A DOMAIN DECOMPOSITION SOLVER FOR A PARALLEL ADAPTIVE MESHING PARADIGM

RANDOLPH E. BANK* AND SHAOYING LU†

Abstract. We describe a domain decomposition (DD) algorithm for use in the parallel adaptive meshing paradigm of Bank and Holst [3, 4]. Our algorithm has low communication, makes extensive use of existing sequential solvers, and exploits in several important ways data generated as part of the adaptive meshing paradigm. Numerical examples illustrate the effectiveness of the procedure.

Key words. Domain decomposition, Bank–Holst algorithm, parallel adaptive grid generation.

AMS subject classifications. 65N50, 65N30

1. Introduction. In this work, we describe a domain decomposition (DD) algorithm for use in the parallel adaptive meshing paradigm described in [3, 4]. The Bank-Holst paradigm provides a general approach to parallel adaptive meshing in which communication costs are kept low, and where sequential adaptive software (such as the software package PLTMG used in this work) can be employed without extensive recoding. This approach has three main components:

Step 1: Load Balancing. We solve a small problem on a coarse mesh, and use a posteriori error estimates to partition the mesh. Each subregion has approximately the same error, although subregions may vary considerably in terms of numbers of elements or gridpoints.

Step 2: Adaptive Meshing. Each processor is provided the complete coarse mesh and instructed to sequentially solve the *entire* problem, with the stipulation that its adaptive refinement should be limited largely to its own partition. The target number of elements and grid points for each problem is the same. Near the end of this step, the mesh is regularized such that the global mesh described in Step 3 will be conforming.

Step 3: DD Solve. A final mesh is computed using the union of the refined partitions provided by each processor. A final solution computed using a domain decomposition or parallel multigrid technique.

With this approach, the load balancing problem is reduced to the numerical solution of a small elliptic problem on a single processor, using a sequential adaptive solver such as PLTMG, without requiring any modifications to the sequential solver. The bulk of the calculation in the adaptive meshing step also takes place independently on each processor and can also be performed with a sequential solver with no modifications necessary for communication. The only parts of the calculation requiring communication are (1) the initial fan-out of the mesh distribution to the processors at the beginning of adaptive meshing step, once the decomposition is determined by

*Department of Mathematics University of California, San Diego La Jolla, California 92093-0112. email:rbank@ucsd.edu. The work of this author was supported by the National Science Foundation under contract DMS-0208449, and through the UCRP program at Lawrence Livermore National Laboratory. The UCSD Scicomp Beowulf cluster was built using funds provided by the National Science Foundation through SCREMS Grant 0112413, with matching funds from the University of California at San Diego.

†Department of Mathematics University of California, San Diego La Jolla, California 92093-0112. email:sylu@math.ucsd.edu. The work of this author was supported by the National Science Foundation under contract DMS-0208449, and through the UCRP program at Lawrence Livermore National Laboratory.

the error estimator in load balancing; (2) the mesh regularization, requiring communication to produce a global conforming mesh in DD solve step; and (3) the final solution phase, that might require local communication (e.g., boundary exchanges). In some circumstances, it might be useful to avoid the initial fan-out communication step by allowing all processors (which are otherwise idle) to simultaneously compute the coarse solution and load balance in Step 1. Note that a good initial guess for the DD solve is provided by the adaptive meshing step by taking the solution from each subregion restricted to its partition.

A more complete discussion of the overall paradigm as well as some numerical illustrations can be found in [3, 4]. In Mitchell [29], a parallel adaptive procedure similar to Step 2 of our procedures is described. See [33, 31, 17, 21] for some other approaches to parallel adaptive meshing. Our focus in this work is on Step 3 of the paradigm. Our domain decomposition solver is motivated by the Bank-Holst paradigm itself, in that it attempts to minimize communication and maximize the use of existing sequential software. Our algorithm is also designed to exploit as much as possible the wealth of data related to the solution generated by Step 2 of the process. For example, our solver uses the final regularized adaptive mesh on each subdomain as the basis of the parallel solve, and the linear systems solved on each processor are quite similar (except for the right hand side) to those solved in the final adaptive step. The sequential multigraph method [8] used by PLTMG is used to solve these linear systems.

Domain decomposition methods are now widely studied. See for example, [20, 19, 22, 23, 24, 16], the survey articles [15, 35, 36] and the book [32]. Our method is similar to those proposed and analyzed in [5, 6]. In particular, the method analyzed in [6] was shown to have a rate of convergence that is independent of N, the order of the global system of equations. However, it can depend on other factors, in particular the number of processors p. Our method differs from that of [6] in the construction of the right hand side, and in the solution update, which are non-standard for a domain decomposition method. Given the generality of the adaptive meshing procedures in PLTMG, we are also unable to exactly satisfy the assumptions on the mesh used in the convergence analysis in [6]. Intuitively, we expect our right hand side and update to improve upon the standard algorithm, and while we do not strictly satisfy the assumptions, we do largely satisfy them in spirit. Thus we expect our method to exhibit convergence rates almost independent of N, and we have observed this behavior in practice. However, there is some dependence on p, although this seems to be at worst logarithmic. See Mitchell [28, 30, 29] for an alternative approach based on a parallel multigrid solver.

We will informally refer to a processor's "fine grid" as the partition that is associated with the processor (Ω_i), despite the fact that it may not be uniformly fine. Similarly its "coarse grid" (on $\Omega - \Omega_i$) refers to the remainder of the domain, even though parts of the mesh outside of Ω_i are not uniformly coarse. As mentioned above, in the adaptive meshing paradigm, Step 2 of the process typically generates a very good initial guess for the solver in Step 3, namely an approximate solution composed of the fine grid parts of the solution generated by each of the processors. Along subdomain interfaces, this approximate solution in general will be multi-valued, even though the global mesh is conforming. In some sense the goal of the domain decomposition solver in this context is to resolve the discontinuities in this approximate solution. We also note that the initial guess already satisfies equations corresponding to interior mesh points in the global system (assuming the sequential solver in Step 2 of the paradigm did a good job of solving the linear systems generated in that phase of the algorithm). Thus in the global system of equations, we expect the residuals to generally be small everywhere except possibly for gridpoints associated with the system of subdomain interfaces.

In our algorithm, each processor contributes equations and unknowns corresponding to all its fine mesh points. From this we formally construct an expanded linear system. Interface unknowns corresponding to the same interface grid point have continuity enforced via a Lagrange multiplier. In this respect, the expanded global system can be viewed as arising from a special mortar element formulation [13, 12, 14, 34, 25] in which the mortar element space consists of Dirac δ functions or scaled hat functions centered at the interface nodes. Once this enlarged system is constructed, the Lagrange multipliers and duplicate unknowns are formally eliminated via block Gaussian elimination. The resulting Schur complement matrix is just the regular stiffness matrix for the conforming finite element space. The right hand side contains the conforming finite element right hand side, but it also has some "jump" terms that penalize the discontinuities along the interface. The details of this algorithm are given in Section 3.

The remainder of this manuscript is organized as follows. In Section 2, we discuss the interaction of the domain decomposition solver and adaptive mesh generation. In particular, we discuss small modifications of the refinement criteria that take into account the assumptions regarding the mesh given in [6]. We also describe some details of the global mesh regularization procedure that have an impact on the domain decomposition solver. In Section 3 we derive the domain decomposition solver. Our notation is mainly that of linear algebra, although most equations have analogs that could be expressed using finite element notation. In Section 4, we present some numerical examples. These problems are chosen to reflect a variety of applications that involve the adaptive solution of elliptic partial differential equations. The complete adaptive paradigm has been implemented in the PLTMG package, using MPI for communication.

The discussion in this manuscript is restricted to two dimensions. This is done mainly for convenience. The basic idea of the Bank-Holst paradigm has been applied equally well in three dimensions [3, 4]. The domain decomposition procedure described here is largely algebraic, and thus formally can be applied to three dimensional problems. However, one aspect that is specific to two dimensions is the mesh regularization procedure described in Section 2. For adaptive procedures employing a refined element tree data structure, mesh regularization in any space dimension is likely to be straightforward, since one can compare trees generated on each processor and adjust as necessary to insure a global conforming mesh. Without such a data structure, as in the PLTMG code used here, a more complicated regularization procedures such as the one described in Section 2 is needed. In three dimensions, the mesh regularization approach described here is problematic if triangular faces on the interface could be refined in incompatible ways on different processors. In such cases, our mortar element formulation can be generalized, and a DD solver based on the principles described here can be developed. See Lu [27] for some work in this direction.

2. Parallel Adaptive Methods and Domain Decomposition.

There are two aspects of the adaptive meshing paradigm that have significant consequences for our domain decomposition algorithm. First, our algorithm for grading the mesh away from the refined subdomain for a given processor is described. Second, the method used to make the global mesh conforming is given. Suppose the global domain Ω is partitioned into p nonoverlapping subdomains Ω_i, $1 \leq i \leq p$, such that

$$\Omega = \cup_i \overline{\Omega}_i,$$
$$\Omega_i \cap \Omega_j = \emptyset, \quad i \neq j.$$

Processor i receives all of Ω, but its adaptive refinement is confined mainly to $\overline{\Omega}_i$. However, some elements outside of Ω_i must also be refined, in order to grade the mesh between small elements in Ω_i and larger elements elsewhere in Ω while simultaneously controlling the shape regularity of the elements.

In the initial implementation of our algorithm, we simply multiplied a posteriori error estimates for elements outside of Ω_i by 10^{-6} so that the sequential adaptive refinement algorithms would be unlikely to choose those elements for refinement on the basis of their error estimate. Some elements not in Ω_i but near $\partial\Omega_i$ are also refined in order to insure a shape regular and conforming mesh on each processor. It seems certain that this initial suggestion, although quite simple and easy to implement, is probably not an optimal strategy. The issue of grading the mesh outside of Ω_i is presently an active area of research that will be reported elsewhere. Here we just summarize the main issues and how they influence our domain decomposition algorithm. We then describe the particular algorithm used in the experiments presented here.

