Geometry based pre-processor for parallel fluid dynamic simulations using a hierarchical basis

Anil Kumar Karanam *
Scientific Computation Research Center, RPI

Kenneth E. Jansen
Scientific Computation Research Center, RPI

Christian H. Whiting †
ABAQUS Inc.

Abstract

The pre-processing stage of finite element analysis of the Navier-Stokes equations is becoming increasingly important as the desire for more general boundary conditions, as well as applications to parallel computers increases. The set up of general boundary conditions and communication structures for parallel computations should be accomplished during the pre-processing phase of the analysis, if possible, to ensure efficient computations for large scale problems in computational fluid dynamics (CFD). This paper introduces a general methodology for geometry based boundary condition application and pre-computing of parallel communication tasks.

1 Introduction

Large scale finite element simulations [11, 12], require boundary condition information, parallel communication data structures, and higher order degree-of-freedom information to be pre-processed for rapid retrieval during the computation phase of the analysis. This is critical for achieving scalability and short turnaround times. Although this is not always possible, (e.g. dynamic mesh adaptivity during computation will require data structure modification during simulations), when it is, it represents a substantial decrease in computation time. Many large scale computations of laminar and turbulent flows, do meet the requirements for pre-processed data structures, and the additional speed gained is often a necessity to acquire a solution in a reasonable time-frame.

The pre-processing method described here is based on recent developments in abstract mesh representation and classification against a geometric model. Mesh-model classification (see Beall and Shephard [3]) provides a connection between the finite element mesh, which may be varied over the course of several analysis runs, and the geometric (topological) model of the physical domain, which remains fixed. Given this relationship, boundary condition attributes are associated with the geometric model(O'bara *et al.* [13]) the topological description of the problem domain, rather than the finite element mesh, thus enabling a more intuitive approach to their application [17]. The finite element mesh then inherits boundary condition attributes from the geometric model. The pre-processor interacts with the geometric model, as well as the finite element mesh, to pre-compute the element degree-of-freedom (mode) connectivity information (including information necessary for higher order computations), boundary condition arrays, and parallel communication data structures. In this paper when we use the term *mode*, we refer an interpolation function which contributes a single degree of freedom to the global system of equations.

Essential and natural boundary condition attributes may be quite complex for fluid dynamics simulations and require substantial pre-processing for accurate specification. In addition, periodic boundary conditions are handled differently than other essential boundary conditions and pose additional difficulties. A method is presented

*Supported by NSF Grant No. 9985340
†Supported by a grant from NASA LaRC

for application of general sets of boundary conditions to the geometric model, which are conferred to the finite element mesh during pre-processing. Classical finite element procedures traditionally associate boundary conditions directly with the nodal degrees of freedom which requires that the boundary conditions be set up for each mesh, even when the geometric model is unchanged. In addition, higher-order computations rely heavily on the additional topology of the mesh (e.g. mesh edges and faces), information which is not readily accessible from the classical data structures which are based on simple element connectivity in terms of global mode numbers. To address these issues, the boundary conditions are instead applied directly to the geometric model (no mesh even needs to exist at this stage) and the pre-processor reads the model and the mesh and associates the model information with the mesh. By pre-processing boundary condition information, the analysis code has no need to make expensive geometric model queries and therefore, can rapidly constrain the required modes.

The increasing complexity of fluid dynamic simulations has lead to the heavy use of parallel computers. While the solvers themselves can be trivially parallelized [11, 23, 22] and have been shown to yield near perfect scaling on large problems, a non trivial, extensive effort goes into the pre-processing of these parallel data structures and communication traces which make the parallelism almost transparent to the solver.

In this paper we attempt to address the most critical aspects required for pre-processing higher order simulations. First we introduce the idea of topological mesh model hierarchy. Then we discuss the concepts used in applying boundary and initial conditions efficiently. At this point we also present the compact data structure which holds all the higher order information and is provided as input for the solver code. Following that, we briefly describe the additional pre-processing required for parallel processing.

2 Topological hierarchy

Finite element meshes have traditionally been described in terms of nodal coordinates and element connectivity. For the purpose of preprocessing, this representation has been dispensed in favor of a richer topological data structure [3]. This data structure or database, maintains information related to all mesh entities, vertices, edges, faces, and regions as well as adjacency relationships between them. The geometric model may be similarly defined in terms of model entities of the same type. The i^{th} mesh entity of dimension d_i will be denoted by $M_i^{d_i}$ similarly $G_i^{d_i}$ for a model entity. For linear basis computations this information is clearly more than necessary, however, preprocessing higher-order computations requires this. The pre-processor also performs the essential task of distilling the available generic data available in the initial stages to an optimal size and form needed by the main analysis code. Therefore the use of the entire mesh data structure is confined to the pre-processing stage, and only the traditional finite element structures (coordinates and connectivity generalized to support higher order mode information) are used within the analysis code. This means that the traditional structures are created during pre-processing, and written to disk to be read by the analysis code. This data structure is identical to the traditional format, with the exception that information on higher order degrees of freedom is included.

Central to the method of pre-processing described here is the idea of the classification of the finite element mesh on the geometric model. Every stage in the pre-processor makes extensive use of the cross referencing capability provided by this idea. Mesh-model classification defines the relationship between the finite element mesh and the physical domain (geometric model) on which the problem is to be solved and is key to the development of geometry based boundary condition specification. Beall and Shephard [3] define *mesh classifi cation against the geometric domain* as the unique association of a mesh entity of dimension d_i, $M_i^{d_i}$ to a geometric model entity of dimension d_j, $G_j^{d_j}$ where $d_i \leq d_j$, denoted $M_i^{d_i} \sqsubset G_j^{d_j}$. The classification symbol $\sqsubset$ indicates that the left-hand entity is classified on the right-hand entity.

