SIAM Rev.
Vol. 45, No. 2, pp. 000–000

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 that 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 [J. Xu and A. Zhou, *Math. Comp.*, to appear] and not so recent [J. A. Nitsche and A. H. Schatz, *Math. Comp.*, 28 (1974), pp. 937–958; A. H. Schatz and L. B. Wahlbin, *Math. Comp.*, 31 (1977), pp. 414–442; A. H. Schatz and L. B. Wahlbin, *Math. Comp.*, 64 (1995), pp. 907–928] results on local a priori and a posteriori error estimation. This revision of the original article [R. E. Bank and M. J. Holst, SIAM J. Sci. Comput., 22 (2000), pp. 1411–1443] updates the numerical experiments, and reflects the knowledge we have gained since the original paper appeared.

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

AMS subject classifications. 65M55, 65N55

PII. S0000000000000000

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 [5] and MC [24] 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 [21, 22, 23, 27, 40, 43]; most of the dynamic strategies involve repeated application of a particular static strategy. One of

*Received by the editors April 7, 1999; accepted for publication (in revised form) December 6, 1999; published electronically November 2, 2000. The work of the first author was supported by the National Science Foundation under contracts DMS-9706090, DMS-9973276, and DMS-0208449. The work of the second author was supported by the National Science Foundation under CAREER award 9875856 and under contracts DMS-9973276 and DMS-0208449. 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.

http://www.siam.org/journals/sirev/45-2/00000.html

†Department of Mathematics, University of California at San Diego, La Jolla, CA 92093 (rbank@ucsd.edu, mholst@math.ucsd.edu).

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 [44]. 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 that 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:

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, that 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 [30, 31, 32] on parallel multigrid methods. A similar approach appeared recently in [18]. The use of a posteriori error estimates in mesh partitioning strategies has also been considered in [36].

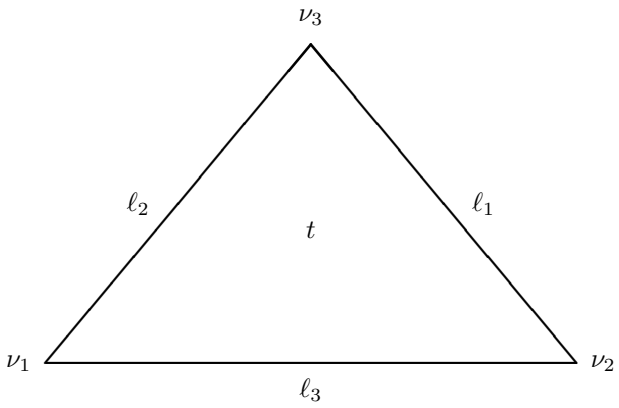


FIG. 1. *A typical element.*

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, that 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 preprocessing 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)

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

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

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

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

Equations (1.1)–(1.5) are derived by comparing the approximation error for linear and quadratic interpolation on t. See [5, 11] for details. The second derivatives in the 2×2 matrix M_t are taken as constants on t. The values of the second derivatives are extracted as a postprocessing step from the a posteriori error estimates. The remaining information needed to compute 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 toward 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 3×3 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 18×18 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., [42]. 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 that form the foundation for adaptive algorithms (see, e.g., the book of Verfürth [42] 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

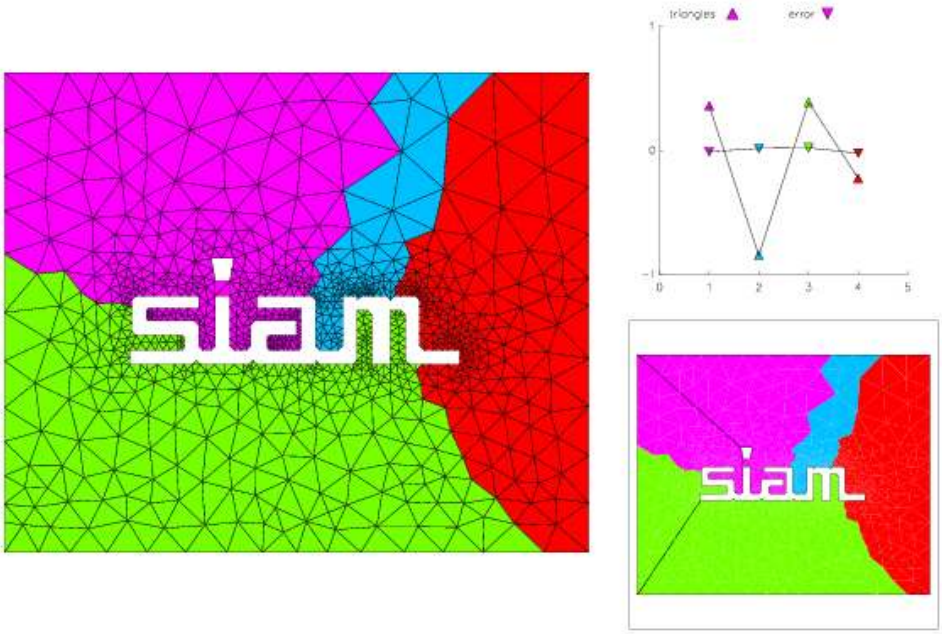
$$\begin{aligned} -\nabla \cdot (\nabla u + \beta u) &= 1 \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{2.1}$$

where $\beta = (0, 10^5)^t$, and Ω is the region depicted in Figure 2. This coarse triangulation has 1676 elements and 1000 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 4000 unknowns (located at triangle vertices) is obtained. The mesh for one subdomain and the corresponding solution is shown in Figure 3. 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 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 24410 triangles and 12808 vertices. This mesh is shown in Figure 4. The final global solution generated by our domain decomposition/multigraph solver [9, 8, 12] is also shown in Figure 4.

We next repeated this experiment but in a more realistic setting. This time the coarse mesh had 15153 triangles and 8000 vertices, and was partitioned into



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

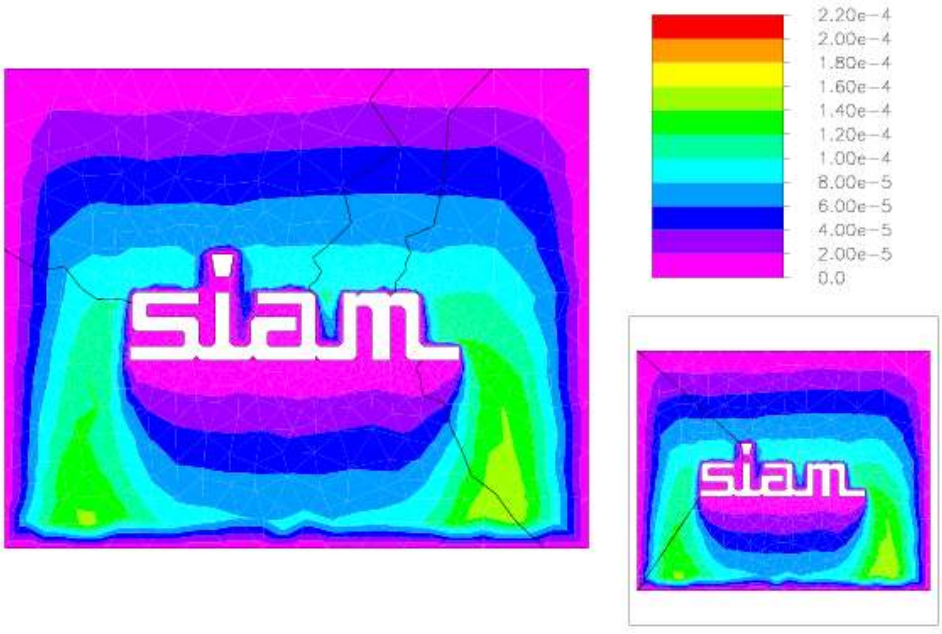
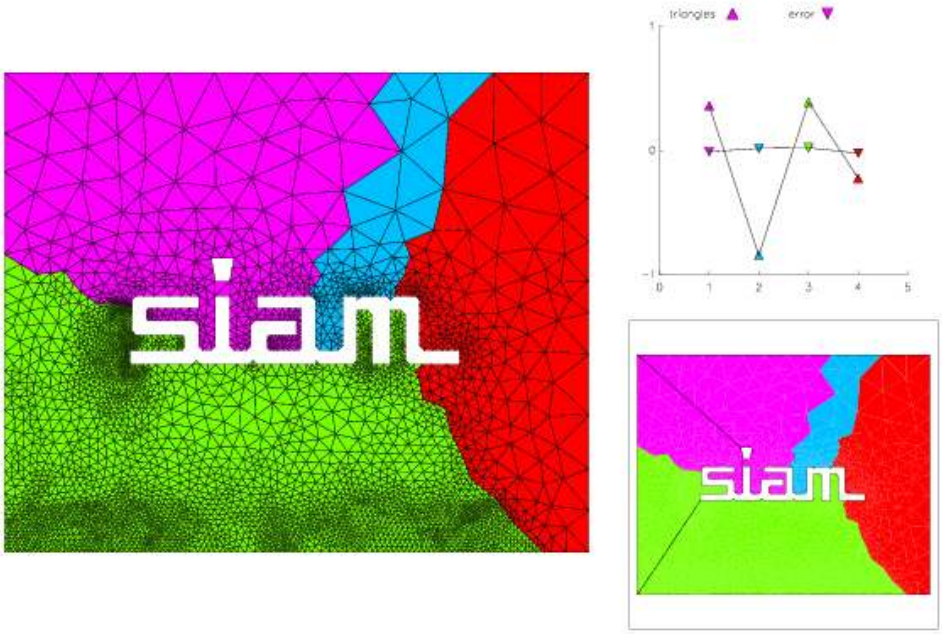


FIG. 2. *The coarse grid solution.*



The refined mesh for problem 3.

