Computing and Visualization in Science manuscript No.
(will be inserted by the editor)

Some Variants of the Bank-Holst Parallel Adaptive Meshing Paradigm

Randolph E. Bank*

Department of Mathematics University of California, San Diego La Jolla, California 92093-0112. (email:rbank@ucsd.edu).

October 5, 2004

Abstract. The Bank-Holst adaptive meshing paradigm is an efficient approach for parallel adaptive meshing of elliptic partial differential equations. It is designed to keep communication costs low and to take advantage of existing sequential adaptive software. While in principle the procedure could be used in any parallel environment, it was mainly conceived for use on small Beowulf clusters with a relatively small number of processors and a slow communication network. A typical calculation on such a machine might involve, say $p = 32$ processors, an adaptive fine mesh with a few million vertices, and use 2–3 minutes of computational time. In this work we, discuss a variant of the original scheme that could be used in situations where a much larger number of processors, say $p > 100$ is available. In this case the problem size could be much larger, say 10–100 million, with still a low to moderate computation time.

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

1 Introduction

In [1,2], we introduced a general approach to parallel adaptive meshing for systems of elliptic partial differential equations. Our approach is designed to keep communications costs low, and to allow sequential adaptive software (such as the software package PLTMG used in this work) to be employed without extensive recoding. Our original paradigm has three main components:

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

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

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

With this approach, the load balancing problem is reduced to the numerical solution of a small elliptic problem on a single processor, using a sequential adaptive solver such as PLTMG without requiring any modifications to the sequential solver. The bulk of the calculation in the adaptive meshing step also takes place independently on each processor and can also be performed with a sequential solver with no modifications necessary for communication. The only parts of the calculation requiring communication are (1) the initial fan-out of the mesh distribution to the processors at the beginning of adaptive meshing step, once the decomposition is determined by the error estimator in load balancing; (2) the mesh regularization, requiring communication to produce a global conforming mesh in preparation for the final global solve in Step 3; and (3) the final solution phase, that might require local communication (e.g., boundary exchanges). In some circumstances, it might be useful to avoid the initial fan-out communication step by allowing all processors (which are otherwise idle) to simultaneously compute the coarse solution and load balance in Step 1. Note that a good initial guess for the final global solve is provided by the adaptive meshing step by taking the solution from each subregion restricted to its partition.

* The work of this author was supported by the National Science Foundation under contract 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.

A more complete discussion of the overall paradigm as well as some numerical illustrations can be found in [1, 2]. A description of a domain decomposition solver used in Step 3 of the paradigm is given in [3]. In Mitchell [10], a parallel adaptive procedure similar to Step 2 of our procedure is described. See [12,11,6–8] for some other approaches to parallel adaptive meshing.

Our goal of this work is to present a variant of the above approach in which the load balancing occurs on a much finer mesh. In Step 1 of the paradigm, we assume that $N_c \gg p$ where N_c is the size of the coarse mesh and p is the number of processors. This is necessary to allow the load balance to do an adequate job of partitioning the domain into regions with approximately equal error. We also assume that N_c is sufficiently large and the mesh sufficiently well adapted for the a posteriori error estimates to accurately reflect the true behavior of the error. For the second step of the paradigm, we assume that $N_p \gg N_c$ where N_p is the target size for the adaptive mesh produced in Step 2 of the paradigm. Taking $N_p \gg N_c$ is important to 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 is 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.

If any of these assumptions is weakened or violated, there might be a corresponding decline the effectiveness of the paradigm. In this case, we consider the possibility of modifying Steps 1 and 2 of the paradigm as follows.

Step 1': Load Balancing. On a single processor we adaptively create a *fine* mesh of size N_p, and use a posteriori error estimates to partition the mesh such that each subregion has approximately equal error, similar to Step 1 of the original paradigm.
Step 2': Adaptive Meshing. Each processor is provided the complete adaptive mesh and instructed to sequentially solve the *entire* problem. However, in this case each processor should adaptively *coarsen* regions corresponding to other processors, and adaptively refine its own subregion. The size of the problem on each processor should remain at N_p, but this adaptive rezoning strategy will concentrate the degrees of freedom in the processor's subregion. At the end of this step, the mesh could be regularized such that the global mesh described in Step 3 will be conforming.
Step 3': Global Solve. This step is the same as Step 3; the global mesh consists of the refined partitions provided by each processor. A final solution is computed using a domain decomposition or parallel multigrid technique.

