Identification Number: DE-FC02-06ER25769

Project Title: Interoperable Technologies for Advanced Petascale Simulations (ITAPS)

Institutional Faculty PI's: Mark S. Shephard and Kenneth E. Jansen

Institution: Rensselaer Polytechnic Institute

Date of Report: February 5, 2010

Report Period Covered: December 2008-December 2009

1. Technical Progress for Current Year

Efforts during the past year have contributed to the continued development of the ITAPS interfaces and services as well as specific efforts to support ITAPS applications.

The ITAPS interface efforts have two components. The first is working with the ITAPS team on improving the ITAPS software infrastructure and level of compliance of our implementations of ITAPS interfaces (iMesh, iMeshP, iRel and iGeom). The second is being involved with the discussions on the design of the iField fields interface.

Efforts to move the ITAPS technologies to petascale computers has identified a number of key technical developments that are required to effectively execute the ITAPS interfaces and services. Research to address these parallel method developments has been a major emphasis of the RPI's team efforts over the past year.

The development of parallel unstructured mesh methods has considered the need to scale unstructured mesh solves to massively parallel computers. These efforts, summarized in section 2.1 show that with the addition of the ITAPS procedures described in sections 2.2 and 2.3 we are able to obtain excellent strong scaling with our unstructured mesh CFD code on up to 294,912 cores of IBM Blue Gene/P which is the highest core count machine available. The ITAPS developments that have contributed to the scaling and performance of PHASTA include an iterative migration algorithm to improve the combined region and vertex balance of the mesh partition, which increases scalability, and mesh data reordering, which improves computational performance. The other developments are associated with the further development of the ITAPS parallel unstructured mesh adaptation procedures. Specific developments include:

- Parallel boundary layer mesh adaptation integrated with parallel anisotropic mesh adaptation (section 2.4.1).
- A new more scalable message packing library (section 2.4.2).
- Support of periodic boundary conditions (section 2.4.3).

We have continued to work closely with both the accelerator applications for COMPASS and fusion application for CEMM. For COMPASS, efforts have focused on providing specific unstructured mesh adaptation tools to deal with curved elements and mesh adaptation. For CEEM, we are working to provide the structures and methods needed for the M3D-C1 to go to full three dimensional configurations.

2. Parallel Methods for Unstructured Meshes Developments

2.1 PHASTA CFD Code Scaling to Petascale Machines

As indicated in the previous progress report, one aspect of getting unstructured meshing techniques to the petascale is demonstrating the ability of PDE discretization and solution systems to scale well on massively parallel computers. This year we continued our work on scaling the PHASTA open source CFD code to near petascale machines. PHASTA (Parallel Hierarchic Adaptive Stabilized Transient Analysis) is an implicit stabilized finite element code that solves its matrix systems using appropriate pre-conditioned iterative solvers on partitioned meshes. PHASTA employs an implicit time integration method that requires the solution of the full set of equations in each step. The linear algebra solver used is based on GMRES, which needs only matrix vector products and can operate on the partitioned mesh. With this combination of methods, the interprocessor communications needed are of two types:

- Organized, substantial, and regular communication between partitions that touch each other in terms of elements on multiple processors that share common boundary mesh entities.
- For each iteration within the linear solve (typically O(10) iterations per time step), there is a required ALL-REDUCE communication.

The combination of stabilized finite elements and iterative solvers in PHASTA allows it to be used with adaptively defined anisotropic unstructured meshes such as shown in Figure 1 that includes anisotropic prismatic boundary layers and anisotropic tet elements [11,12,13, 15,17].

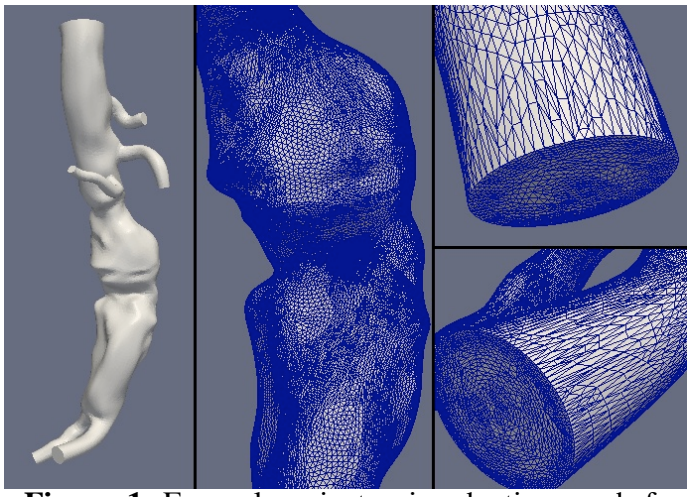

Figure 1. Example anisotropic adaptive mesh for an abdominal aorta aneurysm.

Last year we reported scaling results on 1024 to 16,384 processors of an IBM Blue Gene/L. This year we have continued to extend the ability of PHASTA to scale on even larger core count machines and, with the help of methods described in the next section, have extended the results to machines with more than an order of magnitude more compute cores. Strong scaling results to all 163,840 cores of the Argonne Intrepid IBM Blue Gene/P were reported in reference [14]. This paper was a 2009 Gordon Bell Finalist. More recent results, yet to be published, have shown excellent strong scaling to 294,912 cores of IBM Blue Gene/P and 98,304 cores of Cray XT5 (see Table 1) [12,14,17,18,20].