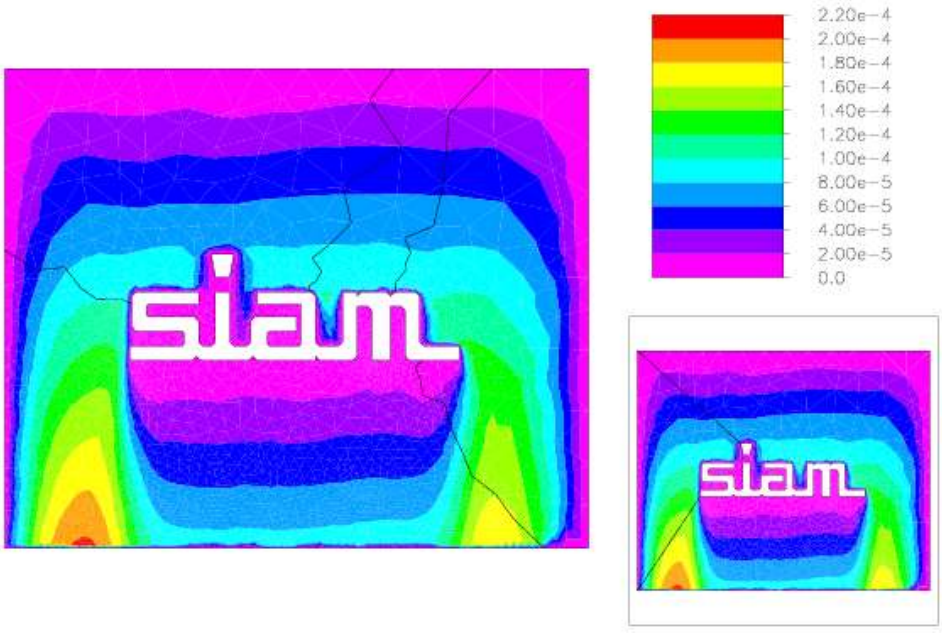
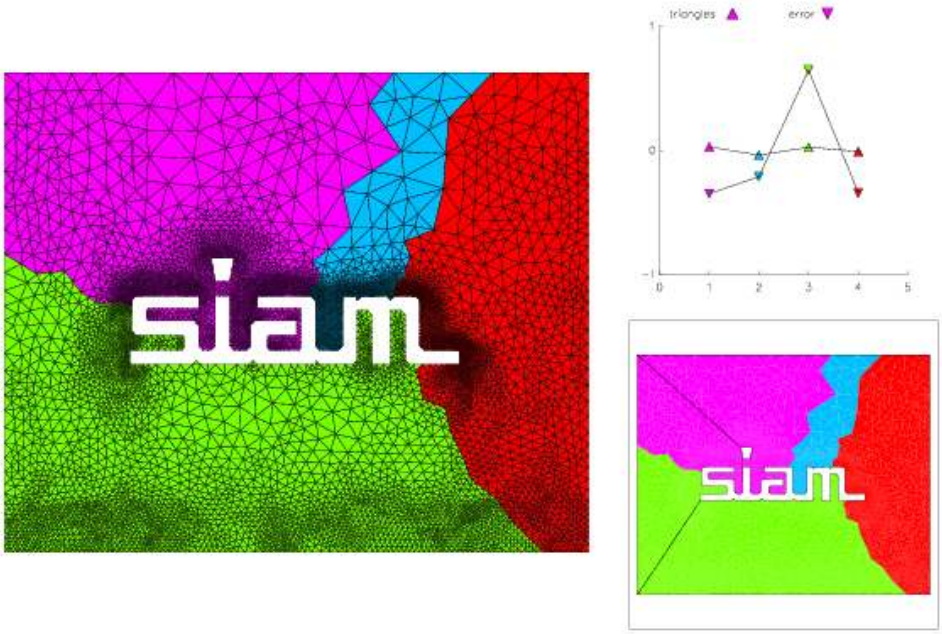


FIG. 3. *The solution for problem* 3.



The global refined mesh.

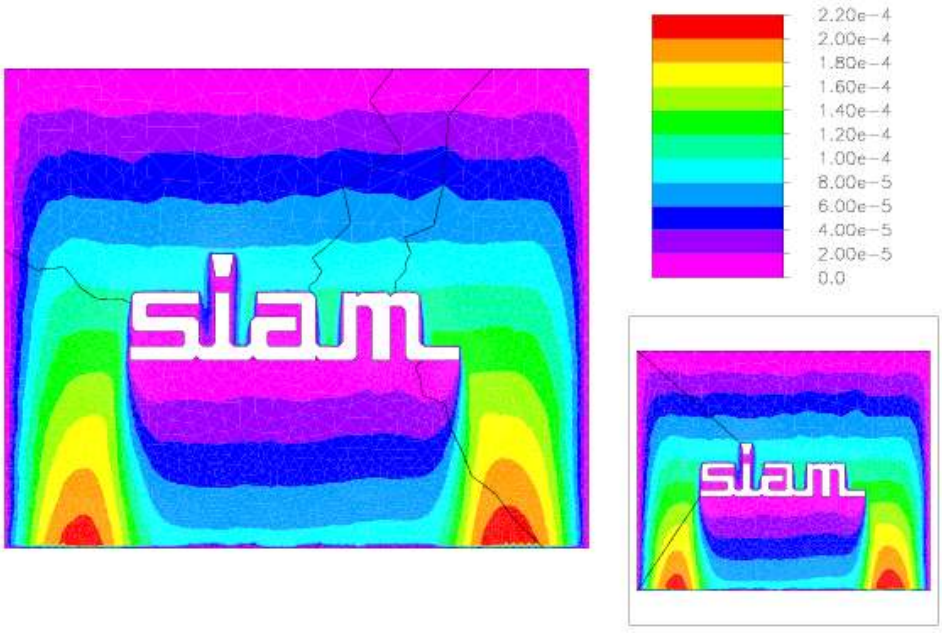


FIG. 4. *The final global solution.*

16 subregions. Each processor then adaptively solved the problem with a target value of 100000 vertices. The composite fine grid solution had 2883660 elements and 1467973 vertices. The final global solution required 5 iterations of our parallel domain decomposition/multigraph procedure.

With approximately 2.9 million elements, one can not display the mesh with triangle edges drawn (indeed we could not even collect the entire mesh on one processor for graphics processing). Thus we rely on color to indicate element size. In Figure 5, we show the original load balance partition and the final mesh. Here we see that the initial load balance created partitions of widely differing size. However, in the final mesh we note that elements along interfaces appear to vary smoothly in size, indicating that the meshes generated on each processor were largely compatible despite the lack of communication.

In Figure 7, we show the final solution and the a posteriori error estimate. The uniformity of color in the a posteriori error graph indicates that our strategy did a reasonable job of globally equilibrating the error despite the lack of communication in the mesh generation phase.

2.2. An elliptic variational inequality. For our second example, we use PLTMG to solve the variational inequality

$$\min_{u\in K} \int_\Omega |\nabla u|^2 - 2f(x)u\,dx \tag{2.2}$$

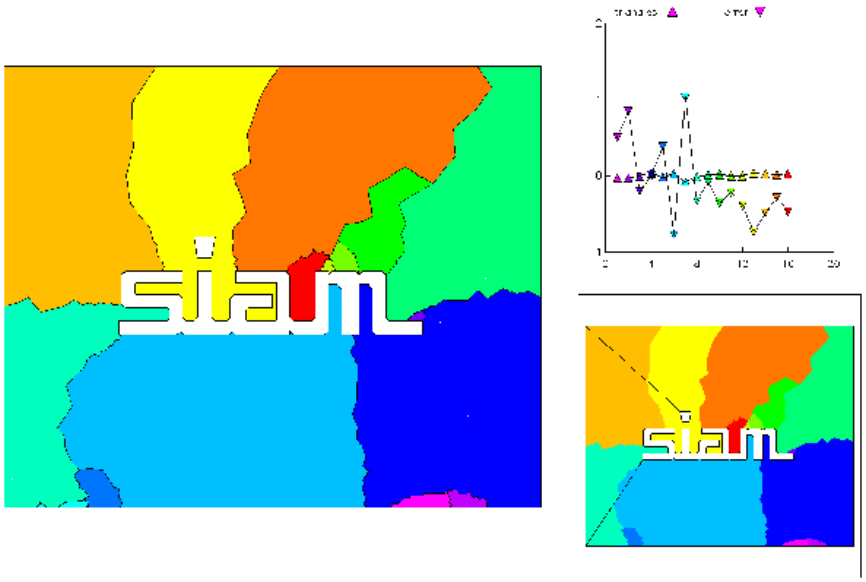
where

$$\Omega = (0,1)\times(0,1),$$
$$K = \left\{ u \in \mathcal{H}_0^1(\Omega) : |u| \le \frac{1}{4} - \frac{1}{10}\sin(\pi x_1)\sin(\pi x_2) \ \text{ for } x \in \Omega \right\},$$
$$f(x) = -\Delta(\sin(3\pi x_1)\sin(3\pi x_2)).$$

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 the simple linear system solves of the previous example. 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 same multigraph technique as the first example. See [6] for details.

In this example, we began with a uniform 3×3 mesh, and adaptively generated a mesh with 8000 vertices and 15676 elements. This calculation was done on a single processor, and corresponds to Step 1 of our paradigm. The mesh with 8000 vertices was then partitioned into 16 subregions on the basis of equal error. Each processor then continued with Step 2 of the paradigm, with a target value of 100000 vertices, as in the first example. The global fine mesh had 2838746 elements and 1429325 vertices. The final global solve was also an interior point iteration, with the domain decomposition/multigraph solver used for the resulting linear systems. The final global solve required 4 interior point iterations, starting from the initial guess provided by Step 2.