For example a mesh vertex can be classified on a model vertex, model edge, model face or model region. Where as a mesh edge cannot be classified on model vertex (an entity of lower order than itself), similarly mesh faces cannot be classified on model edges or vertices and mesh regions can be classified only on model regions. This unique classification can be used to transfer the desired boundary condition information from the model entities

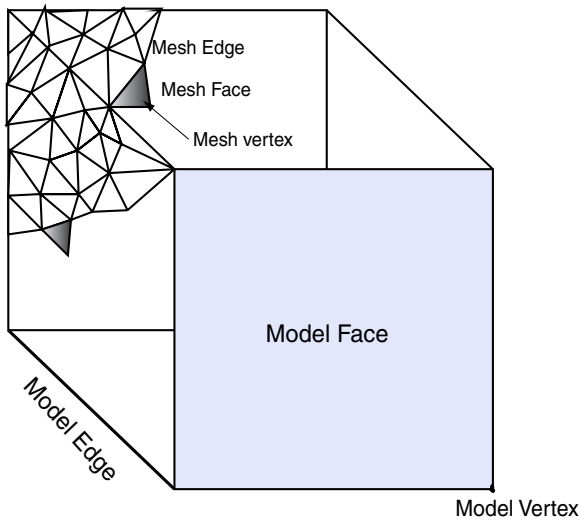


Figure 1: Mesh Model Classification

to the mesh entities that are classified on them. In the remainder of the document geometric model entities are referred as model entities and the finite element mesh entities are referred to as mesh entities.

3 Boundary and Initial conditions

A central part of the finite element formulation is the accurate specification of boundary conditions, which help define the physical problem being solved. Furthermore initial conditions affect the time it takes to reach a steady solution, when that is the goal or strongly influence the solution when solving unsteady problems.

When solving differential equations using numerical or finite element methods, generally we have three kinds of boundary conditions. First there are essential or Dirichlet boundary conditions, which constrain the solution variables, on specified sections of the boundary. Secondly there are natural or Neumann boundary conditions, which constrain the fluxes (which are dependent on the derivatives) of the variables. The third type of boundary condition involves periodicity in which case the solution repeats itself after a certain spatial interval, in a given direction. We can take advantage of this fact by equating the solution variables at the ends of the periodic spatial interval. This periodic interval can be either a translation or a rotation. This type of periodic boundary conditions are really a subtype of a larger class of boundary conditions known as multi-point constraints.

For Navier-Stokes equations essential boundary conditions can be applied to all of the solution variables, pressure, three components of velocity and temperature. There are two specific ways of specifying velocity essential boundary conditions. The first one constrains one component of velocity while leaving the other components free. This will be referred to as *one_component* boundary conditions. The velocity components in the plane perpendicular to this vector are free. The second way completely specifies the complete velocity vector, thus all components are constrained. This will be referred to as *three_component* boundary conditions. When only two components are prescribed, this can be done with the application of two one component conditions. Finally natural boundary conditions are applied in the form of mass or heat fluxes, traction on surfaces and pressure specified over a surface in a weak or integral sense.

3.1 Inheritance of boundary conditions

In section 1, we discussed the capability of applying general sets of boundary conditions to the geometric model, which are then automatically transferred to the finite element mesh during pre-processing. This ability to transfer boundary condition data attached to a topological model, to the finite element mesh is one of the most important attributes of this pre-processing method. This capability is made possible by the mesh-model hierarchy discussed in section 2.

While the mesh-model classification saves us from having to set boundary conditions on individual mesh entities, for complicated models even the model entities could run into the hundreds. The application of boundary conditions, even to a relatively simple geometric model, can become quite cumbersome if the model has many edges and

vertices. Since the physical information is most naturally associated with the model faces, a method has been implemented by which edges inherit from faces and vertices from edges, thus enabling the user to consider only the faces. While this makes the application of boundary conditions much simpler, it places additional demands on the pre-processor to derive the correct set of boundary conditions to be inherited.

To address this problem we introduce the concept of boundary condition inheritance. This allows for the boundary conditions to be set on the highest possible model entities (usually faces, since faces are the highest level entities which can represent a boundary). The pre-processor uses a predefined set of rules to inherit the boundary condition attributes from the faces to the other lower order entities (edges and vertices). Setting boundary conditions on model faces is usually all that is required, but occasionally the user might need to specify boundary conditions on a small number of edges and vertices to correct for any known conditions of conflict of the inheritance.

The task of transferring the boundary condition values from model to mesh entities is performed by using the classification information provided by the mesh data structure. For instance if a particular value of pressure is set on a model face, the pre-processor iterates over all the mesh entities classified on this model face and applies the pressure boundary condition to them. There is no scope for any sort of conflict since each mesh entity is uniquely classified on a single model entity. On the other hand the transfer of information from the model faces (on the boundary conditions were actually specified) to the lower model entities (edges and vertices) is usually more involved. In the topological geometric model, edges are formed by the intersection of two model faces and vertices are defined at locations where at least two model edges meet. When deriving boundary conditions from multiple neighbors there could both resolvable and unresolvable conflicts. For instance when two different directions are specified for the *one_component* velocity on two neighboring faces of a model edge, both directions are preserved and two components are constrained instead of one. Where as, if the conflict occurs in the magnitudes only, then the result is indeterminate and it is suggested that velocity boundary condition be explicitly set on the edge to avoid the ambiguity arising from inheritance.