1.07B elements mesh		Intrepid:IBM BG/P		Kraken:Cray XT5		JuGene:IBM BG/P	
num. of cores	avg. elem./core	time	s-factor	time	s-factor	time	s-factor
4,096 (base)	261,600	844.38	1	311.34	1	845.68	1
8,192	130,800	427.33	0.99	144.23	1.08	–	–
16,384	65,400	217.05	0.97	73.06	1.07	–	–
32,768	32,700	109.87	0.96	39.35	0.97	–	–
65,536	16,350	58.65	0.91	28.04	0.69	–	–
98,304	10,900	39.06	0.90	18.67	0.70	–	–
131,072	8,175	29.68	0.89	–	–	–	–
163,840	6,540	24.12	0.88	–	–	–	–
294,912	3,630	–	–	–	–	14.39	0.82

Table 1. PHASTA scaling results on near petaflop machines.

2.2 Algorithms to Improve Mesh Partitions for Petascale Machines

As the initial PHASTA scaling studies moved to larger core counts some loss of strong scaling was observed. An examination of the two key steps of elements matrix formulations and global systems solve indicated that second step of system solve was the primary source of the loss of scalability. The key source for this problems relates to the fact that the workload associated with that step is a function of the number of vertices in the mesh and that even though the graph partitioners used maintained excellent balance of the number of elements on a part, the imbalance of the number of vertices per part for many of those cases was often 20% or more, leading to an imbalance of the work load. Since the overall partition provided by the graph partitioners is generally a good overall partition, a more local algorithm that employs additional adjacency information that is available through the ITAPS iMesh and iMeshP interfaces has been developed to perform local modifications of the partitions to reduce vertex imbalance while minimizing increases in element imbalance.

The partition improvement procedure referred to as Local Iterative Inter-Part Boundary Modification (LIIPBMod) [20] locally migrates small numbers of mesh elements from parts with relatively more vertices to neighboring parts that are relatively lightly loaded. On the heavily loaded part, the mesh vertices on the part boundary are traversed and the ones bounding a small number of elements are identified. If the neighboring part is lightly loaded, the whole “cavity” (all the adjacent elements to a selected vertex) is migrated to the neighboring part. Figure 2 demonstrates the basic application of this process for selected vertices. The top row shows mesh vertices selected to be migrated and the local mesh before migration while the bottom row shows the mesh after the connected elements have been migrated.

The iterative application of this operation has been found to be effective for reducing the vertex imbalance while having only a small influence on increasing element imbalance. It is also found that this algorithm improved the partition boundary communication requirements by decreasing the ratio of vertices on a part boundary to the total number of vertices on the part. Another advantage of the procedure is it typically is much faster that a general full repartitioning of the mesh. Table 2 compares the element and vertex imbalance ratios for a 42 million element partitioned mesh before and after using the LIIPBMod algorithm along with the time usage of the ParMETIS partitioner and

LIIPBMod algorithm. The number of iterations LIIPBMod needed to improve the vertex imbalance for each partition is included in the table as well.

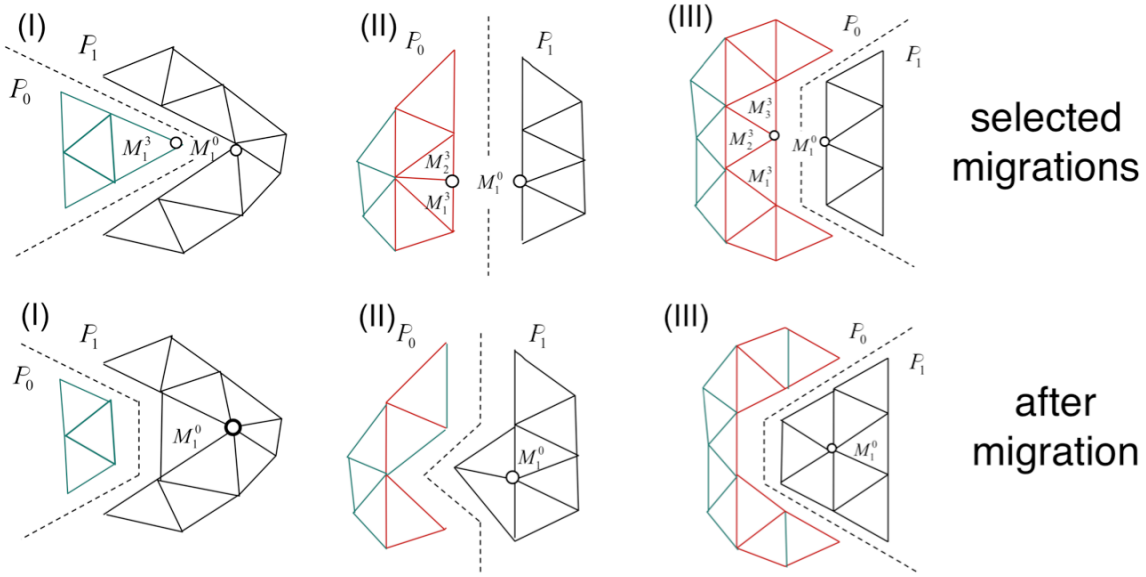

Figure 2. Migration of elements to move vertices from left to right.

