Scalable Parallel Solution Coupling for Multi-Physics Reactor Simulation

Timothy J. Tautges, Alvaro Caceres

Argonne National Laboratory, Argonne, IL 60439 USA

Abstract. Reactor simulation depends on the coupled solution of various physics types, including neutronics, thermal/hydraulics, and structural mechanics. This paper describes the formulation and implementation of a parallel solution coupling capability being developed for reactor simulation. The coupling process consists of mesh and coupler initialization, point location, field interpolation, and field normalization. This capability is tested on two example problems; one of them is a reflector assembly from the ABTR. Performance of this coupler in parallel is reasonable for the problem size chosen and the chosen range of processor counts. The runtime is dominated by startup costs, which amortize over the entire coupled simulation. Future efforts will include adding more sophisticated interpolation and normalization methods, to accommodate different numerical solvers used in various physics modules and to obtain better conservation properties for certain field types.

1. Introduction

Simulation applications are evolving toward more complexity in terms of both the physics modeled as well as the length scales at which they are modeled. These "multi-physics" and "multi-scale" codes share a common need to transfer solutions from one discretization to another. In many cases, a multi-physics or -scale code does not originate at once, but is assembled from several, often stand-alone codes. For these codes, solution transfer must consider the overall process, including not only the mathematical details of mapping from one mesh to another, but also the efficient communication and mapping between meshes whose distribution on the parallel machine may not have been chosen to optimize solution transfer. In this paper, we discuss a modular approach to solution transfer, which accommodates a variety of mesh types, mapping functions, and multi-physics solution frameworks.

The ITAPS project is developing common interfaces to mesh (iMesh), geometry (iGeom), and other data, and services based on them [1]. The interfaces are designed to be datastructure-neutral, and can be called directly from C, C++, and Fortran applications. IMesh defines a data model consisting of entities (the finite element plus polygons and polyhedra), entity sets (arbitrary groupings of entities and other sets), the interface instance or "root set", and tags (data annotated on members of any of the other data types). Services are available for mesh smoothing [2] and adaptive mesh refinement [3], to name a few. The ITAPS interfaces are being used as the basis for the SHARP reactor simulation project [4], which performs the coupled solution of thermal/hydraulics and neutronics for fast reactor cores. Solution transfer plays a major role in this effort.

The solution mapping process consists of transferring a field from a source mesh to a target mesh, where the source and target meshes are each used by a specific physics, are represented in iMesh, and are distributed over a parallel machine. In this work we focus on solution transfer for overlapping 3D

domains. Motivated by a component-based approach to multi-physics code development, we make the following assumptions when designing solution transfer capability:

1. Each physics is free to choose the mesh type and parallel distribution best for that physics, without regard for its suitability for solution transfer.

2. Each physics type and mesh is distributed across a set of processors, defined by an MPI communicator for each mesh

3. If multiple physics types are assigned to a given processor, the mesh for all physics on that processor is stored in a single iMesh instance, and that instance communicates with all other processors containing pieces of any of those meshes.

There has been a great deal of previous work on the topic of solution transfer (e.g. see Refs. [5-6]). Many papers focus on the mathematical properties of mapping solutions, with various constraints imposed which are specific to the physics being solved. Also, various mapping functions are used, depending on the shape functions used to compute the field. Despite these variations, there are also many similarities in the various approaches to parallel solution transfer. For example, most approaches require locating one mesh in another, communicating fields between processors, and normalizing results after interpolation (though the method actually used in that normalization may vary). Hence, we approach the problem of solution transfer from a process point of view, where distinct steps are implemented in a modular fashion. This allows substitution of one or more specific steps, e.g. mapping function, while reusing other parts of the implementation.

2. Parallel Solution Transfer Algorithm

The parallel solution transfer process is broken down into several functions:

Initialization: We assume the mesh has been read onto the machine and is initialized (with parallel interfaces) according to the assumptions above. Parallel details of the mesh (interface and partitioning sets, communicator, etc.) are encapsulated in the iMesh instance. To facilitate point location on a source mesh, a kdtree decomposition is computed for the local part of the source mesh on each processor. In addition, the bounding box corner points for the root node of this tree on each processor is sent to all processors holding target mesh. These box coordinates are used to narrow the search for source elements containing each target point, which reduces parallel communication during this process.