In Figure 8, we show the initial partition in to 16 subregions and the final global refined mesh. Here again the mesh matches very well along the interfaces, even where the interfaces cross the contact zones. We also note that the smallest elements appear



The initial partition into 16 *regions with approximately equal error.*

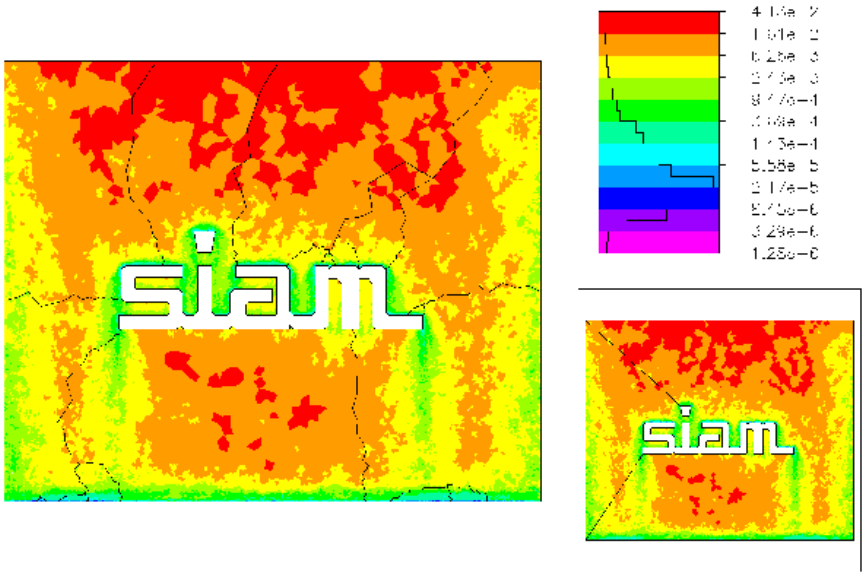


FIG. 5. *The final mesh. Color indicates element size.*

to resolve the details of the free boundary that defines the contact zones, and the largest elements appear in the centers of the contact zones.

In Figure 9, we show plots of the error and solution. The uniformity in color in the error shows the procedure did a reasonable job of equilibrating the error, although we again see evidence of the extra attention the adaptive procedure (automatically!) paid to the boundaries of the contact zones.

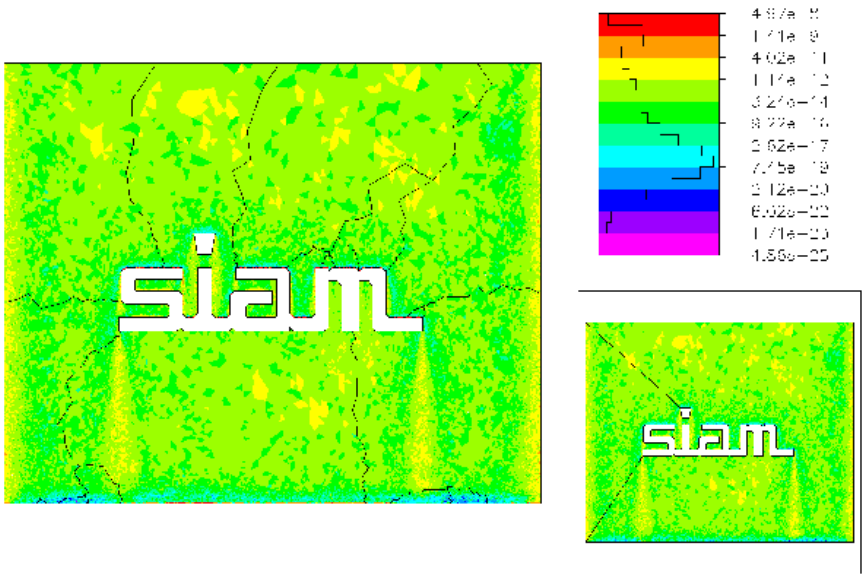


FIG. 6. *A posteriori error estimate for the final solution.*

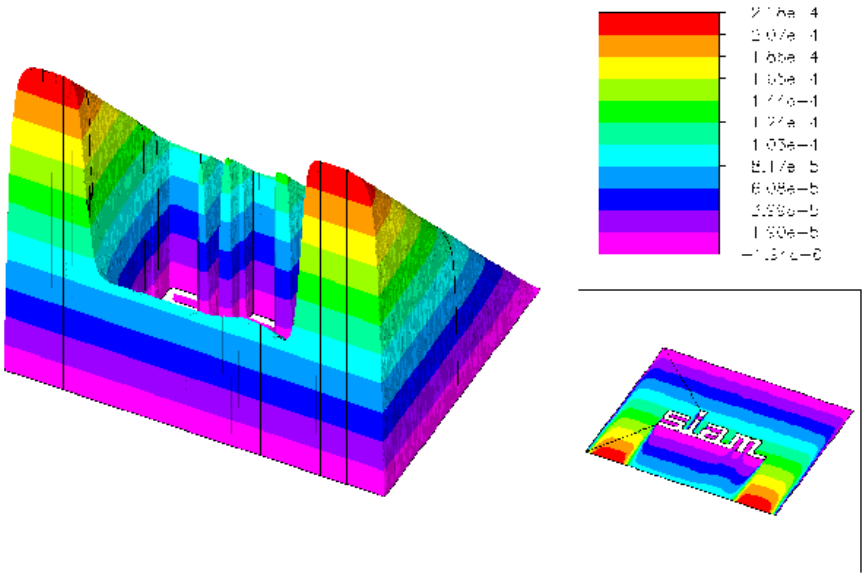
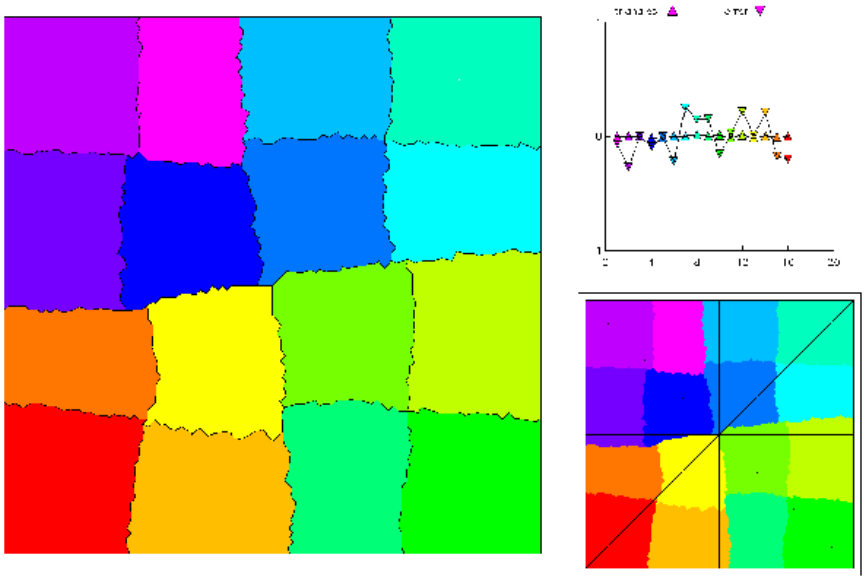


FIG. 7. *The final global solution.*



The initial partition into 16 *regions with approximately equal error.*

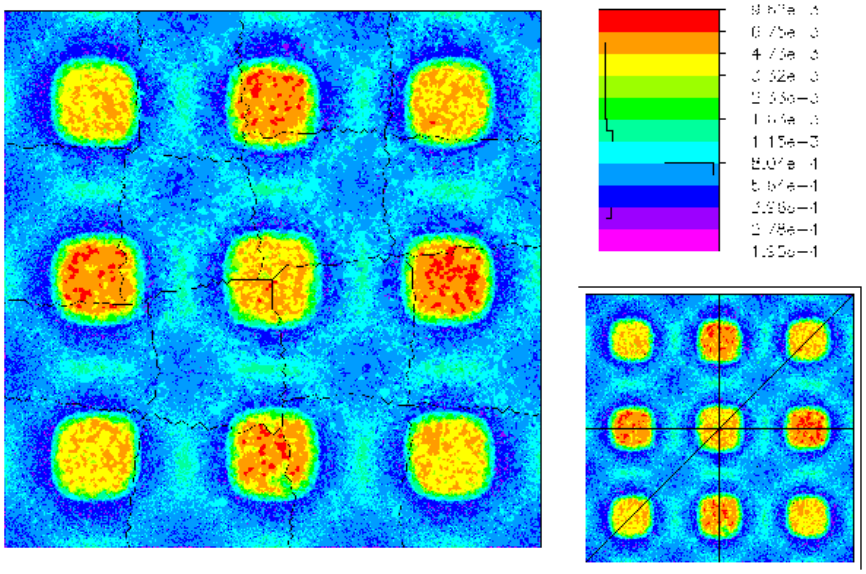
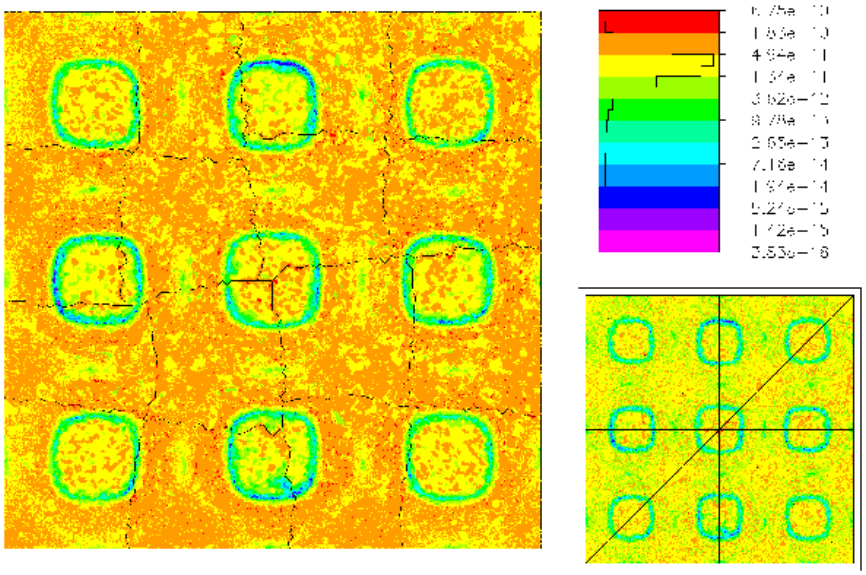


FIG. 8. *The final mesh. Color indicates element size.*



A posteriori error estimate for the final solution.