42M element mesh	elem. imb.		vtx. imb.		time usage (sec.)		num. iter.
num. of parts	ParMETIS	LIIPBMod	ParMETIS	LIIPBMod	ParMETIS	LIIPBMod	
512	1.0226	-	1.0496	-	21.0	-	-
1,024	1.0240	1.0245	1.0516	1.0488	24.6	0.6	1
2,048	1.0208	1.0223	1.0527	1.0496	26.3	0.3	1
4,096	1.0218	1.0335	1.0904	1.0493	26.1	0.7	2
8,192	1.0186	1.0336	1.0840	1.0487	26.6	1.2	3
16,384	1.0264	1.0454	1.1446	1.0492	28.8	2.2	3
32,768	1.0241	1.0653	1.1642	1.0578	37.4	18.5	9

Table 2. Comparison of element and vertex imbalance ratios before and after using LIIPBMod algorithm on a 42M element mesh along with the time usage (including mesh migration) of the ParMETIS PartKWay partitioner and LIIPBMod algorithm.

In the second example [20], LIIPBMod is applied on the partitions produced by Zoltan PHG partition on the abdominal aorta aneurysm geometry shown in Figure 1 with a 1.07 billion element mesh. The same parameters are used as in the previous test (i.e., tol vtx = 1.05 and maxiter = 10). The tests are performed on Kraken at NICS, using 2048 processing cores (4096 cores are used for the case of "4k to 160k"). Table 3 compares the element and vertex imbalance ratios before and after LIIPBMod. It also contains the time usage of Zoltan PHG and LIIPBMod, again this includes the time usage of mesh entity migration. LIIPBmod reduces the vertex imbalance for all seven cases (from 4,096 to 131,072 parts) as well as the "4k to 160k" partition down to 5% (5th column in Table 5.4) with modest impact on element balance and it reduces the vertex imbalance dramatically for the partitions with a high number of parts (e.g., from 21.17% to 4.99%), which is important for time critical simulations. Time usage of LIIPBMod (7th column of Table 5.4) is small compared to that of hypergraph-based partitioner, and is negligible compared to the finite element analysis. As discussed in reference [20], the partitioned meshes produced after the application of LIIPBMod did improve the scalability of the PHASTA, particularly on very large core count with strong scaling improving from

around 0.8 to approaching 0.9. Although such an improvement may not sound extensive, it does correspond to millions of CPU hours off the full simulation of billion element meshes run of vary large core counts.

$1.07B$ element mesh	elem. imb.		vtx. imb.		time usage (sec.)		num. iter.
num. of parts	PHG	LIIPBMod	PHG	LIIPBMod	PHG	LIIPBMod	LIIPBMod
4,096	1.0549	1.0583	1.0736	1.0499	281.1	4.7	3
8,192	1.0549	1.0616	1.0841	1.0499	307.3	9.1	2
16,384	1.0549	1.0704	1.0990	1.0499	325.6	9.9	4
32,768	1.0549	1.0831	1.1210	1.0499	343.6	11.4	5
65,536	1.0549	1.0972	1.1719	1.0497	363.2	17.6	4
98,304	1.0549	1.1070	1.2117	1.0499	376.8	25.7	4
131,072	1.0549	1.1092	1.1782	1.0497	398.4	37.3	4
4k to 160k	1.0554	1.1182	1.1952	1.0497	138.1	28.4	4

Table 3. Comparison of element and vertex imbalance ratios before and after using LIIPBMod algorithm on a 1.07B element mesh. Time usage is also provided along with the number of iterations used in LIIPBMod algorithm.

Its should be noted that in a recently developed version of the Zoltan partitioning procedures are faster and produce better combined vertex and element balance than the version used in the example of Table 3. We are currently applying LIIPBMod to the meshes resulting from the application of the newest procedures and the initial results indicate that improvements continue to be obtained and that the combination of the newest Zoltan partitioner and LIIPBMod are producing the best results to date. This will be reported on in up-coming documentation.

2.3 Data Reordering to Improve Parallel Performance

The effective use of the processor memory hierarchy is an important issue in computing performance. To improve the use of the processor memory hierarchy, specifically overall cache utilization, for the systems formation and solution steps on unstructured meshes a part level mesh topological traversal algorithm was defined that reordered both mesh vertices and regions [19]. This reordering increases the spatial locality of data and improves overall cache utilization during on processor systems formation and solution.

At the heart of the algorithm is the use of the mesh adjacency information to order the data. See reference 19 for a complete description of the algorithm. Note that the algorithm is easy to implement with iMesh compliant tools since iMesh supports the full range of mesh entity adjacencies. With the partition-based solver being used [13,17,19] it is only on-part data ordering that is of importance to improving cache utilization. In the current application only linear elements are used, thus the only mesh entities that must be ordered are the mesh vertices and regions [19]. Since the algorithm developed is based on the traversal of mesh adjacency information across the mesh vertices, edges, faces and regions [19], it is easy to include the ordering of any of the mesh entities. The inclusion of the ordering of mesh edges and faces would be needed in the case of high order p-version finite elements. The parallel execution of the data ordering is trivial since it is embarrassingly parallel. Note that during the process of forming element matrices and the on-part assembly of the part level system for the linear element cases, the coordinated

ordering of both mesh regions and vertices is important, while in the system solution phase it is only vertex ordering that is important.