The following general methodology is used to implement the boundary condition inheritance from higher order to lower order model entities. The pre-processor iterates over all the model entities, faces followed by edges and finally vertices. For each entity, if any boundary condition attribute is explicitly specified, that value is given precedence and used as it is (usually the case for model faces). When nothing is explicitly specified on an entity, the boundary condition attributes are derived from the higher order entities intersecting to create it. All the immediate higher order entities in contact with the current one are visited and the value of the attribute in the current entity is derived as a combination of all the surrounding higher entities. For instance a model edge will derive its boundary condition data from the model faces intersecting to create the edge. Similarly a vertex will derive its information from the model edges coincident on it.

The above described method of boundary condition inheritance can be applied to both essential and periodic class of boundary conditions. It should be noted that the natural boundary conditions are defined only for model faces and are not derived for edges and vertices since they do not make physical sense for edges and vertices on which the formulation does not evaluate boundary integrals.

3.2 Boundary and initial conditions for hierarchic basis

The boundary condition coefficients are calculated at the pre-processing stage where the rich mesh model classification structure is available and then the evaluated values are written out in the traditional form and can be efficiently applied at the solver stage. This also enables us to set boundary conditions on the higher order modes attached to mesh edges and faces. In the current work we have used higher order basis functions based on the Legendre Polynomials. They have been discussed in detail by Shephard *et al.* [18].

Higher order basis functions using Lagrange basis functions enforce essential boundary conditions in a relatively straight forward manner, as the basis coefficients correspond to solution values at nodes, e.g., the Lagrange interpolation equation $N_a(\xi_b) = \delta_{ab}$. Since the solution coefficients with respect to the hierarchical basis

do not correspond to solution values at spatial locations, work must be done to determine the coefficients on the boundaries. To accomplish this, we interpolate the known Dirichlet boundary function with the hierarchical basis by solving a linear system of equations for the unknown basis coefficients. Additionally, a unique set of interpolation points must be chosen since there are no particular spatial locations associated with the higher-order coefficients.

The element level interpolation may be constructed by solving a linear system of equations for the coefficients on each element in the domain. Suppose we wish to specify that $\phi(x_i) = g(x_i)$ over some portion of the boundary, where $\phi(x_i)$ could be any of our solution variables. We can find the coefficients of an approximation to $g(x_i)$ (for each element, e) as

$$g(x_i) \approx \hat{g}^e(x_i) = \sum_{a=1}^{n_{ip}} g_a^e N_a^e(x_i) \tag{1}$$

where N_a^e are the basis functions for element e, g_a^e are the unknown coefficients, and n_{ip} is the number of interpolation points, which must equal n_{es}, the number of element shape functions. Equation (1) can be expressed in the following matrix system.

$$\boldsymbol{M}\boldsymbol{g} = \boldsymbol{R} \tag{2}$$

$$\boldsymbol{M} = [M_{ab}] = N_a^e(\boldsymbol{\xi}_b^{int}), \tag{3}$$

$$\text{and } \boldsymbol{R} = [\boldsymbol{R}_b] = g(\boldsymbol{x}(\boldsymbol{\xi}_b^{int})) \tag{4}$$

where $\boldsymbol{\xi}_b^{int}$ is the b^{th} interpolation point (in element e's coordinates). This system of linear equations is solved for the basis coefficients, g_a^e, which are used when needed by the analysis code to evaluate $\phi(x_i)$ (which is expanded in the same basis as $\hat{g}^e(x_i)$). Even though we are using only element level interpolation, the resulting values are continuous between elements. The hierarchical nature of the higher order basis functions ensures that only the modes common between neighboring element boundaries are coupled and C_0 continuity insures the same answer will be obtained regardless of which element is chosen. For instance, when interpolating the edge coefficients for an edge shared by two or more elements, we will always arrive at the same value regardless of which element we use to do the interpolation since the only contributing shape functions are that of the two vertices associated with the end points of the edge and the functions associated with that edge. For computations using the Lagrange basis, the interpolation points are simply the nodal coordinates, and the matrix in (2) is the identity matrix. Furthermore, the system of equations described above may be simplified somewhat by statically condensing the coefficients since not all functions are coupled. In practice, however, the interpolation is only computed during preprocessing, making the time savings less significant.

The procedure described above for essential boundary conditions may also used to set an initial condition in cases where the exact initial condition is relevant to the simulation. However, experience has shown that in cases where such accuracy is not necessary (as is usually the case), using the linear interpolation of the initial conditions is sufficient to ensure convergence. The linear interpolation is obtained by simply setting all higher-order coefficients equal to zero.

3.3 Periodic boundary conditions

The application of periodic boundary conditions poses several difficulties in the context of hierarchical basis functions since all mesh entities must be identical on periodic planes (including edge and face directions). A general methodology has been developed for the application of periodic boundary conditions. The data necessary to enforce periodic boundary conditions can be contained in a single array which specifies the "periodic master" of each mesh entity. When essential boundary conditions are set, periodic boundary conditions are also set by copying the solution coefficients of the periodic masters to their periodic slaves. This operation is simply an indirect address of the solution array using the periodic boundary conditions array. The equations corresponding to the periodic entities are eliminated from the system by using this array to zero the corresponding residual components.

