiliary_node_update_fct_pt(
       FSI_functions::apply_no_slip_on_moving_wall);
    }
  } // aux node update fct has been (re-)set

} // end_of_actions_after_adapt
```

1.1.5 Actions after distribution

The actions required after distribution are extremely similar to those required after adaptation because distribution deletes certain elements (or replaces them by halo copies). The only difference is that the redundant pressure degrees of freedom do not have to be re-pinned.

```
//==== start_of_actions_after_distribute=============================
/// Actions_after_distribute()
//====================================================================
template<class ELEMENT>
void FSIChannelWithLeafletProblem<ELEMENT>::actions_after_distribute()
{
 // Re-setup FSI
 //-------------

 // Work out which fluid dofs affect the residuals of the wall elements:
 // We pass the boundary between the fluid and solid meshes and
 // pointers to the meshes. The interaction boundary is boundary 4 and 5
 // of the 2D fluid mesh.

 // Front of leaflet: Set face=0 (which is also the default so this argument
 // could be omitted)
 unsigned face=0;
 FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
  (this,4,Fluid_mesh_pt,Wall_mesh_pt,face);

 // Back of leaflet: face 1, needs to be specified explicitly
 face=1;
 FSI_functions::setup_fluid_load_info_for_solid_elements<ELEMENT,2>
  (this,5,Fluid_mesh_pt,Wall_mesh_pt,face);

 // (Re-)apply the no slip condition on the moving wall
 //----------------------------------------------------

 // The velocity of the fluid nodes on the wall (fluid mesh boundary 4,5)
 // is set by the wall motion -- hence the no-slip condition must be
 // re-applied whenever a node update is performed for these nodes.
 // Such tasks may be performed automatically by the auxiliary node update
 // function specified by a function pointer:
 for(unsigned ibound=4;ibound<6;ibound++ )
  {
   unsigned num_nod= Fluid_mesh_pt->nboundary_node(ibound);
   for (unsigned inod=0;inod<num_nod;inod++)
    {
```

```
     Fluid_mesh_pt->boundary_node_pt(ibound, inod)->
      set_auxiliary_node_update_fct_pt(
       FSI_functions::apply_no_slip_on_moving_wall);
    }
  } // aux node update fct has been (re-)set


} // end_of_actions_after_distribute
```

1.1.6 The doc_solution() function

As usual, we add the processor number to the end of the filename for each output file to make sure that the different processors don't over-write each other's output.

The trace file documents the time trace of the imposed influx and the displacement of the node at the tip of the leaflet. It could be written by any processor since all solid elements are retained everywhere. We only write to the trace file from processor 0.

```
// Write trace file (only on processor 0) -- Tip node exists on
// every processor (because all wall elements are kept as halos)
// but we don't want different processors to overwrite (or replicate)
// the trace file.
if (this->communicator_pt()->my_rank()==0)
 {
  trace << time << " "
        << Global_Physical_Variables::flux(time) << " "
        << tip_node_pt->x(0) << " "
        << tip_node_pt->x(1) << " "
        << tip_node_pt->dposition_dt(0) << " "
        << tip_node_pt->dposition_dt(1) << " "
        << doc_info.number() << " "
        << std::endl;
 }
```

1.2 Source files for this tutorial

The source files for this tutorial can be found in

demo_drivers/mpi/multi_domain/fsi_channel_with_leaflet

Similar examples of modified driver codes for FSI problems for a channel with a collapsible wall and an oscillating ring can be found in

demo_drivers/mpi/multi_domain

1.3 PDF file

A pdf version of this document is available.