The procedure has been tested on adaptively created unstructured meshes [19]. In one example, the effect of the data reordering is studied for different phases of an implicit analysis including element-data blocking, element-level computations, sparse-matrix filling and equation solution. The computations are performed on various supercomputers including IBM Blue Gene (BG/L and BG/P), Cray XT (XT3 and XT5) and Sun Constellation Cluster. It is observed that reordering improves the on core performance by up to 24% on Blue Gene/L and up to 40% on Cray XT5. The CrayPat hardware performance tool was used to measure the number of cache misses across each level of the memory hierarchy. It is determined that the measured decrease in L1, L2 and L3 cache misses when data reordering is used, closely accounts for the observed decrease in the overall execution time [19].

2.4 Improvements to Parallel Mesh Representations and Adaptation

2.4.1 Parallel Boundary Layer Meshing

The boundary layer mesh adaptation process is initiated on a mesh that already carries a pre-defined mesh with layered elements on no-slip walls. Subsequent mesh adaptation steps preserve the layer structure normal to the walls while at the same time attaining desired element sizes in different directions as indicated by the a posteriori mesh size field information.

For the parallel development of boundary layer adaptation, the concept of entity groups is used. An entity group is a group of mesh entities that have to stay together and satisfy certain rules. The inherent structure in the boundary layer mesh allows them being decomposed as a product of a layer surface (2D) and thickness (1D) mesh, as shown in Figure 3. Thus, the stack of consistent 3D elements forming the boundary layer fits the definition of an entity group from the mesh standpoint.

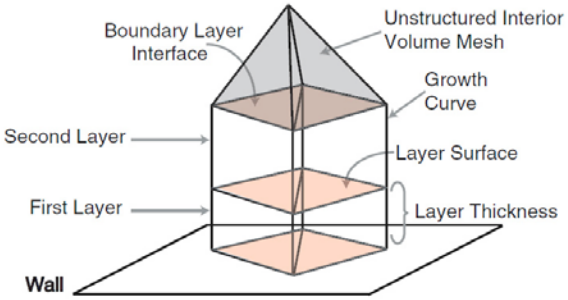

Figure 3. Decomposition of a boundary layer mesh.

Dealing with partitioned mesh, interface elements (interior volume element on top of a boundary layer stack) can be separated from their boundary layers. At the same time, edges shared between the top boundary layer element and interface element are part of the boundary layer. Thus, boundary layer edges must be properly unified across the boundaries to make sure that the communication across the partition boundary is consistent.

To preserve the structure along the normals, mesh adaptation for layered part of the mesh is divided into two steps: surface adaptation and thickness adjustment. The local mesh modification operations of edge split, collapse and swap are utilized to perform the surface mesh adaptation while node movement is applied to adjust the layer thicknesses.

Current development of boundary layer adaptation functionality using entity groups concept includes refinement (serial and parallel), coarsening (serial) and swapping (serial). Figure 4 depicts an example of parallel refinement of the mesh having boundary layers.

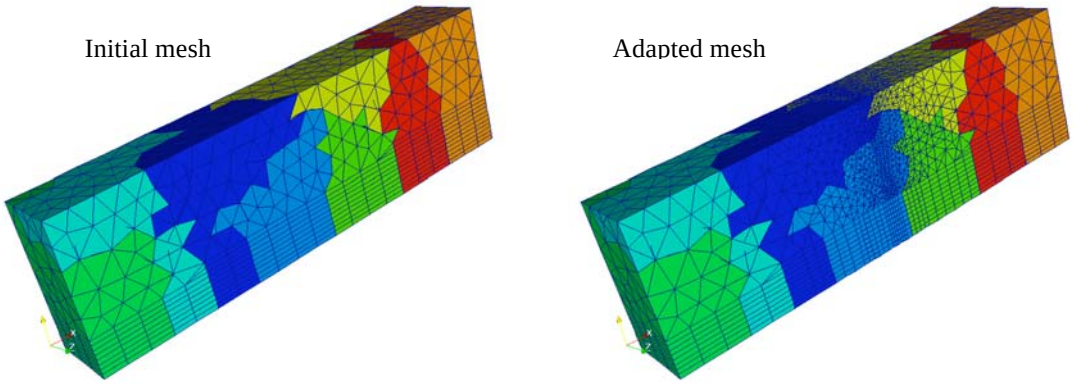

Figure 4. Parallel refinement of a boundary layer mesh with the planar shock size field.

2.4.2 Neighborhood Aware Message Packing Library

The Inter-Processor Communication Manager (IPComMan) [10] is a general-purpose communication package built on top of MPI that aims to reduce data exchange costs by exploiting communications of a local neighborhood for each processor. The neighborhood is the subset of processors exchanging messages with each other during a specific communication round, which in a number of applications is bounded by a constant, typically under 40, independent of the total number of processors. The basic idea of the library is to keep the message-passing within subdomains when possible and greatly reduce the number of collective calls needed.

The library takes care of the message flow with a subset of MPI functions. IPComMan takes advantage of non-blocking message-passing functions, allowing it to post send or receive requests in between computation steps, and automatically manages their completion and delivery. The library provides several useful features i) automatic message packing, ii) management of sends and receives with non-blocking MPI functions, iii) asynchronous behavior unless the other is specified, and iv) support of dynamically changing neighborhood during communication steps.

IPComMan takes care of memory allocation for both sending and receiving buffers, and manages the ones that can be reused without additional allocation. The library stores messages going to the same processor in contiguous memory. Thus, when sending or receiving a package, no additional memory copying is needed.