As with the essential and natural boundary conditions described in section 3.1, periodic boundary conditions are specified on the entities of highest possible dimension, usually faces. The process of boundary condition inheritance is used to transfer these to bounding edges and vertices. As in the previous case, any periodic boundary condition attributes directly specified on an entity take prece-

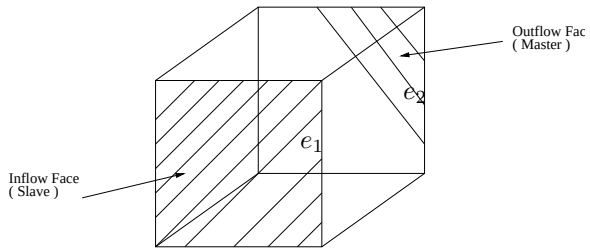


Figure 2: Inheritance of periodic boundary conditions

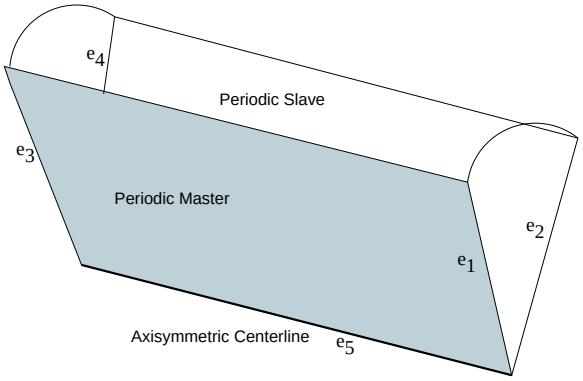


Figure 3: Axisymmetric periodicity

dence over inherited values. We present two examples to illustrate the procedure.

In the first case, let us consider the simple box model shown in Figure 2 and describe the process by which one of the edges inherits the periodicity from the faces. In this example, the inflow and outflow faces have been specified as periodic with each other, with the outflow face as the periodic master. When the edge e_1 is probed for periodic boundary conditions, all the higher order entities (faces) adjacent to edge are queried for periodic boundary condition attributes. In the current example the inflow face is adjacent to the edge e_1 and it returns the tag of the outflow face as the master. The pre-processor uses this outflow face to iterate over all its edges until it finds the edge e_2 which is closest to the edge e_1 and sets it as the master.

For the second example we will consider a case with axisymmetry, Figure 3. Again, there is a master face and a slave face and e_1 finds its master edge, e_2 through inheritance from the face as in the previous example. One significant difference from the box case is that a match is determined by comparing e_1 rotated by the angle determined by the angle between the two periodic faces. This process is repeated for e_3 and e_4 without change but e_5 finds that it matches itself. Such a situation signals that e_5 is in fact an axis of symmetry where the two non-axial components should be constrained to zero.

4 Compact data structure

Recall that the goal of pre-processing was not only to produce the boundary and initial conditions but also the data structures necessary to enable integration over the finite elements of the mesh, when using a hierarchical basis. The traditional finite element mesh structure providing coordinates and connectivity is no longer sufficient. All the edges in the mesh pick up modes for $p \geq 2$, triangular faces pick up modes for $p \geq 3$ and quadrilateral faces for $p \geq 4$. In addition, different element topologies start having region modes starting at different polynomial orders. Tetrahedral mesh regions will have region modes associated with them for $p \geq 4$, hexahedral and pyramidal mesh regions for $p \geq 6$ and wedges for $p \geq 5$ (see Shephard *et al.* [18] for details). Here we exploit the rich data structure provided by mesh data base as described in section (2) to generate all the higher order mode and entity information and condense it into the compact data structure used by the solver code. Also all the information pertaining to any entity (such as a vertex, an edge, a face or a region) are kept local. This information includes the number of modes on the entity, its partition adjacency information, global equation number and its local polynomial order. This increases the memory requirements somewhat but simplifies the process of applying boundary conditions and generating parallel communication traces.

The first step in setting up the data structures is the assignment of global equation numbers to all mesh entities.

This is done by visiting each entity, determining the number of shape functions it contributes based on its polynomial order and assigning a unique equation number for each of these functions. Next all the elements in the mesh are visited and the equation numbers associated with the lower order entities bound by it are collected. For example a tetrahedral region may collect equation numbers from 4 vertices, 6 edges and 4 faces and the region itself if the polynomial order is greater than 3. This procedure is similar to that described by Hughes [10] for meshes of Lagrange elements where the global node numbers associated with each finite element are stored in the data structure.

This connectivity information provides a complete description of the mapping between the element level computations and the global degrees of freedom (where the linear equations are formed and solved). For hierarchical basis functions of degree 3 and higher some of the basis functions need to have their sign reversed since the mapping from the entity to the element coordinate system introduces a sign change for some of the bounding elements. The situation is illustrated by a simple example shown in Figure 4.