Point Location: Point location is the process of choosing points at which data will be coupled (usually derived from a target mesh), and locating the processor(s), element(s), and parametric coordinates in those element(s) that contain each target point. The source element(s) containing a given point can be on the local or remote processor. The algorithm used in this phase is shown in Figure 2.1. In this case, target points are derived from the target mesh, but need not be restricted to vertices in that mesh. This algorithm reduces data communication by storing point location data on the source mesh processor, and passing an index into that list to the target mesh processor requesting the interpolation point. This incurs a small extra storage cost (one tuple of three integers per target point) but reduces communication during the interpolation phase. All processors holding source and target mesh participate in the scatter/gather communication. In theory this communication could be optimized in cases where the target and source processor sets were disjoint (no processor held both source and target mesh). However, it is unclear whether that would save much communication time, or whether it would be worth the added complexity in the application.

1. Choose target points t_i
2. $\forall$ t_i:
   (a) $\forall$ root box B_j containing t_i:
      i. If $j != p$ add tuple *Tu1(j, i, xyz)*, where *xyz* are the coordinates of t_i
      ii. Else add tuple *Tu_tmp(i)*
3. Scatter/gather *Tu1(j, i, xyz)*, using *j* as destination processor; *j* gets replaced on destination processor with source processor
4. $\forall$ *tu1* $\in$ *Tu1*
   (a) Traverse the source mesh kdtree to find the leaf node(s) whose boxes contain t_i
   (b) $\forall$ element *e* in each leaf node, find the natural coordinates *pqr* in *e*
   (c) If *pqr* $\in$ *[0, 1]*, add tuple *Tu2(e, pqr)[k]*, and tuple *Tu3(j, i, k)[l]*
   (d) Else add tuple *Tu3(j, i, -1)[l]*
5. Scatter/gather *Tu3(j, i, k)*, using *j* as destination processor; *j* gets replaced on destination processor with source processor
6. $\forall$ *tu3* $\in$ *Tu3(j, i, k)*, if *k* != -1, add tuple *Tu4(j, i, k, f)[l]*, where *f* is a double-precision parameter used in the interpolation phase

Repeat steps 4a-c for all tuples in *Tu_tmp*; in step 4c, put contained points in *Tu5(i,k)* instead of *Tu3*

Figure 0: Algorithm for point location phase of solution coupling.

Interpolation: During the execution of a physics component, field values are computed and stored on various entities in the source mesh on each processor. This process may be repeated for iterations working towards some steady state, or as part of a time advancement calculation. The field values on this source mesh must be interpolated onto the target mesh. The algorithm used in the interpolation stage is described in Figure 2. As described in the introduction, our approach is modular, allowing the application to choose the type of shape function used for the interpolation. In our current implementation, the shape functions are implemented in the iMesh/MOAB library. This is discussed more later. The only requirement in this implementation is that the location of the target point within the element be represented using three double precision numbers whose bounds are between -1 and 1.

Normalization: After interpolation, a field is assigned values at target points. If repeated interpolations are done from one mesh to another and back again, the values of the field can "drift" due to numerical roundoff error and other factors, depending on the shape function evaluations. In some cases this drift can be disregarded, but in others it can add or remove energy in the physical system. To avoid this, the coupler also provides for normalizing some integrated quantity over the target mesh to match the same integrated quantity on the source mesh. Currently, we only implement

1. Choose field to interpolate, *f*
2. Scatter/gather *Tu4(j, i, k, f)* using *j* as the destination processor; *j* is replaced with the source processor *i*
3. $\forall$ *Tu4(j, i, k, f)[l]*, interpolate *f(e, pqr), e, pqr* $\in$ *Tu3(e, pqr)[k]* $\rightarrow$ *f* $\in$ *Tu4(j, i, k, f)[l]*
4. Scatter/gather *Tu4(j, i, k, f)* using *j* as the destination processor; *j* is replaced with the source processor *i*
5. $\forall$ *Tu4(j, i, k, f)[l], f* $\rightarrow$ *f[i]*
6. $\forall$ *Tu5(i, k)[l],* interpolate *f(e, pqr), e, pqr* $\in$ *Tu3(e, pqr)[k]* $\rightarrow$ *f[i]*

Figure 0: Algorithm for interpolation phase of solution coupling.

the global integrated quantity, according to the integration function associated with the shape function used in the interpolation. This is discussed more later.