The user may specify whether the size of each message is constant or arbitrary during a specific communication step. The fixed message size is taken by the library and used while extracting the messages. The arbitrary message size is put together with every message to correctly unpack the message upon arrival.

Each processor has its own neighborhood, which is different from that of its neighbor(s). While initializing a communication library object, each processor specifies the neighbors it is going to communicate with. From that point, the library is concentrated on delivering messages between processor and its neighbors only, not touching other processors of the domain, when possible. There are no collective calls during each communication round.

There could be situations when it is not possible to *a priori* define all the neighbors for processors, i.e., new neighbors may be encountered during a communication step. For example, mesh modification operations may alter the neighborhood of specific processors. Consider the communication pattern presented in Figure 5. Processor P0 has in its neighborhood processors P1 and P2. P3 has P2 as the only neighbor, but after it has sent all the messages to P2 it finds out that there are some messages to be sent to P0. P3 includes P0 in its list of neighbors and begins to send packages to it. An increment of time before, say P0 finished sending to and receiving all the messages from P1 and P2, extracted them and proceeded to the next communication step. In that case packages from P3 to P0 are lost, which will result in incorrect program behavior.

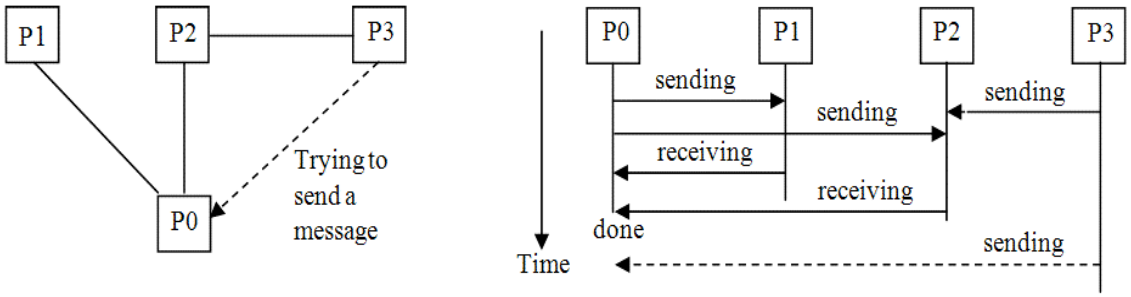

Figure 5. Communication paradigm with the neighborhood being changed.

As new neighbors needing to communicate will be unaware of each other, one collective call at the end of communication round is performed to verify whether the global number of sends and receives match. In case they do not match, the library continues to receive the messages identifying the new neighbors, since all the packages from existent neighbors are already received.

Dynamic and irregular computation often results in an unpredictable number of messages communicated among the processors. Using IPComMan, there is no need to verify and send the number of packages to be received at the end of sending phase. The last message sent from the processor to a neighbor contains the number of buffers expected to be received by the neighbor.

To measure the performance of mesh migration, an essential part of parallel adaptive unstructured mesh procedures, IPComMan was compared with the Autopack communication utility. Initial results on IBM Blue Gene/L show that IPComMan's ability to localize communication to neighborhoods to be independent of the total number of processors and its use of non-blocking functions allows it to continuously reduce the communication time as the number of processors increases. Even though the differences in communication times between Autopack and IPComMan are not substantial at 128 processors, IPComMan is from 3 to 7 times faster in the 4096 processor case.

Although the total communication time for IPComMan is reduced, the scaling for the process does fall with increased processor count. It is important to note though that the

mesh adaptation process that involves mesh migration procedures is not well balanced in terms of the communication load per part. However, additional efforts on the scalability are desired. This includes several aspects of managing sends and receives during the communication phase while interacting with the computation part, and keeping efficient use of the buffer memory. The options are being considered to eliminate collective calls with the neighborhood concept in IPComMan, although additional analysis is required to see if these approaches lead to the reduction of the communication time.

2.4.3 Periodic Boundary Conditions in FMDB

When solving partial differential equations using numerical methods there are many problems of interest where it is possible to reduce the domain to a repeated unit subject to periodic boundary conditions where the solution repeats itself on the periodic boundaries. The most effective means to represent periodic boundary conditions when using mesh based numerical methods is to have identical meshes on the periodic boundaries to be matched since they then support the simple equating of the solution variables associated with the appropriate matched mesh entities.

The definition of a matched mesh on a set of periodic boundaries is one where the mesh entities on the matched domain boundaries are topologically identical and geometrically differ by only the geometric transformation needed to have the matched boundary entities occupy the same place in space.

The specification of periodic boundaries is support by the high-level topological model description used of the problem domain. With this the fully automatic mesh generator used is capable of generating a matched mesh. Given that, we needed to modify the ITAPS parallel mesh adaptation services and underlying mesh representation to maintain information on the matched mesh entities and to ensure that mesh adaptation operations are properly applied to the mesh entities connected to matched mesh entities to maintain a valid mesh that remains matched on the matched boundaries.

The extensions to the FMDB mesh representation included adding information on which mesh entities are matched given the periodic boundary conditions specified on the geometric model level and an initial mesh that was matched. This information is carried within a STL container in the mesh entity structure (matched entity container). Once a matched entity is modified during the process, its matched entity (entities) will be notified and modified accordingly.

Since matched mesh entities are connected by the sharing of common degrees-of-freedom (dof), the procedures that construct the graph used by the Zoltan partitioners needed to be sure the additional graph edges were added for that dof connected mesh entities that are not otherwise (geometrically of based topology) connected.