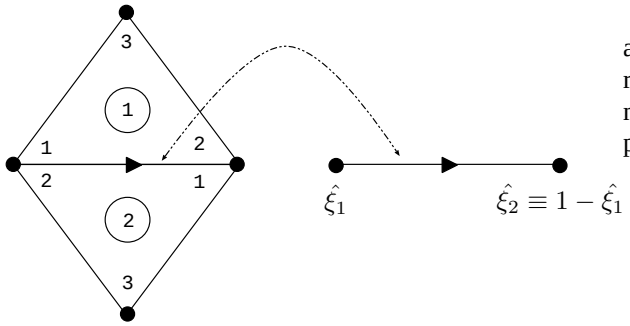


Figure 4: Mesh elements illustrating the reversal of shape functions

In Figure 4 a two dimensional case is shown in which two triangular mesh faces share a common edge; we will consider each face as an element. This figure shows the element numbers in circles, as well as each local degree of freedom number with respect to each of the elements. The edge is also shown along with its local coordinate system, $\hat{\xi}_1$, which is directed as indicated by the arrow, note that the global direction of the edge is determined by the vertex ordering stored in the mesh database. The local coordinates of the edge must be mapped to the coordinate system of each bounding element in order to evaluate the function. It is desirable to evaluate a single set of element shape functions to be used for all elements in the mesh. This enables the basis functions to be pre-computed and tabulated for each quadrature point, as commonly done for Lagrange-type elements. This is not possible without accounting for the edge direction in some way.

Returning to the example, suppose $k = 3$ has been set on the edge depicted in Figure 4. It will therefore contribute two functions, one quadratic and one cubic, to the local basis of each of the two bounding triangular elements (see [6] for a description of the shape functions) given by

$$N_2(\hat{\xi}_i) = -2\hat{\xi}_1\hat{\xi}_2 \tag{5}$$

$$N_3(\hat{\xi}_i) = -2\hat{\xi}_1\hat{\xi}_2(\hat{\xi}_2 - \hat{\xi}_1) \tag{6}$$

where the parametric coordinates for this edge are

$$\hat{\xi}_1 \quad \text{and} \quad \hat{\xi}_2 \equiv 1 - \hat{\xi}_1 \tag{7}$$

and the subscripts on the basis functions refer to their respective polynomial orders. When the coordinates are mapped from the edge to the element coordinates, the problem becomes apparent, i.e.

Element 1:

$$\xi_1 = \hat{\xi}_1, \qquad \xi_2 = \hat{\xi}_2 \tag{8}$$

Element 2:

$$\xi_1 = \hat{\xi}_2, \qquad \xi_2 = \hat{\xi}_1 \tag{9}$$

and the basis functions become:

Quadratic:

$$N_2^{(1)} = -2\xi_1\xi_2 \tag{10}$$

$$N_2^{(2)} = -2\xi_2\xi_1 = N_2^{(1)} \tag{11}$$

Cubic:

$$N_3^{(1)} = -2\xi_1\xi_2(\xi_2 - \xi_1) \quad (12)$$

$$N_3^{(2)} = -2\xi_2\xi_1(\xi_1 - \xi_2) = -N_3^{(1)} \quad (13)$$

where the superscript indicates the element that the function is associated with. The cubic function on element 2 is the negative of that on element 1, while the quadratic function is the same, regardless of the edge direction. This case generally occurs when an element uses an edge in the opposite direction than that edge was defined. Since the cubic edge function is different for each of the bounding elements, a single set of basis functions will clearly not suffice to completely describe the basis. To overcome this difficulty, during pre-processing the local degree of freedom numbers that correspond to shape functions that must be negated are flagged (e.g. there equation numbers are negated). This information is then used in the flow solver to create the correct element basis functions from the pre-computed table of element functions. For quadratic or linear basis, no functions need to be negated, and the data structures may be used as they are. This also implies that when using the hierarchical code with linear elements, no significant penalty is paid for having the generality of higher-order basis functions in the same code.

The compact data structure described above can be easily analyzed regarding its memory usage. When employing piecewise linear functions on tetrahedra, the savings is dramatic (nearly a factor of five less memory used to store the mesh). The savings is reduced, but still present, for quadratic (a factor of nearly two). As expected, when the basis is increased to cubic, the memory advantage is lost ("compact" data structure uses 30% more memory) but this is a modest tradeoff for the significant improvement to algorithmic efficiency and simplicity. Comparisons were also made in terms of computational effort. The code that made use of the compact data structure described above typically required a factor of five less time taken when compared to a flow solver that makes direct use of the mesh database. While it is clear that some of the observed advantage can be attributed to the preprocessing which effectively maps all the data required for the integration of a large set of elements in one operation (as opposed to traversing the mesh data structure to collect this information one element at a time), there were other differences in the codes that make it impossible to attribute all of this gain to the compact data structure.

5 Parallel communication

Finite element methods are very well suited to use on parallel computers as a substantial computational effort is in the calculation of element level integrals. To reduce the computational effort during the analysis phase, the structures specifying the interpartition (sometimes called interprocessor or interprocess to reflect that it is the computer processes that are communicating to sustain a correct parallel advancement of the solution strategy) communications are pre-processed. Each partition then executes a copy of the analysis code, reading the pre-processed input data relating to its portion of the domain, as well as information relating to other partitions it must communicate with. Each partition makes a call to the linear algebra solver with the locally formed matrices. We have used a linear algebra solver based on GMRES [14, 16], which needs only matrix vector products. The parallelism is transparent to the linear algebra solver which works on each individual partition as if it was the whole domain. This section describes the information that partitions must communicate to each other as well as the construction of these communication structures by the pre-processor.