3. Coupling Results

Error! Reference source not found. shows the ABTR reflector assembly model, with different colors used to indicate different geometric volumes. The source mesh consists of 12k hexahedral elements, while the target mesh contains 130k tetrahedral elements. For these examples, an analytic function is defined at hexahedral mesh vertices, and this field is mapped to the target mesh vertices based on trilinear basis functions. The original fields and the coupled results are also shown in

Figure 1: Geometry and target mesh for the ABTR reflector (left); source field defined on hexahedral mesh (middle); transferred field on target tetrahedral mesh (right).

. Note that this shows a "pseudocolor" plot, which interpolates the vertex values to show a smoothly-varying field.

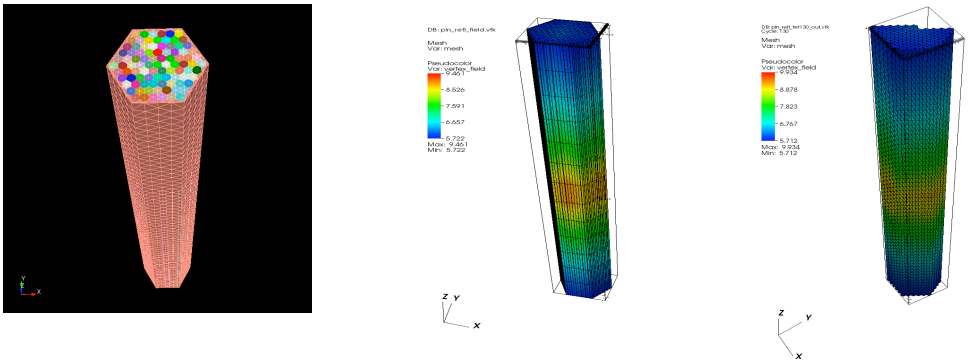

Figure 1: Geometry and target mesh for the ABTR reflector (left); source field defined on hexahedral mesh (middle); transferred field on target tetrahedral mesh (right).

Run times for the various coupling steps are shown in Figure 2; normalization time was not measured. The data shown is for coupling the same meshes on increasing numbers of processors. In absolute terms, the initialization of the problem dominates the other phases in this problem, with the interpolation phase accounting for a maximum of 0.06%, for the 32-processor run. For problems where the mesh is not deforming, the interpolation stage is the only one repeated for each coupling step, therefore these times will be acceptable. Parallel efficiency shows mixed results, with poor scaling for mesh import and interpolation, and moderate to good scaling for instantiation and point location. Although interpolation time appears to scale quite poorly; absolute times for this portion of the example were below 0.001 seconds, which is close to the clock tick size. Timing results should be generated on larger problems to better assess scaling of the this step.

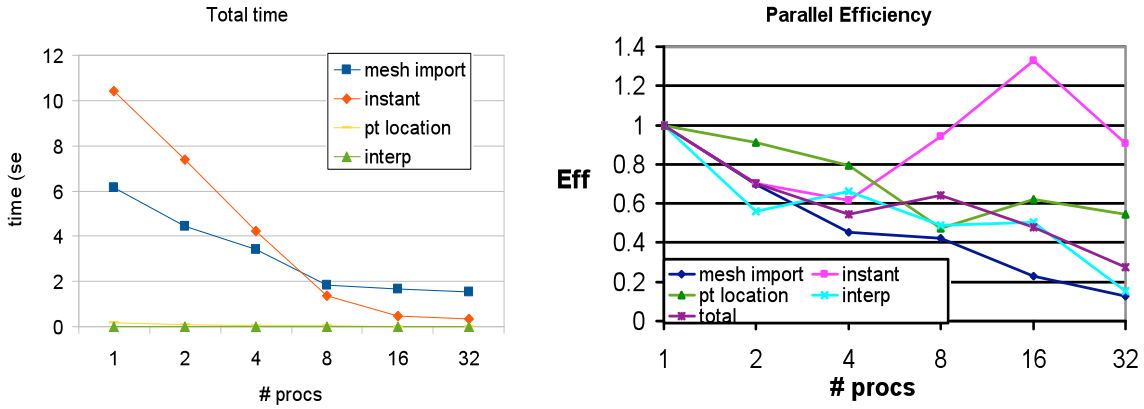

Figure 2: Execution time for coupling (left), parallel efficiency (right).