With this variant, the initial mesh can be of any size. Indeed, our choice of N_p is mainly for convenience and to simplify notation. Of course, allowing the mesh in Step 1' to be finer increases the cost of both the solution and the load balance, but it allows for flexibility in overcoming potential deficiencies of a very course mesh in the original paradigm. As before, all processors could simultaneously carry out Step 1' in order to avoid the initial fan-out communication step. It is also possible to compute the solution in Step 1' in parallel using some variant of the original paradigm.

Another interesting extension is to create a three-stage parallel adaptive procedure by essentially invoking the new procedure in a recursive fashion. In this case, suppose that we now have p^2 processors available.

Step 1a": Initial Load Balancing. On a single processor we adaptively create a fine mesh of size N_p, and use a posteriori error estimates to partition the mesh into p subregions such that each subregion has approximately equal error, exactly as in Step 1'.
Step 2a": Adaptive Meshing. p processors are provided the complete adaptive mesh and instructed to sequentially solve the entire problem, coarsening outside and refining within their subregion. The overall mesh on each processor remains of size N_p as in Step 2'.
Step 1b": Final Load Balancing. Each of the p processors partitions just its own subregion in to p smaller subregions of approximately equal error and broadcasts its entire mesh to p processors (including itself). Now a total of p^2 processors are active.
Step 2b": Adaptive Meshing. Each of the p^2 has one of the p meshes computed in Step2a". Each processor sequentially solves the entire problem, coarsening outside and refining within its subregion. The overall mesh remains of size N_p on each processor. At the end of this step, the mesh could be regularized such that the global mesh described in Step 3 will be conforming.
Step 3': Global Solve. This step is the same as Steps 3 and 3'. The global mesh consists of the refined partitions provided by each of the p^2 processors. A final solution is computed using a domain decomposition or parallel multigrid technique.

In the case of the original paradigm, we envisioned its use mainly in the environment of a small cluster of fast workstations. In this situation the number of available processors typically is small or moderate. For example, with the 32-processor cluster available to us we can solve problems with several million nodes in a few minutes using the original paradigm [1–3]. The three stage scheme described above is really not appropriate or necessary for such an environment, but it could be used to solve very large problems on very large parallel machines. For example, consider the modest choices of $p = 32$ and $N_p = 100000$. Suppose at each of the coarsening–refinement adaptive step half the mesh points are moved from outside to inside the processor's subregion. With this set of relatively conservative assumptions, in the final global mesh each of the $p^2 = 1024$ processors would have at least 50000 mesh points in its refined subregion; so the global refined mesh would have at least

$50000 \times 1024 \approx 5 \times 10^7$ nodes. With less conservative assumptions and/or more available processors, this scheme could be used to adaptively solve problems with billions of nodes on thousands of processors.

Although all these variants have simple communication patterns with small communication costs, it is important to emphasize one caveat. The goal of all these procedures is to create a final global adaptive mesh in which the error is roughly equilibrated among the elements, and the effort needed to create these meshes is roughly the same on each processor. The idea of creating subregions of approximately equal error for the load balancing steps really amounts to the fragile assumption that this corresponds to approximately equal work for each processor, and that the final adaptive mesh will have a reasonable equilibration of the error among the elements. In the three stage algorithm, this assumption is further extrapolated in that the initial load balance determines the p subregions used for the second load balance.

The remainder of this manuscript is organized as follows. In Section 2, we present some numerical illustrations of both the original paradigm and the new variant. In Section 3, we briefly describe how the a posteriori error estimates are modified to produce error indicators appropriate for the parallel adaptive refinement and unrefinement schemes described above.

2 Numerical examples

In this section, we present some numerical results. Our examples were run on a small LINUX-based Beowulf cluster, consisting of 20 dual 1800 Athlon-CPU nodes with 2GB of memory each, a dual Athlon file server, and a 100Mbit CISCO 2950G Ethernet switch. This cluster runs the NPACI ROCKS version of LINUX (based on RedHat 7.1), and employs MPICH1.2.2 as its MPI implementation. The computational kernels of PLTMG are written in FORTRAN; the *g77* compiler (version 2.96) was used in these experiments, invoked using the script *mpif77* and optimization flag *-O*.