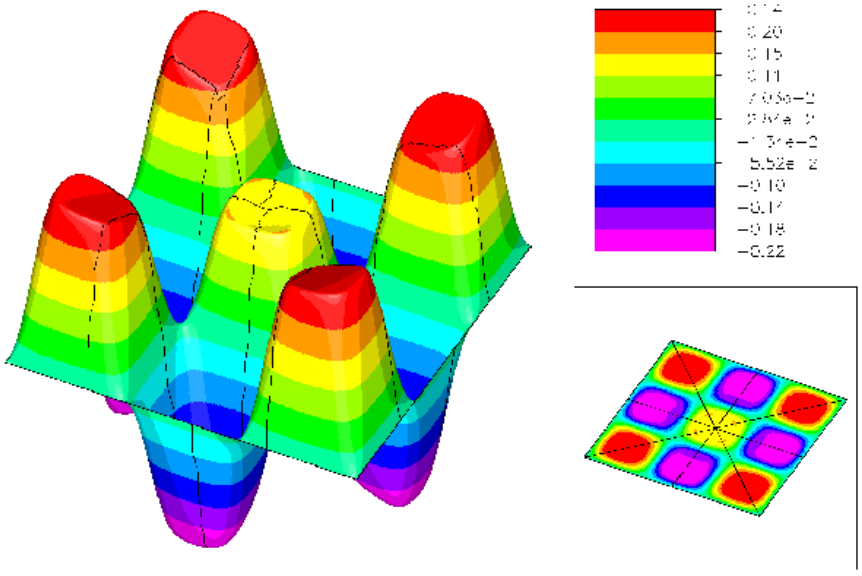


FIG. 9. *The final global solution.*

2.3. A 3D elasticity problem. The two previous examples demonstrated the effectiveness of the parallel algorithm for linear and nonlinear scalar problems and variational inequalities in 2D. To illustrate that the parallel algorithm works equally well for coupled elliptic systems and for 3D problems, for our third example we will use MC to solve the elasticity equations:

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

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

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

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

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

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

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

where the Lamé constants λ and μ 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 10. We apply a load to the top of each letter in downward direction, through the traction force function g above. The bottom surface of the common foundation is held fixed through the essential boundary condition above, leaving the rest of the solid structure to deform under the vertical load. 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.

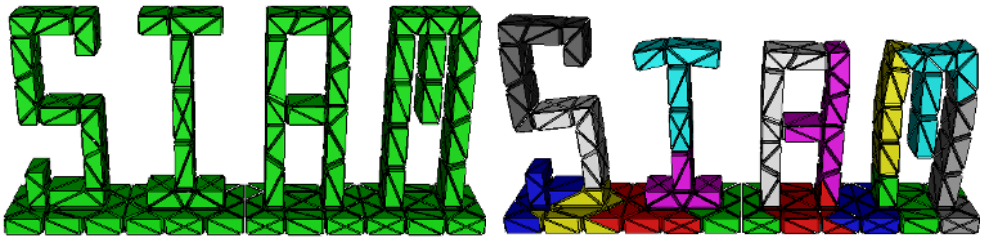


FIG. 10. *Coarse tetrahedrialization is on the left; on the right is the solution (as a deformation), showing the sixteen subdomains after four steps of weighted spectral bisection.*

The initial coarse mesh in the left picture of Figure 10 has 524 tetrahedral elements and 265 vertices. As in the previous examples, we partition the domain into sixteen subdomains with approximately equal error using the recursive spectral bisection algorithm described in section 3 below. The sixteen 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 10,000 vertices is obtained, giving about 30,000 degrees of freedom (the three coordinate deformations) on each processor. The union of the disjoint refined partitions then totals more than one million degrees of freedom.

The resulting refined subdomain mesh for each subdomain and the corresponding solution (plotted as a deformation of the solid) is shown in Figures 11–12. (The

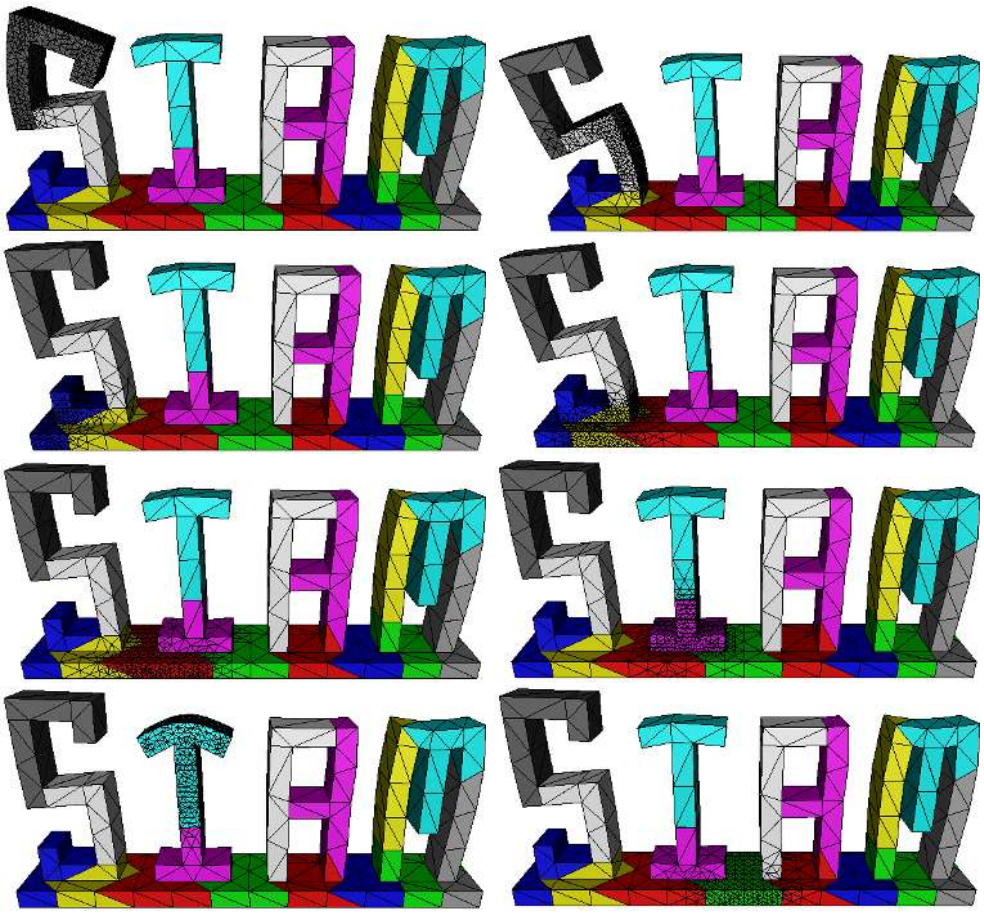


FIG. 11. *Solutions (as deformations) on the final adapted meshes, for subdomains 1 through 8 (left-to-right, then top-to-bottom).*

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 that maintains conformity and shape regularity. The sixteen problems are solved completely independently by the sequential adaptive code MC.

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 $\mathbb{R}^3$, namely, the elliptic constraints in the Einstein equations (cf. [48]):

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

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

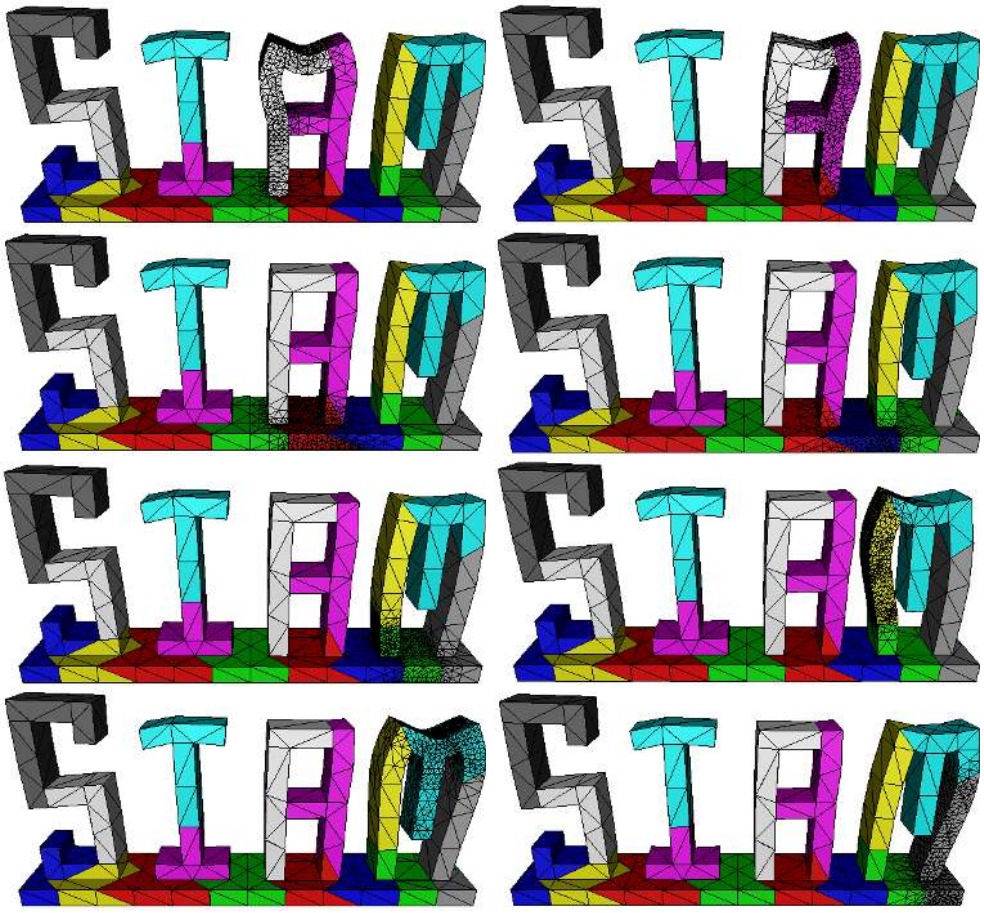


FIG. 12. *Solutions (as deformations) on the final adapted meshes, for subdomains 9 through 16 (left-to-right, then top-to-bottom).*