The parallel mesh adaptation procedures also needed to be modified to ensure that as mesh modifications were applied to mesh entities on a periodic boundary the same operations were applied to the match mesh entities on the matched boundary. Of course, such operations will require mesh modifications be executed on the mesh entities that the matched mesh entity bound. The mesh modification operations on both sides of the periodic boundary must be coordinated to ensure the resulting mesh remains matched and valid.

3. Applications Developments

3.1 Accelerator Modeling with COMPASS

During the current year there were two areas of emphasis in our efforts with COMPASS. The first is the development of an explicit nodal repositioning algorithm to improve the shapes of curved elements. The second is working toward having the full set of curved mesh components operate on parallel computers.

3.1.1 Explicit High-order Nodal Repositioning for Curved Elements

The overall objective is to improve the shape quality of high-order meshes through various local mesh modifications. In order to achieve this goal, the explicit high-order nodal repositioning procedure is under development.

For curved elements we apply Bezier polynomials to represent the geometric shape [7]. The Convex Hull Property of the Bezier polynomial indicated that the determinant of Jacobian of the curved element is bounded by the coefficients computed by the control points of the curved element. This information has been used in our previous work (see [7] and references cited within) to determine the invalid elements and to decide which mesh entities need to be modified to create a valid mesh. The current effort uses the same information to work toward improving the shape of the elements to further improve the conditioning of the resulting system.

A key issue to address in developing such a procedure is to determine an appropriate measure of curved element shape. There is a well established set of equivalent shape measures from straight sided elements, to be referred to as Q_s. However, they have no consideration of the influence of element curving. The common shape measure of element curving, Q_c, is the ratio of the minimum to maximum Jacobian which only measures deviation from a straight-sided element and is equal to the best value of 1.0 for any straight sided element, including one that is nearly invalid. Since each measure individually deals with a component of the element shape, it was found that the product of a normalized version of the two provides a useful single shape measure for curved elements to be referred to as $Q = Q_s * Q_c$.

The input of element shape improvement algorithm is a list of mesh regions (elements) whose shape quality Q is below the desired threshold value Q_{min}. For each region in the list the straight sided, Q_s, and curved shape, Q_c, quality measurements are computed and a set of local mesh modification procedures applied to improve Q to be greater than Q_{min}, if possible. The mesh modification operations, including the explicit nodal positioning procedure that was added, make use of the component shape measures and properties of Bezier polynomials in determining the most appropriate operations to apply.

The explicit nodal positioning procedure repositions either corner or the edge nodes to improve the element by determining the nodes most useful to move and applying a line search to move the selected node to maximize the shape improvement due to moving that node. The computation of Q_c provides the information which mesh entities contribute to the minimum *det(J)*. By increasing the minimum *det(J)*, the curved shape contribution of the element can be effectively improved. The procedure first considers the four region vertices, corner nodes by selecting the one with minimum *det(J)* associated with it.

The approach taken to reposition these nodes is to first determine a good direction of motion and then to move the node in that direction to gain the best improvement. Based on previous work on explicit node positioning, the direction selected is normal to the face opposite the current node. Given that direction, the node is moved in the direction where the shape measure improves until either the shape stops improving, or the shape of some element connected to that node degrades to be equal to the shape of the element being improved. A Golden search is used to converge on that point.

Figure 6 shows a one cavity accelerator test case. The initial curved mesh created by CUBIT had a number of invalid elements with a worst shape of -4.24. A negative number indicates an invalid element caused by a negative Jacobian. The mesh modification procedures alone were able to create all valid elements with a worst element shape of 0.00045. After explicit nodal reposition the worst shape element was improved to 0.0030. Additional investigation is underway to define more complete node positioning procedures.

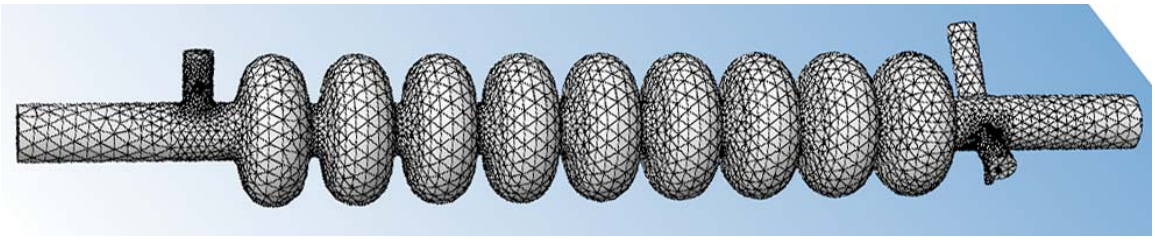

Figure 6. One cavity accelerator test case.

3.1.2 Full Set of Curved Mesh Components Operating on Parallel Computers

Currently the work flow of executing a high-order finite element simulation on complex CAD domains involves a set of steps using a number of tools that operate using a number of conventions. To be able to support the full adaptive simulation process on the parallel computer two developments are needed. They include creation of parallel versions of the curved mesh improvement and curved mesh adaptation tools developed by the RPI ITAPS team and elimination of some data flow inconsistencies in the process of creating the initial mesh and assigning the simulation attributes.