5.1 Mesh partitioning

For a parallel simulation to be efficient, the load distribution on all the partitions should be balanced at all times. This helps avoid some partitions lying idle and some being overloaded. In recent years, a considerable attention has been focused on solving the problem of mesh partitioning [7, 8, 2, 19, 4, 20, 5]. For computations which depend on adaptive refinement, various strategies of dynamic load balancing and migration have been developed [1] and have to be deployed at the solver stage. But for computations not depending on dynamic adaptivity all the load balancing and partitioning can be done at the preprocessing stage. This involves partitioning the mesh and assigning local equation numbers for all the entities on every partition.

Mesh partitioning can be done in many ways to suit specific needs. We have chosen two popular strategies suitable for our needs.

Inertial Partitioning: This is the method where partition boundaries are chosen based on geometrical considerations [9]. The users specify criterion used to divide the mesh into partitions. While this method is not ideal for achieving optimal interface length or perfect load balancing in a generic case, it does give the user a greater control over the description of the partitions. This can prove very helpful in some special cases where specific sections of the mesh have to be on pre-determined partitions. An example for inertial partitioning can be seen in Figure 5.

Dual Based Partitioning: This involves partitioning the dual graph of the mesh using various graph partitioning methods. We have used the generic graph partitioning package, *METIS* [15], which provides extensive opportunities for customization and is known to produce partitions with good load balancing and optimal interface length. An example of a dual based partition can be seen in Figure 6.

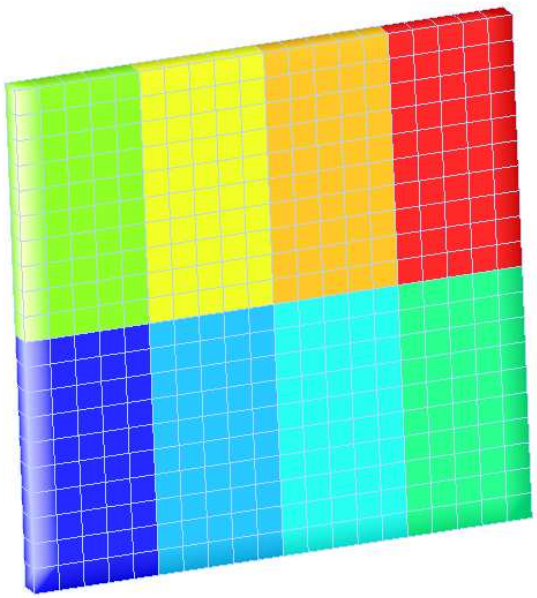


Figure 5: Inertial Partitioning

Each partition maintains a complete collection of data representing its portion of the finite element mesh and analysis information. This includes mode numbers and connectivity information as well as boundary condition data for all modes that are physically on the partition or its interpartition boundary. The finite elements are uniquely divided among the partitions, so each element will be found on only one partition. The mesh entities that contribute to element level integrals, however, appear in multiple partitions if they are on an interpartition boundary. Element level computations are performed completely local to each partition and must communicate only when the element level contributions,such as element integrals, are assembled to the global equations. This global assembly procedure involves the sending and receiving of mode information between partitions. Another case in which partitions must exchange information is when periodic boundary conditions are present with periodic partners residing on different partitions. These two types of communications will be described below.

5.2 Communication

The basic idea behind the parallel communication of finite element information can be described with the aid of Figures 7, 8 and 9. This figure illustrates three partitions and vertices on the interpartition boundaries. For simplicity, only vertices are shown since edges and faces are handled identically. Solid circles denote master vertices (where the equations are actually solved) while hollow circles denote slave vertices where only temporary information is stored. This master-slave relationship is established by creating a hierarchy based on host partition numbers (for example, 2 is greater than 1, so on the boundary between 1 and 2 all the vertex images on partition 2 will be designated masters and the images on 1 will designated slaves. Similar hierarchy can be used on other partition boundaries). The element residuals associated with each vertex are first assembled from elements on each of the bounding partitions. After local assembly [10], these values are sent in the direction of the arrows to the values on the master

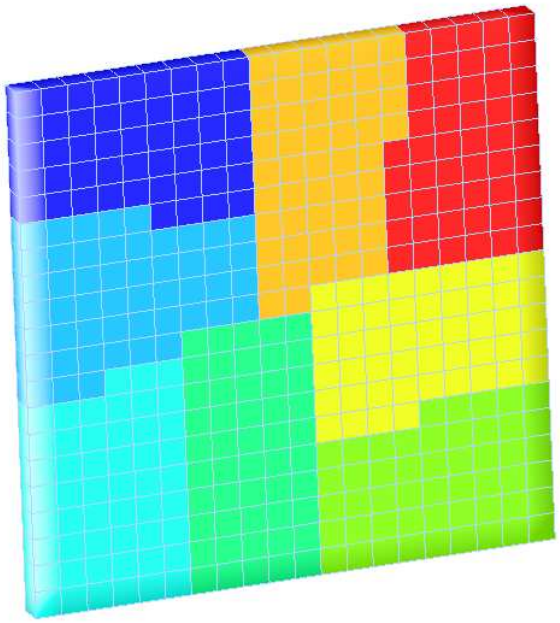


Figure 6: Dual Based Partitioning

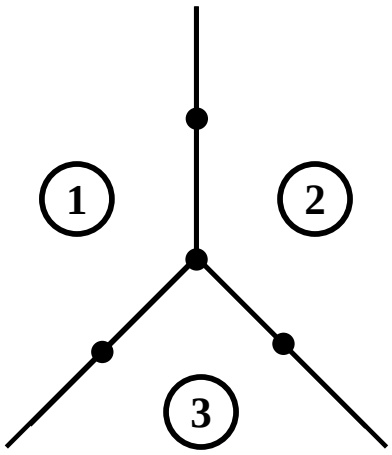


Figure 7: Overview of communication

partition (as defined by the numeric hierarchy described earlier) and added. The sending partition (also known as the slave partition) then zeroes its residual values, essentially removing this vertex from it's system. In this manner, all residuals associated with entities on the interpartition boundary are only solved for by a single partition, known as the entity's master image. After the equations are solved, the solution values are copied from the entity's image on the master partition to all of its slave images. The creation of the necessary data structures and the execution of these tasks is described in the next section.