Clearly, it is advantageous on processor i to confine the refinement as much as possible to Ω_i. Mesh points in Ω_i become mesh points in the final refined global conforming mesh and contribute in a direct way to the overall global solution. On the other hand, mesh points outside of Ω_i do not contribute directly to the overall global solution. To some extent, they can be viewed as a necessary overhead expense in the paradigm. Informally, we substitute local computation on these extra mesh points for interprocessor communication that would otherwise be required.

It is also important to note that the goal of Step 2 of the paradigm is adaptive mesh generation, *not* the computation of an accurate solution. Clearly adaptive meshing and solution accuracy are related, but it is not necessary to have an accurate solution to determine a good adaptive mesh. Indeed, it is the ability to generate good meshes from relatively inaccurate solutions that explains the success of many adaptive methods; see [9, 10] for an analysis and numerical examples of the a posteriori error estimation procedure used in this work. On the other hand, it is also clear that data outside of Ω_i does influence the solution within Ω_i. It is possible that such influence could be sufficiently strong to have a significant adverse effect on the adaptive mesh generation within Ω_i if it is not at least partly resolved. Potential examples are strong point singularities outside of Ω_i, or upstream flow in the case of PDE's involving convection. While it is not necessary for each processor to completely resolve such behavior in order to create a good mesh on its own subregion, it does seem important that each processor determine the approximate influence on the solution in its subregion, and then if necessary resolve it to an appropriate level.

With respect to our domain decomposition solver, the mesh outside of Ω_i plays an important role. Unlike typical domain decomposition algorithms, there is no special global coarse mesh subspace in our procedure. At each iteration, each processor solves a local problem on its final mesh on all of Ω, refined in Ω_i and coarse elsewhere. In effect, each processor has its own "built-in" global coarse mesh, which replaces the global coarse mesh found in other solvers. Thus the mesh outside of Ω_i should be sufficiently fine to play this role in our solver; as a practical matter, we expect that the meshes generated in Step 1 of the paradigm will be more than adequate for this purpose.

Another important point is related to the refinement of the mesh outside of Ω_i but very close to $\partial\Omega_i$. For example, in [5, 6], it is assumed that the refinement is extended to all elements with one or more vertices lying on $\partial\Omega_i$. Informally, one purpose of such assumptions is to insure that stiffness matrix elements corresponding to interface points on $\partial\Omega_i$ are the same as those in the global stiffness matrix of the overall global refined mesh. It is easy to see that if all elements having one or more vertices on $\partial\Omega_i$ correspond exactly to the global refined mesh, then this will be true. Thus there is incentive to extend the refined zone on processor i one or two layers outside $\partial\Omega_i$ to accommodate this aspect of the domain decomposition algorithm.

Of course some refinement of this type happens naturally in the process of grading element size in a controlled fashion in the transition between the small elements in Ω_i and larger elements elsewhere in Ω, but it is still useful to consider this as a separate point. This is especially true near so-called "cross points," mesh points lying on the boundaries of three or more subdomains. From the viewpoint of adaptive refinement on processor i, the main consideration is whether a given point is in $\overline{\Omega}_i$; parts of the global system of interfaces that do not include $\partial\Omega_i$ are of little significance. On the other hand, the entire global system of interfaces is important for the domain decomposition solver, and among interface points, cross points often require special attention. In our algorithm, cross points formally pose no special problems, but we have observed empirically that having the local mesh on processor i and the global refined mesh coincide as much as possible in the vicinity of cross points lying on $\partial\Omega_i$ is more important than for other interface points on $\partial\Omega_i$, in terms of the impact on the convergence rate of the domain decomposition algorithm.

As we see from the above discussion, there are many objectives and issues to be considered and balanced in determining the nature of the coarse mesh outside of Ω_i on processor i. Here we address this problem with a procedure that multiplies the a posteriori error estimate for an element t outside of Ω_i by a factor θ_t, $0 < \theta_t \le 1$. This is the same as our original suggestion if we chose $\theta_t = 10^{-6}$ for all t outside of Ω_i. In the present situation, however, we want $\theta_t = 1$ for elements near $\partial\Omega_i$, and then to have θ_t become small quickly but smoothly as the distance from Ω_i increases. The choice of θ_t also reflects the influence from the behavior of the solution outside of Ω_i.

In our algorithm, for $t \notin \Omega_i$, θ_t is generally given by

$$\theta_t = \frac{\omega_t}{2d_t}, \tag{2.1}$$

where ω_t comes from an "influence function" for Ω_i and d_t is a measure of distance from Ω_i. We consider each of these factors separately.

Assume that the bilinear form for the PDE (or its linearization) is given by $a(u, v)$. Let $\mathcal{V}^i$ denote the space of continuous piecewise linear polynomials associated with the existing global mesh on processor i (fine on Ω_i and coarse elsewhere). Let

$$\begin{aligned} \mathcal{V}_0^i &= \{\phi \in \mathcal{V}^i | \phi(x) = 0 \text{ for all } x \in \overline{\Omega}_i\}, \\ \mathcal{V}_1^i &= \{\phi \in \mathcal{V}^i | \phi(x) = 1 \text{ for all } x \in \overline{\Omega}_i\}. \end{aligned}$$

We consider the local dual problem: find $w \in \mathcal{V}_1^i$ such that

$$a(\phi, w) = 0 \tag{2.2}$$

for all $\phi \in \mathcal{V}_0^i$. Intuitively, $w = 1$ on $\overline{\Omega}_i$, and for $x \notin \overline{\Omega}_i$, $w(x)$ should give some indication of the influence of the solution near x on the solution in Ω_i. Generally, we expect w to decay towards zero as the distance to Ω_i increases, but the specific behavior depends on the details of the PDE and the physical location of Ω_i within Ω. We define ω_t by

$$\omega_t = \min(1, \max_{x \in t} |w(x)|). \tag{2.3}$$

Since $w \in \mathcal{V}_1^i$, ω_t is determined by examining only values at the vertices, and trivially $\omega_t = 1$ for $t \in \Omega_i$.

In terms of linear algebra, problem (2.2) involves the solution of a linear system involving a submatrix of the stiffness matrix. Assume we use the standard nodal basis for $\mathcal{V}^i$. Then the stiffness matrix A has the block 2×2 form

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix}$$

where A_{ff} corresponds to mesh points in $\overline{\Omega}_i$ and A_{cc} corresponds to mesh points strictly outside of Ω_i. As the adaptive refinement proceeds, we expect that the order of A_{ff} will be much larger than that of A_{cc}. The linear system corresponding to (2.2) is

$$A_{cc}^t W + A_{fc}^t e = 0$$

where superscript t stands for transpose, e is the vector of ones, and W is a vector of values of w at the coarse mesh vertices. This linear system is relatively inexpensive to assemble and solve, since we can leverage much of the effort used to assemble and solve linear systems involving the matrix A, required for computing the finite element solution.

We now consider the construction of the metric d_t. Informally, if the elements in Ω_i with vertices lying on $\partial\Omega_i$ are of size h, we want the first few layers of elements outside of Ω_i to also be of size h, and then the elements should grow in size to approach the large elements in most of Ω. For this reason, defining d_t in terms of some simple physical Euclidean metric would be undesirable. Rather, we need to define distances relative to the local value of h. A simple and efficient way to do this, assuming we control shape regularity of the elements, is to use a metric based on the triangulation itself. In particular, we can inductively define distances $\delta(v)$ for all vertices v in the mesh as follows. Initially all vertices have $\delta(v)$ undefined. Then for each vertex $v \in \overline{\Omega}_i$, we set $\delta(v) = 0$. Any vertex on any interface edge associated with a cross point on $\partial\Omega_i$ also has $\delta(v) = 0$ regardless of whether $v \in \overline{\Omega}_i$; the goal is to control coarse grid refinement more carefully in the vicinity of cross points lying on $\partial\Omega_i$. We then make a breadth-first search of the graph corresponding to the mesh, starting from all vertices with $\delta(v) = 0$. All unmarked vertices v' connected by an element edge to a vertex with $\delta(v) = 0$ are assigned $\delta(v') = 1$. Inductively, all unmarked vertices v' connected to a vertex with $\delta(v) = k$ are assigned $\delta(v') = k + 1$. Generally $\delta(v)$ measures the shortest path in the graph from the vertex v' to any vertex with $\delta(v) = 0$. The value of $\delta(v)$ is computed for each vertex at the beginning of each major adaptive step. $\delta(v)$ for any fixed vertex may increase at each major adaptive step as the mesh becomes more refined and its path length increases. This provides a smooth mesh-dependent behavior that is desirable to control refinement outside of Ω_i. For completeness, we remark that PLTMG also has options for mesh unrefinement, which could cause $\delta(v)$ to decrease, and moving mesh points, which leaves $\delta(v)$ invariant.

Let a given element t have vertices v_k, $1 \le k \le 3$. Then

$$d_t = \min_{1 \le k \le 3} \delta(v_k). \tag{2.4}$$

Finally for all elements with $d_t \le 1$ we set $\theta_t = 1$; otherwise, we define θ_t using (2.1). The goal of this weighting strategy is to produce a mesh where most of the refinement occurs in Ω_i, but to allow modest refinement as necessary outside of Ω_i to meet the multiple goals of the adaptive procedure itself, and the global domain decomposition solver used in Step 3 of the paradigm. If PLTMG used a refined element tree data structure (as it did in earlier versions) [7, 11], then forcing specified levels of refinement in various parts of the domain would be quite simple. On the other hand, we believe the greater flexibility afforded by less structured and more general adaptive algorithms in the present version of PLTMG more than compensate for the increased complexity of certain computations.