The first example is the Helmholtz problem

$$\begin{aligned} -\Delta u - 100u &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Here Ω describes the topology of the North Sea; we scaled the coordinates such that $\Omega \subset (0,1) \times (0,1)$. This problem is solved by a standard finite element method. Let S be the space of continuous piecewise linear polynomials corresponding to the triangulation of Ω. The the solution $u_h \in S_0$, where $S_0 = \{v \in S | v = 0 \,\text{on}\, \partial\Omega\}$. Linear systems involving the sparse symmetric indefinite stiffness matrix are solved by the multigraph iteration used in PLTMG. The solution is shown is Figure 1.

The second example is the optimal control problem

$$\min \int_\Omega (u - u_0)^2 + \gamma\lambda^2 \, dx$$

subject to the constraint equation

$$\begin{aligned} -\Delta u &= \lambda \quad \text{in } \Omega \equiv (0,1) \times (0,1), \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

and the inequalities

$$1 \le \lambda \le 10.$$

The target function u_0 is given by

$$u_0 = \sin(3\pi x)\sin(3\pi y)$$

and the regularization parameter $\gamma = 10^{-4}$.

The Lagrangian for this problem is given by

$$L(u,v,\lambda) = \int_\Omega (u - u_0)^2 + \gamma\lambda^2 + \nabla u \cdot \nabla v - v\lambda \, dx$$

for $(u,v,\lambda) \in H_0^1(\Omega) \times H_0^1(\Omega) \times L^2(\Omega)$.

This problem is discretized in standard fashion using piecewise linear finite elements. The approximation $(u_h, v_h, \lambda_h) \in S_0 \times S_0 \times S$, where S and S_0 are as described above. The discrete solution (u_h, v_h, λ_h) is shown in Figure 1.

The discretized problem is solved by an interior point method; at each step the linear algebra problem, based on the second derivatives of the Lagrangian, is of the form

$$\left(\begin{array}{cc|c} H & A & 0 \\ A & 0 & K \\ \hline 0 & K^t & G \end{array}\right) \begin{pmatrix} \delta u \\ \delta v \\ \delta\lambda \end{pmatrix} = \begin{pmatrix} b_u \\ b_v \\ b_\lambda \end{pmatrix}. \tag{1}$$

The standard nodal basis functions are used. The matrix A is the symmetric positive definite stiffness matrix corresponding the the Dirichlet problem constraint. The matrices H, K and G are all related to the mass matrix M corresponding to S. In particular, H is a submatrix of M corresponding to S_0. The matrix K is rectangular; trial basis functions correspond to the space S, while test basis functions are those for the subspace S_0. The matrix $G = \gamma M + D$, where D is a positive definite diagonal matrix with entries arising from the interior point method applied to the inequality constraints for λ.

Linear systems involving A are easily solved using the multigraph iteration. Additionally, linear systems involving G approximately solved using a simple symmetric Gauss-Seidel iteration with conjugate gradient acceleration. Based on these observations we construct a block symmetric Gauss-Seidel preconditioner based on the 2×2 blocking indicated in (1), This is realized as follows:

$$\begin{aligned} A\tilde{c}_u &= b_v, \\ A\tilde{c}_v &= b_u - H\tilde{c}_u, \\ G\delta\lambda &= b_\lambda - K^t\tilde{c}_v, \\ A\delta u &= b_v - K\delta\lambda, \\ A\delta v &= b_u - H\delta u. \end{aligned} \tag{2}$$

If G were replaced by the Schur complement and all linear systems solved exactly, this preconditioner would yield the exact solution.

