

Scalable Parallel Solution Coupling for Multi-Physics Reactor Simulation



Global Nuclear Energy Partnership

*Prepared for
U.S. Department of Energy
Reactor Campaign
T. J. Tautges, A. Cacares,
M. O. Delchini, and D. Kaushik
June 30, 2008
GNEP-REAC-PMO-MI-DV-2008-000156*

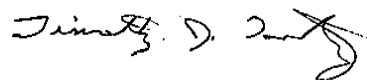


DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

Prepared by:

Principal Investigator



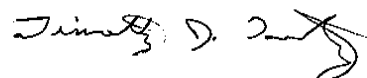
6/30/08

Timothy J. Tautges

Date

Reviewed by:

GNEP Work Package Manager

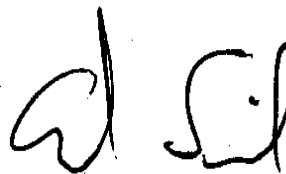


6/30/08

Timothy J. Tautges

Date

GNEP WBS Level 4 Manager



7/1/08

Andrew R. Siegel

Date

Approved by:

GNEP Reactor Campaign Director



7/1/08

Robert N. Hill

Date

ABSTRACT

Reactor simulation involves the solution of a variety of physics types, including neutronics, thermal/hydraulics, and structural mechanics. Codes are being written which solve several of these physics simultaneously. The SHARP project at Argonne National Lab is developing a parallel, component-based reactor simulation framework. The framework is anchored by the representation of the spatial domain or the mesh. This report describes the formulation and implementation of a parallel solution coupling capability being added to the SHARP framework. 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.

CONTENTS

1. Introduction	1
2. The Solution Coupling Problem	2
2.1 Initialization	3
2.2 Point Location	3
2.3 Interpolation	4
2.4 Normalization	5
3. Implementation in MOAB	6
3.1 Coupler API	6
3.2 Interpolation Method	6
3.3 Conservation & Normalization	7
4. Coupling Results	7
4.1 Brick Example	7
4.2 91-pin ABTR Reflector Assembly	7
5. Run-Time Performance	11
6. Summary & Future Work	12

Figures

Figure 1.1.: The SHARP framework.....	2
Figure 2.1: Algorithm for point location phase of solution coupling.....	4
Figure 2.2: Algorithm for interpolation phase of solution coupling.....	5
Figure 4.1.: 64-brick geometry used in first mapping example. Source mesh with 1700 hexahedra (left), target mesh with 12k tetrahedra (right).....	8
Figure 4.2: Field on source mesh (left) mapped to target mesh (right) for 64-brick example.....	9
Figure 4.3.: Source (left) and target (right) meshes for 91-pin reflector assembly example.....	9
Figure 4.4.: Field on source mesh (left) mapped onto target mesh (right) for reflector assembly example.	10
Figure 4.5.: Close-up of source field (left) mapped to target mesh (right) for reflector assembly example.	10
Figure 5.1: Total run-time for reflector assembly example, including initialization of mesh and coupler.	11
Figure 5.2: Run-time for reflector assembly example, not including initialization time.....	12

REACTOR CAMPAIGN SCALABLE PARALLEL SOLUTION COUPLING FOR MULTI-PHYSICS REACTOR SIMULATION

1. Introduction

Reactor simulation involves the solution of a variety of physics types, including neutronics, thermal/hydraulics, and structural mechanics. Codes are being written to carry out several of these physics simulations simultaneously. In most cases, these codes are component-based, with each type of physics solved in a distinct code component, and with components interacting through data stored on the physical domain representation or mesh. For generality, we assume that each physics type is solved on its own mesh distributed on the parallel computer in a way that is optimal for that physics type. To couple various physics calculations, the results from one physics type must be mapped to the mesh of another physics type and then normalized according to application-defined conservation requirements. This report describes an algorithm and its implementation to perform this solution coupling among multi-physics components.

The SHARP project at Argonne National Lab is developing a parallel, component-based reactor simulation framework [1]. A schematic architecture of this code framework is shown in Figure 1.1. The framework is anchored by the representation of the spatial domain, or the mesh. The MOAB library implements the mesh representation, with the ITAPS iMesh interface, which is used to access mesh from most components. MOAB also provides various parallel data services, for example parallel IO and parallel data communication. Other infrastructure services are connected to MOAB through iMesh, including parallel partitioning (using Zoltan [2]), visualization (using Visit [3]), and mesh generation (using CUBIT [4]). Together, the mesh representation and the associated infrastructure services forms the “framework”. This framework is the foundation on which we are building the SHARP reactor simulation code. Components are being developed to solve various physics modules relevant to this problem. Current efforts are focused on the thermal/hydraulics (Nek5000 [5]) and neutronics (UNIC [6]).

