Mesh Regularization in Bank-Holst Parallel *hp*-Adaptive Meshing

Randolph E. Bank [*][1] and Hieu Nguyen[†][2]

[1] Department of Mathematics, University of California, San Diego, La Jolla, California 92093-0112, rbank@ucsd.edu.
[2] Department of Computer Science, University of California, Davis, Davis, California 95616, htrnguyen@ucdavis.edu.

1 Introduction

In this work, we study mesh regularization in Bank-Holst parallel adaptive paradigm when adaptive enrichment in both h (geometry) and p (degree) is used. The paradigm was first introduced by Bank and Holst in [1–3] and later extended to hp-adaptivity in [5]. In detail, the paradigm can be summarized in the following steps.

Step 1 – Load Balancing: The problem is solved on a coarse mesh, and available a posteriori error estimates are used to partition the mesh into subregions. The partition is such that each subregion has approximately the same error although subregions may vary considerably in terms of number of elements, number of degrees of freedom, and polynomial degree.

Step 2 – Adaptive Meshing: Each processor is provided with complete data for the coarse problem and instructed to sequentially solve the *entire* problem, with the stipulation that its adaptive enrichment (in h or p) should be limited largely to its own subregion. The target number of degrees of freedom for each processor is the same.

Step 3 – Mesh Regularization: The local mesh on each processor is regularized such that the mesh for the global problem described in Step 4 is conforming in both h and p.

Step 4 – Global Solve: The final global problem consists of the union of the refined partitions provided by each processor. A final solution is computed using domain decomposition.

This paradigm is attractive as it requires low communication and allows existing sequential adaptive finite element codes to run in parallel environment without

* The work of this author was supported by the U.S. National Science Foundation under contract DMS-0915220. The Beowulf cluster used for the numerical experiments was funded by NSF SCREMS-0619173.
† The work of this author was supported by the National Science Foundation under contract DMS-0915220 and a grant from the Vietnam Education Foundation (VEF).

much effort in recoding. However, it also poses some challenges in mesh regularization (Step 3). Since the adaptive enrichment on each processor (Step 2) is completely independent of what happens on other processors, the global refined mesh, constructed from the meshes associated with the refined regions on each of the processors, is initially non-conforming along the interface system.[3] Thus, we need to efficiently identify and resolve these nonconformities, and ultimately to establish links between degrees of freedom on the fine mesh interface system on a given processor and the corresponding degrees of freedom on other processors which share its interface. These tasks are challenging due to the fact that the meshes are unstructured in geometry (in h), have variable degree (variable p), no element refinement tree is available, and nonconformity exists in both h and p.

2 Data Structures

In our implementation of Bank-Holst paradigm in PLTMG, a relaxed version of longest edge bisection h-refinement and a rather flexible p-refinement strategy are used for hp-refinement, see [7].

2.1 Boundary Edge Data Structure

Each boundary edge is represented by a column in the $6 \times NBF$ integer array IBNDRY, where NBF is the number of boundary edges. For the Ith column of IBNDRY, four of the six entries contain information about the endpoint vertices, and indication of whether the edges is curved or straight, and a user-defined label. One entry indicates edge type (various boundary condition types, or internal interface), and the fifth entry, nonzero only for edges defining the interface system used in the parallel computation, encodes information which is used in the regularization process. This entry is described in more detail in Sect. 2.2 (Table 1).

AQ1

Table 1. Boundary edge information

IBNDRY(1,I)	First vertex number
IBNDRY(2,I)	Second vertex number
IBNDRY(3,I)	Curved edge
IBNDRY(4,I)	Edge type
IBNDRY(5,I)	Parallel information
IBNDRY(6,I)	User label

[3] The term “interface” is used to refer to the system of edges that are shared by two subregions, and the term “boundary” is used to refer to the union of the physical boundary of the domain and the interface.

2.2 Interface Edge Labeling

One approach to solve the nonconformities in the global refined mesh is to build and store refinement trees for all elements. However, such trees lose some of their attractiveness if procedures such as mesh moving and edge flipping destroy some of their properties. In addition, we only need information about the edges on the interface system, which typically is a very small fraction of the total information describing the mesh. Thus, instead of creating refinement trees for all elements, during the regularization phase we recover a refinement tree for each interface edge that defines the initial interface system. To insure that subregions remain geometrically conforming on all processors, we forbid mesh moving and edge flipping for all vertices and edges lying on the interface system.