FIG. 13. *An adapted mesh and solution (as a deformation), illustrating the adaptivity pattern generated by a normal global (sequential, conforming) approach on a single processor for comparison.*

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

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

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 ϕ 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 approximations.) Physically reasonable functions for remaining parameters appearing in the equations are used to completely specify the problem (cf. [26] 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 that 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 in [11]:

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

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 [11], with a slightly different normalization. For $d = 3$, this is the shape-measure given in [28], 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 Figures 14 and 15, generated using the procedure described above, has 31786 tetrahedral elements and 5809 vertices. We partition the domain into four subdomains (shown in Figure 16 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 25000 vertices is obtained (yielding

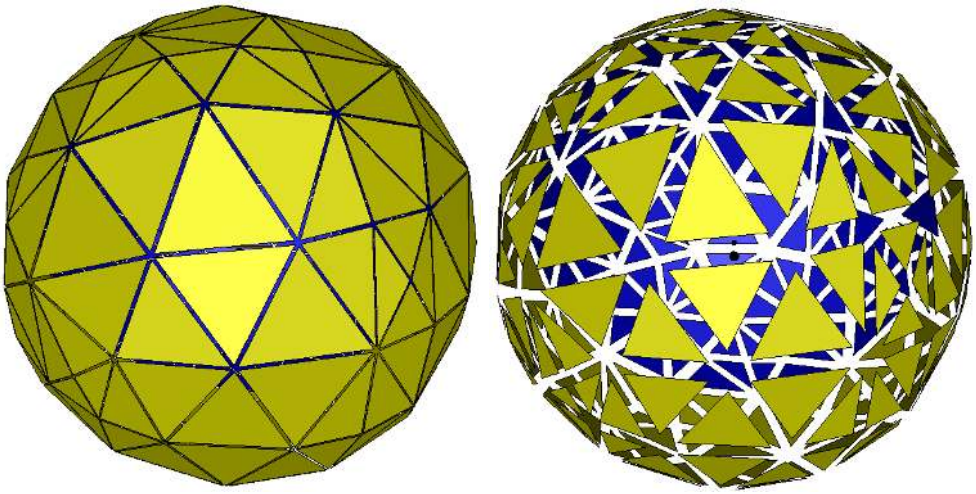


FIG. 14. *The coarse binary black hole mesh (*5809 *vertices and* 31786 *simplices).*

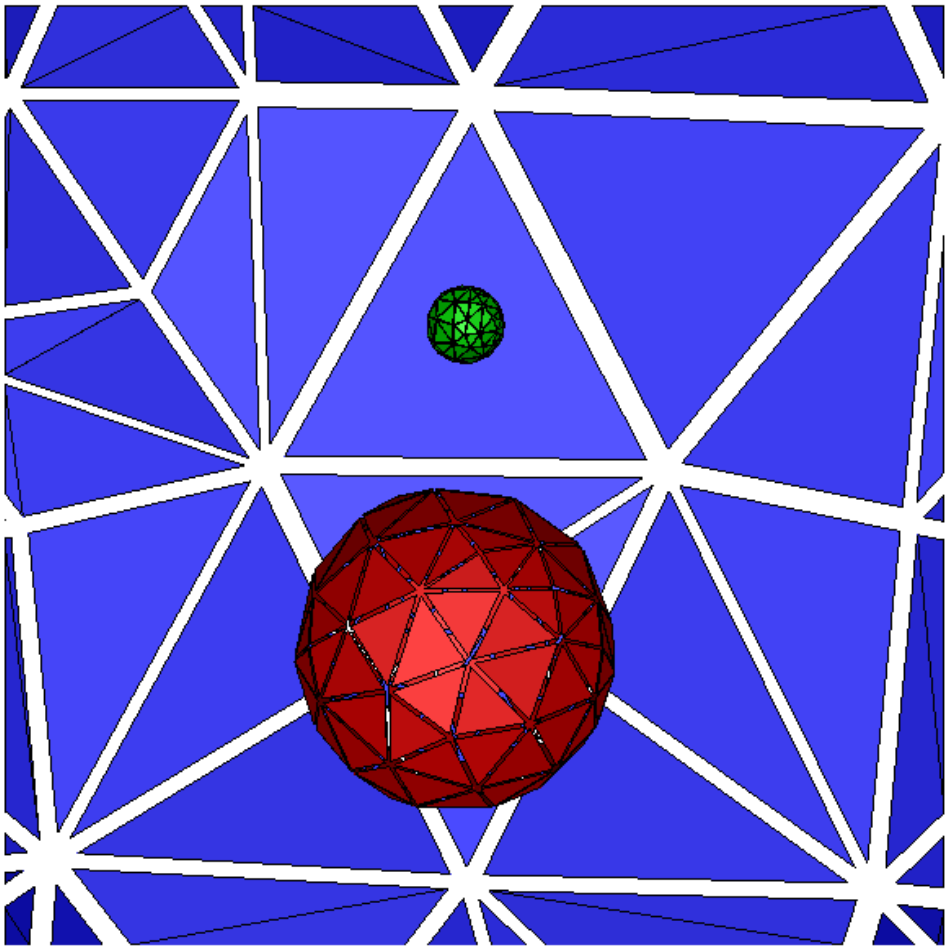


FIG. 15. *Exploded view of the coarse binary black hole mesh showing the two interior hole boundaries.*

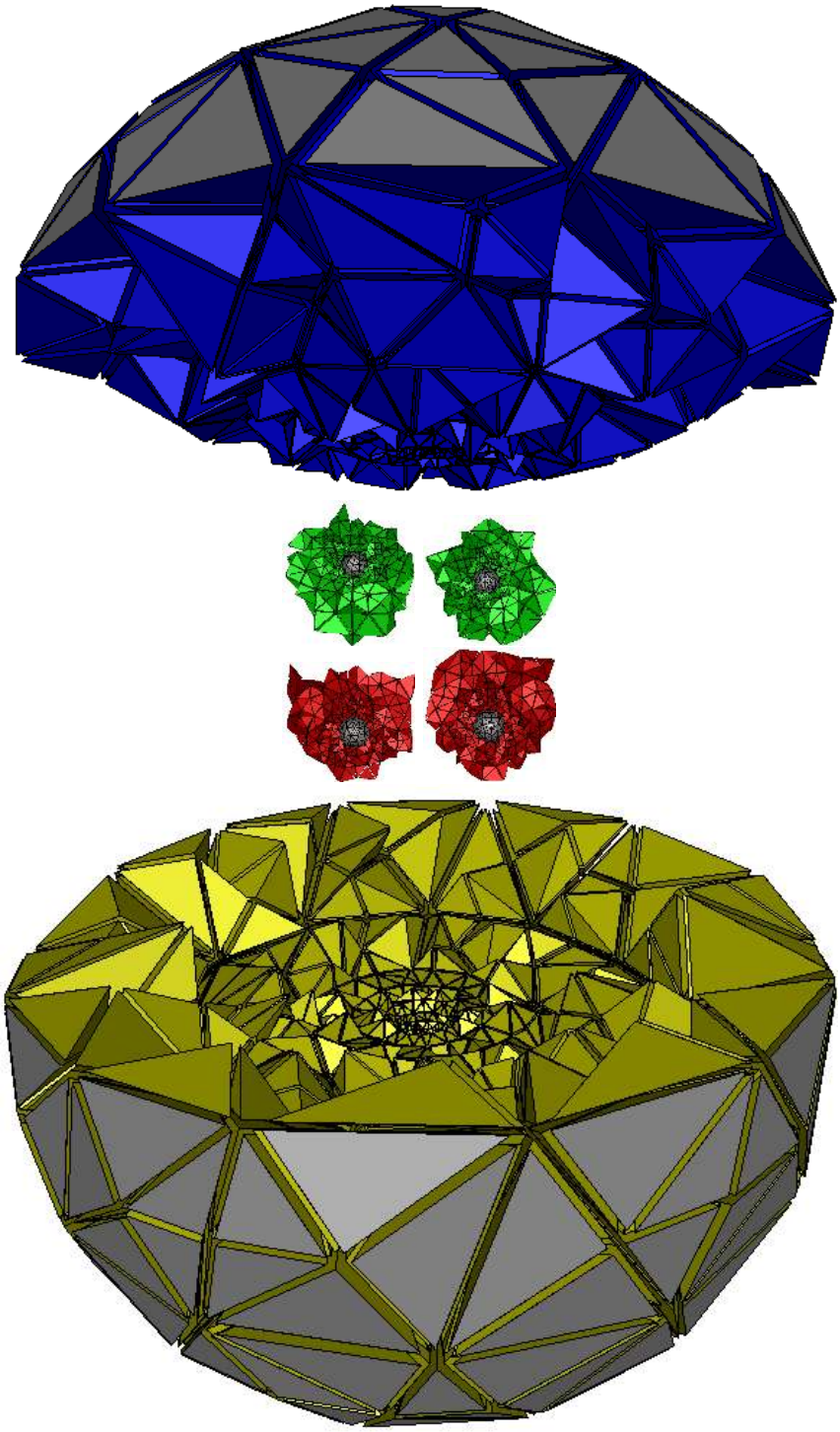


FIG. 16. *Subdomains* 1 *(green),* 2 *(red),* 3 *(blue), and* 4 *(yellow) from spectral bisection of the coarse binary black hole mesh. Exterior boundary and interior hole boundaries are depicted in gray. Each of the small subdomains* 1 *and* 2 *are sliced open to reveal the inner holes they enclose.*

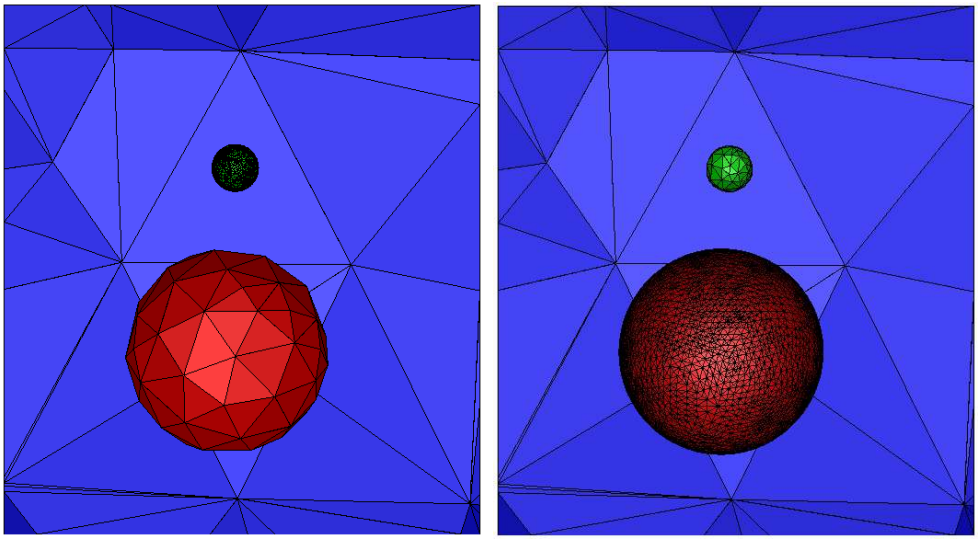


FIG. 17. *Refined mesh for subdomain* 1 *(*28999 *vertices and* 150607 *simplices) and subdomain* 2 *(*25732 *vertices and* 133886 *simplices); only faces of tetrahedra on the boundary surfaces are shown.*

subdomains with about 130000 simplices each), so that the subdomain problems each involve roughly 100000 degrees of freedom.