For the partitions to exchange information during the computation, each must maintain a data structure describing its communications in addition to the other finite element data. The Message Passing Interface [21] is used for exchanging information between the partitions. Let us define a *communication stage* as that which involves all partitions making all their necessary communications. In other words, each communication stage consists of each partition sending to and receiving from each partition with which it must communicate. We will denote by N_P^i the number of partitions with which partition i must communicate. There are two types of communication stages: one in which the residuals are added to masters and zeroed on slaves (type 1), and another in which the solution values are copied from masters to slaves (type 2). Both require the same information and differ only slightly. A single type 1 communication stage is necessary each time the element level residual formation and local assembly is completed and a type 2 communication stage is necessary each time the boundary conditions are set on the solution vector. This two stage communication structure is a result of the method used to enforce boundary conditions, where in the residuals on boundary modes are zeroed.

From the perspective of a single partition, i, a communication stage may be described as a sequence of *tasks*, denoted T_j^i and j iterates over the partitions i interacts with. For MPI to carry out the communications described above, each task, T_j^i, must provide details of the exchange of data between partitions i and j. To this end, T_j^i has associated with it, the following integer data:

tag: A unique tag associated with T_j^i which distinguishes this send and receive for the MPI func-

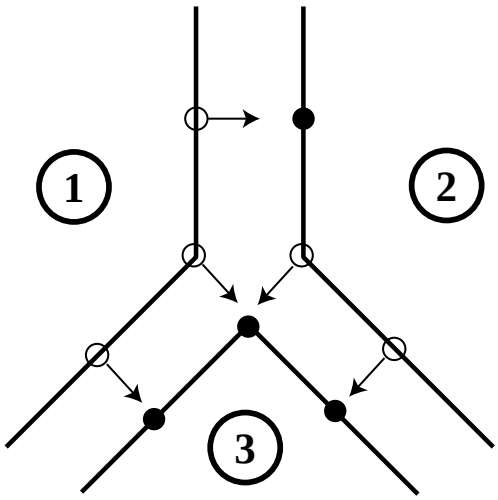


Figure 8: Communication Slave to Master

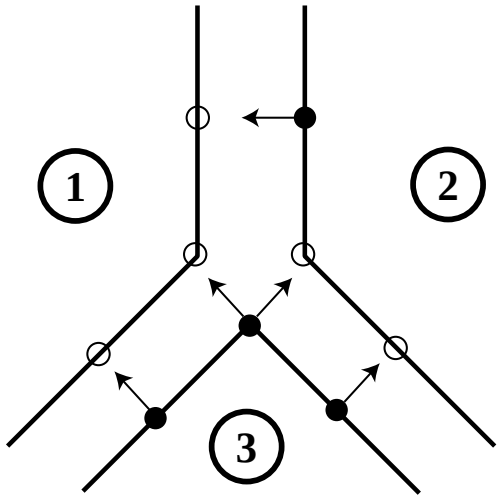


Figure 9: Communication Master to Slave

tions.

type: Denotes whether this partition is master or slave in the current communication. For a type 1 communication, a master calls MPI_receive(...) to receive and add the data, while a slave calls MPI_send(...) to send and zero the data. In a type 2 communication, the slaves receive and copy and the masters send and thus there is no need to zero anything.

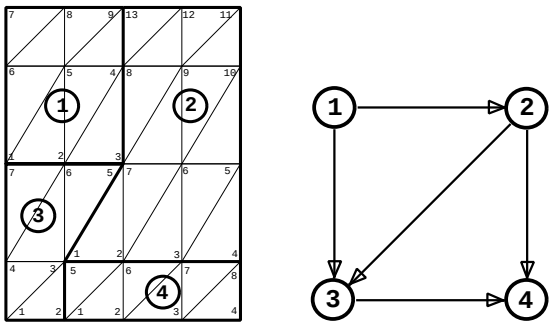
partner: PID number of the partner partition involved in this task.

numSeg: Number of data segments to be sent or received (see below).

segData: local mode numbers on the other partition to send to or receive data from for each of the numSeg data segments.

Here a data segment is defined as a continuously numbered group of mode numbers. Each segment also contains its length and starting index. In the beginning of execution of the analysis code, the segment data is used in conjunction with the MPI_TYPE_HVECTOR(...) function to create new MPI data types which are used during the communication stages. These data types are simply masks that describe where information can be found on the various partitions for each of the segments in T_j^i.