In the current work flow CUBIT is used to create the initial mesh of the ACIS solid model of the domain and is also used to support the specification of the analysis attributes of loads, material properties and boundary conditions. From there the RPI developed ITAPS tools are used to both make the initial meshes valid with improved curved shape elements, and to adapt the meshes. The mesh and attribute information must then be input to the finite element analysis procedure. A problem that arises is that CUBIT creates its own version of the solid model topology and the analysis attributes are associated with that model. However, the curved mesh correction/improvement tool and mesh adaptation interact directly with the ACIS model and its topology. Thus to support the full set of mesh modification operations and properly associate the analysis attribute with the mesh, a mapping between the model topologies is needed. Two approaches are being developed to support that process.

In the first approach the CUBIT developers have provided an extension that for each ACIS model topological model entity that indicates which cubit model entity it represents. In those cases where there is a one-to-one correspondence between entities this allows the easy construction of a map between the entities. We are currently debugging the tool being developed to construct and use this map and the first fully manifold model case has passed the test.

A problem with the first approach is when CUBIT functions are applied that will split model topological entities such that an ACIS model entity may be represented by multiple CUBIT model entities not all information is maintained. Thus the second approach that will be considered will be to use the ANL developed iGeom interface, CGMA, which can interact more directly with the CUBIT model. There are some interaction and mapping complexities involved with this process that will need to be resolved before we can implement this potentially more powerful approach.

3.2 Fusion Modeling with CEMM

The RPI ITAPS team is supporting the CEMM SciDAC activities on the simulation of the Magneto-HydroDynamics (MHD) of plasmas [6]. In addition to supporting new features developed by PPPL in the M3D-C1 code (new type of boundary condition, use of curved mesh elements, …), we are working with CEMM and TOPS on extension of the jointly developed M3D-C1 to the fully 3D case. Key areas of develop include:

- Parallel mesh database management. The mesh topology being supported includes both straight-sided linear and higher order curved mesh elements.
- Anisotropic mesh adaptation based on the computation of the Hessian matrix.
- Parallel Algebraic data container management such as vectors and matrices.
- Linking of PPPL-SCOREC algebraic system to linear algebraic solver packages such as Petsc and SuperLu.

Prior to the ongoing extension of the M3D-C1 code, 2D MHD equations were solved on a unique 2D plane. Design of the code to support the 3D resolution of MHD equations has been focused on supporting the automatic creation of copies of a pattern 2D plane along the toroidal direction (Figure 6) and appropriate methods to support the efficient parallel communication of the sets of information defined on each plane. To this end, we are developing components as follows:

- Creation of a new software component to support the management of the planes. Each plane and the set of information (vectors, matrices, geometry, mesh) attached to that plane are managed by an abstract model that is in charge of coordinating communications between the planes.
- Extension of the parallel mesh database management: Triangle meshes are used to discretize each 2D plane. Extension of FMDB has been implemented in order to automatically generate a 3D parallel mesh (wedges) that connects the triangles located on each plane.
- Extension of the parallel algebraic data container management: Methods and functions to support storage and queries of the data contained in the defined vectors and matrices have been added.

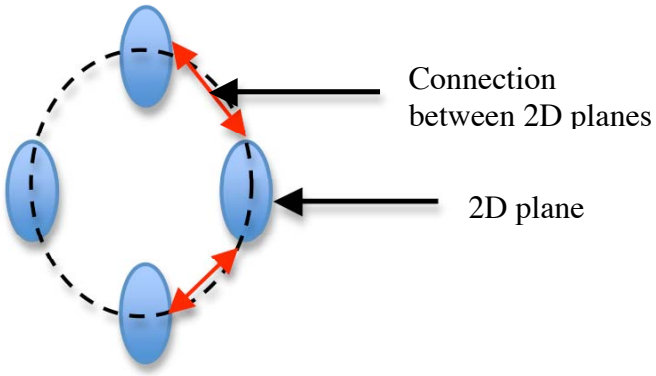

Figure 6. Configuration used to solve 3D MHD equations.
A 2D pattern plane is populated along the toroidal direction.

4. Plans for the Coming Year

During the coming year the RPI ITAPS team will continue to work with the rest of the ITAPS team on the ITAPS interfaces including starting the iFields interface once we jointly complete its design.

We will continue to work on our improving the ability of the parallel ITAPS tools to operate on petascale computers (and beyond). Efforts on partition improvements are being coordinated with the Zoltan developers (including interactions with CSCAPES). Substantial efforts are planned for the improvement of the ITAPS mesh adaptation service both in terms of the scalability of the mesh adaptation algorithms and message packing tool IPComMan.

Efforts will continue with supporting parallel adaptive mesh control methods for COMPASS with the goal of supporting their integration with the SLAC solvers on massively parallel computers and working closely with CEMM developers on the fully 3D implementation of a parallel M3D-C1. These efforts will also be coordinated with the TOPS center that will be providing the core non-linear equation solving components.

5. Publications for the Last Year

[1] F. Delalondre, C. Smith, M.S. Shephard, "Collaborative Software Infrastructure for Adaptive Multiple Model Simulation", *Comp. Meth. Appl. Mech. Engng.*, accepted 2009.

[2] F. Delalondre, C. Smith, and M.S. Shephard, Int. Conf. on Adaptive Modeling and Simulation, *ADMOS 2009*, CIMNE, Barcelona, 2009

[3] Fabien Delalondre, Cameron Smith, Mark S. Shephard, "Collaborative Software Infrastructure for Adaptive Multiple Model Simulation", *Proc. US National Congress on Computational Mechanics*, Columbus, Ohio, July 16-19, 2009 (Abstract).