The resulting refined subdomain meshes are shown in Figure 17. 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 that 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 ϕ) of the elliptic system is depicted in Figure 18 (the subdomain 1 solution) and in Figure 19 (the subdomain 2 solution).

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 [19, 37, 41]. While this particular mesh partitioning algorithm is among the more expensive possibilities, we emphasize that it is used in our algorithm only once, on a small coarse grid problem. As a result, the initial partitioning cost is typically much smaller than the solve time for a single subdomain problem (see Tables 1 and 2 below).

Let $\mathcal{T}$ denote a triangulation of the domain Ω 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 semidefinite M-matrix A by

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

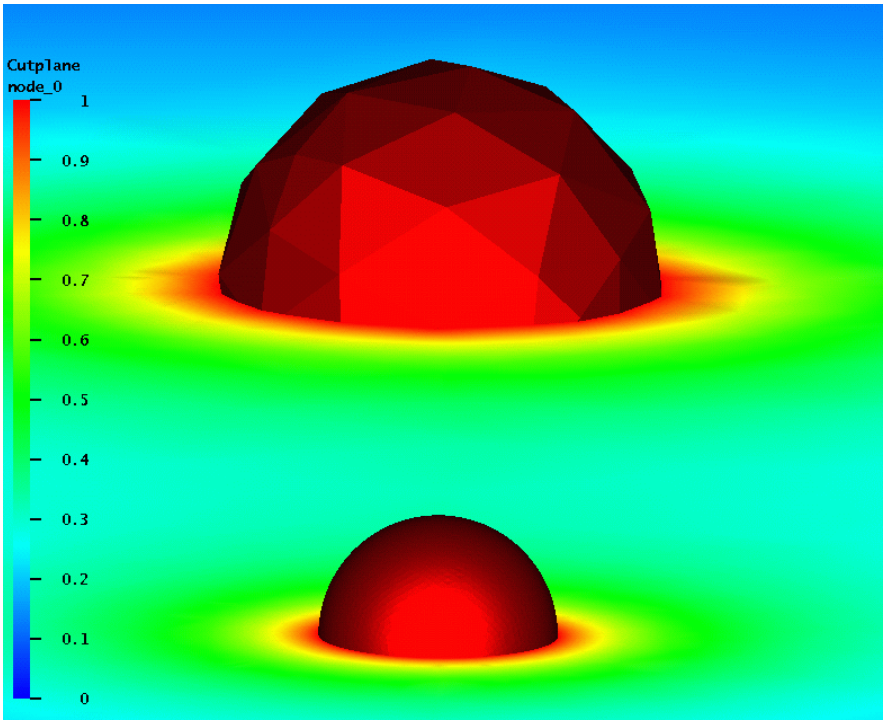


FIG. 18. *The conformal factor ϕ from the adapted subdomain* 1 *solve.*

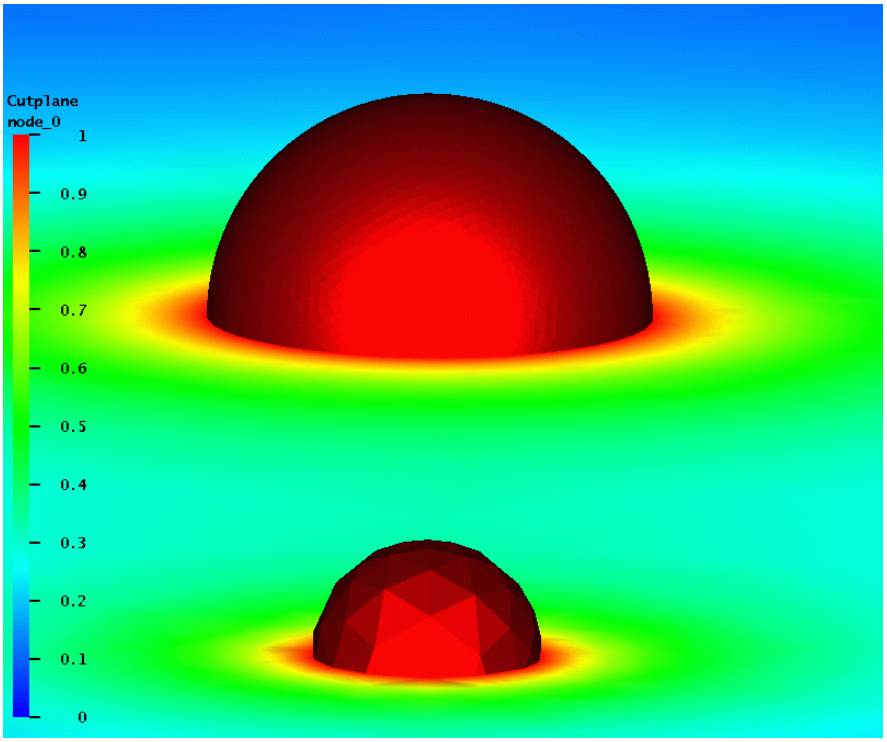


FIG. 19. *The conformal factor ϕ from the adapted subdomain* 2 *solve.*

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 eigenvalue problem

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

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 ψ_2, known as the Fiedler vector.

Let p denote the number of processors; we do not require $p = 2^\ell$. We use a standard binary tree with $2p - 1$ nodes (p leaves and $p - 1$ internal nodes). The root is labeled $i = 1$ and node i has children $2i$ and $2i + 1$, $1 \le i \le p - 1$. Associated with each node is a weight ω_i denoting the number of leaves contained in its subtree. In particular, $\omega_i = 1$, $i = 2p - 1, \ldots, p$ and $\omega_i = \omega_{2i} + \omega_{2i+1}$ for $i = p - 1, \ldots, 1$.

The entire mesh is assigned to root, and it is partitioned among its two children as follows. We first approximately solve the eigenvalue problem (3.1) for the whole mesh, and then create a permutation of the elements $\{q_i\}$ such that

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

We then find the index k that provides the best partition of the form

$$\frac{1}{\omega_2} \sum_{q_i \le k} e_i \approx \frac{1}{\omega_3} \sum_{q_i > k} e_i. \tag{3.2}$$

The corresponding submeshes are assigned to the children nodes. This is similar to the strategy suggested by Chan, Ciarlet, and Szeto [19].

As usual, we apply this approach recursively, at each level dividing each group of elements into two smaller sets by solving an eigenvalue problem of the type (3.1) restricted to that group of elements. Thus after $p-1$ steps, we have created a partition of the elements into p sets 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 a classical Rayleigh quotient iteration [35], modified both to bias convergence to λ_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 ϕ_k that satisfies

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

Using ϕ_k we compute the approximate eigenvalue $\sigma_k \approx \lambda_2$ from the Rayleigh quotient $\sigma_k = \phi_k^t A \phi_k$, form the residual vector r_k, and approximately solve the linear system

$$A\tilde{\delta}_k = r_k \equiv \sigma_k \phi_k - A\phi_k. \tag{3.4}$$

Note that by construction $\psi_1^t r_k = \phi_k^t r_k = 0$. From $\tilde{\delta}_k$, we form the vector δ_k satisfying

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

In the inverse iteration step (3.4), the residual appears as right-hand side, since this system is 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 the iterative method, we use our multigraph iteration [12]. So far, this has proven to be simple and effective, but the issue of the most efficient solver in this context is presently open. Early versions of our algorithm employed an iteration-dependent diagonal shift in (3.4), and a simple iterative scheme. Reinitializing the multigraph method at each iteration to allow for such a shift is so costly it offsets the benefit of improved convergence rate in the overall iteration.

Finally, we solve the 2×2 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 ϕ_{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 means to bias the overall Rayleigh quotient iteration towards convergence to ψ_2. This completes the description of a single Rayleigh quotient iteration. Note that continually imposing orthogonality conditions with respect to ψ_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 ψ_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.

TABLE 1
PLTMG cumulative execution times (seconds) for the convection-diffusion example (SIAM) and the variational inequality example (OBSTACLE).

	SIAM		OBSTACLE	
breakpoint	average	range	average	range
end of Step 1	4.6	4.6 – 4.6	6.3	6.3 – 6.3
end of Step 2	16.4	15.4 – 17.3	53.7	48.6 – 59.8
end of reconcile mesh (Step 3)	22.4	20.4 – 23.8	58.4	52.9 – 64.6
end of DD solve (Step 3)	63.2	59.3 – 66.4	92.8	84.4 – 99.0

TABLE 2
MC cumulative execution times (seconds) for the elasticity example (SIAM 3D) and the binary black hole example (BLACK HOLE).