Only minimal information needed to recover the edge refinement tree is stored for each interface edge. In particular, for each interface edge E, we need the index of its original edge $r(E)$ in the interface system of the broadcast coarse mesh (after Step 1) and its position in the refinement binary tree $s(E)$. Because the original (interface) edges are the same on all processors, we can first match them, and then their descendants based on their positions in the refinement tree. These two pieces of information are combined to make a single integer, $label(E)$, the parallel information for edge E stored in the fifth row of the IBNDRY array:

$$label(E) = r(E) + (s(E) - 1) * base.$$

Here *base* is an integer which is larger than the number of boundary edges *NBF* in the broadcast coarse mesh. For edge E_{org} in the broadcast mesh, $r(E_{org})$ is its number in the IBNDRY system and $s(E_{org}) = 1$. When an edge E is refined into two children E_1 and E_2, their labels are determined from $label(E)$ and the following identities:

$$r(E_1) = r(E_2) = r(E)$$
$$s(E_1) = 2 * s(E)$$
$$s(E_2) = 2 * s(E) + 1$$

For consistency, E_1 and E_2 are ordered in the counterclockwise traversal defined by vertices of E.

2.3 Interface Data Structure

When a boundary edge is refined, its entries in IBNDRY are replaced by those of one of its children. Thus IBNDRY contains only refined boundary edges. To recover the refinement trees of the interface edges, first all of the refined edges are sorted in groups according to $r(E)$. The refined edges in each group are then ordered in a counterclockwise traversal of the interface based on their vertices (end points). Edges in each group will be used to recover a refinement tree whose leaves and root represent themselves and their original edge respectively.

In order to illustrate the construction of the refinement tree of edges sharing the same ancestor, we consider the group of all refined edges associated with the original edge E as shown in Fig. 1. These edges have the same index $r(E)$ and have been ordered via a counterclockwise traversal. For simplicity, only positions of these edges in the binary tree are shown. First, leaf nodes for the refined edges are created. Since the two nodes with largest keys (nodes 15 and 14 in our example) are siblings, their $s(E)$ values are used to create the node of their parent (node 7). Then the parent node for the two nodes with the next largest keys (nodes 10 and 11 in our example) are created and so on. The process is completed when the root node (with key 1) is created.

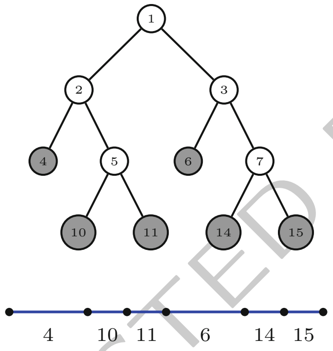

Fig. 1. Refinement tree associated with an original edge

Following the above procedure, we construct the interface data IPATH outlined in Table 2. Each interface edge, including those associated with internal nodes in refinement trees, is represented by a column with six entries in IPATH array. The first entry contains the index $r(E)$ if the edge is original (root) and zero otherwise. When edges from the two sides of the interface are matched, this entry is updated with the index of the corresponding edge. The second entry stores either the index of the edge's first child or its number in IBNDRY array (with minus sign) if it has no child. Sibling edges are put consecutively in IPATH array so storing the index for the second child edge is not necessary. Depending on the stage in the construction of IPATH array, the third and forth entries accommodate the indices of either edges, vertices or degrees of freedom of the two ends of the edge. The fifth entry is either the first or last (with minus sign) index of the interior degree(s) of freedom of the edge. This information together with the degree of the edge stored in the last entry are sufficient to recover all indices of the edge's interior degrees of freedom as they are numbered consecutively. The sign of the fifth entry indicates if they are increase or decrease along the counterclockwise traversal of the interface.

Table 2. Interface data structure: tree section

tree section				
type	root	root/leaf	internal	leaf
IPATH(1,*)	-l/n	-l/n	0/n	0/n
IPATH(2,*)	child	-e	child	-e
IPATH(3,*)	e1/v1/d1	v1/d1	e1/v1/d1	v1/d1
IPATH(4,*)	e2/v2/d2	v2/d2	e2/v2/d2	v2/d2
IPATH(5,*)	+-d	+-d	+-d	+-d
IPATH(6,*)	degree	degree	degree	degree
l=label, n=neighbor, e = edge k, v = vertex, d = dof				

3 Mesh Regularization

The regularization phase requires two all-to-all communication steps. The first describes the initial (non-conforming in *h* and *p*) interface system, and the second describes the final conforming system.

3.1 Data Reordering

At the beginning of the regularization step, each processor reorders its data structures. For processor I, edges, vertices and degrees of freedom on the interface between subregion I and the rest of the domain (fine interface) appear first in their respective arrays. These data are also arranged in a counterclockwise traversal of the interface to aid in the creation of the parallel interface data structure IPATH. Next, in all arrays, appears data corresponding to the interior of subregion I (fine interior); typically this is the majority of the data on processor I. Then appears data corresponding to the coarse part of the interface system on processor I (the interface not bounding region I). Finally appears data corresponding to the interiors of subregions other than I. Note that the first two blocks of this data (fine interface and fine interior) represent the contribution of processor I to the global fine mesh.

The parallel interface data structure IPATH is arranged in two sections; at the beginning is a pointer section with pointers for each processor's contribution to the fine interface system, and then two special sets of pointers, one for the local coarse interface system and one for the global fine mesh as a whole (see Table 3). The second section contains the tree data for individual edges on the interface system. After regularization, each processor has an IPATH array that contains complete data of the two-sided global fine interface system appended with data of local coarse interface system.

3.2 Fine Mesh Regularization

After reordering and a global exchange of interface data, each processor has complete information of the fine interface system. Then each process matches its interface edges against those of it neighbors. First original coarse edges are matched

Table 3. Interface data structure: pointer section

pointer section: 1→p+2	
IPATH(1,I)	first interface tree entry for subregion I
IPATH(2,I)	last interface tree entry for subregion I
IPATH(3,I)	first interface vertex/dof for subregion I
IPATH(4,I)	last interface vertex/dof for subregion I
$I = p+1$: pointers for local coarse system	
$I = p+2$: pointers for global fine system	

based on their labels. Then their descendants are matched following the refinement tree structures. We note here that for two neighboring processors, counterclockwise traversals of the interface are in opposite directions. An example of descendants of two original edges (from two different processors) is shown in Fig. 2.

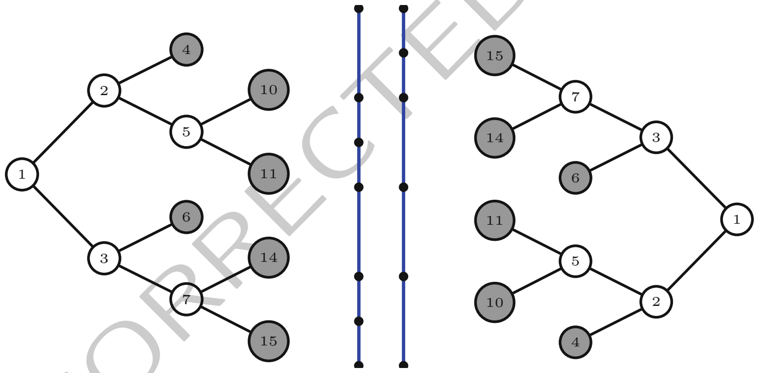

Fig. 2. Edge matching

When a pair of matching edges is determined, their first entries in IPATH are updated to store the indices (also in IPATH array) of their neighbors (change status from "−1" or "0" to "n" as in Table 2). If edges without corresponding neighbors are found, this indicates nonconformity in h. This is resolved by the processor with the less refined interface; it executes appropriate steps of h-refinement to make its interface match that of it neighbor. Although we must allow for arbitrary differences in refinement, it is typical to see at most one level of refinement difference on the fine portion of the interface. An example in Fig. 2 is edge 4 on the left that corresponds to edge 7 on the right with two child edges 14 and 15. In this case, edge 4 on the left will be h-refined one level.

When issues of h-conformity are resolved, the edges are re-examined to eliminate nonconformity in degree. Since the mesh is now h-conforming, each leaf edge on the fine interface system should have exactly one matching neighbor (from another processor). If the degrees of a matching pair are different, this nonconformity is resolved by the processor with the edge of lower degree; it executes appropriate steps of p-refinement in order to achieve the same degree as its neighbor on the interface edge. However, if red-green like refinement rules are applied as in [6], fixing the degree for one interface edge might also change the degree of another interface edge and cause further nonconformity. Thus, multiple communication steps might be required to eliminate nonconformity in degree. This issue was the main motivation for us to find a more flexible p-refinement algorithm and more general nodal basis functions for transition elements, allowing the mesh to be made both h and p conforming with just one communication step. Such approach is described in [5, 7].

When the global mesh is conforming, a second reordering as described above is carried out locally on each processor, followed by a second all-to-all broadcast of the new IPATH array. This time no nonconforming edges will be encountered during the matching process.

3.3 Coarse Mesh Regularization

The coarse part of the local mesh on processor I allows a complete conforming mesh of the whole domain on each processor, thus avoiding otherwise necessary communication steps. Due to constraints of shape regularity, the coarse mesh will typically be reasonably fine in areas near the fine subregion Ω_I and become more coarse in regions more distant from Ω_I. However, in some special situations such as having a singularity outside of Ω_I, the coarse mesh on processor I might be refined [8]. In very unusual circumstances, it is possible for the coarse mesh on some processors to be more refined (in h or in p) than the global fine mesh in some areas. Although this does not influence the global fine mesh solution directly, our DD solver assumes that the coarse mesh on each processor is not more refined than the global fine mesh, see [4, 9].

As described in Sect. 3.1, the IPATH array on each processor has a section for the coarse interface edges; this part of the data structure is local and different on every processor. Following the second and final broadcast of the IPATH data structure, each coarse interface edge is matched with one of the global fine edges. Here, the matching is one-way from a coarse edge to a fine edge only. Based on this type of matching, over-refined coarse edges are identified and then unrefined in either h or p.

We have also observed empirically [5, 9] that the convergence properties of our DD solver are enhanced when elements in the coarse regions having edges on the coarse interface system are more refined than those in the interior parts of the coarse region. To capture this effect, we also allow some limited refinement of elements lying along the coarse interface. The level of refinement on the interface boundary of Ω_J is determined by its distance from Ω_I; distance is measured in a graph in which the Ω_J correspond to vertices and the edge between Ω_I and Ω_J is present if and only if they have a shared interface boundary. The level of allowed refinement decays as 2^{-K}, where K is the distance from Ω_I to Ω_J.

Bibliography

[1] Randolph E. Bank. Some variants of the Bank-Holst parallel adaptive meshing paradigm. *Comput. Vis. Sci.*, 9(3):133–144, 2006.

[2] Randolph E. Bank and Michael Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM J. Sci. Comput.*, 22(4):1411–1443 (electronic), 2000.

[3] Randolph E. Bank and Michael Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Rev.*, 45(2):291–323 (electronic), 2003. Reprinted from SIAM J. Sci. Comput. **2**2 (2000), no. 4, 1411–1443 [MR1797889].

[4] Randolph E. Bank and Shaoying Lu. A domain decomposition solver for a parallel adaptive meshing paradigm. *SIAM J. Sci. Comput.*, 26(1):105–127 (electronic), 2004.

[5] Randolph E. Bank and Hieu Nguyen. Domain decomposition and *hp*-adaptive finite elements. In Yunqing Huang, Ralf Kornhuber, Olof Widlund, and Jinchao Xu, editors, *Domain Decomposition Methods in Science and Engineering XIX*, Lecture Notes in Computational Science and Engineering, pages 3–13, Berlin, 2011. Springer-Verlag.

[6] Hieu Nguyen. *p*-adaptive and automatic *hp*-adaptive finite element methods for elliptic partial differential equations Ph.D. Thesis, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2010.

[7] Randolph E. Bank and Hieu Nguyen. *hp* adaptive finite elements based on derivative recovery and superconvergence. Submitted.

[8] Randolph E. Bank and Jeffrey S. Ovall. Dual functions for a parallel adaptive method. *SIAM J. Sci. Comput.*, 29(4):1511–1524 (electronic), 2007.

[9] Randolph E. Bank and Panayot S. Vassilevski. Convergence analysis of a domain decomposition paradigm. *Comput. Vis. Sci.*, 11(4–6):333–350, 2008.