We now turn to the mesh regularization procedure, which produces the global conforming refined mesh used in the domain decomposition solver. In PLTMG, mesh refinement on interface edges is restricted to simple bisection, although our adaptive refinement procedure generally allows the mesh points to move. Our mesh regularization procedure has two substeps. Each step begins with interprocessor communication, where processor i exchanges information describing the interface edges on $\partial\Omega_i$. Each processor will ultimately create a data structure describing the complete system of interface edges in the global refined mesh. After the first communication step, edges in this global interface system will not necessarily match if there is more refinement in one side of the interface (see Figure 2.1, left).

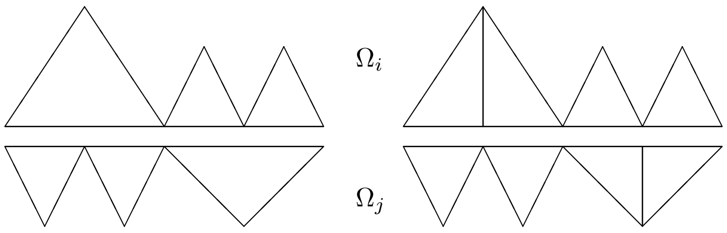

FIG. 2.1. *The coarse side of a non matching interface (left) is refined to make the global mesh conforming (right).*

Using information from its neighbors, each processor creates a mapping of its interface edges (those on $\partial\Omega_i$) to those of its neighbors. Less refined edges on its side of the interface are refined as necessary to be compatible with the neighbor. Less refined edges on the neighbor's side are refined by the neighbor. The boundary exchange and edge matching procedures are repeated, and this time all processors will succeed in matching all their interface edges to those of its neighbors (see Figure 2.1, right).

Following this second boundary exchange, each processor examines its interface system for all local interface edges *not* on $\partial\Omega_i$. These edges play no role at all in the final global conforming mesh, but they do play an important role in the domain decomposition solver. In particular, if some part of the local interface system on processor i is *more* refined than the global interface system, then those edges are coarsened to to make it compatible with the global interface system. Since all interface edges outside $\partial\Omega_i$ lie in the (assumed) coarsely refined part of Ω on processor i, the occurrence of such inconsistencies is rare. Nonetheless they do occur, and our domain decomposition solver assumes that vertices in the local interface system on processor i are a subset of those in the global conforming refined mesh (the sets of course coincide exactly on $\partial\Omega_i$). We also examine interface edges at cross points lying on $\partial\Omega_i$. We require all such edges correspond exactly to the global interface system. Edges lying on $\partial\Omega_i$ already satisfy this criterion, but edges at cross points on $\partial\Omega_i$ but not on $\partial\Omega_i$ may need to be refined. However, given the special definition of $\delta(v)$ used near such points, it is expected that the adaptive procedure itself makes violations rather rare. We note that both refinement and unrefinement of edges in the coarse part of the interface system can be done without communication with other processors.

Finally each processor constructs a mapping of all vertices on its local interface system to corresponding vertices in the global interface system. This is done without further interprocessor communication; it is deduced from edge matching in the global interface system following the second communication step. The resulting data structure forms the basis of the interprocessor communication steps in the domain decomposition solver.

3. A Domain Decomposition Method.

In this section we present a domain decomposition algorithm for solving the global conforming linear systems arising in Step 3 of the Bank-Holst paradigm. In many respects, this algorithm is motivated by and similar to the domain decomposition algorithms described in [5, 6]. Consistent with our overall philosophy, we wish to minimize communication and maximize the use of existing sequential software.

Formally, our domain decomposition procedure is an additive Schwartz method for solving the global saddle point system based on a special mortar element discretization. We construct local saddle point systems on each processor that precondition this global system. Instead of solving these local saddle point systems, we eliminate the Lagrange multipliers from each local system, producing linear systems with matrices similar to those assembled during the final adaptive refinement step. It is these linear systems that are solved in parallel. From these solutions, the fine mesh points of the local solutions are collected to form a global approximate solution of the original saddle system.

To simplify our discussion, initially we restrict attention to the case of just two subdomains. In our scheme, each subregion contributes equations corresponding to all fine mesh points, including its interface. Thus in general there will be multiple unknowns and equations in the global system corresponding to the interface grid points. This is handled by equality constraints that impose continuity at all mesh points on the interface. The result is a mortar-element like formulation, using Dirac δ functions for the mortar element space. With a proper ordering of unknowns, the global system of equations has the block 5×5 form

$$\begin{pmatrix} A_{11} & A_{1\gamma} & 0 & 0 & 0 \\ A_{\gamma 1} & A_{\gamma\gamma} & 0 & 0 & I \\ 0 & 0 & A_{\nu\nu} & A_{\nu 2} & -I \\ 0 & 0 & A_{2\nu} & A_{22} & 0 \\ 0 & I & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta U_\nu \\ \delta U_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma \\ R_\nu \\ R_2 \\ U_\nu - U_\gamma \end{pmatrix}. \tag{3.1}$$

Here A_{11} and A_{22} correspond to the fine grid points on processors 1 and 2, respectively, that are not on the interface, while $A_{\gamma\gamma}$ and $A_{\nu\nu}$ correspond to interface points. The fifth block equation imposes continuity, and its corresponding Lagrange multiplier is Λ. The identity matrix appears because the global fine mesh is conforming. The introduction of the Lagrange multiplier and the saddle point formulation (3.1) are only for expository purposes; indeed, Λ is never computed or updated.

On processor 1, we develop a similar but "local" saddle point formulation. That is, the fine mesh subregion on processor 1 is "mortared" to the remaining coarse mesh on processor 1. This leads to a linear system of the form

$$\begin{pmatrix} A_{11} & A_{1\gamma} & 0 & 0 & 0 \\ A_{\gamma 1} & A_{\gamma\gamma} & 0 & 0 & I \\ 0 & 0 & \bar{A}_{\nu\nu} & \bar{A}_{\nu 2} & -I \\ 0 & 0 & \bar{A}_{2\nu} & \bar{A}_{22} & 0 \\ 0 & I & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta \bar{U}_\nu \\ \delta \bar{U}_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma \\ R_\nu \\ 0 \\ U_\nu - U_\gamma \end{pmatrix}, \tag{3.2}$$

where quantities with a bar (e.g., $\bar{A}_{22}$) refer to the coarse mesh. A system similar to (3.2) can be derived for processor 2. With respect to the right-hand-side of (3.2), the interior residual R_1 and the interface residual R_γ are locally computed on processor 1. We obtain the boundary residual R_ν, and boundary solution U_ν from processor 2; processor 2 in turn must be sent R_γ and U_γ. The residual for the coarse grid interior points is set to zero. This is both a significant and natural assumption. It is significant because it avoids the need to obtain R_2 via communication, and to implement a procedure to restrict R_2 to the coarse mesh on processor 1. It is natural because given our initial guess, we anticipate that $R_1 \approx 0$ and $R_2 \approx 0$ at all iteration steps; if this is not the case, we expect that the convergence of the global iteration to be degraded. We remark that R_γ and R_ν are not generally small, but $R_\gamma + R_\nu \to 0$ at convergence. The last entry of the residual takes the form $U_\nu - U_\gamma$ to indicate that the computed updates δU_γ and $\delta \bar{U}_\nu$ satisfy the equation $\delta U_\gamma - \delta \bar{U}_\nu = U_\nu - U_\gamma$.

As with the global formulation (3.1), equation (3.2) is introduced mainly for exposition. The goal of the calculation on processor 1 is to compute the updates δU_1 and δU_γ, that contribute to the global conforming solution. To this end, we formally reorder (3.2) as

$$\begin{pmatrix} 0 & -I & 0 & I & 0 \\ -I & \bar{A}_{\nu\nu} & 0 & 0 & \bar{A}_{\nu 2} \\ 0 & 0 & A_{11} & A_{1\gamma} & 0 \\ I & 0 & A_{\gamma 1} & A_{\gamma\gamma} & 0 \\ 0 & \bar{A}_{2\nu} & 0 & 0 & \bar{A}_{22} \end{pmatrix} \begin{pmatrix} \Lambda \\ \delta \bar{U}_\nu \\ \delta U_1 \\ \delta U_\gamma \\ \delta \bar{U}_2 \end{pmatrix} = \begin{pmatrix} U_\nu - U_\gamma \\ R_\nu \\ R_1 \\ R_\gamma \\ 0 \end{pmatrix}. \tag{3.3}$$

Block elimination of the Lagrange multiplier Λ and $\delta \bar{U}_\nu$ in (3.3) leads to the block 3×3 Schur complement system

$$\begin{pmatrix} A_{11} & A_{1\gamma} & 0 \\ A_{\gamma 1} & A_{\gamma\gamma} + \bar{A}_{\nu\nu} & \bar{A}_{\nu 2} \\ 0 & \bar{A}_{2\nu} & \bar{A}_{22} \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta \bar{U}_2 \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma + R_\nu + \bar{A}_{\nu\nu}(U_\nu - U_\gamma) \\ \bar{A}_{2\nu}(U_\nu - U_\gamma) \end{pmatrix}. \tag{3.4}$$

The system matrix in (3.4) is the matrix used in the final adaptive refinement step on processor 1 (with possible modifications due to global fine mesh regularization). Thus the final matrix and mesh from Step 2 of the paradigm can be reused once again in the domain decomposition solver. In a sense Lagrange multipliers are introduced and then eliminated as an algebraic device to derive the right-hand-side of (3.4). Other than the right-hand-side, our algorithm is very similar to those analyzed in [5, 6]. To summarize, a single domain decomposition/multigraph iteration on processor 1 consists of:

1. locally compute R_1 and R_γ.
2. exchange boundary data (send R_γ and U_γ; receive R_ν and U_ν).
3. locally compute the right-hand-side of (3.4).
4. locally solve (3.4) via the multigraph iteration of [8].
5. update U_1 and U_γ using δU_1 and δU_γ.

We remark that any sequential solver (including a direct method) may be used to solve the local problems (3.4). The multigraph method used in PLTMG was chosen because it is robust for a wide class of differential equations, can handle the highly nonuniform meshes generated by our procedure, and empirically is observed to be nearly optimal in its complexity [8].

In its most simple form, the update step could be $U_1 \leftarrow U_1 + \delta U_1$, $U_\gamma \leftarrow U_\gamma + \delta U_\gamma$, which requires no communication. Standard acceleration procedures typically require some global communication to compute parameters. We remark that the global iteration matrix corresponding to this formulation is not symmetric, even if all local system matrices are symmetric. Thus conjugant gradient acceleration can not be used, although GMRES could be applied. In PLTMG, our solver is normally used in the context of an approximate Newton method (using just one DD iteration at each Newton step). A Newton line search technique that requires some global communication is used for the update step. In particular, in addition to computing several global norms and inner products, the line search produces the output for steps 1–3 above for the next Newton iteration.

When there are $p > 2$ subdomains, our algorithm remains much the same in outlook, but the description in matrix notation becomes much more complicated. We consider first the global system of equations. As before, each subregion contributes equations and unknowns corresponding to all its fine mesh points, including those on its interface. For interface points contained in two subdomains, typically the most common case, we have one constraint equation that equates solution values on both sides of the interface. At cross points contained in $\ell > 2$ subregions, we have $\ell - 1$ constraints that equate the ℓ different solution values associated with that point, e.g., $U_{i_1} = U_{i_2} = \cdots = U_{i_\ell}$. For each interface point with $\ell \geq 2$, we (arbitrarily) designate one of the values as the *master* value U_{i_m}, and the remaining *slave* values are given by $U_{i_s} = U_{i_m}$, $1 \leq s \leq \ell$, $s \neq m$.

We now consider the block 4×4 global saddle point problem given by

$$\begin{pmatrix} A_{ss} & A_{sm} & A_{si} & I \\ A_{ms} & A_{mm} & A_{mi} & -Z^t \\ A_{is} & A_{im} & A_{ii} & 0 \\ I & -Z & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta U_s \\ \delta U_m \\ \delta U_i \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_s \\ R_m \\ R_i \\ ZU_m - U_s \end{pmatrix}. \tag{3.5}$$

Here U_s refer to slave interface variables, U_m to master interface variables U_i to subregion interior variables and Λ to the Lagrange multipliers. The matrix A_{ii} can be ordered by subregion and will be block diagonal for such an ordering. Since several slave variables can be equated to a single master variable at cross points, the matrix Z will not generally be an identity matrix; however, each row of Z will be zero except for a single entry of 1.0 corresponding to a master variable.

As in the case of two subregions, we reorder (3.5) as

$$\begin{pmatrix} A_{ss} & I & A_{sm} & A_{si} \\ I & 0 & -Z & 0 \\ A_{ms} & -Z^t & A_{mm} & A_{mi} \\ A_{is} & 0 & A_{im} & A_{ii} \end{pmatrix} \begin{pmatrix} \delta U_s \\ \Lambda \\ \delta U_m \\ \delta U_i \end{pmatrix} = \begin{pmatrix} R_s \\ ZU_m - U_s \\ R_m \\ R_i \end{pmatrix}. \tag{3.6}$$

Block elimination of the slave variables and Lagrange multipliers leads to the Schur complement system

$$\begin{pmatrix} A_{mm} + A_{ms}Z + Z^tA_{sm} + Z^tA_{ss}Z & A_{mi} + Z^tA_{si} \\ A_{im} + A_{is}Z & A_{ii} \end{pmatrix} \begin{pmatrix} \delta U_m \\ \delta U_i \end{pmatrix} = \begin{pmatrix} R_m + Z^tR_s - (A_{ms} + Z^tA_{ss})(ZU_m - U_s) \\ R_i - A_{is}(ZU_m - U_s) \end{pmatrix}. \tag{3.7}$$

The matrix appearing on the left-hand-side of (3.7) is the global stiffness matrix corresponding to the conforming finite element approximation. The term $R_m + Z^tR_s$ appearing on the right-hand-side corresponds to the usual residual for the conforming finite element approximation, and is independent of the choice of slave and master variables. However, the "jump" terms involving $ZU_m - U_s$ on the right-hand-side of (3.7) do depend on the choice of master and slave variables.

We now consider the situation on a single processor, which we denote as processor k, $1 \le k \le p$. The problem solved on processor k again involves all p subregions. Subregion k has been refined, and its mesh corresponds to subregion k of the global conforming mesh. The remaining $p-1$ subregions on processor k typically have much coarser meshes than the global mesh. The saddle point problem of processor k has the form

$$\begin{pmatrix} \bar{A}_{ss} & \bar{A}_{sm} & \bar{A}_{si} & I \\ \bar{A}_{ms} & \bar{A}_{mm} & \bar{A}_{mi} & -\bar{Z}^t \\ \bar{A}_{is} & \bar{A}_{im} & \bar{A}_{ii} & 0 \\ I & -\bar{Z} & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta\bar{U}_s \\ \delta\bar{U}_m \\ \delta\bar{U}_i \\ \Lambda \end{pmatrix} = \begin{pmatrix} \bar{R}_s \\ \bar{R}_m \\ \bar{R}_i \\ \bar{Z}\bar{U}_m - \bar{U}_s \end{pmatrix}. \tag{3.8}$$

Here it is notationally cumbersome to distinguish between subregion k and the other subregions, so we simply use $\bar{A}_{ii}$ to refer to all the interior mesh points for all the regions. The part of $\bar{A}_{ii}$ arising from region k is exactly the same as in the global saddle point problem (3.5). Since the remaining parts correspond to coarse meshes, the overall order of $\bar{A}_{ii}$ is typically much smaller than A_{ii}. The residual $\bar{R}_i$ appearing on the right-hand-side of (3.8) is similar to the two subregion case; that is, for points lying in subregion k, it is the residual for the corresponding point in the global saddle point problem, and can be computed without communication on processor k. For points in the interior of the other coarser subregions, we set the residual to zero as in the two subdomain case. As before, this avoids communication and the need to develop an algorithm to restrict the global fine grid interior residuals to coarser meshes.

The interface equations are more interesting. The parts of the interface that involve subregion k correspond exactly to the global saddle point problem. The interface unknowns associated with subregion k are all designated as the master unknowns for their mesh points, since they must be computed and updated as part of the solution process on processor k. The remaining interface points, lying on the interface of two or more subregions other than k form a subset of the interface points of the global system. For these points we define the master and slave unknowns in an arbitrary fashion (in our code, we actually use an average; see below). The residuals $\bar{R}_m$ and $\bar{R}_s$ thus contain a subset of the elements in R_s and R_m in (3.5). Also, the interface solution vectors $\bar{U}_m$ and $\bar{U}_s$ contain of subset of the values in U_s and U_m in (3.5). The parts of R_m and R_s corresponding to subregion k are computed on processor k, and processor k sends these residuals and the parts of U_m and U_s corresponding to subregion k to all other processors. In turn, processor k receives all other fine grid interface residuals and interface solution values from all other processors. This is accomplished in an *all gather* exchange in MPI. Following this exchange, each processor has all the values in R_s, R_m, U_s and U_m, and from this information can extract the subset of information needed to form $\bar{R}_s$, $\bar{R}_m$, $\bar{U}_s$ and $\bar{U}_m$.

We note that the support of test functions in the coarse parts of the interface might be much larger than the corresponding fine grid basis functions used in computing the interface residuals. Thus simply using residuals computed using fine grid test functions for the coarse grid interface is formally inconsistent. On the other hand, our goal for the coarse grid interface points is only to provide some approximation of the fine grid right-hand-side in (3.5). The solution on processor k is used to update only unknowns corresponding to subregion k, and for these points the right-hand side is computed correctly. The less accurate approximation used elsewhere, similar to setting the residual to zero for coarse grid interior points, has proved sufficiently accurate in terms of producing accurate updates in the fine mesh region.

Block elimination of the slave variables and Lagrange multipliers in (3.8) leads to the Schur complement system

$$\begin{pmatrix} \bar{A}_{mm} + \bar{A}_{ms}\bar{Z} + \bar{Z}^t\bar{A}_{sm} + \bar{Z}^t\bar{A}_{ss}\bar{Z} & \bar{A}_{mi} + \bar{Z}^t\bar{A}_{si} \\ \bar{A}_{im} + \bar{A}_{is}\bar{Z} & \bar{A}_{ii} \end{pmatrix} \begin{pmatrix} \delta\bar{U}_m \\ \delta\bar{U}_i \end{pmatrix} = \begin{pmatrix} \bar{R}_m + \bar{Z}^t\bar{R}_s - (\bar{A}_{ms} + \bar{Z}^t\bar{A}_{ss})(\bar{Z}\bar{U}_m - \bar{U}_s) \\ \bar{R}_i - \bar{A}_{is}(\bar{Z}\bar{U}_m - \bar{U}_s) \end{pmatrix}. \tag{3.9}$$

As in the two subregion case, the matrix appearing on the left hand side of (3.9) is the matrix used in the final adaptive refinement step on processor k, with possible modifications due to global fine mesh regularization. The right-hand-side can be computed once the exchange of interface data is complete. As in the two processor case, the parts of $\delta\bar{U}_m$ and $\delta\bar{U}_i$ that correspond to subregion k are extracted from the solution of (3.9) and used to update the global solution.

We note that the choice of master and slave unknowns for points on the coarse parts of the interface on processor k is arbitrary. To resolve this ambiguity, in practice we take the master variable to be the *average* of all values that correspond to the interface point:

$$U_{i_m} \equiv \frac{1}{\ell}\sum_{s=1}^{\ell} U_{i_s}.$$

This is easy to do algorithmically, but awkward to describe in matrix notation. The effect is that the jump terms on the right-hand-side of (3.9) corresponding to coarse interface points are averaged over all choices of master variable. However, recall that for the interface points for subregion k, the master variable is always chosen to be the value from subregion k.

To summarize, a single domain decomposition/multigraph iteration on processor k consists of:

1. locally compute $\bar{R}_i$ and parts of R_s and R_m associated with subregion k.
2. exchange boundary data, obtaining the complete fine mesh interface vectors R_m, R_s, U_m and U_s.
3. locally compute the right-hand-side of (3.9) (using averages as described above).
4. locally solve (3.9) via the multigraph iteration.
5. update the fine grid solution for subregion k using the appropriate parts of $\delta\bar{U}_i$ and $\delta\bar{U}_m$.

We close this section with some discussion of convergence criteria. This is a delicate issue, and there are several points to consider. First, in each DD iteration each processor (simultaneously) solves a problem on its final adaptive mesh. Although these problems may be small in comparison with the linear system for the global fine mesh, they are still the largest problems solved on each processor, so every iteration will require a large fraction of the time used in Step 2 of the paradigm (since a large fraction of the cost in the entire adaptive procedure is due to the last refinement step in which the largest problem is assembled and solved). If a large number of iterations are used, then Step 3 of the paradigm can easily dominate the cost of the entire computation. Second, typically we have a very good initial guess; thus we expect the residuals to be generally small at all iterations; indeed much of the communication efficiency in the procedure relies on approximating the residual by zero for coarse mesh interior points. Third, the goal of the computation is to compute an approximate solution to the PDE, not an approximate solution to the linear system (of course the two are clearly related). Thus we should try to stop the iteration when we have a sufficiently accurate approximation to the PDE. Finally, we are likely to be dealing with very nonuniform meshes, and the norms used in the convergence criterion should take this into account.

We begin with a discussion of norms. Let G_i denote the diagonal entry of the mass matrix corresponding to vertex i,

$$G_i = \|\phi_i\|^2_{\mathcal{L}^2} \equiv \int_\Omega \phi_i^2 \, dx,$$

where ϕ_i is the usual nodal basis function associated with vertex i in the mesh. $G_i = O(h_i^2)$, where h_i is some measure of the size of elements sharing vertex i. Let $\mathcal{U}$ be a grid function; then

$$\|\mathcal{U}\|^2_G = \sum_i \mathcal{U}_i^2 G_i \tag{3.10}$$

With this weighting, formally $\|\mathcal{U}\|_G \sim \|u_h\|_{\mathcal{L}^2}$, where u_h is the finite element function corresponding to the grid function $\mathcal{U}$. Let $\mathcal{R}$ be a residual; then

$$\|\mathcal{R}\|^2_{G^{-1}} = \sum_i \mathcal{R}_i^2 G_i^{-1}. \tag{3.11}$$

With this weighting, intuitively $\|\mathcal{R}\|_{G^{-1}}$ looks like $\|e_h\|_{\mathcal{H}^2}$, where e_h is the error in the finite element solution and $\mathcal{H}^2$ is the usual Sobolev space. This must only be formal since generally $e_h \notin \mathcal{H}^2$.

Norms are computed with respect to the global fine mesh; each processor computes its contribution to the global norm (the contribution from vertices in $\overline{\Omega}_i$) and then a communication step is necessary to form the global norm. The main convergence criterion is

$$\frac{\|\delta\mathcal{U}^k\|_G}{\|\mathcal{U}^k\|_G} \le \max\left(\frac{\|\delta\mathcal{U}^0\|_G}{\|\mathcal{U}^0\|_G}, \frac{\|\nabla e_h\|_{\mathcal{L}^2}}{\|\nabla u_h\|_{\mathcal{L}^2}}\right) \times 10^{-1}. \tag{3.12}$$

Here $\mathcal{U}^k$ and $\delta\mathcal{U}^k$ are the global grid function and update, respectively, at iteration k, while $\|\nabla e_h\|_{\mathcal{L}^2}$ and $\|\nabla u_h\|_{\mathcal{L}^2}$ are the a posteriori error estimate and the initial solution (corresponding to grid function $\mathcal{U}^0$). In words, the iteration stops when the relative error in the solution is reduced by a factor of ten, or when the relative error in the solution of the linear system is a smaller by a factor of ten than the error in the PDE at the beginning of the iteration. The norm $\|\nabla e_h\|_{\mathcal{L}^2}$ appears instead of, e.g., $\|e_h\|_{\mathcal{L}^2}$ because it arises naturally in the context of a posteriori error estimation and it is the norm for which the strongest theoretical results are available. On the other hand, the use of different norms does introduce some inconsistency into (3.12). One could systematically replace $\|\cdot\|_G$ with $\|\nabla\cdot\|_{\mathcal{L}^2}$ at an increased computational cost in order to resolve the inconsistency should that prove necessary. It created no problems in the numerical experiments presented in this work. A secondary convergence criterion is

$$\frac{\|\mathcal{R}^k\|_{G^{-1}}}{\|\mathcal{R}^0\|_{G^{-1}}} \le 10^{-2}. \tag{3.13}$$

In our experiments, (3.12) was always satisfied before (3.13).

Finally, on each processor the multigraph iterative method was used to solve local problems of the form (3.9). The convergence for the multigraph iteration was

$$\frac{\|\overline{\mathcal{R}}^j\|_{\ell^2}}{\|\overline{\mathcal{R}}^0\|_{\ell^2}} \le 10^{-4}. \tag{3.14}$$

Here $\overline{\mathcal{R}}^j$ denotes the local residual at multigraph iteration j. The choice of $\|\cdot\|_{\ell^2}$ arose because the multigraph solver was part of a stand-alone package for solving linear systems [1] that was incorporated into PLTMG. As an algebraic multilevel method, it had no information about the linear system beyond the matrix and right hand side, and hence no basis to choose another norm. One could of course provide additional information and use another norm if necessary. The use of the more stringent tolerance 10^{-4} in (3.14) was to try to insure that the approximation of the interior residuals by zero at course grid points remained valid throughout the domain decomposition iteration.

4. Numerical examples.

In this section, we present some numerical results. Our examples were run on a small LINUX-based Beowulf cluster, consisting of 16 dual 1800 Athlon-CPU nodes with 2GB of memory each, a dual Athlon file server, and a 100Mbit CISCO 2950G Ethernet switch. This cluster runs the NPACI ROCKS version of LINUX (based on RedHat 7.1), and employs MPICH1.2.2 as its MPI implementation. The computational kernels of PLTMG are written in FORTRAN; the G77 compiler (version 2.96) was used in these experiments.

For our first example problem, we consider the simple Poisson equation

$$\begin{aligned} -\Delta u &= 1 && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

where Ω is a region in the shape of Lake Superior.

The second example problem is based on the one-dimensional Burger's equation. The differential equation is

$$-\epsilon\Delta u + u_y + uu_x = 0,$$

with $\epsilon = 10^{-3}$. If $\epsilon = 0$, this is the one dimensional Burger's equation with y playing the role of time. The domain Ω is the quarter circle of radius one. Homogeneous Neumann boundary conditions are applied along the circular arc, while Dirichlet boundary conditions are specified on the left side ($x = 0$) and the bottom ($y = 0$) as

$$u = \begin{cases} 1 & x = 0, \quad 0 \le y \le 1 \\ 1 & 0 \le x \le .25, \quad y = 0 \\ 1.5 - 2x & .25 \le x \le .75, \quad y = 0 \\ 0 & .75 \le x \le 1, \quad y = 0 \end{cases}.$$

This combination of boundary conditions gives rise to a solution similar to the so-called "λ shock" of Burger's equation. Newton's method is used to solve the systems of nonlinear equations. The multigraph method is used to solve the linear systems arising at each Newton step.

In our third test problem, we solve the linear elliptic problem

$$\begin{aligned} -a_1u_{xx} - a_2u_{yy} - f &= 0 && \text{in } \Omega, \\ (a_1u_x, a_2u_y) \cdot \mathbf{n} &= c - \alpha u && \text{on } \partial\Omega, \end{aligned}$$

where the piecewise constant values of the coefficients are given in Table 4.1. The domain Ω is shown in Figure 4.1, where the five subregions are denoted by color. We chose this problem because of its very anisotropic coefficients. The data for this problem was supplied by Leszek Demkowicz; the problem originated at Sandia National Laboratories, Albuquerque, New Mexico, and is a model for a battery [18, 26].

Region	a_1	a_2	f	side	c	α
1	25	25	0	left	0	0
2	7	0.8	1	top	1	3
3	5.0	10^{-4}	1	right	2	2
4	0.2	0.2	0	bottom	3	1
5	0.05	0.05	0			

TABLE 4.1
Coefficient definitions.

For our fourth example, we use PLTMG to solve the variational inequality

$$\min_{u \in K} \int_{\Omega} |\nabla u|^2 - 2f(x)u\,dx$$

where the domain $\Omega = (0,1) \times (0,1)$, and

$$K = \left\{ u \in \mathcal{H}_0^1(\Omega) : |u| \le \frac{1}{4} - \frac{1}{10}\sin(\pi x_1)\sin(\pi x_2) \ \text{ for } x \in \Omega \right\},$$
$$f(x) = -\Delta(\sin(3\pi x_1)\sin(3\pi x_2)).$$

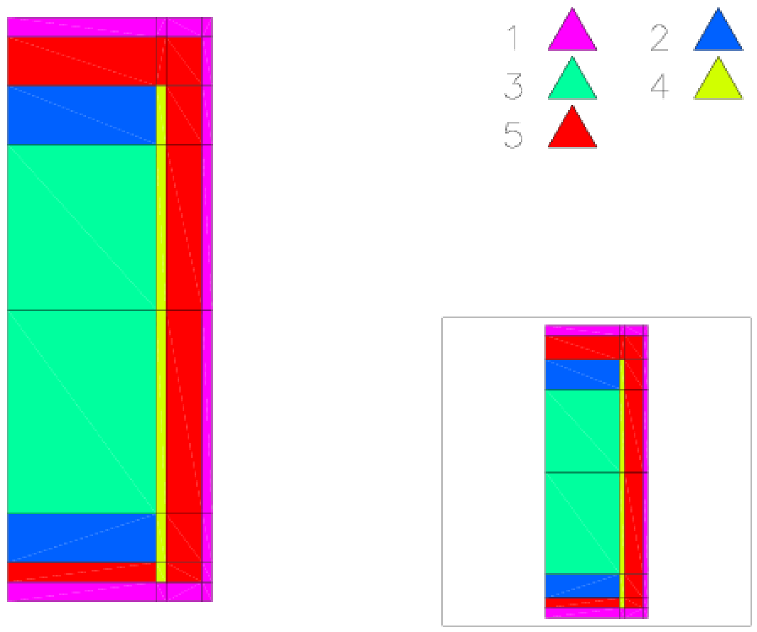

Fig. 4.1. *Domain for third test example. The tensor product partition is given by* $(0, 6.1, 6.5, 8, 8.4) \times (0, 0.8, 1.6, 3.6, 12, 18.8, 21.2, 23.2, 24)$.

In the absence of the obstacle, this is a simple elliptic equation with exact solution $u = \sin(3\pi x_1)\sin(3\pi x_2)$. This problem was solved using an interior point method. In this case, the interior point iteration systematically replaces simple linear system solves. However, the linear systems for each iteration step of the interior point method formally have the same structure as the unconstrained elliptic PDE, and are solved using the multigraph technique. See [2] for details.

The solutions for all test problems are shown in Figure 4.2. These problems were solved using our adaptive meshing paradigm for various values of p. During Step 1 of the paradigm, we began with an initial triangulation; the size varied between $N = 1685$ vertices for the Lake Superior domain to $N = 9$ vertices for the obstacle problem. For the first three problems, the initial mesh was generated by PLTMG from a description of the boundary $\partial\Omega$; for the obstacle problem we began with a uniform 3×3 mesh. These meshes were then adaptively refined (on a single processor) to create a mesh with $N_c = 8000$ vertices. This mesh was then partitioned into p subregions ($p = 2, 4, 8, 16, 32$) using a spectral bisection algorithm. This completed Step 1 of the paradigm. The load balance for the case $p = 32$ for each of the examples is shown in Figure 4.3. The a posteriori error estimation procedure used in PLTMG for adaptive refinement and the load balancing is described in [9, 10].

In Step 2 of the paradigm, the coarse mesh and solution were broadcast to all processors, and each processor independently continued with the adaptive refinement, creating an adaptively refined mesh with $N_p = 100000$ vertices, with most of the refinement concentrated in its own region. Prior to solving the final linear system with $N_p = 100000$, the global mesh was made conforming as described in Section 2. If the mesh were made conforming following this last solve, the initial interior residuals for the domain decomposition solve would generally be larger. (Because PLTMG is a

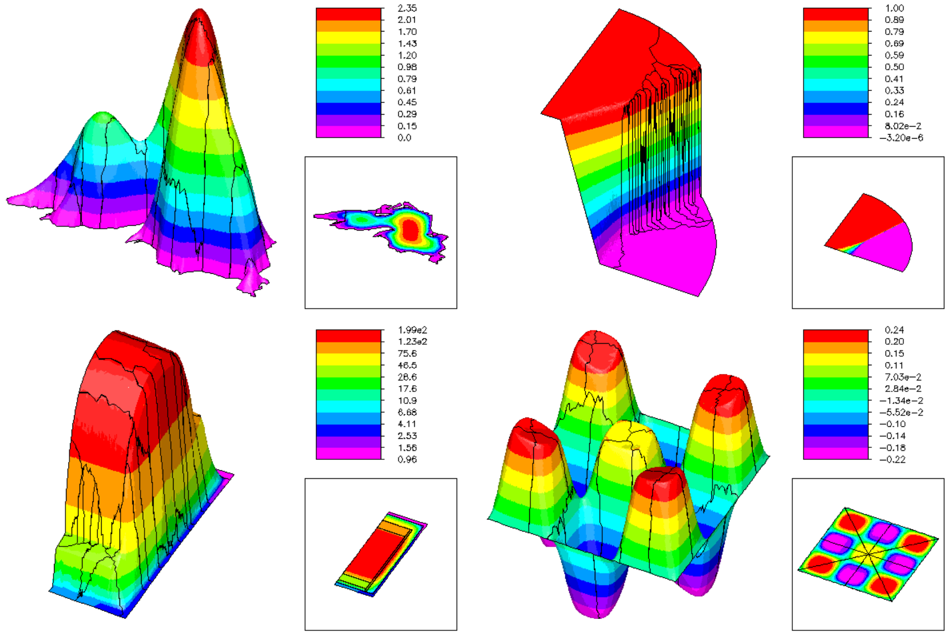

FIG. 4.2. *Solutions for the test problems. The lines denote the load balance for the case $p = 32$.*

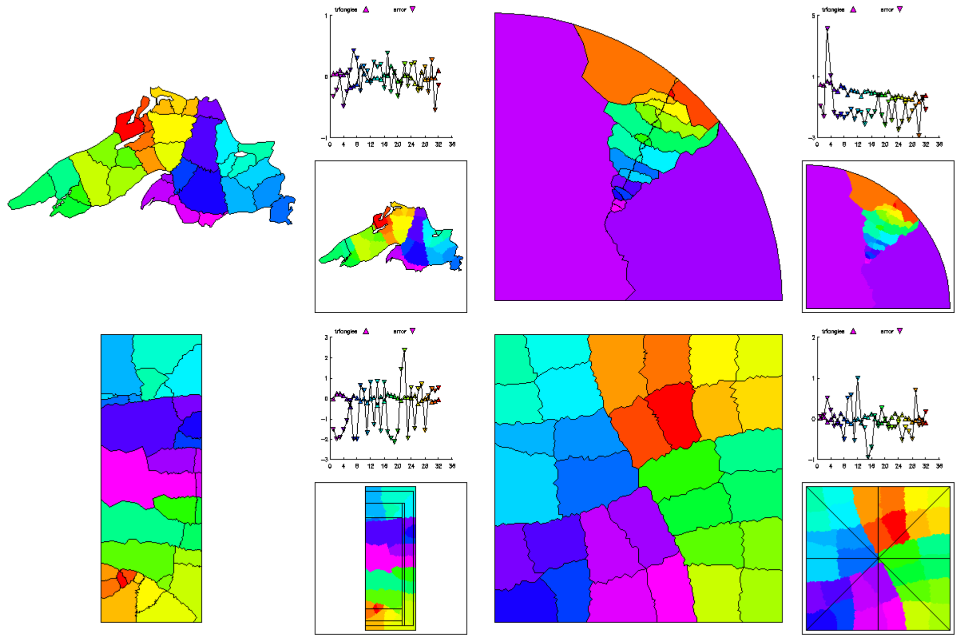

FIG. 4.3. *Load balance for the case $p = 32$. The legend in the upper right on each picture provides histograms showing the range of numbers of elements and a posterior error estimates for each region in the global conforming mesh.*

script driven program, this involved interchanging two command lines in the script, not any change in the program itself.) One could achieve essentially the same result by solving the local problem again after the global mesh was made conforming, but this would add significantly to the overall computation time. In Figure 4.4, we show the mesh density for the global fine mesh in the case $p = 32$. Since these global meshes each have several million elements, one can not draw individual elements. Thus each element is colored by size, and the edges are not drawn, yielding an image that shows the general refinement pattern. Most important, note that the overall refinement patterns seem reasonable given the nature of the problems. The shock in Burger's equations is highly refined as are the internal layers in the anisotropic problem. The contact zones in the obstacle problem are quite visible due to high refinement to resolve the free boundary and relatively low refinement in the centers of the contact zones. Another point to note is the relative smoothness of the mesh density across the subdomain interfaces; this is an a posteriori indication that the load balance computed in Step 1 of the paradigm was good.

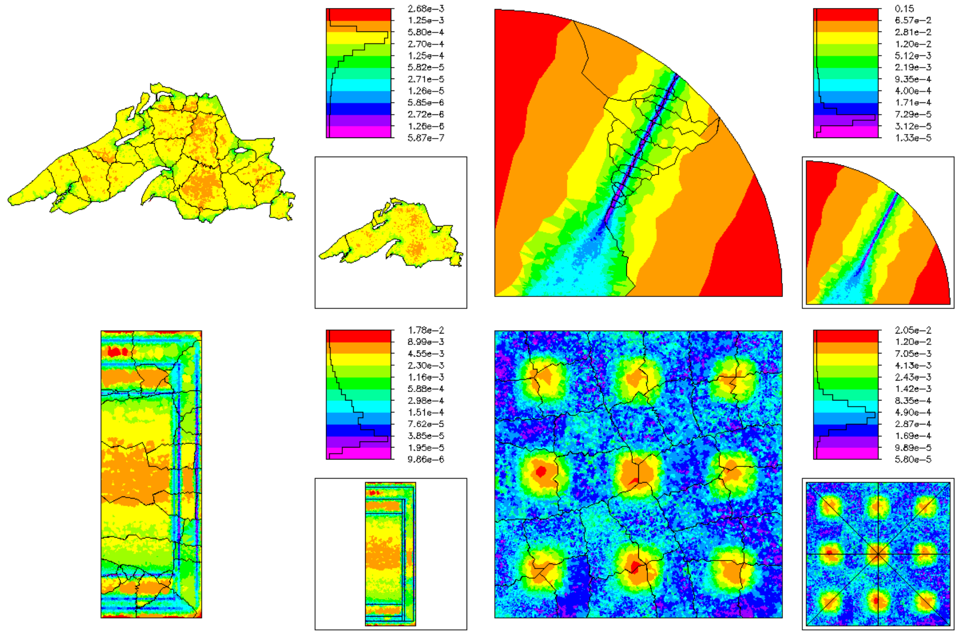

FIG. 4.4. *Mesh density for global refined mesh for the case* $p = 32$.

In Step 3 of the paradigm, a global system of linear equations of the form (3.5) was solved using the domain decomposition algorithm described in Section 3. The starting guess derived from the fine grid parts of the solution provided by the p processors. For the nonlinear problem (second example) the DD solver was used in a standard Newton iteration, while for the obstacle problem, it was embedded in an interior point iteration.

In Figure 4.5, we plot a posterior error estimates computed on the global refined mesh following Step 3 of the paradigm. Here we see that the overall procedure did a reasonable job of equilibrating the error. Even for Burger's equation, the error within the "λ-shock" is generally equilibrated (reflected by the uniformity of color). Error outside the shock is of course much smaller, but relatively little refinement was done outside of the shock. Here again, these error estimates provide some a posteriori indication of the quality of the initial load balance.

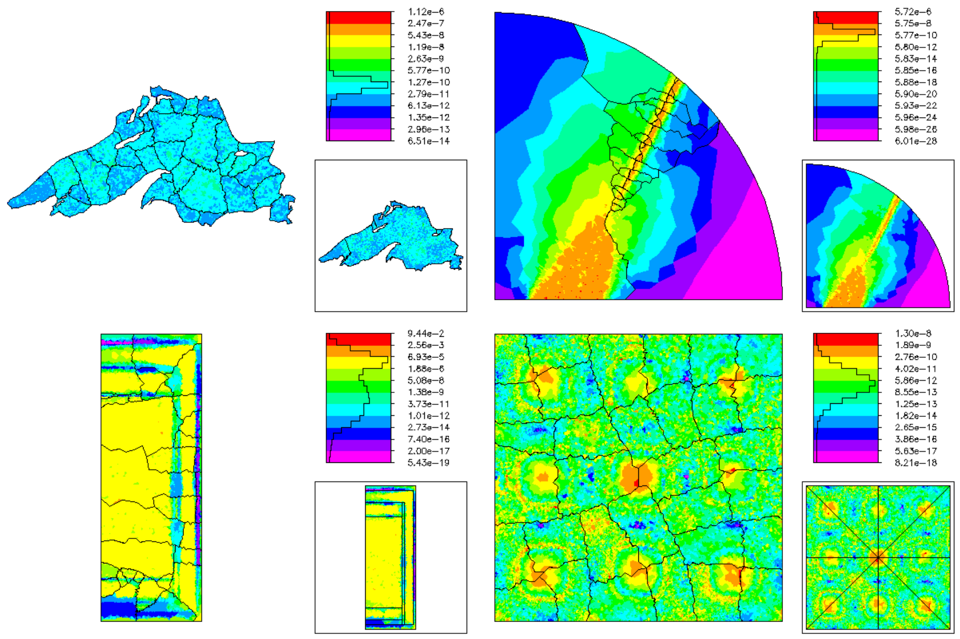

Fig. 4.5. *A posterior error estimates for the global refined mesh for the case $p = 32$.*

In Table 4.2, we provide a summary of the calculation. The global value of N can be approximated by the formula [3]

$$N \approx pN_p - (p-1)N_c, \tag{4.1}$$

where N_c is the size of the original coarse mesh ($N_c = 8000$ in this example) and N_p is the target value used by each processor in Step 2 of the parallel adaptive paradigm ($N_p = 100000$ in this example). As described in Section 2, each processor refines some elements outside of its subdomain Ω_i, to maintain shape regularity in the transition between small elements within its subregion and generally larger elements in the remainder of Ω, to control "pollution" and other effects coming from the PDE outside of Ω_i, and to generally satisfy requirements imposed to make the domain decomposition solver efficient. All of this implies that generally

$$N < pN_p - (p-1)N_c.$$

For example, when $p = 32$, equation (4.1) predicts $N \approx 2952000$ when the actual values ranged between $N = 2480831$ and $N = 1591742$. Because of the hyperbolic nature of the Burger's equation example, it is not surprising that many processors did a lot of refinement on the "upstream" side of Ω_i in order to produce a good mesh within Ω_i. In Figure 4.6, we illustrate the influence functions for typical subdomains in the Lake Superior and the Burger's equation example. In the case of the Poisson equation, the influence function decays relatively quickly outside of the given region. In contrast, for Burger's equation there is little or no decay in the upstream direction. In the upstream direction, the θ_t of (2.1) were reduced mainly due to the distance factor d_t. Thus there was substantially more refinement outside of Ω_i for many of the subregions for this problem, resulting in a the smaller size of the global fine mesh.

TABLE 4.2
p is the number of processors, N the number of vertices in the global fine mesh, DD the number of domain decomposition iterations. The breakpoints are accumulated execution times, in seconds. The range of times for all processors for Step 2 and Step 3 appear in parentheses.

p	N	DD	Breakpoints		
			End of Step 1	End of Step 2	End of Step 3
Poisson Equation					
2	188672	1	2.2	46.6 (43.3-49.8)	54.2 (52.4-56.0)
4	359736	1	2.8	45.4 (43.7-47.4)	54.0 (51.8-58.5)
8	683620	2	3.2	46.5 (44.0-50.7)	62.5 (58.6-69.5)
16	1304403	2	3.5	47.2 (41.5-52.3)	64.1 (58.6-67.6)
32	2480831	3	3.8	50.3 (48.9-54.1)	77.3 (70.8-83.5)
Burger's Equation					
2	188604	2	3.2	50.7 (47.1-54.4)	72.5 (67.5-77.5)
4	351027	2	4.0	55.8 (51.7-59.5)	76.1 (72.1-81.9)
8	623372	2	4.6	63.9 (52.5-75.0)	89.1 (74.2-100.9)
16	1034663	2	5.1	66.3 (55.0-76.3)	90.5 (76.3-102.2)
32	1591742	3	5.5	64.6 (51.4-83.2)	104.8 (81.2-130.0)
Anisotropic Equation					
2	190152	1	2.3	42.0 (41.5-42.6)	52.6 (52.4-52.7)
4	351765	1	3.0	43.6 (42.8-44.8)	50.0 (48.1-51.6)
8	662754	1	3.5	43.5 (41.3-46.1)	50.1 (46.3-53.8)
16	1196489	1	3.9	44.1 (40.2-47.7)	50.0 (45.5-56.1)
32	2127832	2	4.3	46.8 (41.9-52.9)	59.8 (53.6-66.9)
Obstacle Problem					
2	185972	1	3.5	60.8 (60.6-61.1)	69.6 (67.6-71.6)
4	355046	1	4.1	61.8 (58.3-64.1)	72.6 (69.8-75.2)
8	672753	1	4.6	61.6 (57.2-65.8)	73.1 (70.7-78.1)
16	1292638	1	5.1	61.1 (52.2-71.2)	72.2 (65.4-79.3)
32	2473078	1	5.4	63.0 (58.0-71.8)	75.7 (69.0-85.0)

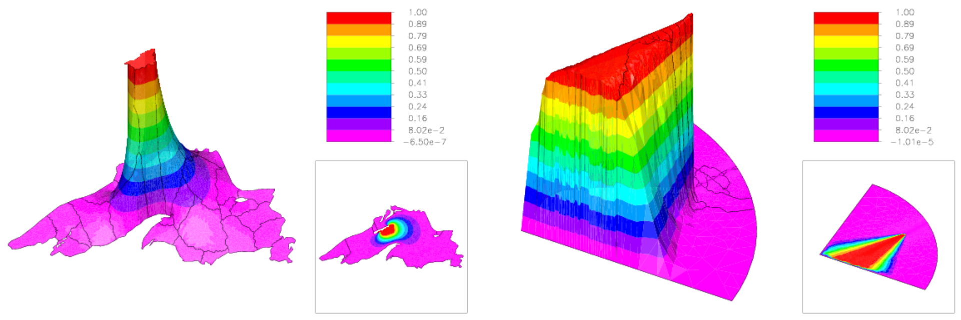

FIG. 4.6. *Typical influence functions for a single subdomain.*

In Table 4.2 we see that the number of domain decomposition iterations is quite stable over the range of this set of problems. The method analyzed in [6] has a rate of convergence independent on N. Our method differs from that in [6] mainly in the definition of the right-hand-side and in the update procedures. Also, our method only approximately satisfies the assumptions in [6] regarding mesh coarsening outside of Ω_i. The convergence rate here appears to show some logarithmic dependence on p, which is probably due to the increasing size and complexity of the global interface system as p increases. In Figure 4.7 we show the initial residuals for the global fine mesh for the Poisson and Burger's equations. From these images, it is clear that the biggest residuals correspond to points on the interfaces $\partial\Omega_i$. This shows that setting the residual to zero at coarse interior points is a reasonable approximation, at least for these classes of problems.

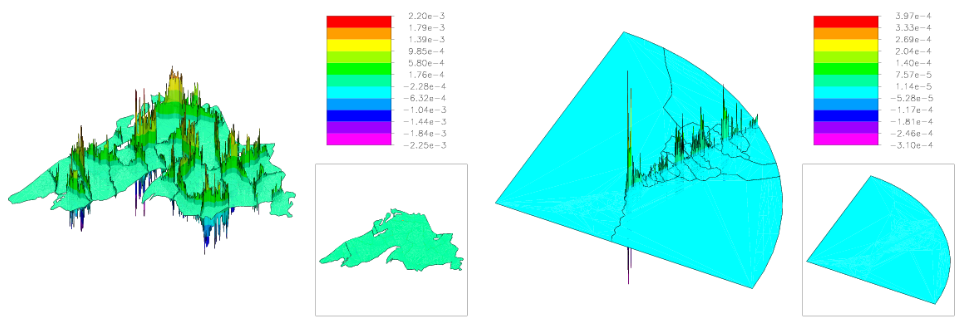

FIG. 4.7. *The initial fine mesh residual from (3.7) for Step 3 of adaptive paradigm for the case* $p = 32$.

In Table 4.2 we also record average execution times at various breakpoints in the computation. The computations done in Step 1 were done on only one processor, but the time was charged equally and fully to all processors. The time used for Step 1 increases slowly as a function of p. In all the runs, all the computations involved were the same in Step 1, except for the load balance, so the increase in time can be attributed in full to the increasing number of eigenvalue problems solved in the recursive spectral bisection algorithm as p increased. However, in terms of the the overall computation, Step 1 has very minimal impact, so this is not a large effect. The cost of Step 2 was very stable over the entire range of p, although there was some increase. This was mainly due to the the increasing cost of reconciling the mesh as p increased. This illustrates good scalability and near perfect speedup. In addition, the execution times of different processors only vary in a small range in Step 2, indicating good load balance from mesh partition. Step 3 refers to the domain decomposition solve using the procedure described in Section 3. Here we see some increase in time with increasing p. This seems to reflect mostly the increasing number of domain decomposition iterations required. Also, the range of times increases in some cases; this reflects differing numbers of multigraph iterations used on different processors to satisfy (3.14).

To better illustrate the convergence behavior of the method, in Table 4.3, we present some results for the Poisson equation for three target values N_p and different numbers of processors. This allows us to see more clearly the dependence of the rate of convergence on N and p. In this case we used the modified convergence criteria

$$\frac{\|\delta\mathcal{U}^k\|_G}{\|\mathcal{U}^k\|_G} \le \frac{\|\delta\mathcal{U}^0\|_G}{\|\mathcal{U}^0\|_G} \times 10^{-3}. \tag{4.2}$$

This results in discrete solutions with errors on the order of $O(10^{-6})$–$O(10^{-7})$. This is well below the approximation error, but it shows the effectiveness of our method in terms of solving the linear system. We also remark that taking $N_p = 25000$ with $N_c = 8000$ is very inefficient for the parallel solution. Even $N_p = 50000$ could be considered inefficient. However, our goal here was to provide several values of N

for each value of p, and these choices provided a simple way to achieve this limited objective. The results show some logarithmic-like dependence of the convergence rate on p, but only a very weak dependence on N.

TABLE 4.3
p is the number of processors, N_p is the target size of for the mesh on each processor, N the number of vertices in the global fine mesh, and DD the number of domain decomposition iterations required to satisfy $\|\delta\mathcal{U}^k\|_G/\|\mathcal{U}^k\|_G \le 10^{-3}\|\delta\mathcal{U}^0\|_G/\|\mathcal{U}^0\|_G$.

N_p	$p=2$		$p=4$		$p=8$		$p=16$		$p=32$	
	N	DD	N	DD	N	DD	N	DD	N	DD
25000	40928	2	66949	5	119946	6	205385	7	344216	9
50000	90015	3	165433	5	302946	6	556735	8	1012872	10
100000	188672	3	359736	5	683620	6	1304403	9	2480831	10

Acknowledgment. We would like to thank the referees, whose thoughtful comments greatly improved the manuscript.

REFERENCES

[1] R. E. BANK, *Multigraph users' guide - version 1.0*, tech. report, Department of Mathematics, University of California at San Diego, 2001.
[2] R. E. BANK, P. E. GILL, AND R. F. MARCIA, *Interior methods for a class of elliptic variational inequalities*, in Proceedings of the First Sandia Workshop on Large-scale PDE Constrained Optimization, to appear.
[3] R. E. BANK AND M. J. HOLST, *A new paradigm for parallel adaptive meshing algorithms*, SIAM J. on Scientific Computing, 22 (2000), pp. 1411–1443.
[4] ———, *A new paradigm for parallel adaptive meshing algorithms*, SIAM Review, 45 (2003), pp. 292–323.
[5] R. E. BANK AND P. K. JIMACK, *A new parallel domain decomposition method for the adaptive finite element solution of elliptic partial differential equations*, Concurrency and Computation: Practice and Experience, 13 (2001), pp. 327–350.
[6] R. E. BANK, P. K. JIMACK, S. A. NADEEM, AND S. V. NEPOMNYASCHIKH, *A weakly overlapping domain decomposition preconditioner for the finite element solution of elliptic partial differential equations*, SIAM J. on Scientific Computing, 23 (2002), pp. 1817–1841.
[7] R. E. BANK, A. H. SHERMAN, AND A. WEISER, *Refinement algorithms and data structures for regular local mesh refinement*, in Scientific Computing (Applications of Mathematics and Computing to the Physical Sciences) (R. S. Stepleman, ed.), North Holland, 1983, pp. 3–17.
[8] R. E. BANK AND R. K. SMITH, *An algebraic multilevel multigraph algorithm*, SIAM J. on Scientific Computing, 25 (2002), pp. 1572–1592.
[9] R. E. BANK AND J. XU, *Asymptotically exact a posteriori error estimators, part I: Grids with superconvergence*, SIAM J. Numerical Analysis, (to appear).
[10] ———, *Asymptotically exact a posteriori error estimators, part II: General unstructured grids*, SIAM J. Numerical Analysis, (to appear).
[11] M. W. BEALL AND M. S. SHEPHARD, *A general topology-based mesh data structure*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 1573–1596.
[12] F. BELGACEM, *The mortar finite element method with Lagrange multipliers*, 1997. Preprint.
[13] C. BERNARDI, Y. MADAY, AND A. PATERA, *A new nonconforming approach to domain decomposition: the mortar element method*, in Nonlinear partial differential equations and their applications, H. B. adn J.L. Lions, ed., Pitman Research Notes in Mathematics, New York, 1994, John Wiley and Sons, pp. 13–51.
[14] D. BRAESS, W. DAHMEN, AND C. WEINERS, *A multigrid algorithm for the mortar finite element method*, SIAM J. on Numerical Analysis, 37 (1999), pp. 48–69.
[15] T. CHAN AND T. MATTHEW, *Domain decomposition algorithms*, in Acta Numerica, Cambridge University Press, 1994, pp. 61–143.
[16] T. F. CHAN AND J. ZOU, *A convergence theory of multilevel additive Schwarz methods on unstructured meshes*, Numerical Algorithms, 13 (1996), pp. 365–398.
[17] H. L. DECOUGNY, K. D. DEVINE, J. E. FLAHERTY, R. M. LOY, C. OZTURAN, AND M. S. SHEP-

HARD, *Load balancing for the parallel adaptive solution of partial differential equations*, Appl. Num. Math., 16 (1994), pp. 157–182.

[18] D. DOBRANICH, *Advances in modeling thermally-activated batteries*, tech. report, Sandia National Laboratories internal memo, Albuquerque, New mexico, 1995.

[19] M. DRYJA AND O. B. WIDLUND, *Some domain decomposition algorithms for elliptic problems*, in Iterative Methods for Large Linear Systems, Boston, 1990, Academic Press, pp. 273–291.

[20] M. DRYJA AND O. B. WIDLUND, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Third International Symposium on Domain Decomposition Methods, T. F. C. et al, ed., Philadelphia, 1990, SIAM, pp. 3–21.

[21] J. E. FLAHERTY, R. M. LOY, C. OZTURAN, M. S. SHEPHARD, B. K. SZYMANSKI, J. D. TERESCO, AND L. H. ZIANTZ, *Parallel structures and dynamic load balancing for adaptive finite element computation*, Appl. Num. Math., 26 (1998), pp. 241–263.

[22] G. HAASE, U. LANGER, A. MEYER, AND S. V. NEPOMNYASCHIKH, *Hierarchical extension operators and local multigrid methods in domain decomposition preconditioners*, East-West J. Numer. Math., 2 (1994), pp. 172–193.

[23] G. HAASE AND S. V. NEPOMNYASCHIKH, *Explicit extension operators on hierarchical grids*, East-West J. Numer. Math., 5 (1997), pp. 231–248.

[24] D. C. HODGSON AND P. K. JIMACK, *A domain decomposition preconditioner for a parallel finite element solver on distributed unstructured grids*, Parallel Computing, 23 (1997), pp. 1157–1181.

[25] J. HUANG AND J. ZOU, *A mortar element method for elliptic problems with discontinuous coefficients*, IMA J. Numer. Anal., 22 (2002), pp. 549–576.

[26] R. R. LOBER, *Thermal battery life test problem for residual method error measure development*, tech. report, Sandia National Laboratories internal memo, Albuquerque, New mexico, 1999.

[27] S. LU, *Parallel Adaptive Multigrid Algorithms*, PhD thesis, Department of Mathematics, University of California at San Diego, 2004.

[28] W. MITCHELL, *The full domain partition approach to distributing adaptive grids*, Applied Numerical Mathematics, 26 (1998), pp. 265–275.

[29] ———, *The full domain partition approach to parallel adaptive refinement*, in Grid Generation and Adaptive Algorithms, IMA Volumes in Mathematics and its Applications, Springer-Verlag, Heidelberg, 1998, pp. 152–162.

[30] ———, *A parallel multigrid method using the full domain partition*, Electronic Transactions on Numerical Analysis, 6 (1998), pp. 224–233.

[31] P. M. SELWOOD, M. BERZINS, AND P. M. DEW, *3D parallel mesh adaptivity : Data structures and algorithms*, in Parallel Processing for Scientific Computing, Philadelphia, 1997, SIAM.

[32] B. SMITH, P. BJORSTAD, AND W. GROPP, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[33] C. WALSHAW AND M. BERZINS, *Dynamic load balancing for pde solvers on adaptive unstructured meshes*, Concurrency: Practice and Experience, 7 (1995), pp. 17–28.

[34] B. I. WOHLMUTH, *A mortar finite element method using dual spaces for the Lagrange multiplier*, SIAM J. Numer. Anal., 38 (2000), pp. 989–1012 (electronic).

[35] J. XU, *Iterative methods by space decomposition and subspace correction*, SIAM Review, 34 (1992), pp. 581–613.

[36] J. XU AND J. ZOU, *Some nonoverlapping domain decomposition methods*, SIAM Review, 40 (1998), pp. 857–914.