	SIAM 3D		BLACK HOLE	
breakpoint	average	range	average	range
end of Step 1	5.2	5.2 – 5.2	31.3	31.0 – 32.3
end of Step 2	146.8	110.2 –198.3	1055.1	923.4 – 1206.3

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 1, we give the accumulated execution execution times (in seconds) for the two PLTMG example calculations described in section 2 (examples 1 and 2 in sections 2.1 and 2.2, respectively). These 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 MPICH as its MPI implementation. The computational kernels of PLTMG are written in FORTRAN. The convection-diffusion example took a total time of about a minute and the variational inequality less than two minutes. The columns labeled "range" record the smallest and largest accumulated times at each breakpoint; all 16 processors were equally and fully charged for Step 1, even though the computation was actually done on only one processor.

In comparing the results in these two examples, the biggest difference occurred in Step 2, which was about 4 times more costly for the variational inequality. This largely reflects the difference between the simple linear system solves in the convection-diffusion problem compared with the interior point iteration for the variational inequality. The latter required a linear system solve for each iteration step. The time spent reconciling the mesh in Step 3 is perhaps a bit surprising (5-6 seconds of the total time). This mostly reflects the computational cost of making the mesh conforming following the boundary exchange, rather than the communication cost itself

(see section 4.1). Interestingly, the final global solve was faster for the variational inequality. This is because the variational inequality global solve used 4 interior point iterations, each requiring a domain decomposition/multigraph inner iteration. For each of interior point iteration, we used just one inner DD iteration, for a grand total of 4 domain decomposition/multigraph iterations. On the other hand, the convection diffusion equation used 5 domain decomposition/multigraph iterations in the final solve of its linear system, and this made the overall DD solve slightly more costly for the convection diffusion example.

In Table 2, we give the execution times (in seconds) for the two MC example calculations described in section 2 (examples 3 and 4 in sections 2.3 and 2.4, respectively). These times are again from the 16-processor LINUX-based Beowulf cluster. The first example used all 16 processors; the second example used 4 of the 16 processors. The initialization time includes generating a coarse mesh from the geometry description, solving the coarse mesh problem, computing a posteriori error estimates, and partitioning the domain. In both examples, the initialization step was done identically on all of the processors; it could also have been done on a single processor, with the results sent to the remaining processors. 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.

The MC implementation does not form a global problem; this can be justified to some extent using the arguments in [24, 25], which are summarized briefly in section 4.3. It is interesting to note that when it is possible to skip the global problem, 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 a global problem of the same overall resolution. 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 Table 2 we see that although the mix of calculations was different for each subproblem, the overall times do not vary too much. 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. Finally, we note that the black hole calculation has a much larger initialization time than the elasticity problem due to the size of the initial coarse mesh (265 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 Ω_i will refine some elements *not* in Ω_i. This will occur mainly near the interface boundaries of Ω_i, where the mesh will be graded in a smooth fashion from the smaller elements of Ω_i to larger elements that cover the remainder of Ω. 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 2D, and of order $(N_p^{2/3})$ in 3D. 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. For example, in the PLTMG examples in Section 2, we had $p = 16$, $N_c = 8000$, and $N_p = 100000$. Equation (3.5) then predicts

$$N_f \approx pN_p - (p-1)N_c = 1480000,$$

where the actual values were $N_f = 1467973$ and $N_f = 1429325$.

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. Using our procedure effectively with, say, $p = 128$, would require a more difficult problem that could be decomposed into at least 128 regions, with most including some interesting behavior to resolve.

In many realistic application problems, the features of the domain and/or the equation coefficients provide more than enough complexity to lead to good scalability. For example, several extremely large biomolecular modeling problems have now been successfully solved through the use of the parallel algorithm on massively parallel computers with hundreds of processors such as the Blue Horizon. This work, which is described in detail in [3, 4], showed a very high degree of parallel efficiency even for hundreds of processors, due to the lack of need for a final global solve (see Section 4.3).

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 domain decomposition. Here we briefly describe our implementation of Step 3 of the paradigm in PLTMG. In the original paper [7], we presented a scheme to collect the global fine mesh on one processor at the conclusion of the calculation. However, it soon became clear that a more useful scenario is to keep the global fine mesh and solution distributed across the processors. Furthermore, the coarse parts of the mesh on each processor play an important role in our domain decomposition global solver. Thus here we describe a procedure that creates a distributed conforming fine mesh by updating the local mesh as it exists at the conclusion of Step 2 of our paradigm.

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 information describing the interface edges is exchanged. After the first communication step, boundary edges will not match if there is more refinement in one side of the interface (see Figure 20, left).

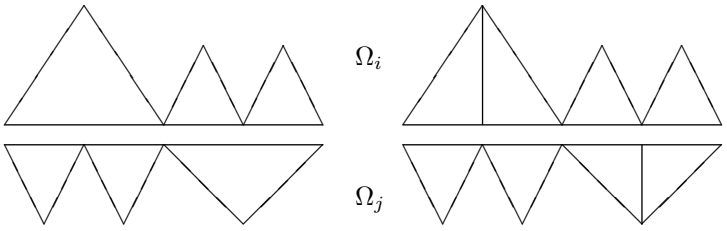


FIG. 20. *The coarse side of a non matching interface (left) is refined to make the global mesh conforming (right).*

Using information from its neighbors, each processor creates a mapping of its interface edges 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 20, right). The resulting data structure (mappings of corresponding vertices deduced from the matching edges) forms the basis of the interprocessor communication steps of our domain decomposition solver.

If one uses a refined element tree data structure for the refinement process [10, 13], as in previous versions of PLTMG, then this procedure is greatly simplified, since each independent problem starts from the same element tree. A simple postprocessing step can enforce equal (and hence conforming) levels of refinement for elements sharing an interface edge.

To compute the global solution, one can of course simply assemble and solve the global problem on a single processor. This was the original choice made in PLTMG.

This was done to quickly obtain a working parallel code to validate Steps 1 and 2 of the paradigm, rather than for reasons of efficiency. Subsequently, we implemented the domain decomposition approach described below. One could also apply parallel multigrid or another parallel iterative method in this situation. Here the parallel multigrid method of Mitchell [30, 31, 32] 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.

We now describe the domain decomposition algorithm presently implemented in PLTMG. In many respects, this algorithm is motivated by and similar to the domain decomposition algorithms described in [9, 8]. Consistent with our overall philosophy, we wish to minimize communication and maximize the use of existing sequential software.

For simplicity, we restrict attention to the case of just two subdomains. In our scheme, each subregion contributes equations corresponding 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. In any event, with a proper ordering of unknowns, the global system of equations has the block 5×5 form

$$\begin{pmatrix} A_{11} & A_{1\gamma} & & & \\ A_{\gamma 1} & A_{\gamma\gamma} & & & I \\ & & A_{\nu\nu} & A_{\nu 2} & -I \\ & & A_{2\nu} & A_{22} & \\ & I & -I & & \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta U_\nu \\ \delta U_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma \\ R_\nu \\ R_2 \\ U_\nu - U_\gamma \end{pmatrix}. \tag{4.1}$$

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 (4.1) 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 course mesh on processor 1. This leads to a linear system of the form

$$\begin{pmatrix} A_{11} & A_{1\gamma} & & & \\ A_{\gamma 1} & A_{\gamma\gamma} & & & I \\ & & \bar{A}_{\nu\nu} & \bar{A}_{\nu 2} & -I \\ & & \bar{A}_{2\nu} & \bar{A}_{22} & \\ & I & -I & & \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta \bar{U}_\nu \\ \delta \bar{U}_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma \\ R_\nu \\ 0 \\ U_\nu - U_\gamma \end{pmatrix}, \tag{4.2}$$

where quantities with a bar (e.g., $\bar{A}_{22}$) refer to the coarse mesh. A system similar to (4.2) can be derived for processor 2. With respect to the right hand side of (4.2), 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 course grid interior points is set to zero. This 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. Given our

initial guess, we expect $R_1 \approx 0$ and $R_2 \approx 0$ at all iteration steps. R_γ and R_ν are not generally small, but $R_\gamma + R_\nu \to 0$ at convergence.

As with the global formulation (4.1), equation (4.2) 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 (4.2) as

$$\begin{pmatrix} & -I & & I & \\ -I & \bar{A}_{\nu\nu} & & & \bar{A}_{\nu 2} \\ & & A_{11} & A_{1\gamma} & \\ I & & A_{\gamma 1} & A_{\gamma\gamma} & \\ & \bar{A}_{2\nu} & & & \bar{A}_{22} \end{pmatrix} \begin{pmatrix} \Lambda \\ \delta\bar{U}_\nu \\ \delta U_1 \\ \delta U_\gamma \\ \delta\bar{U}_2 \end{pmatrix} = \begin{pmatrix} U_\nu - U_\gamma \\ R_\nu \\ R_1 \\ R_\gamma \\ 0 \end{pmatrix}. \tag{4.3}$$

Block elimination of the Lagrange multiplier Λ and $\delta\bar{U}_\nu$ in (4.3) leads to the block 3×3 Schur complement system

$$\begin{pmatrix} A_{11} & A_{1\gamma} & \\ A_{\gamma 1} & A_{\gamma\gamma} + \bar{A}_{\nu\nu} & \bar{A}_{\gamma 2} \\ & \bar{A}_{2\nu} & \bar{A}_{22} \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta\bar{U}_2 \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma + R_\nu + \bar{A}_{\nu\nu}(U_\nu - U_\gamma) \\ \bar{A}_{2\nu}(U_\nu - U_\gamma) \end{pmatrix}. \tag{4.4}$$

The system matrix in (4.4) 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 (4.4). Other than the right hand side, our algorithm is very similar to those analyzed in [9, 8]. 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 (4.4).
4. locally solve (4.4) via the multigraph iteration.
5. update U_1 and U_γ using δU_1 and δU_γ.

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.

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 [49, 34, 17]. We consider two approaches for dealing with this difficultly in the present context. The first of these is the mortar elements, which is discussed here; the second approach is described in the next section.

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* [16, 15]. Although originally developed as a technique to couple spectral and finite element methods, it can be used to couple finite element discretizations that are conforming within subdomains but have nonmatching meshes at the interfaces of the subdomains.