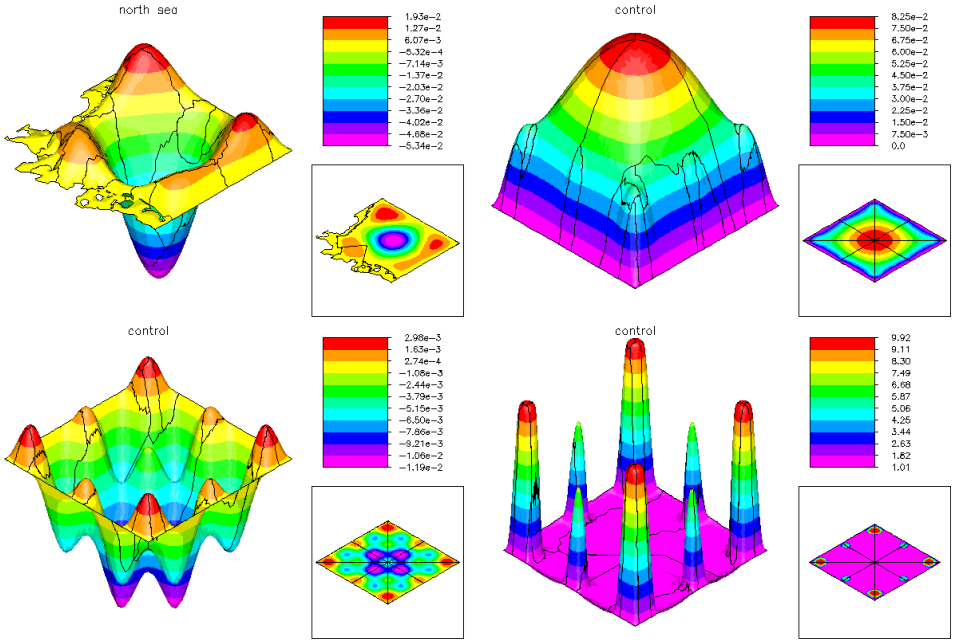

Fig. 1. Top left: solution of the Helmholtz equation. Top right: solution of the optimal control problem. Bottom left: the Lagrange multiplier. Bottom right: the control function λ. The lines denote the load balance for the case $p = 32$.

In our first experiment, we solved the two example problems using the original Bank-Holst paradigm for various values of p. During Step 1 of the paradigm, we began with an initial triangulation. For the Helmholtz equation, the initial mesh was generated by PLTMG from a description of the boundary $\partial\Omega$ and contained $N = 2900$ vertices; for the obstacle problem we began with a uniform 9×9 mesh. These meshes were then adaptively refined (on a single processor) to create a mesh with $N_c = 8000$ vertices. This mesh was then partitioned into p subregions ($p = 2, 4, 8, 16, 32$) using a spectral bisection algorithm. This completed Step 1 of the paradigm. The load balance for the case $p = 32$ for each of the examples is shown in Figure 2. The a posteriori error estimation procedure used in PLTMG for adaptive refinement and the load balancing is described in [4,5].

In Step 2 of the paradigm, the coarse mesh and solution were broadcast to all processors, and each processor independently continued with the adaptive refinement, creating an adaptively refined mesh with $N_p = 100000$ vertices, with most of the refinement concentrated in its own region. Prior to solving the final linear system with $N_p = 100000$, the global mesh was made conforming. Unlike the similar experiments described in [2,3], the solution of the final system of equations with $N_c = 100000$ is deferred until the beginning Step 3 of the paradigm.

In Step 3 of the paradigm, the global system of linear equations was solved using the domain decomposition algorithm described in [3]. Prior to the domain decomposition iteration, the final system of equations left unsolved in Step 2 is solved independently on each processor. Since PLTMG is a script driven program, and various procedures can be called in any order, forcing this solve as part of the call to the DD solver insures that a good initial guess is always provided to the DD solver.

For the Helmholtz equation, the solve is just a simple linear system involving the finite element stiffness matrix; the main point of interest is that the linear systems were symmetric but indefinite. For the control problem, the linear system was the block 3×3 system described above, and each each approximate solution step consisted of approximately solving the five linear systems described in (2). This block solve was the inner iteration of the interior point method used to solve the overall problem.

In Figure 3, we show the mesh density for the global fine mesh in the case $p = 32$. Since these global meshes each have several million elements, one can not draw individual elements. Thus each element is colored by size, and the edges are not drawn, yielding an image that shows the general refinement pattern. Most important, note that the overall refinement patterns seem reasonable given the nature of the problems. Another point to note is the relative smoothness of the mesh density across the subdomain interfaces; this is an a posteriori indication that the load balance computed in Step 1 of the paradigm was good. In Figure 4, we plot a posterior error estimates computed on the global refined mesh following Step 3 of the paradigm. Here we see that the overall procedure did a reasonable job of equilibrating the error. Here again, these error estimates provide some a posteriori indication of the quality of the initial load balance.

In Table 1, we provide a summary of the calculation. The global value of N can be approximated by the formula [1]

$$N \approx pN_p - (p-1)N_c, \tag{3}$$

where N_c is the size of the original coarse mesh ($N_c = 8000$ in our example) and N_p is the target value used by each processor in Step 2 of the parallel adaptive paradigm