Several other conclusions can be drawn from this data. First, the mesh import time is not being reduced as quickly as other times with increasing processor counts. The capability to read only those portions of mesh assigned to a given processor is already under development.

Second, the point location algorithm described in Figure 0 includes a sequential search over the bounding box extents for each processor for each point being interpolated. This step will grow in cost as the number of processors increases. We intend to develop a tree-based search of those boxes as well, bringing scaling closer to O(log(p)) than to O(p) for this part of point location. Since the tree structure is already implemented, the development of this capability will be very straightforward.

Finally, we note that normalization of the solution over all processors is currently not done, since a vertex-based field is being mapped in these examples. However, that step should not require a great deal of execution time, since most of the element-based calculations are local to each processor, followed by a reduction of results over all processors

4. Summary & Future Work

This report describes the formulation and implementation of a parallel solution coupling capability. The coupling process consists of mesh and coupler initialization, point location, interpolation, and normalization. Coupling results show good qualitative agreement with the field being coupled. Scaling varies for the process steps; the interpolation step scales poorly, but takes a negligible amount of absolute time compared to other steps. In the case of static mesh, this step is the only one repeated for each iteration or time step. Larger examples must be run to investigate whether this step scales well.

The implementation of solution transfer in MOAB will be extended in various ways. More interpolation methods will be added as needed, based on the physics components being coupled. Higher-order basis functions are needed immediately, and have already been implemented. Normalization over specific subsets in the mesh is also needed, to focus on specific sub-regions in the problem. This extension will be straightforward based on material sets defined in the source and target meshes. Finally, tuning the interpolation and normalization methods to achieve various conservation properties will also be studied. This is a research topic, which will be of great interest to the scientific community as coupled multi-physics simulations become more common.

Acknowledgments

This work was supported by the US Department of Energy's Scientific Discovery through Advanced Computing program under Contract DE-AC02-06CH11357. Argonne National Laboratory (ANL) is managed by UChicago Argonne LLC under Contract DE-AC02-06CH11357.

References

[1] K.K. Chand, L.F. Diachin, X. Li, C. Ollivier-Gooch, E.S. Seol, M.S. Shephard, T. Tautges, and H. Trease, "Toward interoperable mesh, geometry and field components for PDE simulation development," *Engineering with Computers*, vol. 24, 2008, pp. 165-182.

[2] Patrick M. Knupp, "Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes," *International Journal for Numerical Methods in Engineering*, vol. 45, 1999, pp. 37-45.

[3] M.S. Shephard, K.E. Jansen, O. Sahni, and L.A. Diachin, "Parallel adaptive simulations on unstructured meshes," *Journal of Physics: Conference Series*, vol. 78, month> </month> <month>08</month>/<year>2007</year. , p. 012053.

[4] A. Siegel, T. Tautges, A. Caceres, D. Kaushik, P. Fischer, G. Palmiotti, M. Smith, and J. Ragusa, "Software design of SHARP," *Joint International Topical Meeting on Mathematics and Computations and Supercomputing in Nuclear Applications, M and C + SNA 2007*, La Grange Park, IL 60526, United States: American Nuclear Society, 2007, p. 13.

[5] J. Grandy, "Conservative Remapping and Region Overlays by Intersecting Arbitrary Polyhedra," *Journal of Computational Physics*, vol. 148, 1999, pp. 433-466.
[6] L. Margolin, "Second-order sign-preserving conservative interpolation (remapping) on general grids," *Journal of Computational Physics*, vol. 184, 2003, pp. 266-298.