To keep the discussion simple, suppose that there are only two subregions with a single interface Γ. 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

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

where $\mathcal{V}$ is some suitably chosen mortar space. See [16, 15, 1, 14] for details on the selection of $\mathcal{V}$. When assembled, the resulting system of linear equations will have the classic saddle point structure

$$\begin{pmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^t & B_2^t & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ 0 \end{pmatrix}, \tag{4.5}$$

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 (4.5) has a physical interpretation in terms of an approximation to the normal component of ∇u on Γ.

From its structure, it is clear that one may apply classical domain decomposition techniques to the solution of (4.5) (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 subdomainwise. 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 [47] and not so recent [33, 38, 39] theoretical results actually support this. In particular, it was shown in [24, 25] that optimal approximation order in the H^1-norm can be achieved by simply combining the disjoint refined solutions using a partition of unity, without the solution of a global problem. The argument in [25] uses the partition of unity framework of Babuška and Melenk [2] to combine the local estimates from [47] into a global estimate. In many applications (cf. [3, 4]), one is primarily interested in a high-quality approximation to the derivatives of a function rather than the function itself; in some situations these problems can be well-approximated without the final global solution phase.

The principle idea underlying the key local estimates in [47] is that while elliptic problems are globally coupled, this global coupling is essentially a "low-frequency" coupling, and can be handled on a mesh that is much coarser than that required for approximation accuracy considerations. This idea has been exploited for example in [29, 46], and is, in fact, why the construction of a coarse problem in overlapping domain decomposition methods is the key to obtaining convergence rates that are independent of the number of subdomains (cf. [45]).

The key results in [47] for our purposes are the following a priori and a posteriori error estimates. To explain, let Ω_k be the collection of disjoint subregions of the domain Ω defined by the weighted spectral bisection algorithm of the previous section, and let Ω_k^0 to be an extension of the disjoint Ω_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 Ω_k. Under some reasonable assumptions about the approximation properties of a finite element space S_0^h defined over Ω (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 Ω_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 Ω_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 Ω_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 Ω_k^0 does not pollute the accuracy of the adapted solution in the subdomain Ω_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 Ω_k can be estimated in terms of a (computable) jump function $\eta(\cdot)$ over the extended subdomain Ω_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 [47] outlined above were derived for self-adjoint linear problems in the plane, and as a result they do not apply directly to the examples presented in this paper (nonlinear scalar problems and elliptic systems in both 2D and 3D). Moreover, the local refinement strategy described here tends

to produce very little overlap of the extended subdomains Ω_k^0, and we also do not explicitly enforce a refinement limitation such as $h = O(H^2)$. Small overlap violates one of the basic assumptions in [47], and it also violates the stable partition of unity assumption for using the framework in [25] to justify skipping the global problem. However, the approach provides a very good initial approximation to an overlapping domain decomposition procedure for solving the final global problem.

An alternative to *a priori* enforcing explicit large overlap for avoiding the solution of a global problem is to use duality indicators to determine the minimal overlap automatically. This approach is described and analyzed in [25]; one computes the solution to a linearized dual problem with a characteristic function for the subdomain as the source term for the dual problem. The dual solution then weights a residual indicator for driving adaptive refinement in the subdomain. Global estimates (for a linear functional of the error) are established in [25] for the resulting global solution, recovered from local duality-driven adaptive solutions using only a partition of unity. This approach to determining necessary subdomain overlap automatically is investigated more thoroughly in [20], using the idea of generalized Green's functions.

Acknowledgments. We would like to thank the San Diego Supercomputer Center for access to their NPACI ROCKS Beowulf cluster management software. We would also like to thank Dr. Steven Bond for lending us his expertise with the configuration and management of the ROCKS software.

REFERENCES

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

[2] I. Babuška and J. M. Melenk, *The partition of unity finite element method*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 727–758.

[3] N. Baker, D. Sept, M. Holst, and J. A. McCammon, *The adaptive multilevel finite element solution of the Poisson-Boltzmann equation on massively parallel computers*, IBM Journal of Research and Development, 45 (2001), pp. 427–438.

[4] N. Baker, D. Sept, S. Joseph, M. Holst, and J. A. McCammon, *Electrostatics of nanosystems: Application to microtubules and the ribosome*, Proc. Natl. Acad. Sci. USA, 98 (2001), pp. 10037–10041.

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

[6] 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.

[7] 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.

[8] 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, (to appear).

[9] 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.

[10] R. E. Bank, A. H. Sherman, and A. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, in IMACS Transactions on Scientific Computation I, R. S. Stepleman, ed., North–Holland, Amsterdam, 1983, pp. 3–17.

[11] R. E. Bank and R. K. Smith, *Mesh smoothing using a posteriori error estimates*, SIAM J. Numer. Anal., 34 (1997), pp. 979–997.

[12] R. E. Bank and R. K. Smith, *An algebraic multilevel multigraph algorithm*, SIAM J. on Scientific Computing, 25 (2002), pp. 1572–1592.

[13] M. W. Beall and M. S. Shephard, *A general topology-based mesh data structure*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 1573–1596.

[14] F. Belgacem, *The mortar finite element method with Lagrange multipliers*, Numer. Math. 84 (1999), pp. 173–197.

[15] C. Bernardi, N. Debit, and Y. Maday, *Coupling finite element and spectral methods: First results*, Math. Comp., 54 (1990).
[16] 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. and J.L. Lions, eds., Pitman Res. Notes Math. Ser. 302, Longman Scientific & Technical, Harlow; copublished with John Wiley, New York, 1994, pp. 13–51.
[17] J. Bey, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378.
[18] X. Cai and K. Samuelsson, *Parallel Multilevel Methods with Adaptivity on Unstructured Grids*, preprint, 1999.
[19] T. F. Chan, P. Ciarlet, Jr., and W. K. Szeto, *On the optimality of the median cut spectral bisection graph partitioning method*, SIAM J. Sci. Comput., 18 (1997), pp. 943–948.
[20] D. Estep, M. Holst, and M. Larson, *Generalized Green's Functions and the Effective Domain of Influence*, SIAM J. Sci. Statist. Comput., 2002, submitted.
[21] 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. Numer. Math., 16 (1994), pp. 157–182.
[22] 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. Numer. Math., 26 (1998), pp. 241–263.
[23] G. Fox, R. Williams, and P. Messina, *Parallel Computing Works!*, Morgan-Kaufmann, San Francisco, 1994.
[24] M. Holst, *Adaptive numerical treatment of elliptic systems on manifolds*, Advances in Computational Mathematics, 15 (2001), pp. 139–191.
[25] M. Holst, *Applications of domain decomposition and partition of unity methods in physics and geometry (plenary paper)*, in Proceedings of the Fourteenth International Conference on Domain Decomposition Methods, Mexico City, Mexico, 2002, I. Herrera, D. E. Keyes, O. B. Widlund, and R. Yates, eds., 2002.
[26] M. Holst and D. Bernstein, *Adaptive Finite Element Solution of the Constraint Equations in General Relativity*. In preparation.
[27] 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 Proceedings of the Workshop on Structured Adaptive Mesh Refinement Grid Methods, Institute for Mathematics and Its Applications, University of Minnesota, Minneapolis, MN, 1997.
[28] A. Liu and B. Joe, *Relationship between tetrahedron shape measures*, BIT, 34 (1994), pp. 268–287.
[29] 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.
[30] W. Mitchell, *The full domain partition approach to distributing adaptive grids*, Appl. Numer. Math., 26 (1998), pp. 265–275.
[31] W. Mitchell, *A parallel multigrid method using the full domain partition*, Electron. Trans. Numer. Anal., 6 (1998), pp. 224–233.
[32] W. Mitchell, *The full domain partition approach to parallel adaptive refinement*, in Grid Generation and Adaptive Algorithms, IMA Vol. Math. Appl. 113, Springer-Verlag, New York, 1998, pp. 151–162.
[33] J. A. Nitsche and A. H. Schatz, *Interior estimates for Ritz-Galerkin methods*, Math. Comp., 28 (1974), pp. 937–958.
[34] M. E. G. Ong, *Uniform refinement of a tetrahedron*, SIAM J. Sci. Comput., 15 (1994), pp. 1134–1144.
[35] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice–Hall, Englewood Cliffs, NJ, 1980.
[36] A. K. Patra and D. W. Kim, *Efficient mesh partitioning for adaptive hp finite element meshes*, in Proceedings of the Eleventh International Conference on Domain Decomposition Methods, Greenwich, UK, 1998, C.-H. Lai, P. E. Björstad, M. Cross, and O. B. Widlund, eds., 1998.
[37] 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.
[38] A. H. Schatz and L. B. Wahlbin, *Interior maximum-norm estimates for finite element methods*, Math. Comp., 31 (1977), pp. 414–442.
[39] A. H. Schatz and L. B. Wahlbin, *Interior maximum-norm estimates for finite element methods* II, Math. Comp., 64 (1995), pp. 907–928.
[40] P. M. Selwood, M. Berzins, and P. M. Dew, *3D parallel mesh adaptivity: Data structures*

and algorithms, in the Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, 1997, CD-ROM, SIAM, Philadelphia, PA, 1997.

[41] H. D. Simon and S.-H. Teng, *How good is recursive bisection?*, SIAM J. Sci. Comput., 18 (1997), pp. 1436–1445.

[42] R. Verfürth, *A Posteriori Error Estimation and Adaptive Mesh Refinement Techniques*, Teubner Skripten zur Numerik, G. B. Teubner, Stuttgart, 1995.

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

[44] R. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency, 3 (1991), p. 457.

[45] J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM Rev., 34 (1992), pp. 581–613.

[46] J. Xu, *Two-grid discretization techniques for linear and nonlinear PDEs*, SIAM J. Numer. Anal., 33 (1996), pp. 1759–1777.

[47] J. Xu and A. Zhou, *Local and parallel finite element algorithms based on two-grid discretizations*, Math. Comp., to appear.

[48] 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.

[49] S. Zhang, *Multi-level Iterative Techniques*, Ph.D. thesis, Department of Mathematics, Pennsylvania State University, College Park, PA, 1988.