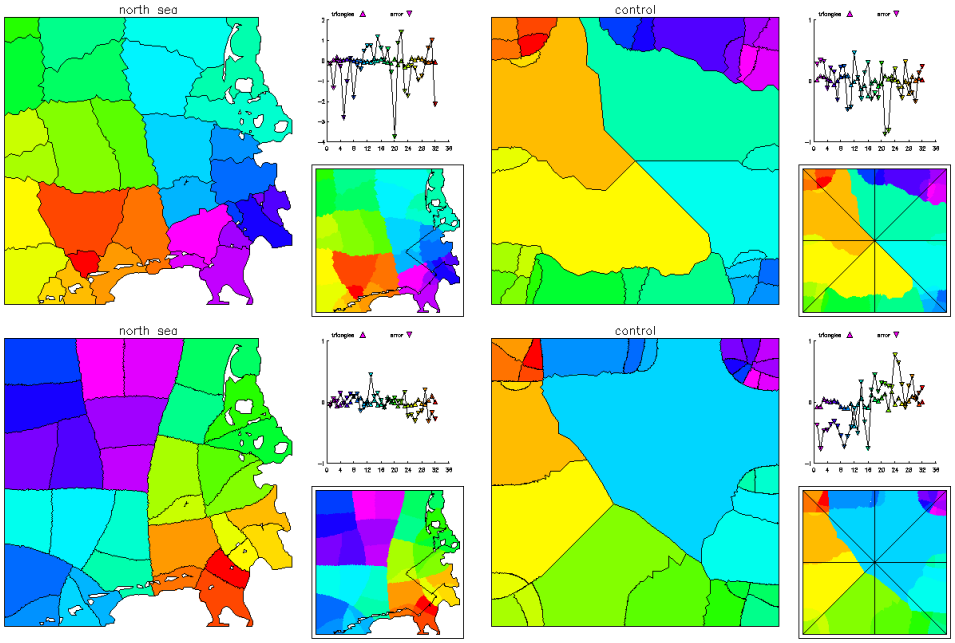

Fig. 2. Top: the load balance for the case $p = 32$ for the original bank-Holst algorithm. Bottom, the load balance for the case $p = 32$ for the variation.

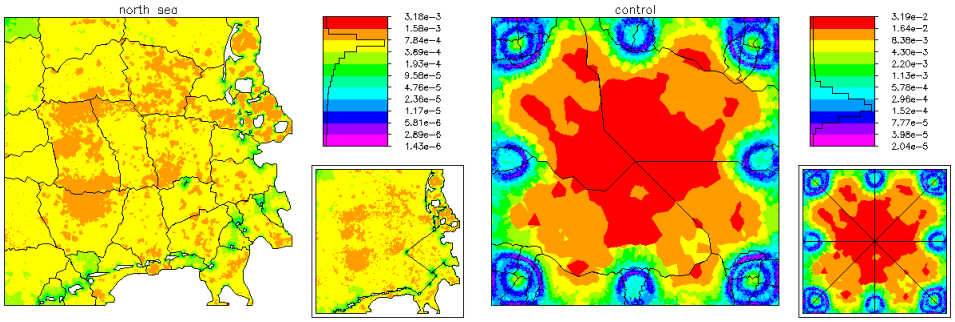

Fig. 3. Mesh density for the final adaptive mesh. Both figures refer to the case $p = 32$ for the original Bank-Holst algorithm.

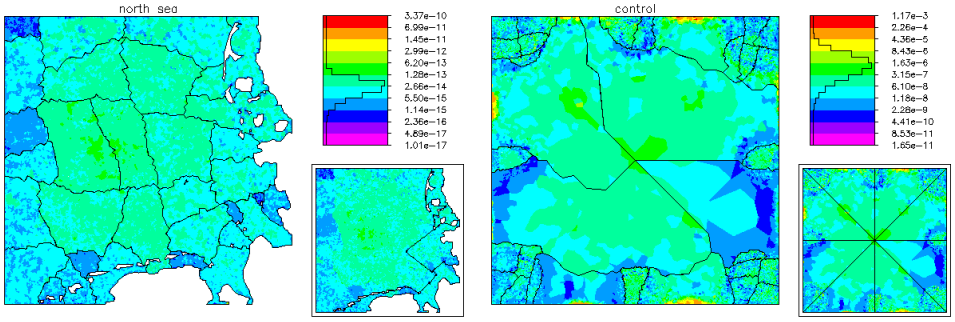

Fig. 4. A posteriori error estimate for the solution. Both figures refer to the case $p = 32$ for the original Bank-Holst algorithm.