Common functional interfaces or APIs have been developed to access mesh, geometry, and other data relevant to the simulation process, by the ITAPS project [7]. These interfaces abstract the details of data representation from the applications, which use them. The ITAPS mesh interface, iMesh, defines a simple data model consisting of:

- entities (topological entities in the finite element zoo, e.g. vertices, triangles, hexahedra)
- entity sets (arbitrary collections of entities and other sets)
- tags (annotations of data to the other data types)
- the mesh interface (the object through which the other mesh data is accessed)

Although quite simple, this data model is capable of holding a wide variety of semantic data relevant to parallel simulations, including parallel partitions, geometric model topology used to generate a mesh, and boundary condition groupings. This data model is also instrumental in storing the data important to the solution coupling approach described in this report.

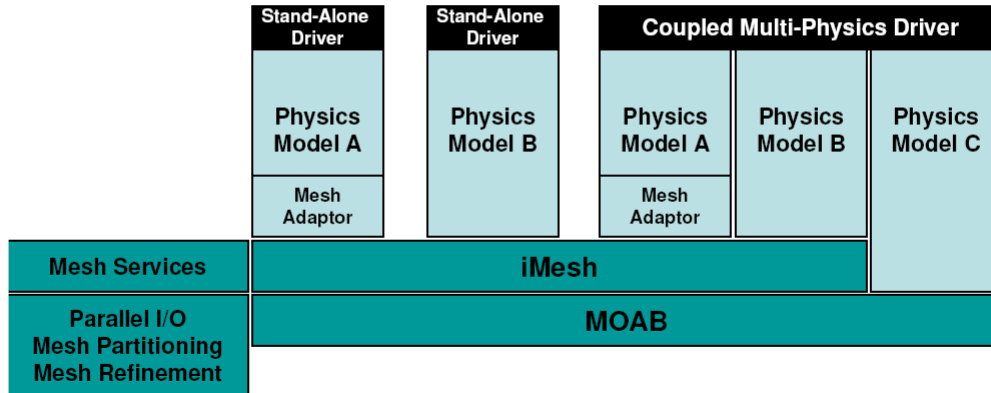


Figure 1.1.: The SHARP framework.

To facilitate solution mapping, we use a spatial decomposition of mesh data called a kd-tree. The kd-tree is a binary tree, with each node linked to at most one parent node and zero or two child nodes. The tree has a single *root node* (with no parent nodes), and one or more *leaf nodes* (with no child nodes). Each leaf node contains a small number of entities from the local mesh; all other nodes do not contain entities. An axis-aligned bounding box is defined for each node which defines the maximum spatial extent of any entities contained in leaf nodes below the node. Thus, the bounding box for the root node defines the spatial extent of the mesh for that tree. Given a spatial location, the kdtree code traverses the tree, and for each leaf node containing the point, evaluates the natural coordinates of the point in each of the elements in that leaf node. An element contains a point if the natural coordinates are all between the lower and upper bounds (usually -1 and 1, respectively).

The arrangement of mesh data on a parallel machine is as follows. On a given processor, the *local mesh* is the data stored local to that processor, for the source or target mesh. MOAB initializes the parallel mesh such that each processor knows about mesh vertices, edges and faces shared with one or more processors, and the handles of those entities on any sharing processor. MOAB also supports *ghost entities*, or local copies of entities owned by neighboring processors. For the purposes of solution coupling, a given processor is responsible for coupling only on its locally owned portion of the source mesh.

This report begins by describing the coupling problem, starting with an abstract definition and working toward a concrete statement of the problem. Algorithms used to perform solution coupling between a pair of meshes are then described, along with the implementation of those algorithms. Performance of this implementation in parallel is analyzed next. Finally, this report concludes with future directions of this effort.

2. The Solution Coupling Problem

The solution coupling problem requires mapping data from a *source* mesh onto locations determined on a *target* mesh. The data on the target mesh is used as a boundary condition or in determining material properties used in the solution of another type of physics. The results of this other physics solution are mapped to another mesh and physics type, continuing the sequence until the solution maps back to the original mesh. This process is repeated either for a series of time steps, or to iterate to a steady-state coupled solution.

We make several important assumptions about the data being coupled:

1. Each physics type is solved on its own mesh, whose characteristics (element type, refinement, distribution on the parallel computer, etc.) are chosen to optimize solution of that physics. We make no attempt to restrict the mesh or distribution of that mesh for one physics type in order to simplify the solution of other physics types or the coupling of results between those meshes.
2. Each physics type and mesh is distributed across a set of processors, defined by an MPI communicator for each mesh. This communicator can include all or only part of the parallel machine. Communicators for both meshes being coupled are available to the coupler.
3. On a given processor, all meshes are stored in a single iMesh instance, and that instance communicates with all other processors containing pieces of any of those meshes.