[4] K. Devine, L. Diachin, J. Kraftcheck, K. E. Jansen, V. Leung, X. Luo, M. Miller, C. Ollivier-Gooch, A. Ovcharenko, O. Sahni, M.S. Shephard, T. Tautges, T. Xie, M.

Zhou, “Interoperable Mesh Components For Large-Scale, Distributed-Memory Simulations”, *Journal of Physics: Conference Series*, 180, 012011, 11 pages, 2009.

[5] A.Y. Galimov, O.Sahni, R.T. Lahey, Jr., M.S. Shephard, D.A. Drew, K.E. Jansen, “Parallel Adaptive Simulation of a Plunging Liquid Jet”, *Acta Mathematica Scientia*, submitted 2009.

[6] S. C. Jardin, J. Breslau, J. Chen, S. Gerhardt, N. Ferraro, X. Luo, K. Jansen, M. Shephard, “Initial application of the M3D-C1 code to the study of non-ideal modes in NSTX”, APS, Nov. 2009 (Abstract).

[7] X.-J. Luo, M.S. Shephard, L.-Q. Lee and C. Ng, “Moving Curved Mesh Adaption for Higher Order Finite Element Simulations”, *Engineering with Computers*, accepted, 2009.

[8] X. Luo, T. Stylianopoulos, V.H. Barocas and M.S. Shephard, “Multiscale computation for soft tissues with complex geometries”, *Engineering with Computers*, 25(1):87-96, 2009.

[9] C. Ollivier-Gooch, L.F. Diachin, M.S. Shephard, T. Tautges, J. Kraftcheck, V. Leung, X.-J. Luo and M. Miller, “An Interoperable, Data-Structure-Neutral Component for Mesh Query and Manipulation”, *Transactions on Mathematical Software*, accepted, 2009

[10] Ovcharenko, A., Shephard, M.S., Sahni, O, Jansen. K.E. and Carothers, C.D., “Subdomain Communication to Increase Scalability in Large-Scale Scientific Applications”, *ICS’09*, June 8–12, 2009, ACM, York Town Heights, NY, USA (Abstract).

[11] O. Sahni, X.J. Luo, K.E. Jansen, M.S. Shephard, “Curved Boundary Layer Meshing for Adaptive Viscous Flow Simulations”, *Finite Elements in Analysis and Design*, 46:132-139, 2010.

[12] O. Sahni, C.D. Carothers, M.S. Shephard and K.E. Jansen, “Strong Scaling Analysis of a Parallel, Unstructured, Implicit Solver and the Influence of the Operating System Interference”, *Scientific Programming*, 17:261-274, 2009.

[13] O. Sahni, K.E. Jansen, C.A. Taylor and M.S. Shephard,“Automated Adaptive Cardiovascular Flow Simulations” *Engineering with Computers*, 25(1):25-36, 2009.

[14] O. Sahni, M. Zhou, M.S. Shephard and K.E. Jansen, “Scalable Implicit Finite Element Solver for Massively Parallel Processing with Demonstration to 160K cores”, *Proceedings of the SC09*, Gordon Bell Finalist, 2009.

[15] Onkar Sahni, Michael Amitay, Mark S. Shephard and Kenneth E. Jansen, “Investigation of Active Control of 3D Flows using Adaptive Simulations”, *Proc. US National Congress on Computational Mechanics*, Columbus, Ohio, July 16-19, 2009, (Abstract)

[16] M.S. Shephard, M.A. Nuggehally, B. FranzDale, C.R. Picu, J. Fish, O. Klaas, J. Tourtellott and M.W. Beall, “Component Software for Multiscale Simulation”, *Bridging the Scales in Science and Engineering*, Oxford University Press, pp. 393-421, 2009.

[17] Mark S. Shephard, Kenneth E. Jansen Onkar Sahni, Xiao-Juan Luo, Min Zhou, Aleksandr Ovcharenko, and Ting Xie, “Recent Advances and Experiences in

Adaptive Mesh Generation and Control for Large Scale Simulations", *Proc. US National Congress on Computational Mechanics, Columbus*, Ohio, July 16-19, 2009, (Abstract).

[18] M. Zhou, O. Sahni, H. J. Kim, C.A. Figueroa, CA. Taylor, M.S. Shephard, and K.E. Jansen, "Cardiovascular Flow Simulation at Extreme Scale, *Computational Mechanics*, accepted 2009.

[19] M. Zhou, O. Sahni, M.S. Shephard, C.D. Carothers and K.E. Jansen, "Adjacency based reordering algorithm for acceleration of finite element computations", submitted *Scientific Programming*, 2009.

[20] M. Zhou, O. Sahni, M.S. Shephard, K.D. Devine and K.E. Jansen, "Controlling unstructured mesh partitions for massively parallel simulations", *SIAM J. Sci. Comp.*, submitted 2009.

6. Project Personnel

The following individuals are involved with the Rensselaer component of the ITAPS project:

- Kenneth E. Jansen, Rensselaer Principal Investigator
- Xiaojuan Luo, Senior Research Associate
- Alexander Ovcharenko, Graduate Student
- Onkar Sahni, Senior Research Associate
- Mark S. Shephard, Rensselaer Project Director
- Ting Xie, Graduate Student
- Min Zhou, Graduate Student, Post Doctoral Research Associate