($N_p = 100000$ in our example). As described in [2], each processor refines some elements outside of its subdomain Ω_i, to maintain shape regularity in the transition between small elements within its subregion and generally larger elements in the remainder of Ω, to control "pollution" and other effects coming from the PDE outside of Ω_i, and to generally satisfy requirements imposed to make the domain decomposition solver efficient. All of this implies that generally

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

For example, when $p = 32$, equation (3) predicts $N \approx 2952000$ when the actual values were $N = 2304303$ for the Helmholtz equation and $N = 2474511$ for the control problem. Note for the control problem, the size of the global block 3×3 linear system was approximately $3N$.

In Table 1 we see that the number of domain decomposition iterations is quite stable over the range of this set of problems. As with the example problems in [3], empirically the convergence rate appears to be largely independent of N but has some logarithmic-like dependence on p.

In Table 1 we also record average execution times at various breakpoints in the computation. The computations done in Step 1 were done on only one processor, but the time was charged equally and fully to all processors. The time used for Step 1 increases slowly as a function of p. In all the runs, all the computations involved were the same in Step 1, except for the load balance, so the increase in time can be attributed in full to the increasing number of eigenvalue problems solved in the recursive spectral bisection algorithm that forms the basis of the load balancing procedure. However, in terms of the the overall computation, Step 1 has very minimal impact, so this is not a large effect. The cost of Step 2 was very stable over the entire range of p, although there was some increase. This was mainly due to the the increasing cost of reconciling the mesh as p increased. The times reported here are relatively less than in similar Tables in [2,3] because the solve for the last problem of on the mesh of size $N_p = 100000$ was deferred until Step 3. The results illustrate good scalability and near perfect speedup. In addition, the execution times of different processors only vary in a small range in Step 2, indicating good load balance from mesh partition. Step 3 refers local solve on the final adaptive mesh, followed by the global domain decomposition solve using the procedure described in [3]. Here we see some increase in time with increasing p. This seems to reflect mostly the increasing number of domain decomposition iterations required. Also, the range of times increases in some cases; this largely reflects differing numbers of multigraph iterations used on different processors for the approximate solution of linear systems involving A.

In the second experiment, we solved each problem again using the modified strategy. This time we took $p = 16, 32, 64, 128, 256$. We note that since our cluster had only 20 dual nodes, for larger values of p each processor had multiple processes; in effect this experiment is only a simulation of such a calculation on a larger machine. For this experiment, in Step 1', we started from the same initial mesh as in the previous experiment, but adaptively refined on one processor to a mesh of size $N = 100000$ (instead of $N = 8000$). The mesh with $N = 100000$ was then load balanced using the spectral bisection algorithm, and the resulting global mesh broadcast to all processors.

In Step 2', each processor used a rezone option available in PLTMG to unrefine the mesh of size $N_p = 100000$ to one of size $\hat{N} = 50000$, and the refine this coarsened mesh back to one of size $N_p = 100000$. In the coarsening phase, all or most of the vertices deleted from the mesh were in regions outside of the subdomain associated with that processor. In the refinement phase, most of the refinement took place within the regions subdomain. The net result was a transfer of about 50000 from outside to inside the processors subdomain. Following this rezone, the global mesh was made conforming as in Step 2 of the original paradigm.

Step 3' is essentially the same as Step 3 in the first experiment. Each processor locally solved the equations associated with its rezoned mesh of size $N_p = 100000$. This was followed by a global domain decomposition solve.

The parallel implementation in PLTMG was directed mainly towards the original paradigm. While it can be used with no change for the new variant, this experiment illustrates several deficiencies in the current parallel implementation, although not in the basic paradigm itself. In particular, a central assumption in the current PLTMG implementation is that the load balance is done in a coarse mesh, while in these experiments, the load balance was done on a mesh of size $N_p = 100000$.

In Table 2 we summarize the results. In comparing the times for Step 1', we see that the times are much larger than in the original paradigm, and increase slightly with increasing p. The calculation was done on one processor, and the time should be the same for each value of p, since the same computations were performed. Much of the time spent in Step 1' was devoted to solving eigenvalue problems as part the spectral bisection algorithm used for load balancing. The results indicate that while the current algorithm is certainly adequate for the original paradigm (the largest problem was approximately of order 16000 in the first experiment) it is very inefficient for the new variant, because the orders of the eigenvalue problems are much larger (the largest was approximately 200000 in this example). The load balance for the case $p = 32$ is shown in Figure 2; we note that the overall partitioning is quite similar despite the large difference in the size of the mesh.