These assumptions reflect the basic philosophy of the SHARP project and code, which tries to avoid any unnecessary constraints on the various physics components in any coupled simulation.

The solution coupling problem can be broken down into four distinct phases:

1. **Initialization:** Read the mesh in parallel (if necessary), and initialize searching structures on the mesh
2. **Point Location:** For each interpolation point, derived from the target mesh, find the element(s) and natural coordinates of the point in those elements in the source mesh
3. **Interpolation:** Interpolation of data from the source mesh onto points from the target mesh
4. **Normalization:** Normalization of the mapped data, on the target mesh, to satisfy specific properties defined by the application

For problems not involving mesh movement, steps 1 and 2 need only be done once, with the results re-used for all time steps or iterations. In other cases, the point location step must be repeated to account for mesh movement. For the purposes of this report, we restrict our consideration to the static mesh case.

We describe each of these steps individually in the following sections.

2.1 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 MBParallelComm class object, which is passed to the coupler at instantiation.

To facilitate point location on a source mesh, a kd-tree decomposition is computed for the local part of the target 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.

2.2 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 natural coordinates in those element(s) that

1. Choose target points t_i
2. $\forall t_i$:
 - (a) \forall root box B_j containing t_i :
 - i. If $j \neq p$ 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 kd-tree 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 $k \neq -1$, add tuple $Tu4(j, i, k, f)[l]$, where f is a double-precision parameter used in the interpolation phase
7. Repeat steps 4a-c for all tuples in Tu_tmp ; in step 4c, put contained points in $Tu5(i, k)$ instead of $Tu3$

Figure 2.1: Algorithm for point location phase of solution coupling.

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. Some important notes about this process are:

- Target points are derived from the target mesh, but need not be restricted to vertices in that mesh. Other examples include elements for which integrated power generation is required, or entity sets for which isotope depletion is required. These capabilities will be added over time, with vertex-based points implemented first.
- This algorithm is designed to minimize data communication. In particular, details of the point location (i.e. element and natural coordinates for each point) are kept on the processor owning the source element, rather than the target point. This incurs a small extra storage cost (one tuple of three integers per target point) but reduces communication during the interpolation phase.
- By default, the entire set of target points, and their point location results, are cached in the coupler. An additional option is provided to pass that information back to the application.
- 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, and would increase the complexity of communication code in the application.

After this phase is finished, a tuple list containing the results is stored in the coupler, indexed in the order target points were input. This tuple list also serves as the communication mechanism for the interpolation step, described next.

2.3 Interpolation

During the execution of a physics component, field values are computed and stored on various entities in the local 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 2.2.

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
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
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 2.2: Algorithm for interpolation phase of solution coupling.

Some important notes about this process are:

- The interpolation results are stored in the same tuple, $Tu4$, computed during the point location stage. The size of each field value therefore needs to be known before any interpolation is done. Currently, this field is assumed to be a single double precision number. In the future, this will be expanded to allow applications to request different size fields to be interpolated.
- As described, this algorithm is silent on what type of shape functions are used to interpolate the field. The only requirement 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.
- Various overloaded definitions of the functions in the coupler are defined to either pass back the interpolated field to the application in an argument, or store the field in a tag specified by the application.

2.4 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.

A discussion of the various normalization methods used in solution coupling is beyond the scope of this report. The implementation described in this document allows for a normalization phase; this initial capability will focus on normalizing to a quantity integrated over the entire mesh. If time allows, it will also be possible to normalize integrated quantities over specified subsets of the mesh, provided those subsets exist in both the source and target meshes (although the contents of those subsets need not be, and generally will not be, the same).

3. Implementation in MOAB

Solution coupling is implemented in the `MBCoupler` class in the MOAB code. Functions for accessing this capability are described next. These functions closely mirror Steps 1-4 described in Section 2. Following the function description, various other implementation issues are discussed.

3.1 Coupler API

- `MBCoupler(MBInterface *impl, MBParallelComm *pc, MBRange &local_elems, int coupler_id, bool init_tree = true);`

Constructor for the coupler; input arguments are the MOAB instance, the parallel communicator object for the source mesh, the target entities to be interpolated onto, the coupler id for this instance, and a flag telling whether to initialize the kd-tree for the source mesh.

If tree initialization is requested, the tree is constructed and its root node stored on the instance. The min and max bounding box corners for the tree on each processor are gathered to all processors, such that each processor holds the box extents for all other processors.

- `MBCoupler::locate_points(double *xyz, int num_points, tuple_list *tl = NULL, bool store_local = true);`

The application inputs spatial points to serve as the target points. These points are located in the source mesh, and (by default) information on that location is stored in the coupler instance. This information is indexed by the order in which the points appear in the input data. This step involves several gather/scatter communication operations.

- `MBCoupler::locate_points(MBRange &ents, tuple_list *tl = NULL, bool store_local = true);`

This form of the `locate_points` function passes in a range of target mesh entities. If these entities are mesh vertices, the target point locations are determined from those vertices; if other type of entities, entity centroids are chosen. Point location information is indexed by the order in which the points appear in the input range. Passing in target locations in this form also results in the interpolated quantities being assigned to the target entities as tags.

- `MBCoupler::interpolate(MBCoupler::Method method, MBTag tag, double *interp_vals, tuple_list *tl = NULL, bool normalize = true);`

In this function, tag values stored on the source mesh in the indicated tag are interpolated onto the target points, using the input interpolation method. Interpolated values are passed back to the caller.

- `MBCoupler::interpolate(MBCoupler::Method method, std::string &tag_name, double *interp_vals, tuple_list *tl = NULL, bool normalize = true);`

In this overloaded version of the interpolation function, the tag is specified by name instead of tag handle; otherwise the behavior of this function is the same as the other version.

3.2 Interpolation Method

One of the input parameters in the `interpolate` function is the interpolation method. Two methods have been implemented thus far in `MBCoupler`:

- **LINEAR_FE:** In this method, a vertex-based field defined on the source mesh is mapped to target points. This mapping is done by locating the source element containing each target point, and computing its natural coordinates according to linear iso-parametric finite element shape functions. These coordinates are used to interpolate a field defined at source vertex locations onto the target point. This interpolation method is used frequently in finite element analysis.
- **PLAIN_FE:** This method maps element-based variables in the source mesh to target points in the target mesh. This mapping is done by simply finding the source element in which the target point is located, and assigning the variable from that source element to the target point.

Since the interpolation method is an input parameter in the coupler, it will be possible to add more methods without changing the basic structure of the code. This will be necessary, for example, to interpolate using higher-order basis functions.

3.3 Conservation & Normalization

Which quantities are conserved when mapping solutions between meshes is influenced by both the interpolation and the normalization methods. In principal, the mapping method should use the same shape functions used to generate the solution on the source mesh. Normalization can be done based on the solution itself, or based on moments of the solution. Either of these quantities can be normalized over the entire mesh, or for subsets in the source and target meshes.

It is beyond the scope of this report to discuss the shape functions used in various FE computations, or the normalization methods used to conserve various quantities in a mapped solution [8]. Currently the only normalization method implemented is over the entire mesh, and only for element-based variables. The implementation described here does, however, allow the addition of more normalization methods, and specification of the method by the application. This functionality will be expanded in the future as various types of physics are coupled using the coupler.

4. Coupling Results

Due to time constraints, coupling was tested on data residing on typical mesh types, but which was generated using a known function. For the purposes of coupling, this data is sufficient to show that coupling is working as designed. Two specific examples are discussed, one with a brick-like structure, and one more typical of reactor geometries.

4.1 Brick Example

A sample problem with 64 bodies was used to generate a source and target mesh, shown in Figure 4.1. This mesh was decomposed for parallel solution mapping using the geometric model volumes (each shown in a separate color in the figure). The field assigned to vertices in the source mesh, and its mapping to the target mesh, are shown in Figure 4.2. Note that this shows a “pseudocolor” plot, which interpolates the vertex values to show a smoothly-varying field. A linear finite element shape function, identical to that used for the solution mapping, is used in the visualization tool.

4.2 91-pin ABTR Reflector Assembly

A 91-pin reflector assembly from the ABTR was used to test coupling on realistic geometry. The geometry and source/target meshes for this problem are shown in Figure 4.3. This model contains 92 volumes (the 91 pin homogenized cells and the duct). The hexahedral mesh (left) is meshed with

approximately 13k hexahedral elements; the tetrahedral mesh (right) contains approximately 130k elements.

The field on the source mesh is shown in Figure 4.4 (left). This field mapped to the target mesh is shown in Figure 4.4 (right). The same field and mapped field are shown on a smaller portion of the domain in Figure 4.5. Note that for the target mesh, only the partition solved by one processor (out of a 2-processor partitioned mesh) is shown. Qualitatively, the mapped solution is close to the mapped field. In the future, we will obviously need the means to compare solution fields quantitatively; this is discussed in Section 6 of this report.

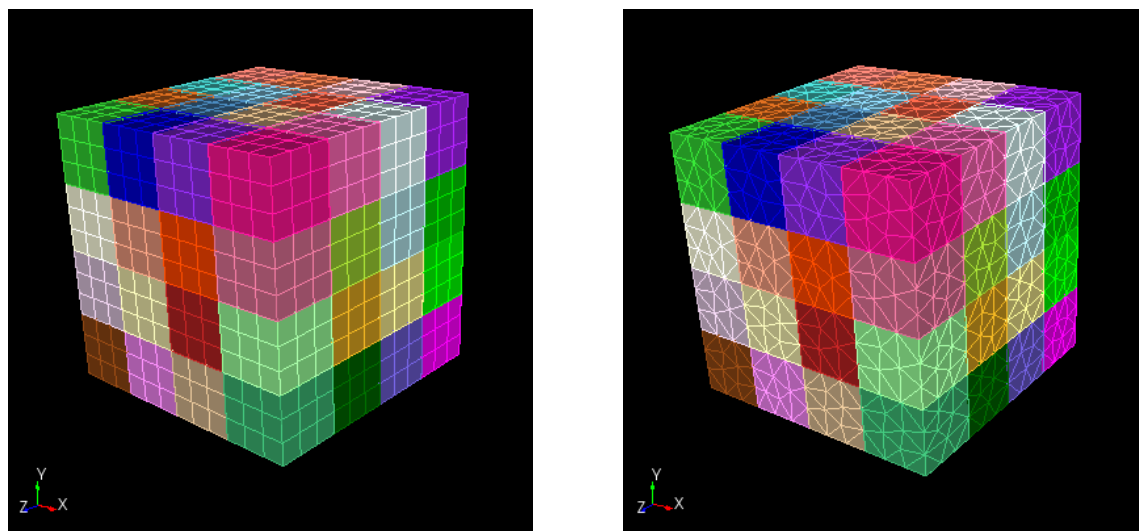


Figure 4.1.: 64-brick geometry used in first mapping example. Source mesh with 1700 hexahedra (left), target mesh with 12k tetrahedra (right).

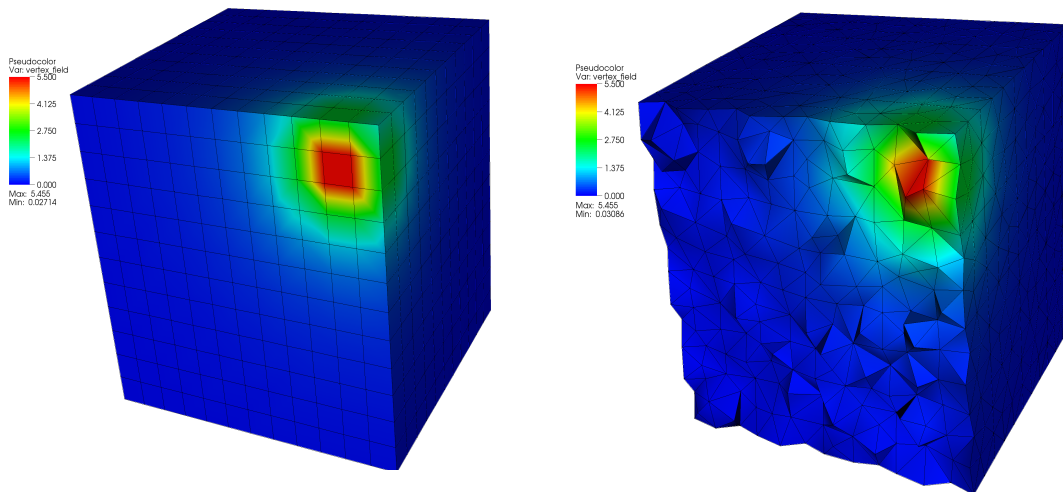


Figure 4.2: Field on source mesh (left) mapped to target mesh (right) for 64-brick example.

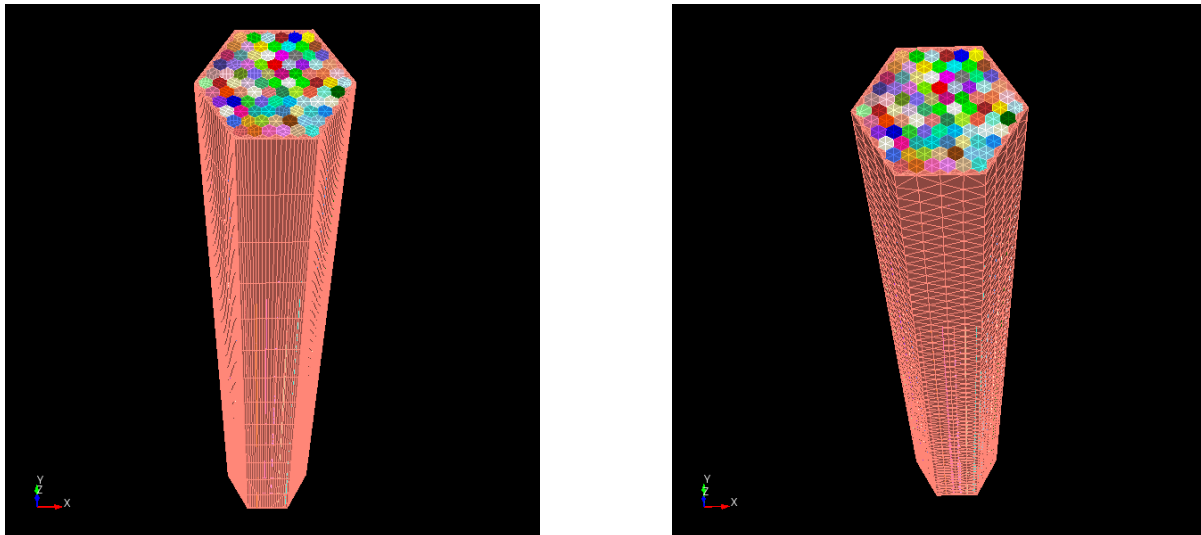


Figure 4.3.: Source (left) and target (right) meshes for 91-pin reflector assembly example.

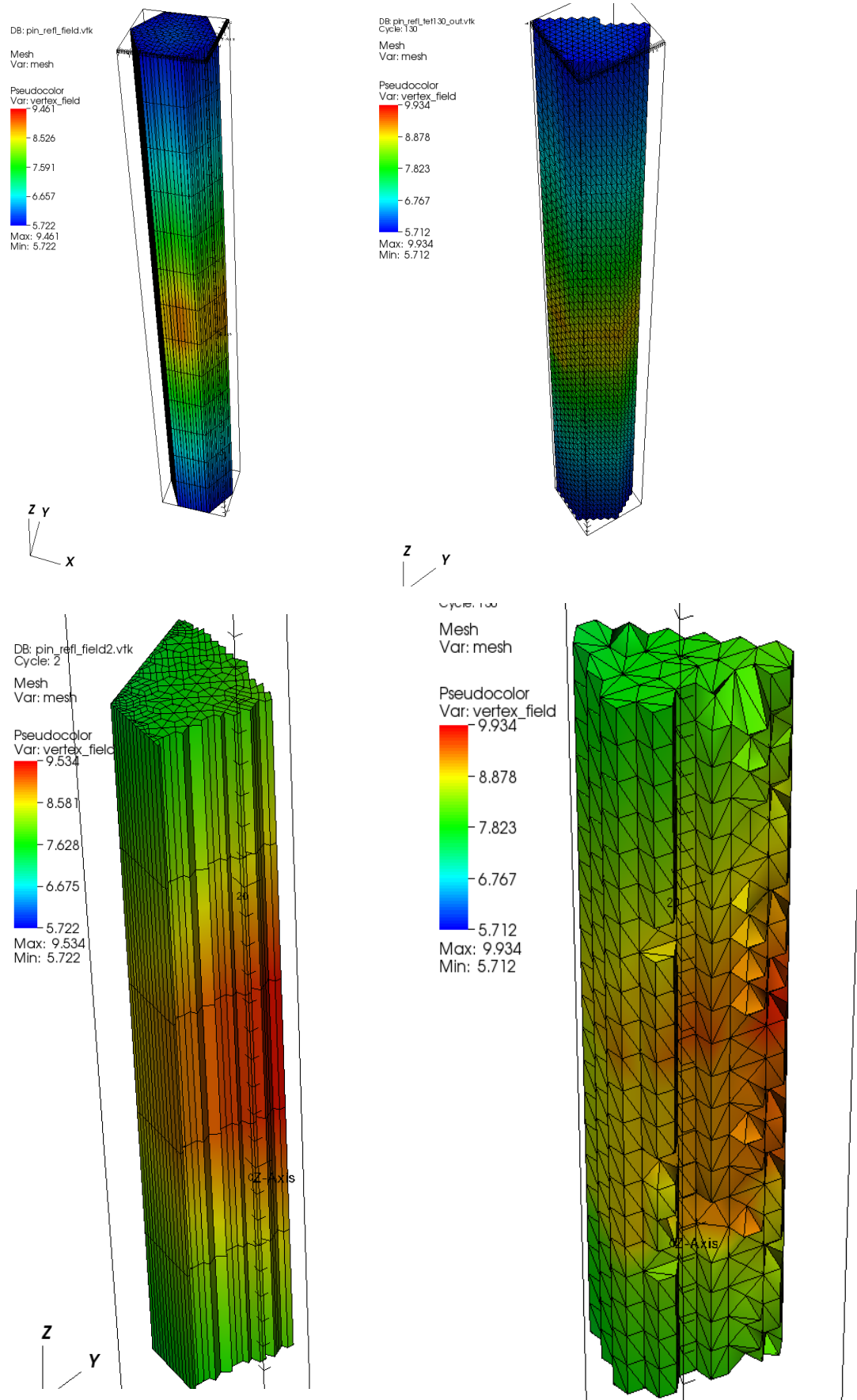


Figure 4.5.: Close-up of source field (left) mapped to target mesh (right) for reflector assembly example.

5. Run-Time Performance

The solution coupling process consists of four phases, described in Section 2: initialization, point location, interpolation, and normalization. The times for mesh import, coupler instantiation, point location, and interpolation are shown together in Figure 5.1; normalization time was not measured. Run-times for the point location and interpolation phases, without initialization times, are shown in Figure 5.2. Clearly, the initialization of the problem dominates the other phases in this problem. In particular, the interpolation phase accounts for a maximum of 6%, 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, we conclude that the cost of coupling will not be significant to the overall execution time.

Several other conclusions can be drawn from this data. First, the mesh import time is not being reduced

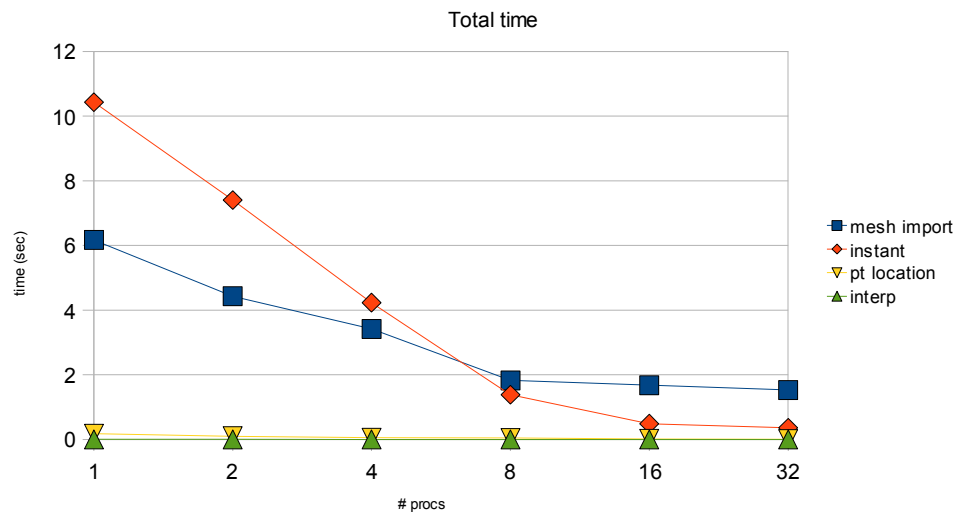


Figure 5.1: Total run-time for reflector assembly example, including initialization of mesh and coupler.

as quickly as other times with increasing processor counts. Although not an unreasonable run time now, this time will become problematic as larger problems are run on greater numbers of processors. The method currently used to initialize the mesh on the parallel machine is to read the entire mesh on each processor, then delete the parts of the mesh not assigned to that processor. To be truly scalable, the capability must be developed to read only those portions of mesh assigned to a given processor. That capability is already under development.

Second, the point location algorithm described in Section 2.1 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 modify this step such that the boxes containing mesh on each processor are arranged in another tree, such that that secondary tree can be traversed similarly to the primary tree. In this way, computing the box(es) containing a given point will scale as $O(\log(p))$ rather than as $O(p)$. 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 an aggregation of results over all processors.

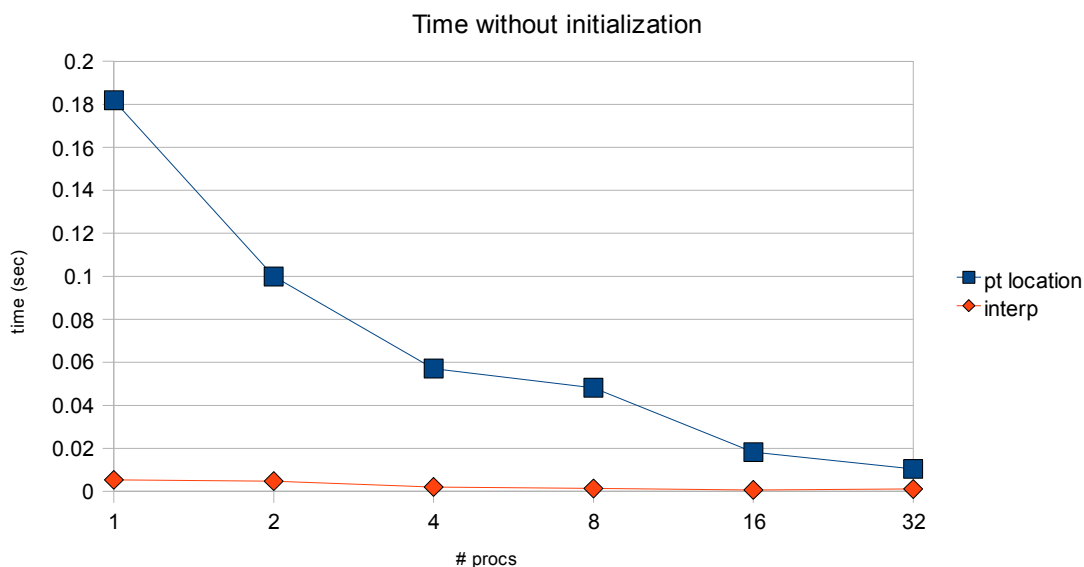


Figure 5.2: Run-time for reflector assembly example, not including initialization time.

6. Summary & Future Work

Reactor simulation involves the solution of a variety of physics types, including neutronics, thermal/hydraulics, and structural mechanics. The SHARP project at Argonne National Lab is developing a parallel, component-based reactor simulation framework.

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. Point location computations are done partly on the processor containing each source point, to choose the processor(s) whose source mesh may contain the target point, and that source processor, to compute the actual location. Actual point location data are stored with the source mesh, with indices into that storage returned to the processor containing each target point. This minimizes communication at a small cost in memory. Interpolation is done using either a linear finite element basis function (for vertex-centered fields) or simple assignment based on source element (for element-based fields). Normalization is currently done over the entire mesh, but only for element-based fields.

Coupling results show good qualitative agreement with the field being coupled. However, the means to compare solutions on disparate meshes quantitatively are needed. This capability may be able to be implemented based on the Intrepid field interface library being developed at Sandia National Laboratories [9]. Collaborations are already underway with that project to explore this area.

The run-time for this coupling tool is dominated by the time to read the mesh and initialize the spatial searching structures in the coupler. The performance of the mesh and tree initialization improved with larger processor counts for the examples described in this report; this was due to the mesh size per processor decreasing. This time would probably remain nearly constant for cases where the mesh size per processor were kept the same as processors were added (weak scaling). This represents a one-time cost for problems with static mesh and it can be amortized over the entire coupled calculation. The

interpolation phase of the coupling problem requires a maximum of 6% of the overall execution time, and in absolute terms was quite low (well below one second). Thus, it appears that the run-time for this coupling capability will be a small fraction of the overall execution time for typical reactor simulations. More effort will be made to compare runtime against the actual simulations once a fully coupled simulation capability is available.

The implementation of this capability in MOAB will be extended in various ways in the near future. More interpolation methods will be added as need arises, based on the physics components being coupled. There will be an immediate need for interpolating higher-order basis functions, and part of this implementation has already been done. Normalization over specific subsets in the mesh is also needed, to focus on specific material, depletion, and other 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. This work is sponsored by the DOE AFCI program.

References

1. A. Siegel and T. Tautges and A. Caceres and D. Kaushik and P. Fischer, "Software Design of SHARP", Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007), Monterey, California, April 15-19, 2007.
2. Karen Devine et. al, "Zoltan Data Management Services for Parallel Dynamic Applications, Computing in Science and Engineering, 4:2, pp 90-97 (2002).
3. K. Joy et. al, "Frameworks for Visualization at the Extreme Scale", Journal of Physics Conference Series, SciDAC 2007, (Editor, David Keyes, et. al.), IOP Publishing, Volume 78, 2007.
4. T. D. Blacker et al., 'CUBIT mesh generation environment, Vol. 1: User's manual', SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994, http://endo.sandia.gov/cubit/release/doc-public/Cubit_UG-4.0.pdf
5. P. Fischer, "Large Eddy Simulation of Wire Wrapped Fuel Pins I: Hydrodynamics in a Periodic Array", Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007), Monterey, California, April 15-19, 2007
6. G. Palmiotti et. al, "UNIC: Ultimate Neutronic Investigation Code", Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007), Monterey, California, April 15-19, 2007.

7. L.A. Diachin, A.C. Bauer, B. Fix, J. Kraftcheck, K. Jansen, X. Luo, M. Miller, C. Ollivier-Gooch, M.S. Shephard, T. Tautges, H. Trease, Interoperable Tools for Advanced Petascale Simulations (ITAPS), Journal of Physics Conference Series, SciDAC 2007, (Editor, David Keyes, et. al.), IOP Publishing, Volume 78, 2007.
8. Pavel Bochev, Mikhail Shashkov, Constrained interpolation (remap) of divergence-free fields, [Computer Methods in Applied Mechanics and Engineering, Volume 194, Issues 2-5](#), 4 February 2005, Pages 511-530 .
9. P. Bochev and M. Hyman, Principles of compatible discretizations, In: Compatible discretizations, Proceedings of IMA Hot Topics workshop on Compatible discretizations, IMA 142, Springer Verlag, 2006, pp.89–120.