The above concepts can be clarified through a simple example. Consider the two dimensional mesh shown in Figure 10(a) which is decomposed into four partitions. This mesh can be considered as a single geometric face of a three dimensional model. The bold encircled numbers indicate partition ID numbers and the small numbers indicate local mode numbers on each partition. For simplicity, only vertex numbers are shown, however, edges (if $p \geq 2$) and faces (if $p \geq 3$) also get mode numbers associated with them and are handled identically to vertices. We also assume, for simplicity of discussion, that there are no periodic boundary conditions applied to the geometric model. A consistent graph corresponding to this mesh, created using the algorithm described above, is shown in Figure 10(b). Let us consider only the tasks associated with partition 2, T_j^2, where $j = 1, 3, 4$, $N_P^i = 3$. T_1^2 involves a communication where partition 2 is master, and modes 8 and 13 on partition 2 will receive contributions from modes 4 and 9, respectively, from partition 1. The other two tasks associated with partition 2, T_3^2 and T_4^2, are both slave communications. Here, mode 7 is sent to partition 3 where they are assembled into local mode



(a) Mesh (b) Graph

Figure 10: Communication example

5, and 1 through 4 are sent to partition 4, where they are assembled to local modes 5 through 8. Then these values are zeroed on partition 2.

Let us now consider that the two vertical edges in Figure 10(a) to be periodic, and the right edge as the master. T_1^2 now receives contributions from partition 1 modes 6 and 7 and adds them to local modes 10 and 11. Partition 2 does not receive any contributions for local mode 5 since the master vertex image for that vertex is mode 7 on partition 3 and both mode 5 on partition 2 and mode 1 on partition 1 add to mode 7 on partition 3 via T_3^2 and T_3^1. The above examples represent type 1 communication and they are reversed for type 2, as noted earlier.

6 Summary and Conclusions

In this paper we have presented a generalized method for statically pre-processing finite element computations. The mesh-model hierarchy around which this preprocessor was designed, was described. This hierarchy was then used to provide an intuitive way to apply a variety of boundary conditions. Finally, the details regarding the pre-processing necessary for the parallel communications were introduced.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 9985340.

References

[1] Y. Kallinderis A. Vidwans and Venkatakrishnan. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA Journal*, 23:497–505, 1994.

[2] M. Al-Nasra and D. T. Nguyen. An algorithm for domain decomposition in finite element analysis. *Computers and Structures*, 39:277–289, 1991.

[3] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *Int. J. Numer. Meth. Engng.*, 40(9):1573–1596, 1997.

[4] S. T. Bernard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *Proceedings of Sixth SIAM Conference on Parallel Processing for Scientifi c Computing*, pages 711–718, 1993.

[5] M. Cross C. Walshaw and M. Everett. Mesh partitioning and load-balancing for distributed memory parallel systems. In *Proceedings of International Conference on Parallel and Distributed Computing for Computational Mechanics*, 1997.

[6] S. Dey. *Geometry-based three-dimensional hp-fi nite element modelling and computations*. PhD thesis, Rensselaer Polytechnic Institute, 1997.

[7] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers and Structures*, 28:579–602, 1988.

[8] C. Farhat. On mapping of massively parallel processors onto finite element graphs. *Computers and Structures*, 32:347–354, 1989.

[9] J.R. Gilbert, G.L. Miller, and S.H. Teng. Geometric mesh partitioning: Implementation and experiments. *SIAM Journal of Scientifi c Computing*, 19(2091-2110), 1998.

[10] T. J. R. Hughes. *The finite element method: Linear static and dynamic finite element analysis*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[11] K. E. Jansen. A stabilized finite element method for computing turbulence. *Comp. Meth. Appl. Mech. Engng.*, 174:299–317, 1999.

[12] K. E. Jansen, C. H. Whiting, and G. M. Hulbert. A generalized-α method for integrating the filtered Navier-Stokes equations with a stabilized finite element method. *Comp. Meth. Appl. Mech. Engng.*, 190:305–319, 1999.

[13] Mark W. Beall Robert M. O'Bara and Mark S. Shephard. Attribute management system for engineering analysis. *Engineering With Computers*, 18:339–351, 2002.

[14] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[15] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion algorithms for repartitioning of adaptive meshes. Technical Report #97-013, U. Minnesota, Dept. of Comp. Sci. and Army HPC Center, 1997.

[16] F. Shakib, T. J. R. Hughes, and Z. Johan. A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis. *Comp. Meth. Appl. Mech. Engng.*, 75:415–456, 1989.

[17] M. S. Shephard. The specification of physical attribute information for engineering analysis. *Eng. Comput.*, 4:145–155, 1988.

[18] M. S. Shephard, S. Dey, and J. E. Flaherty. A straight forward structure to construct shape functions for variable p-order meshes. *Comp. Meth. Appl. Mech. Engng.*, 147:209–233, 1997.

[19] H. D. Simon. Partitioning of unstructured meshes for parallel processing. *Computer Systems in Engineering*, 2:135–148, 1991.

[20] D. Vanderstraeten and R. Keunings. Optimized partitioning of unstructured computational grids. *International Journal of Numerical Methods in Engineering*, 38:433–450, 1995.

[21] E. Lusk W. Gropp and A. Skjellum. *Using MPI – Portable Paralle Programming with the Message Passing Interface*. The MIT Press, Cambridge, Massachusetts, 1994.

[22] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *International Journal of Numerical Methods in Fluids*, 35:93–116, 2001.

[23] C. H. Whiting, K. E. Jansen, and S. Dey. Hierarchical basis in stabilized finite element methods for compressible flows. *Comp. Meth. Appl. Mech. Engng.*, accepted, 1999. SCOREC Report 11-1999, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY.