# A NEW PARADIGM FOR PARALLEL ADAPTIVE MESHING ALGORITHMS

RANDOLPH E. BANK* AND MICHAEL HOLST†

**Abstract.** We present a new approach to the use of parallel computers with adaptive finite element methods. This approach addresses the load balancing problem in a new way, requiring far less communication than current approaches. It also allows existing sequential adaptive PDE codes such as PLTMG and MC to run in a parallel environment without a large investment in recoding. In this new 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, without requiring any modifications to the sequential solver. The small elliptic problem is used to produce *a posteriori* error estimates to predict future element densities in the mesh, which are then used in a weighted recursive spectral bisection of the initial mesh. The bulk of the calculation then takes place independently on each processor, with no communication, using possibly the same sequential adaptive solver. Each processor adapts its region of the mesh independently, and a nearly load-balanced mesh distribution is usually obtained as a result of the initial weighted spectral bisection. Only the initial fan-out of the mesh decomposition to the processors requires communication. Two additional steps requiring boundary exchange communication may be employed after the individual processors reach an adapted solution, namely the construction of a global conforming mesh from the independent subproblems, followed by a final smoothing phase using the subdomain solutions as an initial guess. We present a series of convincing numerical experiments which illustrate the effectiveness of this approach. The justification of the initial refinement prediction step, as well as the justification of skipping the two communication-intensive steps, is supported by some recent [37] and not so recent [23, 28, 29] results on local *a priori* and *a posteriori* error estimation.

**Key words.** adaptivity, finite element methods, a posteriori error estimation, parallel computing

**AMS subject classifications.** 65M55, 65N55

**1. Introduction.** One of the most difficult obstacles to overcome in making effective use of parallel computers for adaptive finite element codes such as PLTMG [2] and MC [15] is the load balancing problem. As an adaptive method adjusts the mesh according to the features of the solution, elements in some areas are refined, whereas others are not. If an initial mesh is distributed quite fairly among a number of processors, a very good error estimator (coupled with adaptive refinement) quickly produces a very bad work load imbalance among the processors.

A number of static and dynamic load balancing approaches for unstructured meshes have been proposed in the literature [12, 13, 14, 17, 30, 33]; most of the dynamic strategies involve repeated application of a particular static strategy. One of the difficulties in all of these approaches is the amount of communication that must be performed both to assess the current load imbalance severity, and to redistribute the work among the processors once the imbalance is detected and an improved distribution is calculated. The calculation of the improved work distribution can be quite inexpensive (such as geometric or inertia tensor-based methods), or it may be a costly procedure, with some approaches requiring the solution of an associated eigenvalue problem or evolution of a heat equation to near equilibrium [34]. These calculations may themselves require communication if they must be solved in parallel using the existing (poor) distribution.

In recent years, clusters of fast workstations have replaced the more traditional parallel computer of the past. While this type of parallel computer is now within reach of an organization with even a modest hardware budget, it is usually difficult to produce an efficient parallel implementation of an elliptic PDE solver; this is simply due to the fact that elliptic continuum mechanics problems necessarily lead to tightly coupled discrete problems, requiring substantial amounts of communication for their solution. The load balancing problem is also more pronounced on workstation clusters: even at 100 Mbit/sec speed, the cluster communication speeds are quite slow compared to modern workstation CPU performance, and the communication required to detect and correct load imbalances results in severe time penalties.

**1.1. A new approach to parallel adaptive finite element methods.** In this work, we present an alternative approach which addresses the load balancing problem in a new way, requiring far less communication than current approaches. This approach also allows existing sequential adaptive PDE codes such as PLTMG and MC to run in a parallel environment without a large investment in recoding.

Our approach has three main components:

*Department of Mathematics, University of California at San Diego, La Jolla, CA 92093. The work of this author was supported by the National Science Foundation under contract DMS-9706090.

†Department of Mathematics, University of California at San Diego, La Jolla, CA 92093. The work of this author was supported by the National Science Foundation under CAREER Award 9875856.

1. 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 grid points.
2. 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 gridpoints for each problem is the same.
3. A final mesh is computed using the union of the refined partitions provided by each processor. This mesh is regularized and a final solution computed, using a standard domain decomposition or parallel multigrid technique.

The above approach has several interesting features. First, the load balancing problem (step 1) is reduced to the numerical solution of a small elliptic problem on a single processor, using a sequential adaptive solver such as PLTMG or MC, without requiring any modifications to the sequential solver. Second, the bulk of the calculation (step 2) takes place independently on each processor, and can also be performed with a sequential solver such as PLTMG or MC with no modifications necessary for communication. (In PLTMG, one line of code was added, which artificially multiplied *a posteriori* error estimates for elements outside a processor's partition by $10^{-6}$. In MC, two lines were added to prevent elements outside the processor's partition from entering the initial refinement queue.) Step 2 was motivated by recent work of Mitchell [20, 21, 22] on parallel multigrid methods. A similar approach appeared recently in [10]. The use of *a posteriori* error estimates in mesh partitioning strategies has also been considered in [26].

The only parts of the calculation requiring communication are (1) the initial fan-out of the mesh distribution to the processors, once the decomposition is determined by the error estimator, (2) the mesh regularization, requiring local communication to produce a global conforming mesh, and (3) the final solution phase, which might require local communication (boundary exchanges). Note that a good initial guess for step 3 is provided in step 2 by taking the solution from each subregion restricted to its partition. Note also that the initial communication step to fan-out the mesh decomposition information is not actually required, since each processor can compute the decomposition independently (with no communication) as a pre-processing step.

**1.2. Justification.** Perhaps the largest issue arising in connection with this procedure is whether it is well founded, particularly in light of the continuous dependence of the solution of an elliptic equation on data throughout the domain. To address this issue, we first note that the primary goal of step 2 above is adaptive mesh generation. In other words, the most important issue is not how accurately the problem is solved in step 2 of this procedure, but rather the quality of the (composite) adaptively generated mesh. These two issues are obviously related, but one should note that it is not necessary to have an accurate solution in order to generate a well adapted mesh. Indeed, the ability to generate good meshes from relatively inaccurate solutions explains the success of many adaptive methods.

A secondary goal of step 2 is the generation of an initial guess for the solution on the final composite mesh. This aspect of the algorithm will be addressed in Section 4. Here we focus on the primary issue of grid generation, and in particular on *a posteriori* error estimates, as such estimates provide the link between the computed solution and the adaptive meshing procedure. Here we consider in detail the schemes used in PLTMG and MC, but similar points can be made in connection with other adaptive algorithms.

PLTMG uses a discretization based on continuous piecewise linear triangular finite elements. The error is approximated in the subspace of discontinuous piecewise quadratic polynomials that are zero at the vertices of the mesh. In particular, let $u - u_h$ denote the error and let $t$ denote a generic triangle in the mesh. In its adaptive algorithms, PLTMG approximates the error in triangle $t$ using the formula

$$\|\nabla(u - u_h)\|_t^2 \equiv \int_t |\nabla(u - u_h)|^2 \, dx \approx v^t B v \tag{1.1}$$

where (see Figure 1.1)

$$\nu_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \qquad \text{for } 1 \le i \le 3, \tag{1.2}$$

$$\ell_i = \nu_j - \nu_k \qquad \text{for } (i,j,k) \text{ a cyclic permutation of } (1,2,3), \tag{1.3}$$

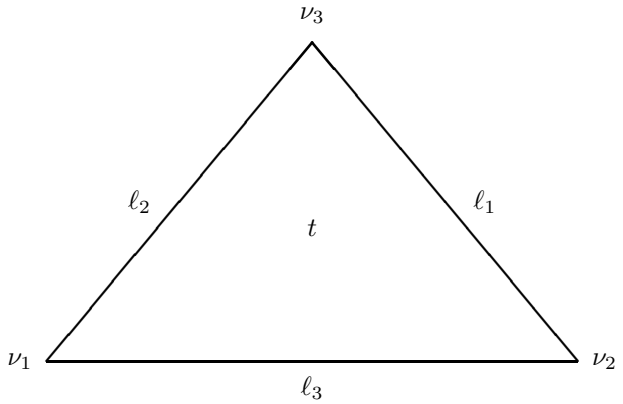



FIG. 1.1. *A typical element.*

$$v = \begin{pmatrix} \ell_1^t M_t \ell_1 \\ \ell_2^t M_t \ell_2 \\ \ell_3^t M_t \ell_3 \end{pmatrix}, \qquad M_t = -\frac{1}{2} \begin{pmatrix} u_{xx} & u_{xy} \\ u_{xy} & u_{yy} \end{pmatrix} \tag{1.4}$$

$$B = \frac{1}{48|t|} \begin{pmatrix} \ell_1^t \ell_1 + \ell_2^t \ell_2 + \ell_3^t \ell_3 & 2\ell_1^t \ell_2 & 2\ell_1^t \ell_3 \\ 2\ell_2^t \ell_1 & \ell_1^t \ell_1 + \ell_2^t \ell_2 + \ell_3^t \ell_3 & 2\ell_2^t \ell_3 \\ 2\ell_3^t \ell_1 & 2\ell_3^t \ell_2 & \ell_1^t \ell_1 + \ell_2^t \ell_2 + \ell_3^t \ell_3 \end{pmatrix}. \tag{1.5}$$

Equations (1.1)-(1.5) are derived by comparing the approximation error for linear and quadratic interpolation on $t$. See [2, 4] for details. The second derivatives in the $2 \times 2$ matrix $M_t$ are taken as constant on $t$. The values of the second derivatives are extracted as a post-processing step from the *a posteriori* error estimates. The remaining information needed to compute the the right hand side of (1.1) is generated directly from the geometry of element $t$.

To be effective, the approximation of the derivatives need not be extremely accurate. Many adaptive algorithms, and in particular those in PLTMG, are directed towards creating meshes in which the errors in all elements are equilibrated. Typically, adaptive algorithms develop refined meshes starting from rather coarse meshes. For reasons of efficiency, often many elements are refined between recomputing the approximate solution $u_h$.

MC also discretizes the solution over piecewise linear triangular or tetrahedral elements, and as in PLTMG, error estimates for each element are produced by solving a local problem using the edge-based quadratic bump functions. However, while these local problems involve inverting $3 \times 3$ matrices for scalar problems in 2D, the local problems are substantially more costly in 3D. In particular, for 3D elasticity, the local problems require the inversion of $18 \times 18$ matrices (6 bump functions and 3 unknowns per spatial point). Therefore, MC also provides an alternative less-expensive error estimator, namely the residual of the strong form of the equation, following e.g. [32]. In this paper, the numerical results involving MC are produced using the residual-based estimator.

While there is considerable theoretical support for the *a posteriori* error bounds which form the foundation for adaptive algorithms, (see for example the book of Verfürth [32] and its references) the adaptive algorithms themselves are largely heuristic, in particular those aspects described above. However, there is a large and growing body of empirical evidence that such algorithms are indeed robust and effective for a wide class of problems. In particular, they are effective on coarse meshes, and on highly nonuniform meshes. The types of meshes likely to be generated in our parallel algorithm are qualitatively not very different from typical meshes where *a posteriori* error estimates are known to perform quite well.

In our procedure, we artificially set the errors to be very small in regions outside the subregion assigned to a given processor, so the standard refinement procedure is "tricked" into equilibrating the error on just one subregion. Since the target size of all problems solved in step 2 is the same, and each subregion initially has approximately equal error, we expect the final composite mesh to have approximately equal errors, and approximately equal numbers of elements, in each of the refined subregions created in step 2. That is, the

composite mesh created in step 3 should have roughly equilibrated errors in all of its elements. This last statement is really just an expectation, since we control only the target number of elements added in each subregion, and do not control the level of error directly. This and other assumptions forming the foundation of our load balancing algorithms are discussed in more detail in Section 3.2.

We note the standard adaptive procedures in PLTMG and MC have additional refinement criteria to insure conforming and shape regular meshes. Thus some elements outside the given subregion but near its interface are typically refined in order to enforce shape regularity; the result is a smooth transition from the small elements of the refined region to larger elements in the remainder of the domain. If the target number of elements is large in comparison with the number of elements on the coarse mesh, this should be a relatively small effect.

To summarize, we expect that for any given problem, our algorithm should perform comparably to the standard algorithm applied to the same initial mesh in terms of the quality of the adaptive local mesh refinement.

**2. Examples.** In this section, we present some simple examples of the algorithm presented in Section 1.

**2.1. A Convection-Diffusion Equation.** In this example, we use PLTMG to solve the convection-diffusion equation

$$\begin{aligned} -\nabla \cdot (\nabla u + \beta u) &= 1 \qquad \text{in } \Omega \\ u &= 0 \qquad \text{on } \partial\Omega \end{aligned} \tag{2.1}$$

where $\beta = (0, 10^5)^t$, and $\Omega$ is the region depicted in Figure 2.1. This coarse triangulation has 2148 elements and 1368 vertices.

We partitioned the domain into four subregions with approximately equal error using the recursive spectral bisection algorithm, described in more detail in Section 3. Then four independent problems were solved, each starting from the coarse grid and coarse grid solution. In each case the mesh is adaptively refined until a mesh with approximately 6000 unknowns (located at triangle vertices) is obtained. These mesh for one subdomain and the corresponding solution are shown in Figure 2.2. Notice that the refinement is largely confined to the given region, but some refinement in adjacent regions is needed in order to maintain shape regularity. We emphasize that these four problems are solved independently, by a the standard sequential adaptive solver PLTMG. The only change to the code used for problem $k$, $(1 \le k \le 4)$ was to multiply *a posteriori* error estimates for elements in regions $j \ne k$ by $10^{-6}$, causing the adaptive refinement procedure to rarely choose these elements for refinement, except to maintain shape regularity.

The meshes from these four subproblems are combined to form a globally refined mesh with 37575 triangles and 19828 vertices. This mesh is shown in Figure 2.3.

The solutions from the four problems are combined to form a global solution that serves as initial guess for a global smoothing process. The final solution, as well as the *a posteriori* error estimates for this solution, are shown in Figure 2.4.

**2.2. A Full Potential Flow Problem.** For our second example, we use PLTMG to solve the full potential flow equation

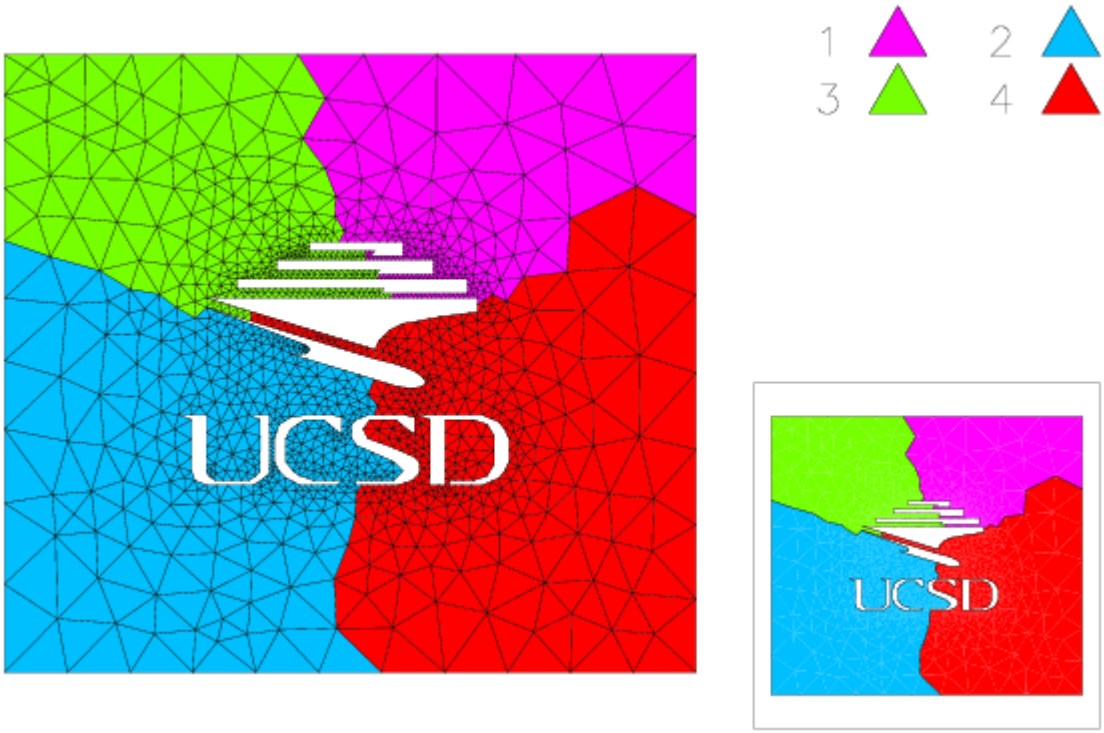
$$\begin{aligned} -\nabla \cdot \rho(\nabla u)\nabla u &= 0 \qquad \text{in } \Omega \\ \rho \partial u/\partial n &= g \qquad \text{on } \partial\Omega_1 \\ \rho \partial u/\partial n &= 0 \qquad \text{on } \partial\Omega_2 \end{aligned} \tag{2.2}$$

where

$$\rho(\nabla u) = (1 - u_x^2 - u_y^2)^{\frac{1}{\gamma - 1}}$$

and $\gamma = 1.4$. $\partial\Omega_1$ is the outer boundary, where the solution assumes its asymptotic behavior, while $\partial\Omega_2$ is the inner boundary, in this example the profile of a Naca0012 airfoil. The initial mesh, generated from the geometry description, had 384 triangles and 221 vertices. In Figure 2.5 we illustrate the initial mesh and a detail of the local Mach number $M(\nabla u)$ given by

$$M(\nabla u) = \sqrt{\frac{2c}{\gamma - 1}},$$




*The initial triangulation, partitioned into four subregions with approximately equal error.*

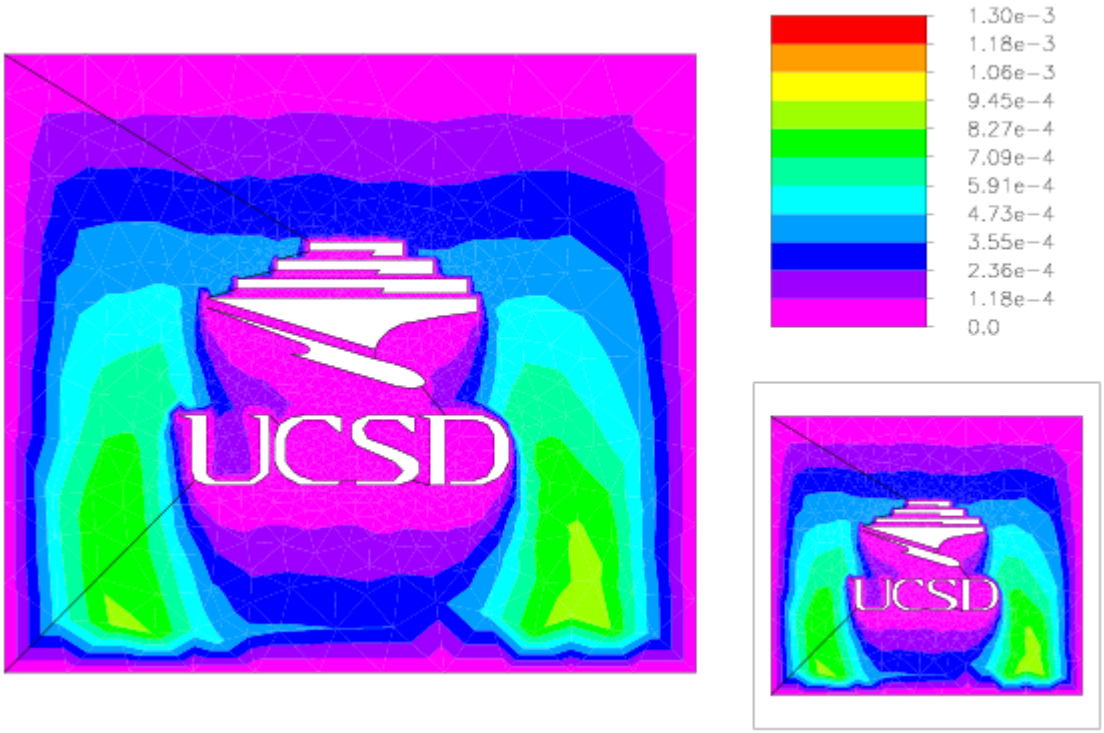
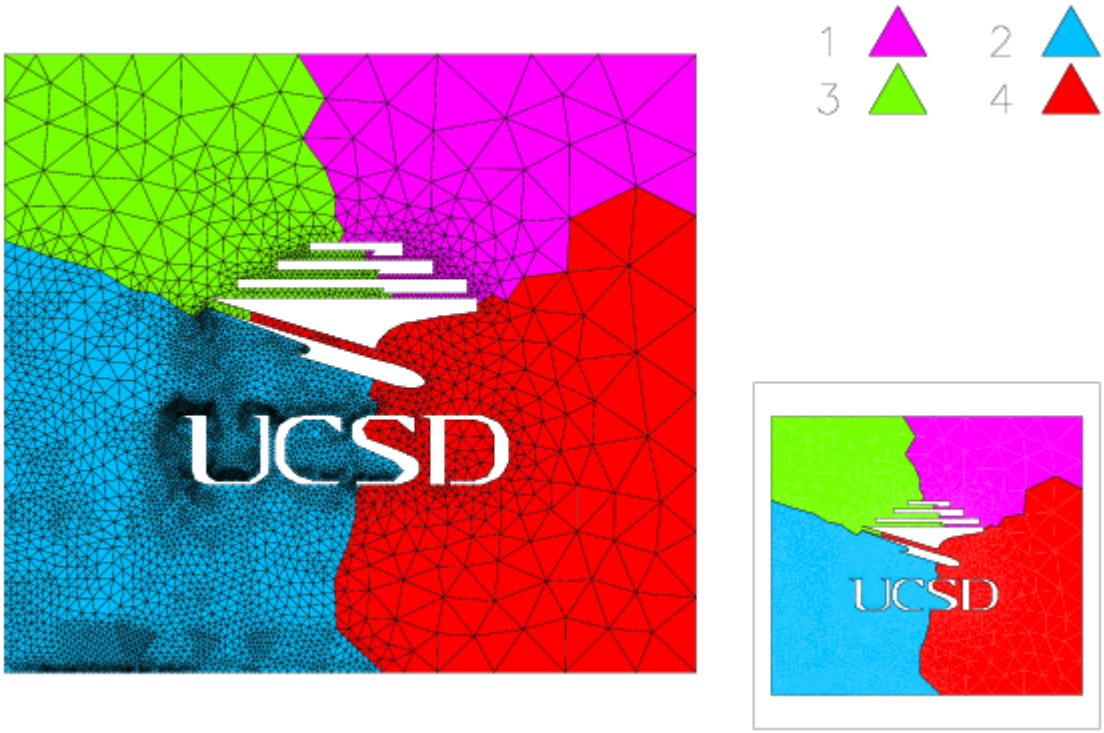



FIG. 2.1. *The coarse grid solution.*




*The refined mesh for problem 2.*

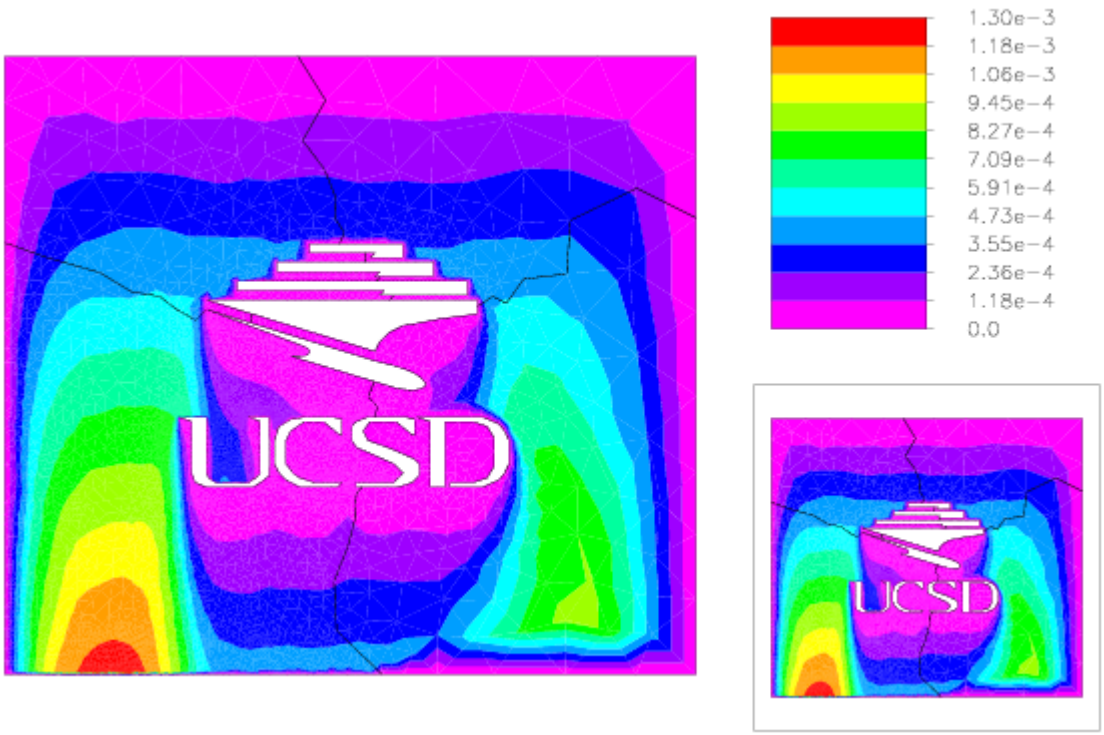
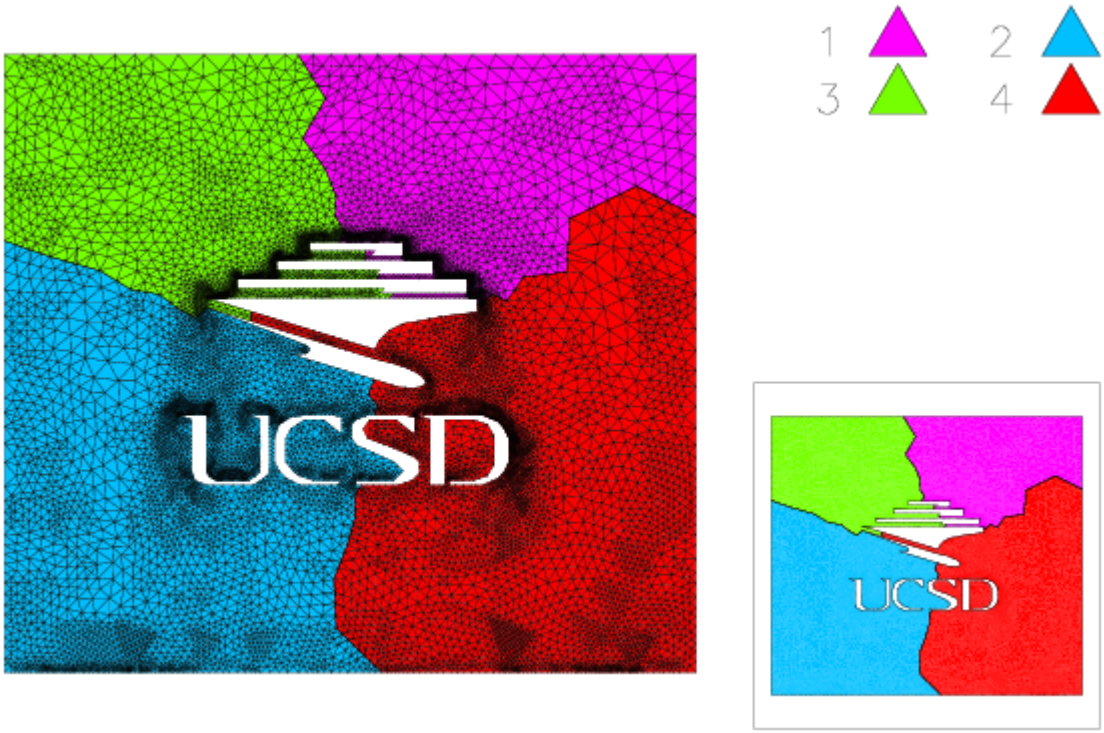



FIG. 2.2. *The solution for problem 2.*




*The global refined mesh.*


FIG. 2.3. *The initial guess for the global solution.*


*The final global solution.*

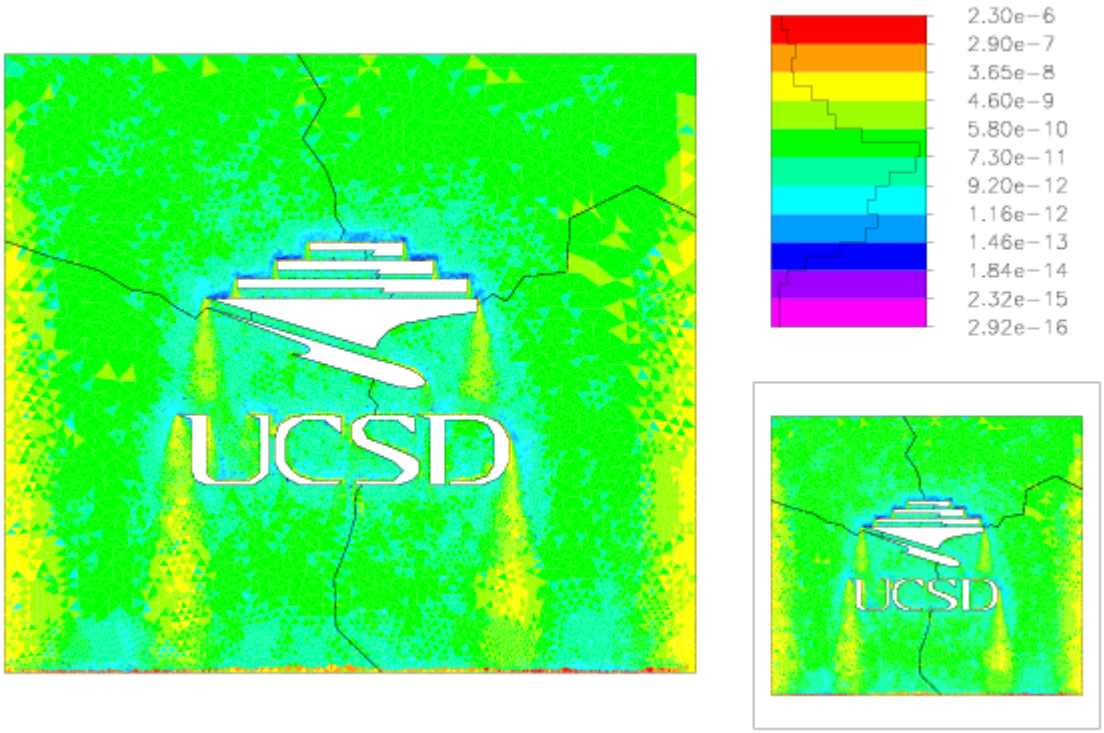



FIG. 2.4. A posteriori *error estimate for the final solution.*

$$c = \frac{1}{1 - u_x^2 - u_y^2} - 1.$$

$M(\nabla u)$ is one of the main quantities of interest in this calculation, and since it depends on $\nabla u$, it represents a more severe test of our algorithm.

As in the first example, we partitioned the domain into four subregions with approximately equal error using the recursive spectral bisection algorithm. Then four independent problems were solved with the mesh adaptively refined to one with approximately 5000 unknowns. The mesh for one subdomain and the corresponding $M(\nabla u)$ are shown in Figure 2.6.

The meshes from the four subproblems are combined to form a globally refined mesh with 37913 triangles and 18723 vertices. This mesh is shown in Figure 2.7, along with $M(\nabla u)$ computed from the initial guess for the the global solution. $M(\nabla u)$ computed from the global solution, as well as the *a posteriori* error estimates for the final global solution, are shown in Figure 2.9.

**2.3. A 3D Elasticity Problem.** The two previous examples demonstrated the effectiveness of the parallel algorithm for linear and nonlinear scalar problems in two dimensions. To illustrate that the parallel algorithm works equally well for coupled elliptic systems and for three-dimensional problems, we will use MC to solve the elasticity equations for our third example:

$$-\nabla \cdot T(\nabla u) = f \quad \text{in } \Omega \subset \mathrm{R}^3, \tag{2.3}$$

$$n \cdot T(\nabla u) = g \quad \text{on } \Gamma_1, \tag{2.4}$$

$$u = 0 \quad \text{on } \Gamma_0, \qquad \partial\Omega = \Gamma_0 \cup \Gamma_1, \qquad \{\} = \Gamma_0 \cap \Gamma_1. \tag{2.5}$$

The stress tensor $T$ is a function of the gradient $\nabla u$ of the unknown displacement $u$, and the corresponding deformation mapping $\varphi$ and deformation gradient $\nabla\varphi$ are given by:

$$\varphi = id + u : \overline{\Omega} \mapsto \mathrm{R}^3, \qquad \nabla u : \overline{\Omega} \mapsto \mathrm{M}^3, \qquad \nabla\varphi = I + \nabla u : \overline{\Omega} \mapsto \mathrm{M}^3.$$

We will use a linearized strain tensor and a linear stress-strain relation:

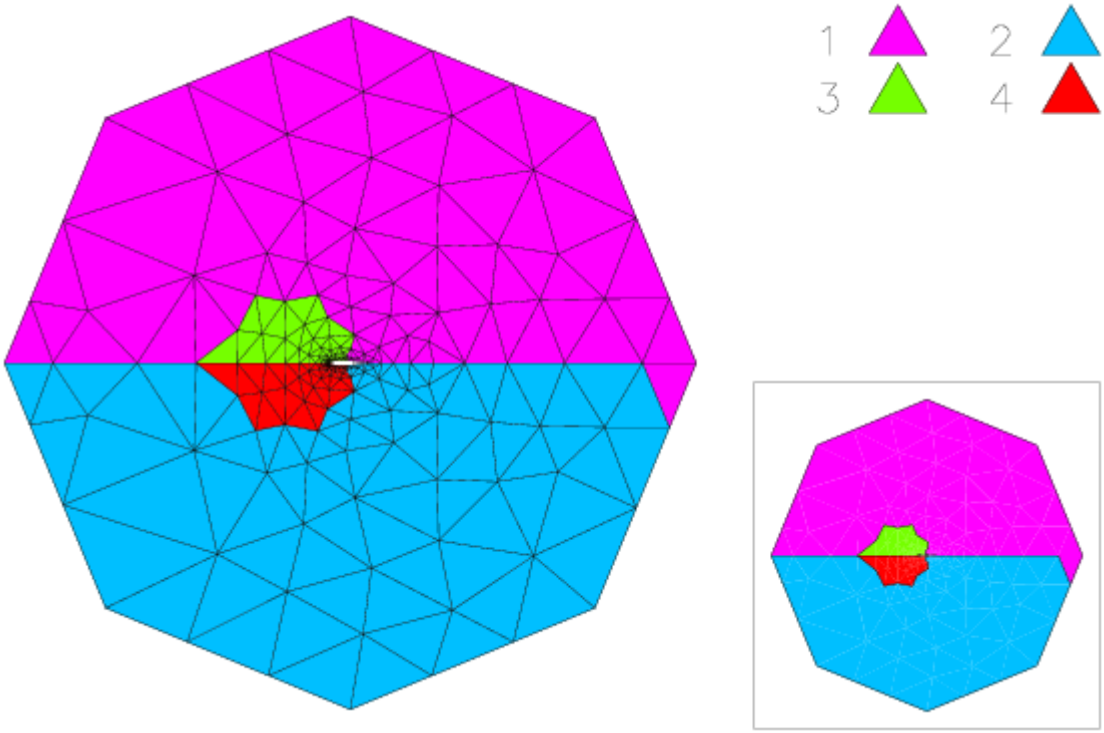
$$E(\nabla u) = \frac{1}{2}(\nabla u + \nabla u^T) : \overline{\Omega} \mapsto \mathrm{S}^3, \qquad T(\nabla u) = \lambda(\mathrm{tr}E)I + 2\mu E,$$

where the Lamé constants $\lambda$ and $\mu$ are taken to be those of steel ($\lambda \approx 10.4403$, $\mu \approx 8.20312$). The solid object forming the domain is depicted in Figure 2.11. We apply traction forces to the back and top of each letter in a horizontal and slightly upward direction, through the function $g$ above. The base of each letter is fixed to a solid foundation, through the essential boundary condition above. No volume forces are present, so that the function $f$ above is zero. We then solve the resulting equations (2.3–2.5) adaptively using MC.

The initial coarse mesh in the left top picture of Figure 2.11 has 296 tetrahedral elements and 159 vertices. As in the previous examples, we partition the domain into four subdomains with approximately equal error using the recursive spectral bisection algorithm described in Section 3 below. The four subdomain problems are then solved independently by MC, starting from the complete coarse mesh and coarse mesh solution. In this example, the mesh is adaptively refined in each subdomain until a mesh with about 5000 vertices is obtained.

The resulting refined subdomain mesh for one subdomain and the corresponding solution (the deformation mapping $\varphi$) is shown in Figure 2.11. (The deformation is taken to be larger than the range of validity of the linear elasticity equations for visualization purposes.) As in the PLTMG examples, the refinement performed by MC is confined primarily to the given region, with some refinement into adjacent regions due to the closure algorithm which maintains conformity and shape regularity. The four problems are solved completely independently by the sequential adaptive code MC. This decomposition approach is especially effective for this particular problem due to the structure of the object, which leads to very weak couplings between the deformations of the individual letters, and due to the fact that the spectral bisection algorithm happens to decomposes the mesh close to the boundaries of the letters.

**2.4. A 3D Nonlinear Elliptic System Arising in Gravitation.** The fourth and final example involves the use of MC to solve a coupled nonlinear elliptic system in $\mathrm{R}^3$, namely the elliptic constraints in




*The initial triangulation, partitioned into four subregions with approximately equal error.*

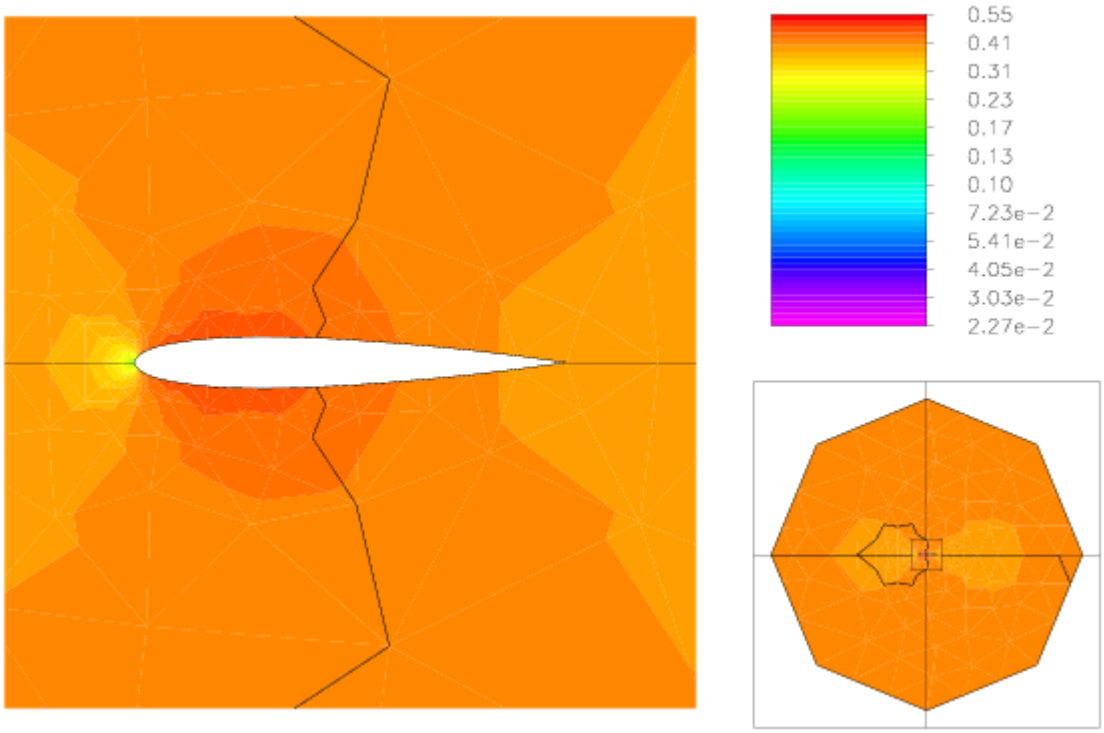
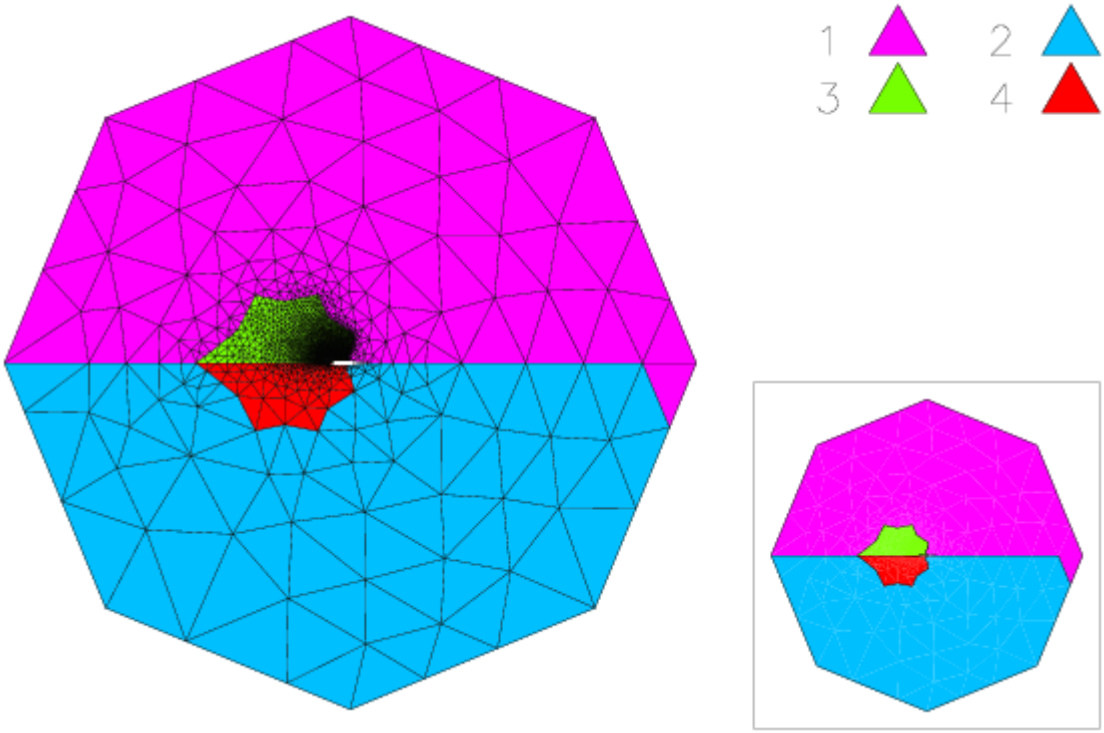



FIG. 2.5. $M(\nabla u)$ *computed from the coarse grid solution.*




*The refined mesh for problem 3.*

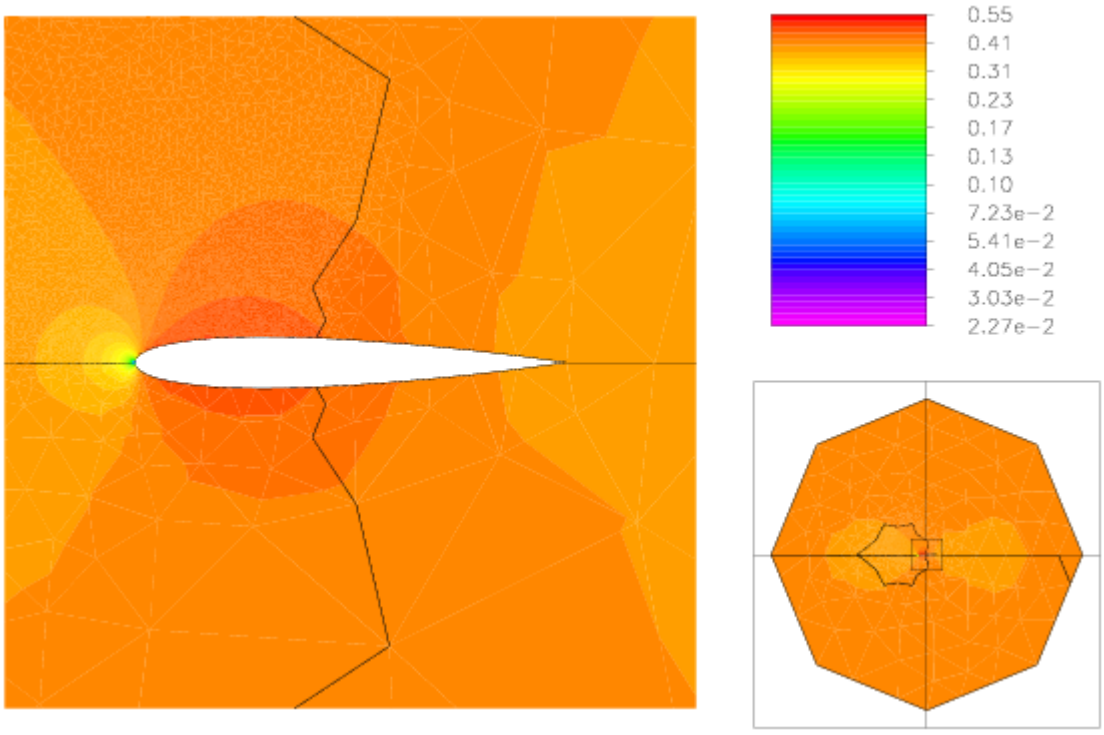
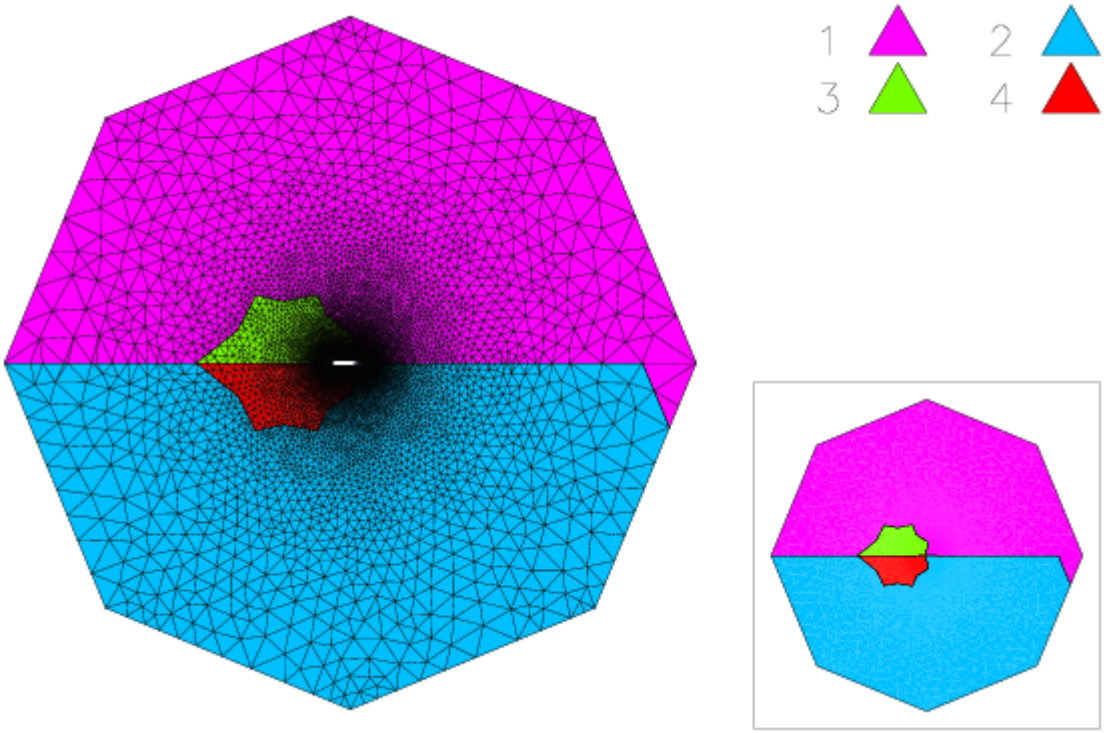



FIG. 2.6. $M(\nabla u)$ *computed from the solution for problem 3.*




*The global refined mesh.*

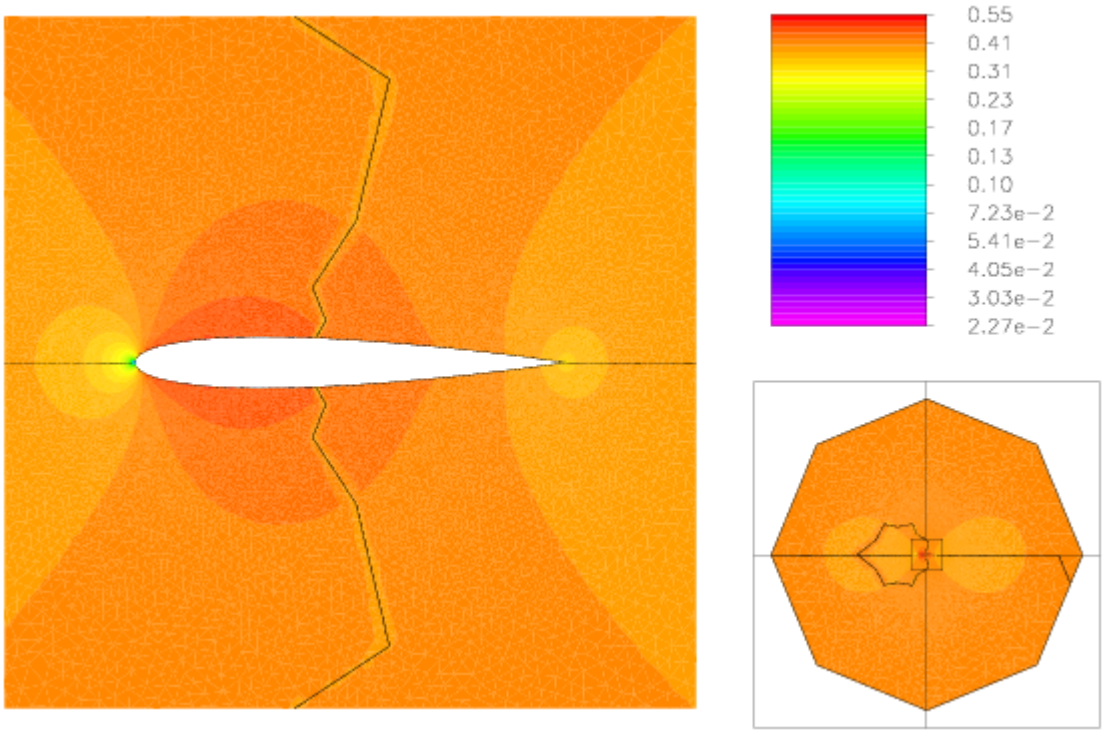



FIG. 2.7. $M(\nabla u)$ *computed from the initial guess for the global solution.*

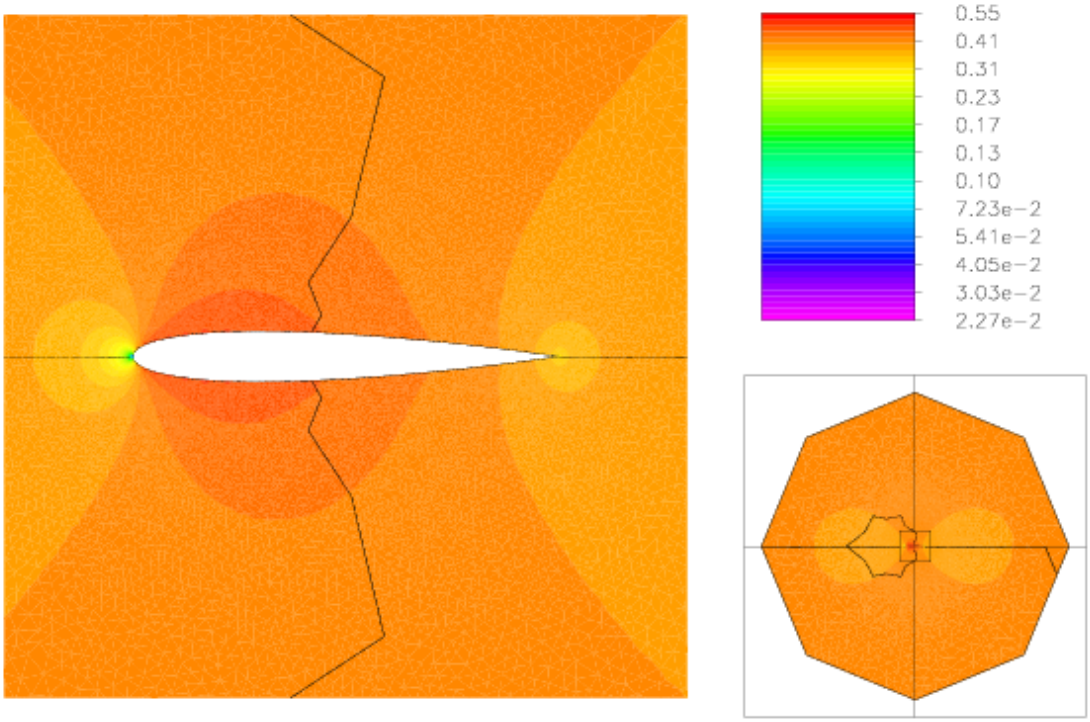



FIG. 2.8. $M(\nabla u)$ *computed from the final global solution.*

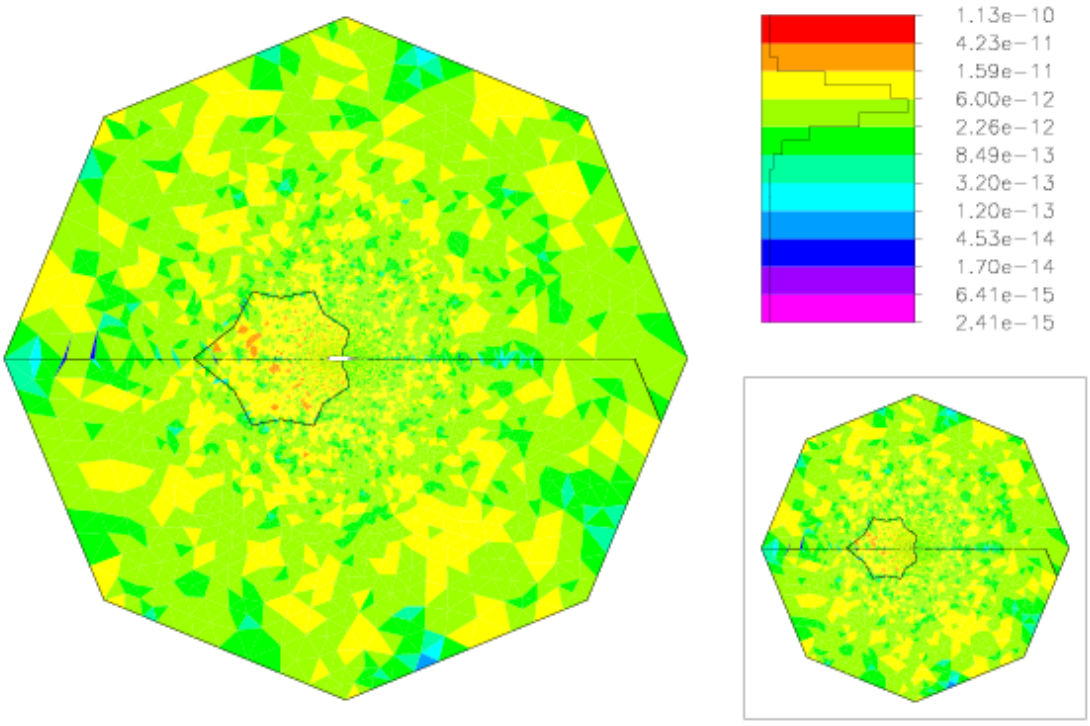



FIG. 2.9. A posteriori *error estimate for the final solution.*

the Einstein equations (cf. [38]):

$$\hat{\gamma}^{ab}\hat{D}_a\hat{D}_b\phi = \frac{1}{8}\hat{R}\phi - \frac{1}{8}\phi^{-7}(\hat{A}^*_{ab} + (\hat{l}W)_{ab})^2 + \frac{1}{12}(\mathrm{tr}K)^2\phi^5 - 2\pi\hat{\rho}\phi^{-3}, \tag{2.6}$$

$$\hat{D}_b(\hat{l}W)^{ab} = \frac{2}{3}\phi^6\hat{D}^a\mathrm{tr}K + 8\pi\hat{j}^a. \tag{2.7}$$

The unknowns are the "conformal factor" $\phi$ and the vector potential $W^b$. The $(\hat{l}W)^{ab}$ operator above is a certain symmetrized gradient operator for tensors:

$$(\hat{l}W)^{ab} = \hat{D}^aW^b + \hat{D}^bW^a - \frac{2}{3}\hat{\gamma}^{ab}\hat{D}_cW^c. \tag{2.8}$$

The Einstein summation convention is used above, so that all repeated symbols in products imply a sum over that index. The gradient operator $\hat{D}^a$ is covariant, meaning that its application requires the use of Christoffel symbols due to the curvilinear nature of the coordinate system required to represent the underlying domain manifold. The Christoffel symbols are formed with respect to an underlying background metric $\hat{\gamma}^{ab}$, so that the left-hand side of the first equation for the conformal factor $\phi$ is essentially a covariant Laplace operator.

To use MC to calculate the initial bending of space and time around two massive black holes separated by a fixed distance by solving the above constraint equations, we place two spherical objects in space, the first object having unit radius (after appropriate normalization), the second object having radius 2, separated by a distance of 20. Infinite space is truncated with an enclosing sphere of radius 100. (This outer boundary may be moved further from the objects to improve the accuracy of boundary condition approximations.) Physically reasonable functions for remaining parameters appearing in the equations are used to completely specify the problem (cf. [16] for details).

We then generate an initial (coarse) mesh of tetrahedra inside the enclosing sphere, exterior to the two spherical objects within the enclosing sphere. The mesh is generated by adaptively bisecting an initial mesh consisting of an icosahedron volume filled with tetrahedra. The bisection procedure simply bisects any tetrahedron which touches the surface of one of the small spherical objects. When a reasonable approximation to the surface of the spheres is obtained, the tetrahedra completely inside the small spherical objects are removed, and the points forming the surfaces of the small spherical objects are projected to the spherical surfaces exactly. This projection involves solving a linear elasticity problem (nearly identical to the problem solved in Example 3 above), together with the use of a shape-optimization-based smoothing procedure. The smoothing procedure locally optimizes the following shape measure function for a given $d$-simplex $s$, in an iterative fashion, similar to the approach taken in [4]:

$$\eta(s,d) = \frac{2^{2(1-\frac{1}{d})}3^{\frac{d-1}{2}}|s|^{\frac{2}{d}}}{\sum_{0\le i<j\le d}|e_{ij}|^2}$$

The quantity $|s|$ represents the (possibly negative) volume of the $d$-simplex $s$, and $|e_{ij}|$ represents the length of the edge that connects vertex $i$ to vertex $j$ in the simplex. For $d = 2$, this is the shape-measure used in [4], with a slightly different normalization. For $d = 3$, this is the shape-measure given in [18], again with a slightly different normalization. Unlike Laplace smoothing, this local shape optimization approach is guaranteed to improve the shape of elements locally at each step, and always maintains a mesh of simplices with positive volumes.

The initial coarse mesh in Figure 2.15, generated using the procedure described above, has 31786 tetrahedral elements and 5809 vertices. As in the previous examples, we partition the domain into four subdomains with approximately equal error using the recursive spectral bisection algorithm described in Section 3 below. The four subdomain problems are then solved independently by MC, starting from the complete coarse mesh and coarse mesh solution. The mesh is adaptively refined in each subdomain until a mesh with roughly 50000 vertices is obtained (yielding subdomains with about 250000 simplices each).

The resulting refined subdomain meshes are shown in Figure 2.19. As in the previous examples, the refinement performed by MC is confined primarily to the given region, with some refinement into adjacent regions due to the closure algorithm which maintains conformity and shape regularity. The four problems are solved completely independently by the sequential adaptive code MC. One component of the solution (the conformal factor $\phi$) of the elliptic system is depicted in Figure 2.20 (the subdomain zero solution) and in Figures 2.21 (the subdomain two solution).

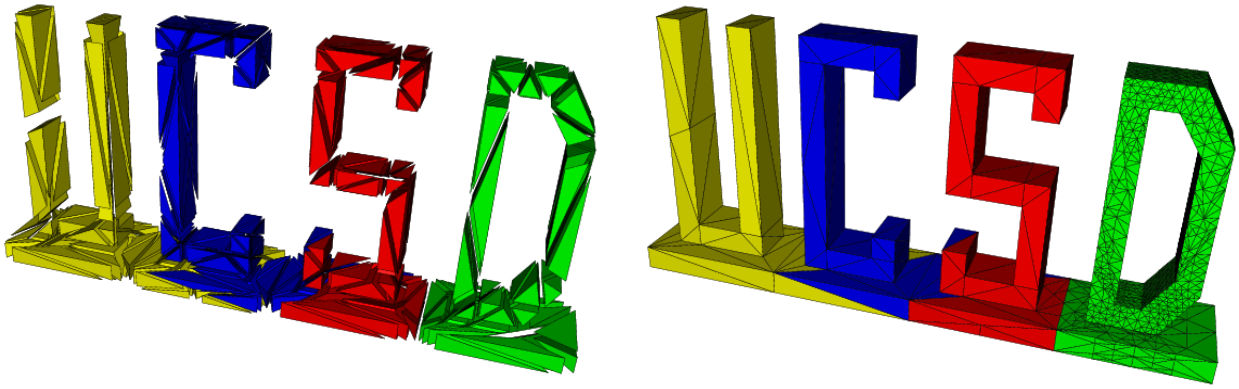


FIG. 2.10. *Initial spectrally bisected UCSD domain and the D subdomain mesh.*

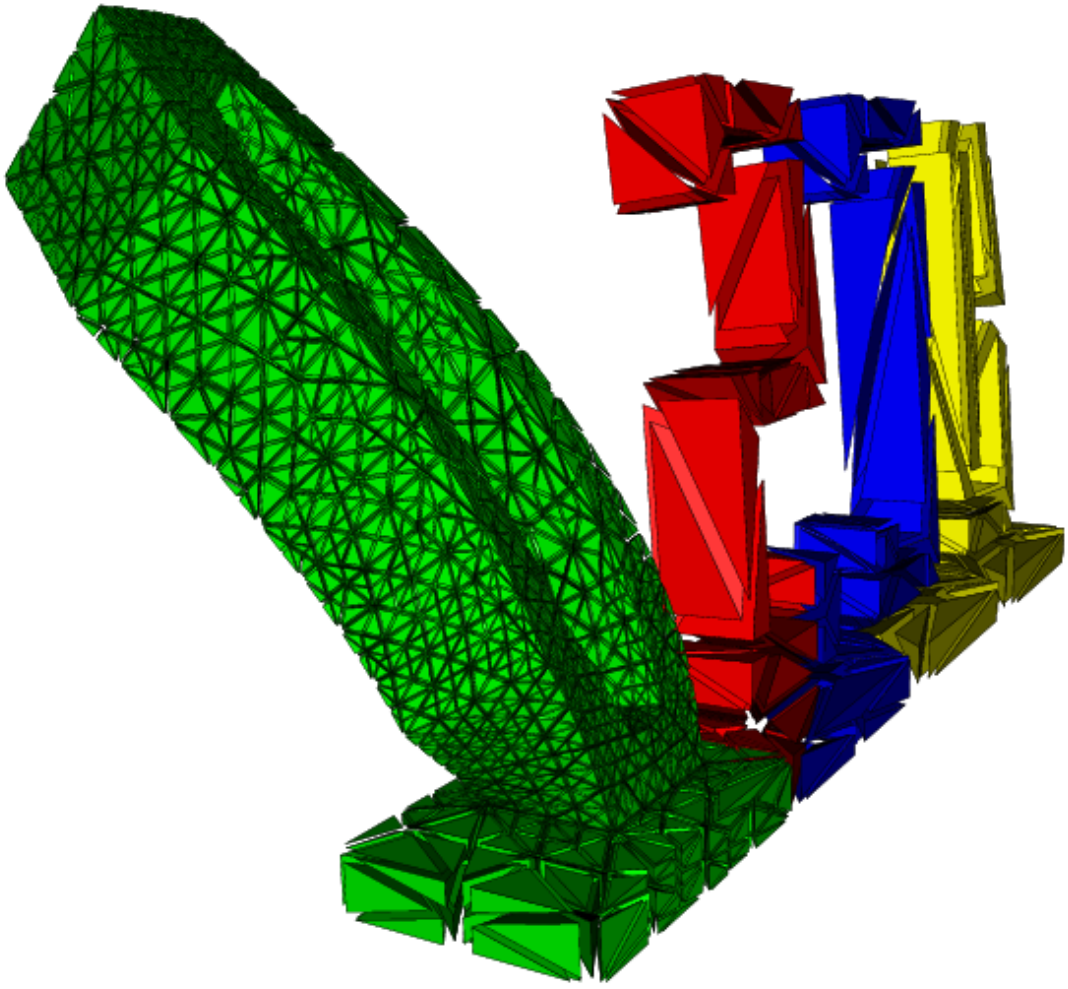


FIG. 2.11. *The exploded deformation of the D subdomain mesh (4838 vertices and 20905 simplices).*

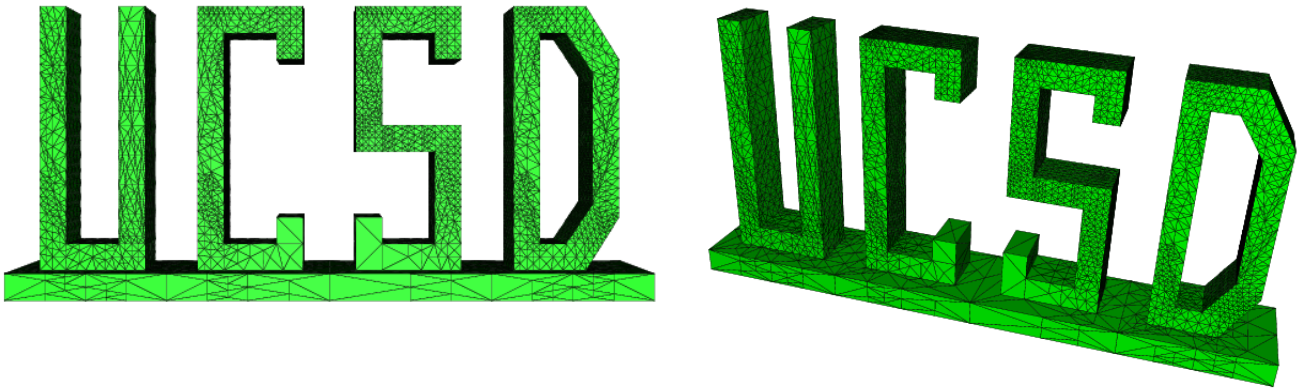


FIG. 2.12. *A global fine (comparison solution) mesh.*

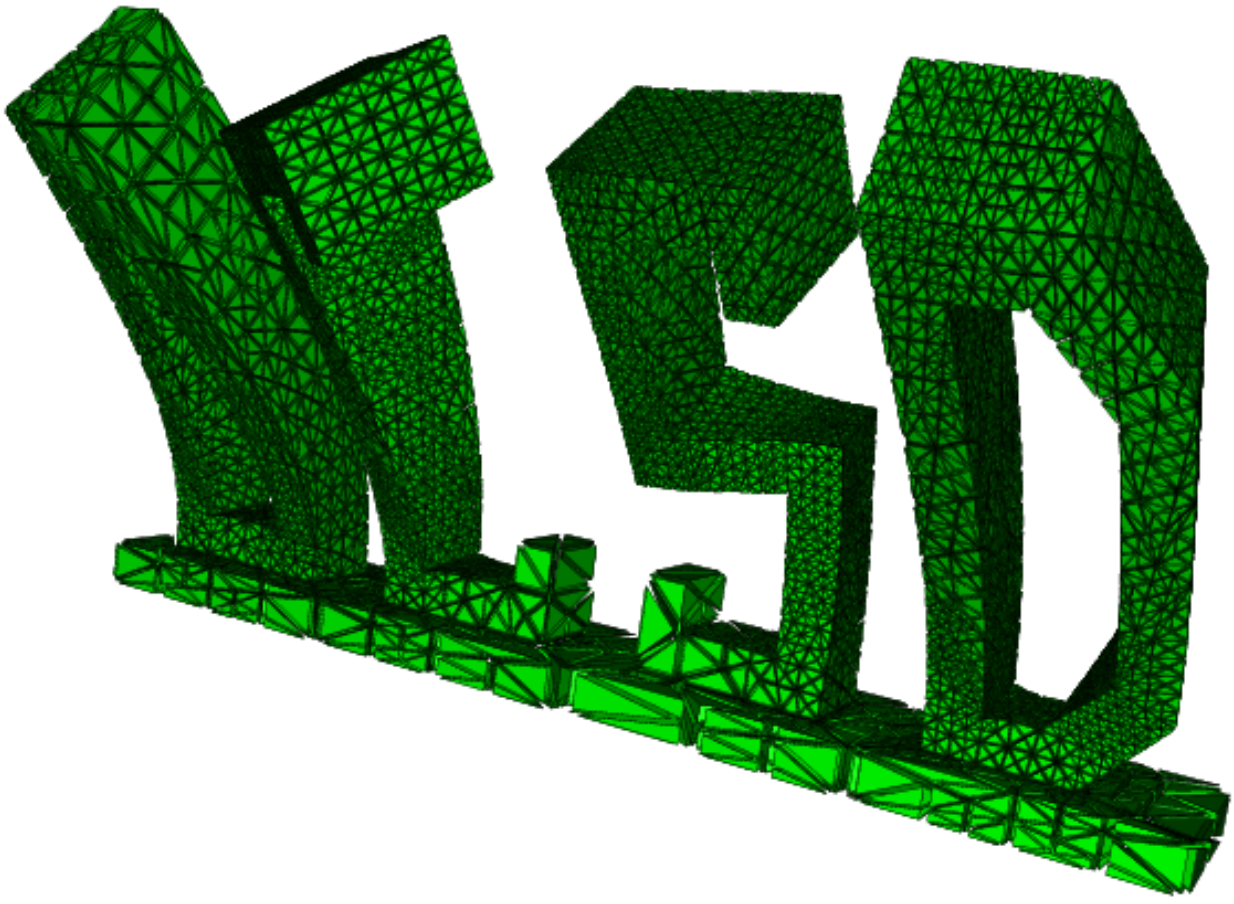


FIG. 2.13. *The exploded deformation of the global fine mesh (19884 vertices and 87769 simplices).*

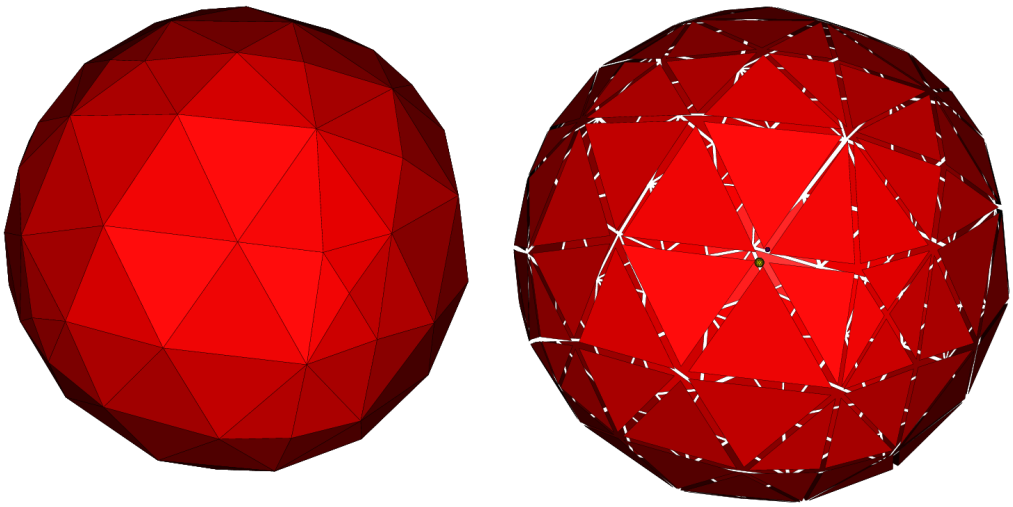


FIG. 2.14. *The coarse binary black hole mesh (5809 vertices and 31786 simplices).*

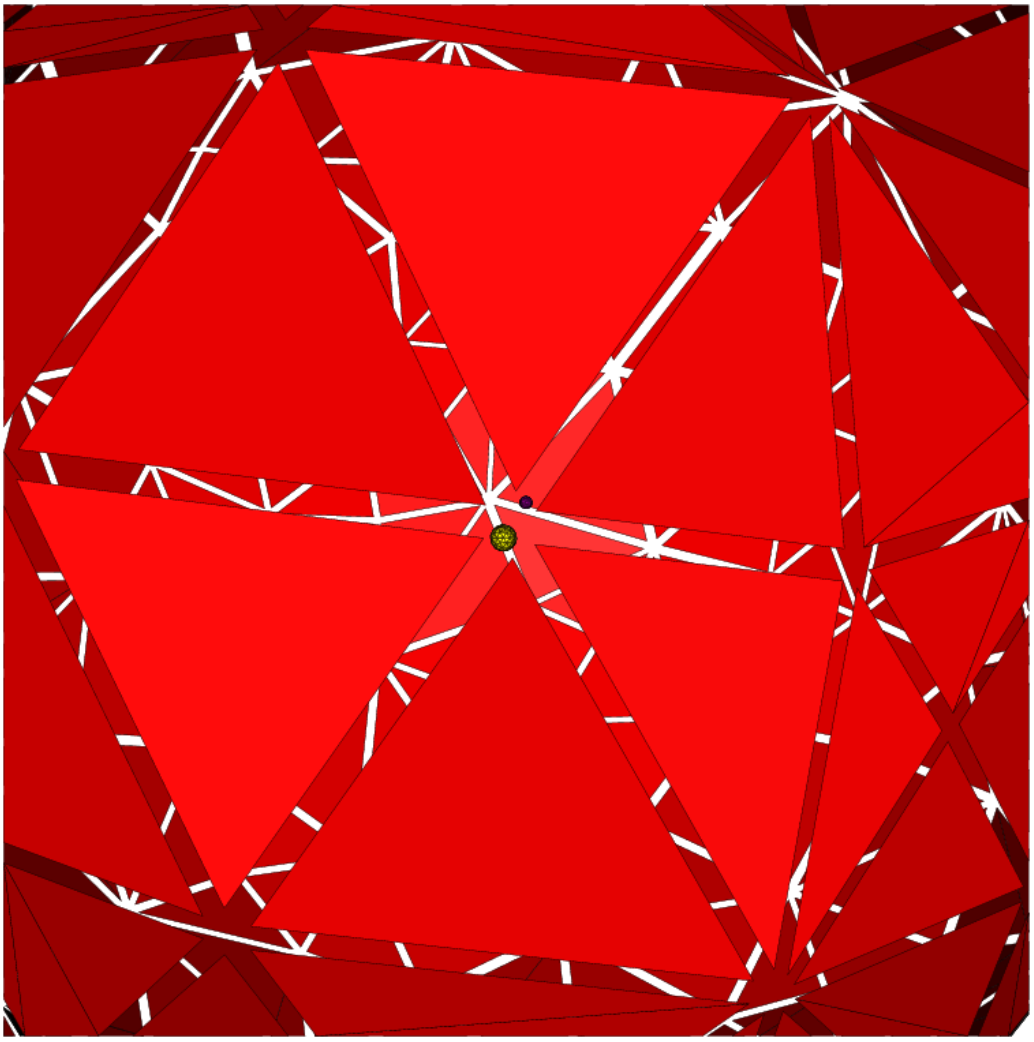


FIG. 2.15. *Exploded view of the coarse binary black hole mesh showing the two interior hole boundaries.*

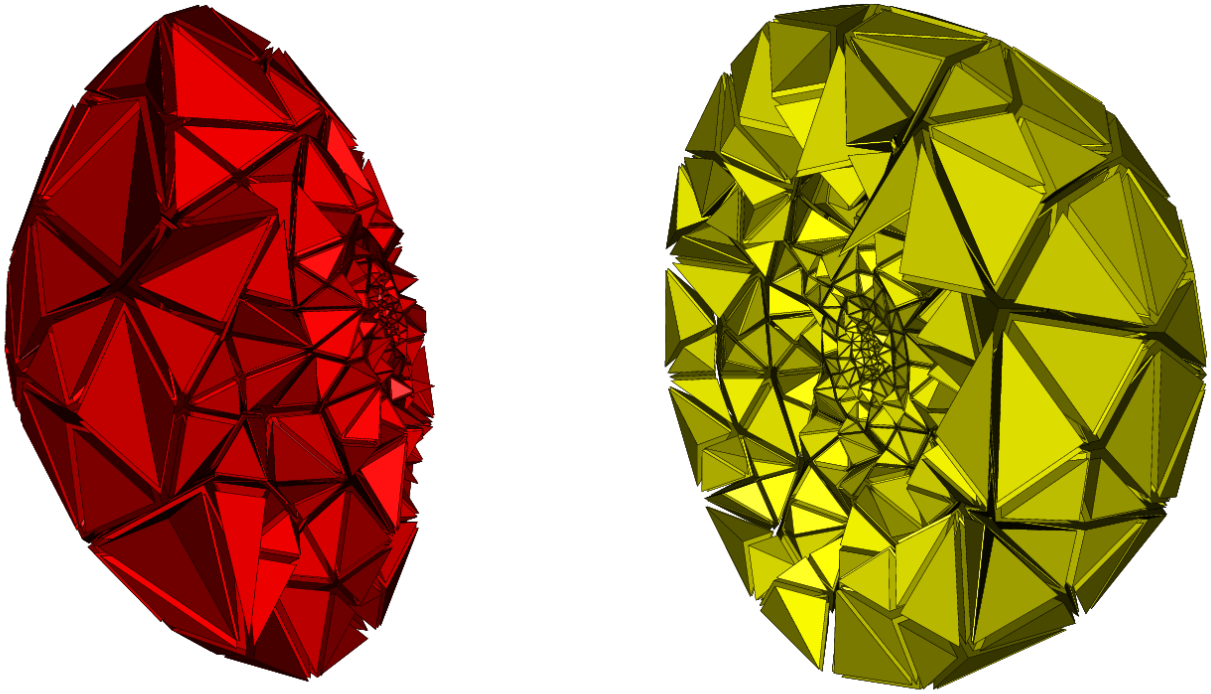


FIG. 2.16. *Subdomains 2 (red) and 4 (yellow) from spectral bisection of the coarse binary black hole mesh; these subdomains enclose two smaller subdomains that contain the inner holes.*

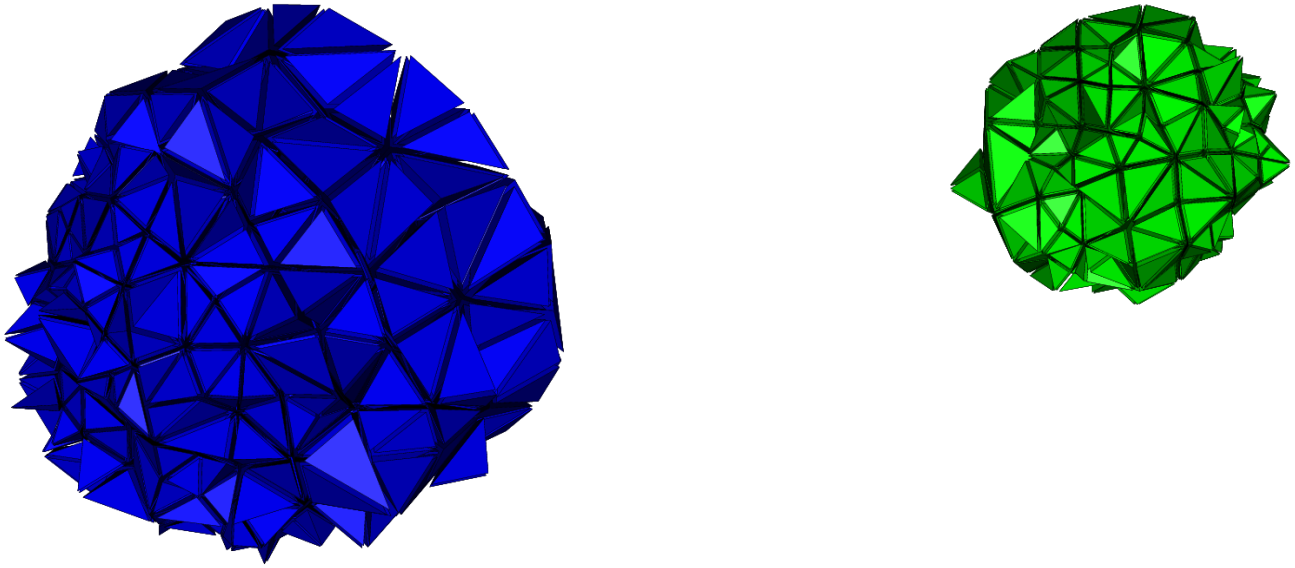


FIG. 2.17. *Subdomains 3 (blue) and 1 (green) from spectral bisection of the coarse binary black hole mesh; these subdomains each contain one of the inner holes.*

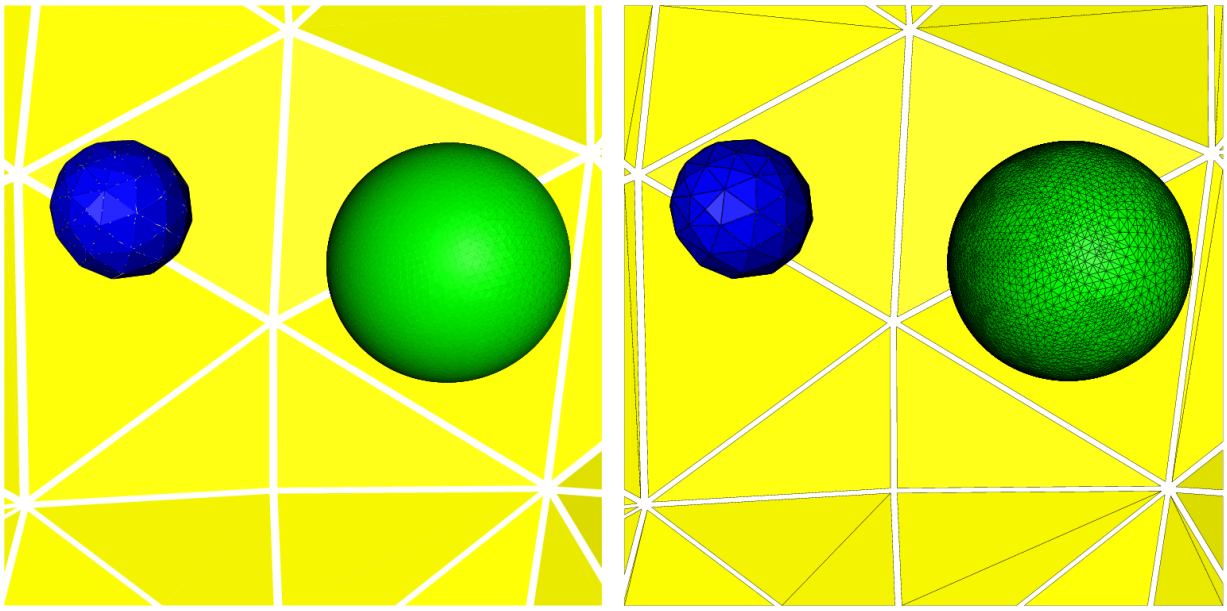


FIG. 2.18. *Refined mesh for subdomain 1. (51915 vertices and 266114 simplices; only faces of tetrahedra on the boundary surfaces are show).*

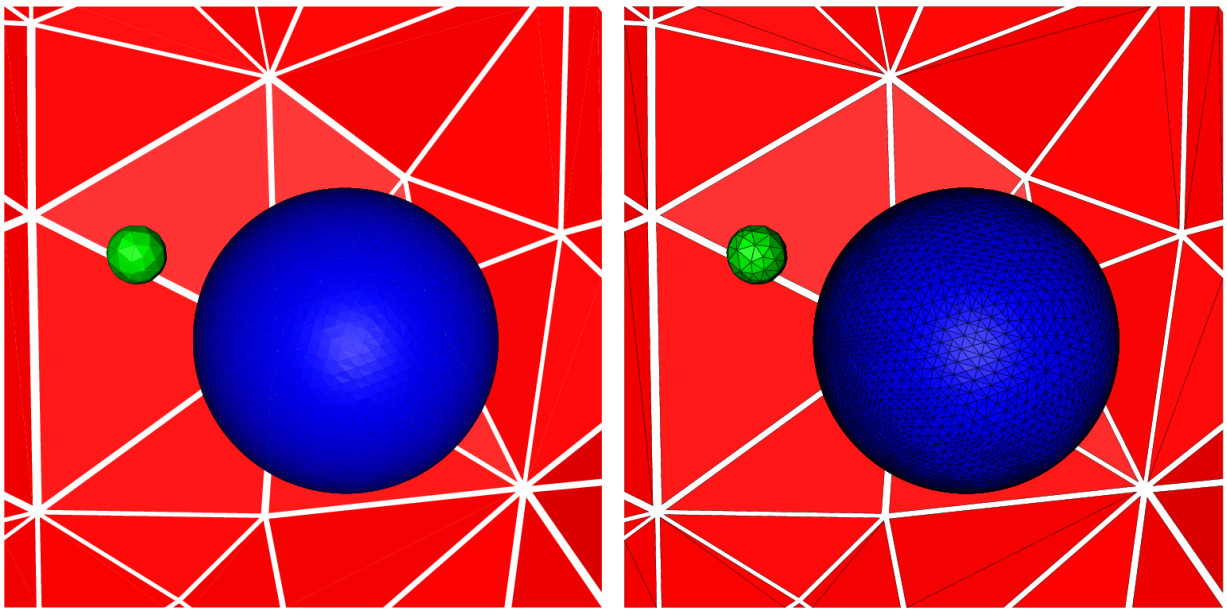


FIG. 2.19. *Refined mesh for subdomain 3. (44550 vertices and 228194 simplices; only faces of tetrahedra on the boundary surfaces are show).*

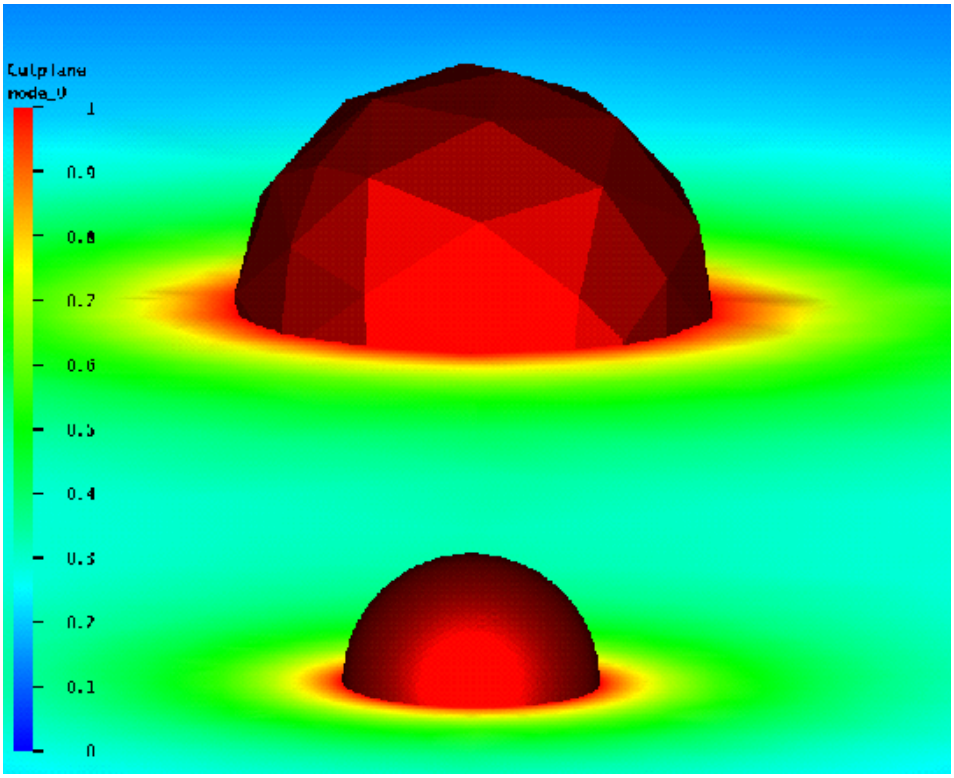



FIG. 2.20. *The conformal factor $\phi$ from the adapted subdomain 1 solve.*

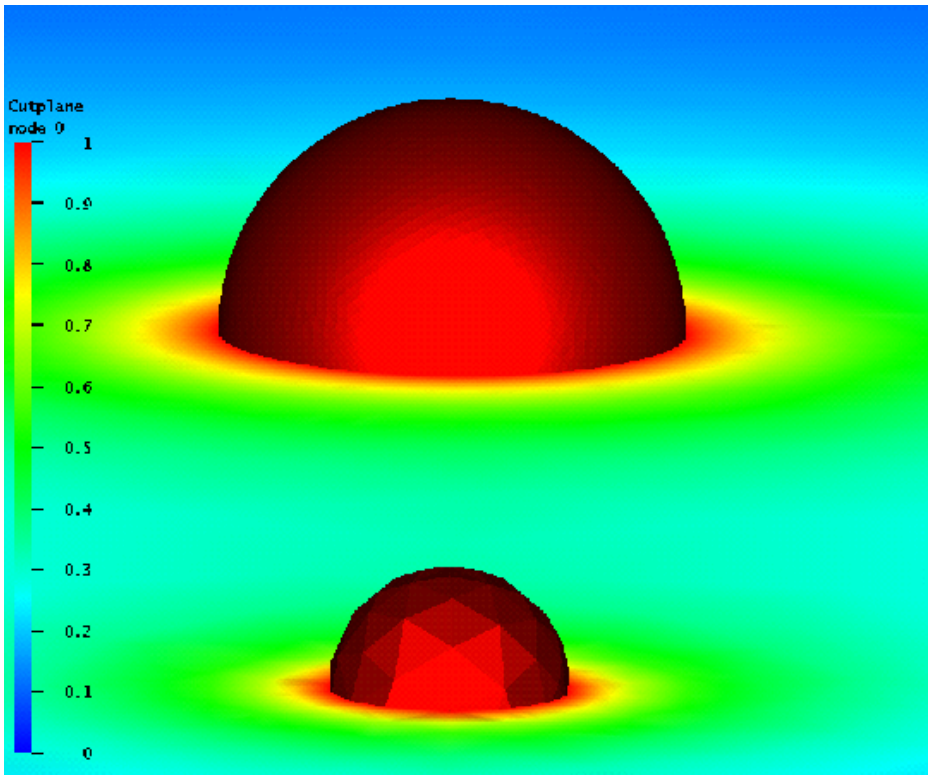



FIG. 2.21. *The conformal factor $\phi$ from the adapted subdomain 3 solve.*

## 3. Computational Considerations.

**3.1. A Spectral Bisection Algorithm.** In this section we describe the algorithm we use for partitioning the coarse mesh so that each subregion has approximately equal error. This algorithm is a variant of the recursive spectral bisection algorithm [11, 27, 31]. While this particular mesh partitioning algorithm is one of the more expensive of the choices that we could make, we emphasize that it is used in our algorithm *only once*, and it is only used on a very small coarse grid problem. As a result, the initial partitioning cost is much smaller than the solve time for a single subdomain problem (see Tables 3.1 and 3.2 below).

Let $\mathcal{T}$ denote a triangulation of the domain $\Omega$ with triangular elements $t_i \in \mathcal{T}$, $1 \le i \le N$, and let $e_i$ denote the *a posteriori* error estimate for $t_i$,

$$e_i \approx \|\nabla(u - u_h)\|^2_{t_i}.$$

Define the $N \times N$ symmetric, positive semi-definite $M$-matrix $A$ by

$$A_{ij} = \begin{cases} -1 & i \neq j \text{ and } t_i \text{ and } t_j \text{ share a common edge} \\ 0 & i \neq j \text{ and } t_i \text{ and } t_j \text{ do not share a common edge} \\ s_i & i = j,\ s_i = -\sum_{k \neq i} A_{ik} \end{cases}$$

and the positive diagonal matrix $D$ by

$$D_{ii} = e_i / e_{max},$$

where

$$e_{max} = \max_i e_i.$$

A typical row of $A$ will have three nonzero off-diagonal elements and $A_{ii} = 3$; this is the so-called *discrete Laplacian* for the dual graph for the triangulation (the triangles themselves are the nodes of the dual graph, and the edges are defined by the adjacency relation). We consider the generalized eigenvalue problem

$$A\psi = \lambda D\psi \tag{3.1}$$

Our approach is standard; by construction, the smallest eigenvalue for (3.1) is $\lambda_1 = 0$ and $\psi_1 = (1, 1, \ldots, 1)^t$. Our interest is in the second eigenvector $\psi_2$, known as the Fiedler vector.

Let $\mathcal{S}^+$ and $\mathcal{S}^-$ denote the index sets corresponding to positive and nonpositive components of $\psi_2$. Then from the orthogonality relation

$$\psi_1^t D \psi_2 = 0$$

we have

$$\sum_{i \in \mathcal{S}^+} e_i \psi_{2,i} = -\sum_{i \in \mathcal{S}^-} e_i \psi_{2,i}.$$

This indicates that the total error for the two groups of triangles is approximately balanced (and would be exactly balanced if all entries of $\psi_2$ were $\pm 1$). Using this observation as motivation, to construct a partition of the elements we first find a permutation of the elements $\{p_i\}$ such that

$$\psi_{2,i} \le \psi_{2,j} \quad \text{ if and only if } \quad p_i < p_j.$$

We then find the index $k$ which provides the best partition of the form

$$\sum_{p_i \le k} e_i \approx \sum_{p_i > k} e_i. \tag{3.2}$$

Often this partition is very close to the partition based on $\mathcal{S}^{\pm}$. This is similar to the strategy suggested by Chan, Ciarlet and Szeto [11].

As usual, we apply this approach recursively, at each level dividing each group of elements into two smaller groups by solving an eigenvalue problem of the type (3.1) restricted to that group of elements. Thus after $k$ steps, we have created a partition of the elements into $2^k$ groups of roughly equal error.

We now briefly describe some details of our procedure for computing the second eigenvector of (3.1). Our procedure is essentially just a classical Rayleigh quotient iteration [25], modified both to bias convergence to $\lambda_2$, and to account for the fact that the linear systems arising in the inverse iteration substep are solved (approximately) by an iterative process. To simplify notation and avoid multiple subscripts, we let $\phi_k \approx \psi_2$, where $k$ now denotes the iteration index.

We suppose that we have a current iterate $\phi_k$ which satisfies

$$\phi_k^t D \phi_k = 1 \qquad \text{and} \qquad \psi_1^t D \phi_k = 0. \tag{3.3}$$

Using $\phi_k$ we compute the approximate eigenvalue $\sigma_k \approx \lambda_2$ from the Rayleigh quotient

$$\sigma_k = \phi_k^t A \phi_k$$

and form the residual vector

$$r_k = \sigma_k D \phi_k - A \phi_k.$$

Note that by construction $\psi_1^t r_k = \phi_k^t r_k = 0$. Next, we approximately solve the linear system

$$(A - \theta_k \sigma_k D) \tilde{\delta}_k = r_k, \tag{3.4}$$

where $0 \leq \theta_k \leq 1$ is chosen to try to insure $0 \leq \theta_k \sigma_k \leq \lambda_2$. A simple strategy such as $\theta_k = 1 - 2^{-k}$, is often effective (although, of course, offers no guarantees). From $\tilde{\delta}_k$, we form the vector $\delta_k$ satisfying

$$\delta_k^t D \delta_k = 1 \qquad \text{and} \qquad \psi_1^t D \delta_k = \phi_k^t D \delta_k = 0.$$

The inverse iteration step uses a conservative shift policy in order to strongly bias the calculation in favor of convergence to the desired second eigenvector. The residual appears as right hand side, since this system is to be solved by iteration rather than by direct Gaussian elimination. In this circumstance only a few iterations are used, and the effect is mainly to attenuate unwanted eigenvectors rather than "blow up" the desired eigenvector. For our iterative method, we use a conjugate gradient procedure with symmetric Gauss-Seidel preconditioner. So far, this has proven to be simple and effective, but the issue of the most efficient solver in this context is presently open.

Finally, we solve the $2 \times 2$ eigenvalue problem

$$\hat{A} v = \lambda v$$

where

$$\hat{A} = \begin{pmatrix} \phi_k^t \\ \delta_k^t \end{pmatrix} A \begin{pmatrix} \phi_k & \delta_k \end{pmatrix}$$

If $v = (\alpha, \beta)^t$ is an eigenvector corresponding to the smaller eigenvalue, we form

$$\tilde{\phi}_{k+1} = \alpha \phi_k + \beta \delta_k$$

and then $\phi_{k+1}$ is formed from $\tilde{\phi}_{k+1}$ by imposing conditions (3.3). The use of this subspace-iteration-like calculation rather than a simple eigenvector update provides a second means to bias the overall Rayleigh quotient iteration is favor of convergence to $\psi_2$. It also compensates to some extent for the loss in convergence rate due to the conservative shift policy and incomplete solution of (3.4) by iteration.

This completes the description of a single Rayleigh quotient iteration. Note that continually imposing orthogonality conditions with respect to $\psi_1$ is mathematically unnecessary, but is important in practice because this direction is reintroduced by roundoff error. Without systematically and continually excluding this eigenvector, the Rayleigh quotient iteration could easily converge to $\psi_1$.

**3.2. Load Balancing.** Partitioning the domain to achieve approximately equal error in each subregion is not really the optimal approach. The optimal strategy is to partition the domain such that each subregion requires equal *work* for the ensuing calculation, and the errors are approximately equal in each element of the global composite mesh. Estimating the work is problematic for many reasons, among them:

- The cost of function evaluations and numerical integration used in computing matrices, right hand sides, and *a posteriori* error estimates might vary significantly in various regions. Moreover, the number of such quadratures depends on the number of elements, and the number of instances when such assembly steps are required.
- The number of iterations of Newton's method for nonlinear systems and linear iterative methods for linear systems may vary slightly from problem to problem, even though all are derived from the same continuous problem. Because the number of iterations is usually small, the percentage change in the work can be quite large (e.g., 3 rather than 2 multigrid cycles represents a 50% increase in the work for that part of the computation). It should also be clear that such small differences are difficult to predict in advance of the actual calculation. The cost per iteration will also vary due to differing orders of the problems.
- The cost of grid management (refining, unrefining, moving the mesh points, and maintaining the relevant data structures) will vary with the number of elements involved and the particular mix of tasks.
- The number of major iterations through the adaptive feedback loop may differ from problem to problem, even if the final meshes all have about the same number of unknowns.

Creating subregions of approximately equal error for the initial partition really amounts to the fragile assumption that this corresponds to approximately equal work for each processor. Although one can hope that more sophisticated models of work will lead to improvement, it seems certain that the overall flexibility and complexity of current adaptive solvers will still make this aspect of the initial load balancing phase problematic.

On the positive side, despite all the dangers mentioned above, our load balancing procedure using *a posteriori* error estimates has empirically been observed to be much better than one might at first expect, at least for the classes of problems we address. For example, in Table 3.1, we give the overall execution times (in seconds) for the two PLTMG example calculations described in Section 2 (Examples 1 and 2). These times are for an SGI Octane 195mhz R10000, using the f77 compiler options -O -32.

| | UCSD | NACA |
|---|---|---|
| Initialization | 2.49 | 0.82 |
| Solve Subproblem 1 | 6.58 | 12.9 |
| Solve Subproblem 2 | 6.75 | 13.0 |
| Solve Subproblem 3 | 6.92 | 13.0 |
| Solve Subproblem 4 | 6.78 | 12.9 |
| Postprocessing | 0.89 | 0.81 |

TABLE 3.1
*PLTMG Execution times (seconds) for convection-diffusion example (UCSD) and the potential flow example (NACA).*

In Table 3.2, we give the execution times (in seconds) for the two MC example calculations described in Section 2 (Examples 3 and 4). These times are for a single 195Mhz R10000 processor of an SGI Octane, using the IRIX C compiler with optimization -O2. While the numbers of vertices and elements in the subdomain meshes of the 3D UCSD example are similar to the two PLTMG examples, there are actually three unknowns at each vertex (the three displacement degrees of freedom), so the discrete problem sizes are triple the number of vertices.

The initialization times in both tables includes generating a coarse mesh from the geometry description, solving the coarse mesh problem, computing *a posteriori* error estimates, and partitioning the domain. The initialization was done on one processor. The times for each of the subproblems include all aspects of the adaptive feedback loop, including matrix and right hand side assembly, solution of linear and nonlinear systems, *a posteriori* error estimation, and adaptive refinement and mesh smoothing. In Table 3.1, this also includes some relatively inexpensive clean-up performed by PLTMG, mainly removing unwanted parts of

| | UCSD3D | BHOLE |
|---|---|---|
| Initialization | 6 | 74 |
| Solve Subproblem 1 | 71 | 1905 |
| Solve Subproblem 2 | 95 | 1523 |
| Solve Subproblem 3 | 73 | 1811 |
| Solve Subproblem 4 | 82 | 1467 |
| Global test solution | 422 | — |

TABLE 3.2
*MC Execution times (seconds) for the elasticity example (UCSD3D) and the binary black hole example (BHOLE).*

each mesh and solution in preparation for creating the global composite mesh. (These calculations are not performed by MC and do not appear in the timing figures in Table 3.2.)

The postprocessing costs given in Table 3.1 for the PLTMG implementation includes the creation of a global conforming mesh and forming an initial guess for the the solution on that mesh using the procedures described in Section 4.1. In our present code, this is also done on one processor. In contrast, the MC implementation does not form a global problem; this is justified to some extent in Section 4.3. The global test solution costs listed in Table 3.2 for the first MC example reflects the cost of solving the entire problem sequentially on one processor without decomposition. (The refined black hole subdomain problems were so large that it was not possible to solve a single global problem on an SGI Octane with roughly four times the number of simplices of each subdomain problem.) What is interesting to note is that the overhead required to decompose the problem is amortized by the gain due to the reduced subproblem sizes; solving the subproblems sequentially is actually faster than solving the global problem. In other words, if the solution quality of the decomposition algorithm is reliable, then the decomposition algorithm actually reduces the sequential solution time when viewed purely as a sequential algorithm. Note that if the decomposition algorithm is used in conjunction with solvers with less favorable complexities than the $O(N \log N)$ complexities in PLTMG and MC, then the decomposition algorithm would show an even larger gain over solving the global problem.

From Tables 3.1 and 3.2 we see that although the mix of calculations was different for each subproblem, the overall times do not vary too much. And since these problems were solved completely independently, there was no time spent in communication between processors, synchronization, etc. Thus to some extent, the time potentially saved by not having the processor communicate during the bulk of the computation offsets the time potentially lost by imperfect load balancing. The potential flow problem has a smaller initialization time than the convection-diffusion problem, mainly due to the smaller size of its coarse mesh problem (221 vertices compared to 1368). The solution times for the subspace problems are larger, due mainly to the facts that the potential flow problem is nonlinear and involves a Newton iteration, and it has more expensive coefficient function evaluations in the assembly phases and the *a posteriori* error estimates. In the three-dimensional examples, the black hole calculation has a much larger initialization time than the elasticity problem due to the size of the initial coarse mesh (159 vertices compared with 5,809 vertices).

**3.3. Scalability.** We now consider some aspects of the scalability of our procedure. Let $N_c$ denote in the size of the coarse grid problem (number of elements or grid points). Let $N_f$ denote the target size of the global fine grid problem, $p$ denote the number of processors, and $N_p$ denote the target problem size for each of the processors.

We have, approximately,

$$N_p \approx N_c + \frac{N_f - N_c}{p} \tag{3.5}$$

Relation (3.5) does not take into account the fact that the processor given the task of refining subregion $\Omega_i$ will refine some elements *not* in $\Omega_i$. This will occur mainly near the interface boundaries of $\Omega_i$, where the mesh will be graded in a smooth fashion from the smaller elements of $\Omega_i$ to larger elements that cover the remainder of $\Omega$. Such grading is necessary to maintain a conforming, shape regular mesh. In a typical situation, one would expect this to be an effect of order $O(N_p^{1/2})$ in two dimensions, and of order $(N_p^{2/3})$ in

three dimensions. Nevertheless, in practical situations, choosing $N_p > N_c + (N_f - N_c)/p$ is generally needed to achieve a fine grid problem size of $N_f$.

It seems clear that generally one should have

$$N_c \gg p. \tag{3.6}$$

A requirement like (3.6) is important to give the partitioning algorithm enough flexibility to construct regions of approximately equal error. For example, in the extreme case where the number of elements and the number of processors are equal, then the only partition is to provide one element to each processor, regardless of the error. This would likely result in a very uneven distribution of the error and poor performance of our adaptive refinement strategy.

It also is important to have

$$N_p \gg N_c. \tag{3.7}$$

This will marginalize the cost of redundant computations. For example, if $N_p = 2N_c$, then one could expect that about half of the computation on each processor would be redundant, which is a significant fraction of the total cost. By solving the problem on the entire domain, using a coarse mesh in all but one subregion, we are in effect substituting computation for communication. This trade-off will be most effective in situations where $N_p$ is much larger than $N_c$ (e.g., $N_p > 10N_c$) so that the redundant computation represents a small fraction of the total cost.

Taken together (3.5)-(3.7) indicate that for good performance, the difficulty of the problem must in some sense scale in proportion to the number of processors. That is, "simple" problems with only a few significant features that need to be resolved through refinement can most efficiently be solved using a few processors. Such problems could be handled on a small network of powerful workstations. To use our procedure effectively with, say $p = 128$, would require a more difficult problem which could be decomposed into at least 128 regions, with most including some interesting behavior to resolve.

**4. The Global Solution.** In this section we discuss some options for combining the independent calculations to form a global composite mesh and solution.

**4.1. Conforming Meshes and Global Smoothing.** In PLTMG, we have implemented an option where the independently generated meshes are glued together to form a global conforming mesh. We begin by simply deleting unwanted parts of the mesh from each of the independently solved problems. The union of the remaining submeshes forms a global mesh. Mesh points on the interfaces are restricted to move only along the interface during mesh smoothing phases of the independent solution process, so the overall geometry of the the subregions remains conforming. However, along the interfaces the meshes will generally be nonmatching, leading to a global nonconforming triangulation, as shown in Figure 4.1. This mesh is made globally conforming through a combination of the refinement of triangles with boundary edges on the interface, and adjusting the locations of some interface vertices. The result is a matching grid along the interface and a globally conforming triangulation. These alternatives are illustrated in Figure 4.1. Once the mesh is globally conforming, local mesh improvement routines of PLTMG (e.g., edge swapping and mesh smoothing) can be employed to improve the shape regularity of the elements as needed.

We remark that the element sizes on both sides of an interface should be approximately equal if the load balancing algorithm is working properly, so that this gluing process, although technically complicated in terms of data structures and implementation, is both simple and natural at a more abstract level.

If one uses a refined element tree data structure for the refinement process [3, 5], as in previous versions of PLTMG, then this procedure is greatly simplified, since each independent problem starts from the same element tree. A simple post processing step can enforce equal (and hence conforming) levels of refinement for elements sharing an interface edge. Also, if mesh smoothing is disallowed for vertices on the interface, the post processing is further simplified, since each interface vertex created during the independent solution process will be exactly the midpoint of the edge it refines. Assuming the refinement levels are approximately the same on both sides of an interface, it is likely that most vertices along an interface from one independent problem will have matching counterparts on the other side of the interface from another independent problem. Furthermore, the locations of nonmatching vertices cannot be arbitrary, but must lie e.g., at midpoints of unrefined edges. Such additional constraints, if present in the refinement algorithm, greatly simplify the algorithm for rendering a conforming mesh.

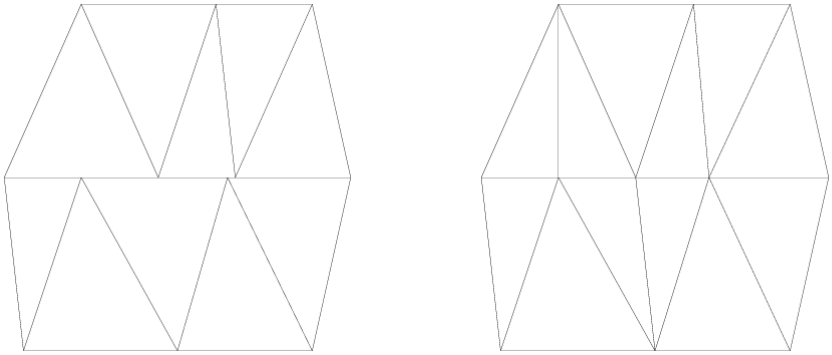


FIG. 4.1. *Nonmatching interface, shown on the left, is made conforming by refining elements and adjusting vertex locations. The resulting conforming interface is shown on the right.*

In any event, once a conforming mesh is created, a solution on the globally refined mesh can be defined as follows. For vertices not on the interface, we use the solution from the independent problem corresponding to that region. For vertices on the interface, the solution is defined by a simple averaging of the solutions from the relevant independent problems. This procedure seems adequate if the resulting solution is to serve only as an initial guess for a subsequent global solution technique. However, one could conceive of more sophisticated and more accurate procedures. For example, one could assemble and solve a problem for the interface values, holding all other values fixed. This results in a low-dimensional system consisting mainly of tridiagonal blocks, with some interblock coupling due to so-called *cross points*. One could also include points adjacent or near to the interfaces as well, yielding larger, but still easily solved systems of equations.

In our present code, all postprocessing is done on one processor, following the solution phase. There are several ways one could make this parallel. For example, one could initially have several processors each glue together the results from two subproblems. These larger pieces then could be combined (in parallel) to make composite meshes arising from four subproblems. Continuing in this fashion leads to an overall algorithm with logarithmic complexity. However, we prefer a simple alternative strategy in which some processors solve subproblems while other processors simultaneously postprocess results from earlier computations.

An example is given in Table 4.1. Here we assume that the domain is partitioned into 16 subproblems ($P_1 - P_{16}$) to be distributed among four processors. Rather than provide four problems to each processor, we provide five problems to three of the processors and only one to the fourth. After solving its subproblem, this processor then begins the task of postprocessing the previously computed solutions. We remark that the cost of postprocessing a problem is usually less than 1/3 the cost of solving it. Indeed it was much smaller in both examples given in Table 3.1. This ratio was assumed for convenience in this illustration; in practice, one might assign Processor 0 several subproblems before switching to postprocessing calculations.

To compute the global solution, one can of course simply assemble and solve the global problem on a single processor. This was the choice made in PLTMG, but it was made mainly to maximize the use of the existing code, rather than for reasons of efficiency. More in keeping with the overall strategy of parallel processing, there are a variety of standard domain decomposition approaches that could be applied in this situation, both with and without overlap. The relevant decomposition and communication channels can be based on the same partitioning used to create the independent problems, which greatly simplifies the implementation. One could also apply parallel multigrid or another parallel iterative method to this situation. Here the parallel multigrid method of Mitchell [20, 21, 22] seems particularly appropriate. In any event, by creating a good initial guess from the solutions of the independent problems, very little work (e.g. few iterations) should be required to compute the global solution.

**4.2. Mortar elements.** In the 3D case, producing a global conforming mesh is much more problematic, in that face matching simply through bisection is not achievable as it is in the 2D case. This is a well-known problem, and impacts adaptive tetrahedral subdivision algorithms based on octasection of tetrahedra [39, 24, 9]. We consider two approaches for dealing with this difficultly in the present context. The first of these is the mortar elements, discussed here; the second approach is described in the next section.

| Processor 0 | Processor 1 | Processor 2 | Processor 3 |
|---|---|---|---|
| Load Balance | – | – | – |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Solve $P_1$ | Solve $P_2$ | Solve $P_3$ | Solve $P_4$ |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Glue $P_2 - P_4$ | Solve $P_5$ | Solve $P_6$ | Solve $P_7$ |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Glue $P_5 - P_7$ | Solve $P_8$ | Solve $P_9$ | Solve $P_{10}$ |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Glue $P_8 - P_{10}$ | Solve $P_{11}$ | Solve $P_{12}$ | Solve $P_{13}$ |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Glue $P_{11} - P_{13}$ | Solve $P_{14}$ | Solve $P_{15}$ | Solve $P_{16}$ |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Glue $P_{14} - P_{16}$ | – | – | – |
| ⇓ | ⇓ | ⇓ | ⇓ |
| Global Solve | | | |

TABLE 4.1
*One possible strategy for overlapping the postprocessing and the solution phases.*

One approach for making a global solution from subdomain solutions on nonconforming subdomains is to simply use the global nonconforming mesh and establish weak continuity of the solution on the interface using so-called *mortar elements* [8, 7]. Although originally developed as a technique to couple spectral and finite element methods, it can be used to couple finite element discretizations which are conforming within subdomains, but have non-matching meshes at the interfaces of the subdomains.

To keep the discussion simple, suppose that there are only two subregions with a single interface $\Gamma$. Let $u_h^{(1)}$ and $u_h^{(2)}$ denote the approximate solutions for the two subregions. Rather than forcing the mesh along the interface to become conforming, we impose continuity of the computed solution weakly, as

$$\int_\Gamma (u_h^{(1)} - u_h^{(1)})\phi \, dx = 0 \quad \text{for all } \phi \in \mathcal{V}, \tag{4.1}$$

where $\mathcal{V}$ is some suitably chosen mortar space. See [8, 7, 1, 6] for details on the the selection of $\mathcal{V}$. When assembled, the resulting system of linear equations will have the classic saddle point structure

$$\begin{pmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^t & B_2^t & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ 0 \end{pmatrix} \tag{4.2}$$

where as usual $A_i$ correspond to individual subregions and $B_i$ corresponds to the coupling of the subregions through the mortar space $\mathcal{V}$. The space $\mathcal{V}$ plays the role of Lagrange multipliers in the saddle point problem, and the element of $\mathcal{V}$ corresponding to the solution of (4.2) has a physical interpretation in terms of an approximation to the normal component of $\nabla u$ on $\Gamma$.

From its structure, it is clear that one may apply classical domain decomposition techniques to the solution of (4.2) (and the more general case of many subregions). All of the information related to each subdomain is already present on the processor responsible for adaptively creating that portion of the composite mesh. Communication between processors is necessary for coupling through the mortar elements, but as usual, the required information is related to the solution and the structure of the mesh only along the interface, generally a small amount of data in comparison with the size of the subdomains.

**4.3. Overlapping decompositions and interior estimates.** The simplest way to form a global solution is not to form one, meaning that the subdomain solutions themselves are taken to be the final discrete solution represented subdomain-wise. To evaluate the solution at any point $x \in \bar{\Omega}$, one simply must determine which subdomain contains the point $x$, and then fetch the solution value from the particular subdomain. While this approach seems naive, some recent [37] and not so recent [23, 28, 29] theoretical results actually support this.

The principle idea underlying the results in [37] is that while elliptic problems are globally coupled, this global coupling is essentially a "low-frequency" coupling, and can be handled on a mesh which is much coarser than that required for approximation accuracy considerations. This idea has been exploited for example in [19, 36], and is in fact why the construction of a coarse problem in overlapping domain decomposition methods is the key to obtaining convergence rates which are independent of the number of subdomains (cf. [35]).

The key results in [37] for our purposes are the following *a priori* and *a posteriori* error estimates. To explain, let $\Omega_k$ be the collection of disjoint subregions of the domain $\Omega$ defined by the weighted spectral bisection algorithm of the previous section, and let $\Omega_k^0$ to be an extension of the disjoint $\Omega_k$, such that $\Omega_k \subset\subset \Omega_k^0$, and so that the sizes of the overlap regions $\Omega_i^0 \bigcap \Omega_j^0$ are on the order of the sizes of the regions $\Omega_k$. Under some reasonable assumptions about the approximation properties of a finite element space $S_0^h$ defined over $\Omega$ (existence of superapproximation, inverse, and trace inequalities), the following *a priori* error estimate holds for the global Galerkin solution $u_h$ to a Poisson-like linear elliptic equation:

$$\|u - u_h\|_{H^1(\Omega_k)} \leq C \left( \inf_{v \in S_0^h} \|u - v\|_{H^1(\Omega_k^0)} + \|u - u_h\|_{L^2(\Omega)} \right),$$

and the following *a posteriori* error estimate holds (where $\eta(u_h)$ is a locally computable jump function):

$$\|u - u_h\|_{H^1(\Omega_k)} \leq C \left( \|h\eta(u_h)\|_{L^2(\Omega_k^0)} + \|u - u_h\|_{L^2(\Omega)} \right).$$

The *a priori* result states that the error in the global Galerkin solution $u_h$ restricted to a subdomain $\Omega_k$ can be bounded by the error in the best approximation from the finite element space $S_0^h$ measured only over the extended subdomain $\Omega_k^0$, plus a higher-order global term (the global $L^2$-norm of the error). In the context of the algorithm in this paper, if the global coarse mesh is quasi-uniform and shape-regular with element diameter $H$, and if we assume that $u \in H^2(\Omega)$, then standard interpolation theory and $L^2$-lifting can be used to bound the global term by an $O(H^2)$ factor. Moreover, if the mesh produced by adaptivity in the extended subdomain $\Omega_k^0$ is again quasi-uniform and shape-regular, but now has element diameter $h$, then the local term can be bounded using standard interpolation theory by an $O(h)$ factor. If our adaptive method respects the relationship $h = O(H^2)$ between the coarse and refined regions, then asymptotically the global term based on the much coarser mesh outside $\Omega_k^0$ does not pollute the accuracy of the adapted solution in the subdomain $\Omega_k$. A similar argument can be applied in the less regular case of $u \in H^{1+\alpha}(\Omega)$.

The *a posteriori* estimate states that the error in the global Galerkin solution restricted to a subdomain $\Omega_k$ can be estimated in terms of a (computable) jump function $\eta(\cdot)$ over the extended subdomain $\Omega_k^0$, plus a higher-order term (the global $L^2$-norm of the error). Through the same argument above, this means that asymptotically, the global error can be controlled by the local computable jump function estimate in each subdomain, so that reliable *a posteriori* error estimates can be computed in isolation from the other subdomains.

The *a priori* and *a posteriori* estimates from [37] outlined above were derived for self-adjoint linear problems in the plane, and as a result do not apply directly to the examples presented in this paper (nonlinear scalar problems and elliptic systems in both two and three dimensions). Moreover, the local refinement strategy described here tends to produce very little overlap of the extended subdomains $\Omega_k^0$, violating one of the key assumptions in [37], and we do not explicitly enforce a refinement limitation such as $h = O(H^2)$. However, the estimates in [37] indicate that this approach will likely provide a very good initial approximation to an overlapping domain decomposition procedure for solving the final global problem, and in an ideal situation (certain classes of elliptic problems with large subdomain overlap), it might be possible to skip the global solution altogether.

## REFERENCES


[1] Y. Achdou, Y. Kuznetsov, and O. Pironneau, *Substrucuring preconditioners for the Q1 mortar element method*, Numer. Math., 71 (1995), pp. 419–449.

[2] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*, Software, Environments and Tools, Vol. 5, SIAM, Philadelphia, 1998.

[3] 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.
[4] R. E. Bank and R. K. Smith, *Mesh smoothing using a posteriori error estimates*, SIAM J. Numerical Analysis, 34 (1997), pp. 979–997.
[5] M. W. Beall and M. S. Shephard, *A general topology-based mesh data structure*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 1573–1596.
[6] F. Belgacem, *The mortar finite element method with Lagrange multipliers*, 1997. Preprint.
[7] C. Bernardi, N. Debit, and Y. Maday, *Coupling finite element and spectral methods: first results*, Math. Comp., 54 (1990).
[8] 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.
[9] J. Bey, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378.
[10] X. Cai and K. Samuelsson, *Parallel multilevel methods with adaptivity on unstructured grids*, 1999. Preprint.
[11] T. F. Chan, P. Ciarlet, and W. K. Szeto, *On the optimality of the median cut spectral bisection method*, SIAM J. Sci. Comput., 18 (1997), pp. 943–948.
[12] H. L. deCougny, K. D. Devine, J. E. Flaherty, R. M. Loy, C. Ozturan, and M. S. Shephard, *Load balancing for the parallel adaptive solution of partial differential equations*, Appl. Num. Math., 16 (1994), pp. 157–182.
[13] 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.
[14] G. Fox, R. Williams, and P. Messina, *Parallel Computing Works!*, Morgan-Kaufmann, San Francisco, 1994.
[15] M. Holst, *Adaptive multilevel finite element methods on manifolds and their implementation in MC*. (In preparation; currently available as a technical report and User's Guide to the MC software).
[16] M. Holst and D. Bernstein, *Finite element solution of the initial-value problem in general relativity: Theory and algorithms*, Comm. Math. Phys., (Submitted in Sept 1999).
[17] S. Kohn, J. Weare, M. E. Ong, and S. B. Baden, *Software abstractions and computational issues in parallel structured adaptive mesh methods for electronic structure calculations*, in Workshop on Structured Adaptive Mesh Refinement Grid Methods, Institute for Mathematics and Its Applications, University of Minnesota, Minneapolis, MN., 1997, p. to appear.
[18] A. Liu and B. Joe, *Relationship between tetrahedron shape measures*, BIT, 34 (1994), pp. 268–287.
[19] M. Marion and J. Xu, *Error estimates on a new nonlinear Galerkin method based on two-grid finite elements*, SIAM J. Numer. Anal., 32 (1995), pp. 1170–1184.
[20] W. Mitchell, *The full domain partition approach to distributing adaptive grids*, Applied Numerical Mathematics, 26 (1998), pp. 265–275.
[21] ———, *A parallel multigrid method using the full domain partition*, Electronic Transactions on Numerical Analysis, 6 (1998), pp. 224–233.
[22] ———, *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, to appear.
[23] J. A. Nitsche and A. H. Schatz, *Interior estimates for Ritz-Galerkin methods*, Math. Comp., 28 (1974), pp. 937–958.
[24] E. G. Ong, *Uniform refinement of a tetrahedron*, Tech. Rep. CAM 91-01, Department of Mathematics, UCLA, 1991.
[25] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, New Jersey, 1980.
[26] A. K. Patra and D. W. Kim, *Efficient mesh partitioning for adaptive hp finite element meshes*, in Eleventh International Conference on Domain Decomposition Methods, C.-H. Lai, P. E. Björstad, M. Cross, and O. B. Widlund, eds., 1998.
[27] A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.
[28] A. H. Schatz and L. B. Wahlbin, *Interior maximum-norm estimates for finite element methods*, Math. Comp., 31 (1977), pp. 414–442.
[29] ———, *Interior maximum-norm estimates for finite element methods II*, Math. Comp., 64 (1995), pp. 907–928.
[30] 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.
[31] H. D. Simon and S.-H. Teng, *How good is recursive bisection?*, SIAM J. Sci. Comput., 18 (1997), pp. 1436–1445.
[32] R. Verfürth, *A Posteriori Error Estimation and Adaptive Mesh Refinement Techniques*, Teubner Skripten zur Numerik, B. G. Teubner, Stuttgart, 1995.
[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] R. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency, 3 (1991), p. 457.
[35] J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM Rev., 34 (1992), pp. 581–613.
[36] ———, *Two-grid discretization techniques for linear and nonlinear PDEs*, SIAM J. Numer. Anal., 33 (1996), pp. 1759–1777.
[37] J. Xu and A. Zhou, *Local and Parallel Finite Element Algorithms Based on Two-Grid Discretizations*, 1998. Preprint.
[38] J. W. York, *Kinematics and dynamics of general relativity*, in Sources of Gravitational Radiation, L. L. Smarr, ed., Cambridge University Press, Cambridge, MA, 1979, pp. 83–126.
[39] S. Zhang, *Multi-level Iterative Techniques*, PhD thesis, Dept. of Mathematics, Pennsylvania State University, 1988.