Step 2' of the variant is quite comparable to Step 2 of the original paradigm, in terms of time. However, here we encounter a second deficiency in the current implementation. In the current PLTMG, vertices lying at endpoints of the original (coarse) edges defining the interface system are not allowed to be deleted in the unrefinement step. If an interface edge is refined, the new interface vertex is allowed to be considered for deletion in a subsequent unrefinement step. If the original interface system is defined on a coarse mesh, this typically causes no problems. In this case, however, it caused an unusual pattern in the unrefinement step. In particular, most of the unrefinement occurred in the interiors of the regions other than that associated with the given processor, with the unrefinement being graded towards the interface.

Prohibiting deletion of the original interface vertices allows for an efficient algorithm for regularizing the global mesh at the end of the second step. Since this regularization process mainly involves the edges defining the boundary of the processor's subdomain, allowing unrefinement on parts of the interface system *not* associated with the given processor's subregion seems to be feasible, and is a topic for future research.

We can predict the size of the global mesh in an idealized situation as follows. Assume that the original mesh has N_p vertices, and that θN_p, $0 < \theta < 1$ vertices are

Table 1. Summary of computation for the original Bank-Holst paradigm. p is the number of processors, N the number of vertices in the global fine mesh, DD the number of domain decomposition iterations. The breakpoints are accumulated execution times, in seconds. The range of execution times among all processors for Step 2 and Step 3 appear in parentheses.

p	N	DD	Breakpoints		
			End of Step 1	End of Step 2	End of Step 3
Helmholtz Equation					
2	186416	2	1.7	19.4 (18.6-20.2)	77.0 (73.5-80.5)
4	354123	2	2.0	19.2 (16.2-21.7)	82.3 (68.6-105.0)
8	666160	3	2.2	20.5 (18.4-23.3)	105.2 (82.4-141.6)
16	1232550	4	2.4	19.9 (15.7-24.4)	124.7 (82.1-189.3)
32	2304303	4	2.6	20.1 (15.5-25.3)	130.5 (83.2-163.1)
Optimal Control Problem					
2	190087	1	5.1	25.3 (25.2-25.4)	97.6 (97.3-97.8)
4	365117	1	5.3	27.1 (26.0-28.3)	107.6 (100.4-115.2)
8	689242	1	5.6	27.0 (25.7-28.5)	109.1 (97.9-117.8)
16	1317877	1	5.9	27.4 (25.6-29.2)	114.1 (97.2-126.5)
32	2474511	1	6.1	28.1 (25.6-29.5)	124.1 (99.6-138.9)

rezoned in the unrefinement/refinement step. Most of these vertices will be moved from outside to inside the processor's subdomain. Thus the global value of N can be approximated by

$$N \approx p\theta N_p + N_p. \tag{4}$$

In this experiment $\theta = 1/2$ and $N_p = 100000$. When $p = 32$, equation (4) predicts $N = 1700000$, where the actual values were $N = 1472638$ for the Helmholtz equation and $N = 1482322$ for the optimal control problem. For the larger values of p, this underestimate becomes more pronounced. We attribute this mainly to restrictions on the unrefinement discussed above.

In Step 3', the DD solver was used to solve the global conforming system of equations, as in Step 3 of the original paradigm. In these examples, the DD solver remains quite effective and shows a slight logarithmic dependence on p.

Overall, while these results are certainly preliminary in nature, they demonstrate the potential of the variant paradigm. In the largest optimal control problem, we created and adaptive mesh with about 7.5 million vertices, and solved the corresponding equations with about 22.5 million unknowns, in about 5 minutes (simulated).

3 Adaptive Refinement and Unrefinement

An important aspect of our algorithm is the criteria used in adaptive refinement and unrefinement. In this section we give a brief discussion of these points.

Suppose the global domain Ω is partitioned into p non overlapping subdomains Ω_i, $1 \le i \le p$, such that

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

In the case of simple refinement, on processor i the mesh should be adaptively refined in the a more or less standard fashion within $\overline{\Omega}_i$. However, some elements outside of Ω_i must also be refined, in order to grade the mesh between small elements in Ω_i and larger elements elsewhere in Ω while simultaneously controlling the shape regularity of the elements. In the case of PLTMG, the adaptive refinement is based on a posteriori error estimates yielding local error indicators based on the $\mathcal{H}^1$ norm. For a detailed description and analysis of the error estimator used here see [4, 5].

In the case of adaptive unrefinement, on processor i, the mesh should be coarsened in all subregions other than $\overline{\Omega}_i$. Once again, the need to keep the mesh both conforming and shape regular means that the coarsening should also be graded away from $\overline{\Omega}_i$.

In the initial implementation of our algorithm, we simply multiplied a posteriori error estimates for elements outside of Ω_i by 10^{-6} so that the sequential adaptive refinement algorithms would be unlikely to choose those elements for refinement on the basis of their error estimate. Similarly, such elements would become obvious targets for unrefinement. The refinement and unrefinement subroutines in PLTMG consider shape regularity as part of their decision process. Also, each intermediate mesh in the refinement or unrefinement process is always conforming. So this simple strategy, used in conjunction with reasonably sophisticated refinement and unrefinement procedures, is sufficient to generated the types of graded meshes described above.

However, it seems certain that this initial suggestion, although quite simple and easy to implement, is probably not an optimal strategy. The issue of grading the mesh outside of Ω_i is presently an active area of research that will be reported elsewhere. Here we just summarize the main issues.

First, it is clearly advantageous for processor i to concentrate as many degrees of freedom as possible within Ω_i. Mesh points in Ω_i become mesh points in the final refined global conforming mesh and contribute in a direct way to the overall global solution. On the other hand, mesh points outside of Ω_i do not contribute directly to the overall global solution. To some extent, they can be viewed as a necessary overhead expense. Informally, we

Table 2. Summary of computation for the variant of the Bank-Holst paradigm. p is the number of processors, N the number of vertices in the global fine mesh, DD the number of domain decomposition iterations. The breakpoints are accumulated execution times, in seconds. The cost of Step 1 exclusive of the spectral bisection load balance and the range of execution times among all processors for Step 2 and Step 3 appear in parentheses.

p	N	DD	Breakpoints		
			End of Step 1'	End of Step 2'	End of Step 3'
Helmholtz Equation					
16	829932	2	81.5	98.4 (95.1-100.5)	159.5 (140.0-186.7)
32	1472638	2	86.7	103.6 (99.1-106.5)	167.6 (139.8-198.8)
64	2610992	3	91.0	108.3 (104.3-110.4)	187.1 (152.9-216.8)
128	4641145	3	95.1	112.1 (108.3-115.4)	191.4 (157.3-224.0)
256	7941852	3	100.8	117.1 (113.0-125.1)	199.6 (163.9-229.0)
Optimal Control Problem					
16	830641	1	144.3	161.8 (158.4-164.7)	223.1 (212.3-237.1)
32	1482322	1	149.6	167.1 (162.7-170.6)	234.6 (216.7-251.0)
64	2600084	2	140.3	158.1 (154.8-161.1)	242.4 (226.4-260.4)
128	4507853	2	153.2	171.1 (167.3-174.1)	260.1 (240.1-282.8)
256	7539839	2	146.6	164.5 (160.8-175.9)	265.0 (237.9-296.6)

substitute local computation on these extra mesh points for interprocessor communication that would otherwise be required.

Second, the goal of Step 2 (or Step 2') is adaptive mesh generation, *not* the computation of an accurate solution. Clearly adaptive meshing and solution accuracy are related, but it is not necessary to have an accurate solution to determine a good adaptive mesh. On the other hand, it is also clear that data outside of Ω_i does influence the solution within Ω_i. It is possible that such influence could be sufficiently strong to have a significant adverse effect on the adaptive mesh generation within Ω_i if it is not at least partly resolved. Potential examples are strong point singularities outside of Ω_i, or upstream flow in the case of PDE's involving convection. While it is not necessary for each processor to completely resolve such behavior in order to create a good mesh on its own subregion, it does seem important that each processor determine the approximate influence on the solution in its subregion, and then if necessary resolve it to an appropriate level.

Third, the mesh outside Ω_i plays an important role in our domain decomposition solver, providing what amounts to a "built-in" coarse mesh on each processor. We will not discuss this point in detail here but rather refer the interested reader to [3,9].

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

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

$$\theta_t = \frac{\omega_t}{2d_t}, \tag{5}$$

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

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

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

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

$$a(\phi, w) = 0 \tag{6}$$

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

In Figure 5, we show the function w corresponding to several different processors for the case $p = 32$. These dual functions were computed as part of the first numerical example (original paradigm) but the ones for the variant would be quite similar in character. For the case of the optimal control problem, the differential operator involved is just $-\Delta$, and the dual functions display exactly the behavior described above. Dual functions for the Helmholtz equation are more interesting. The decay away from Ω_i is not monotonic in this case, and each dual function reveals a more complicated structure that might be difficult to predict in advance.

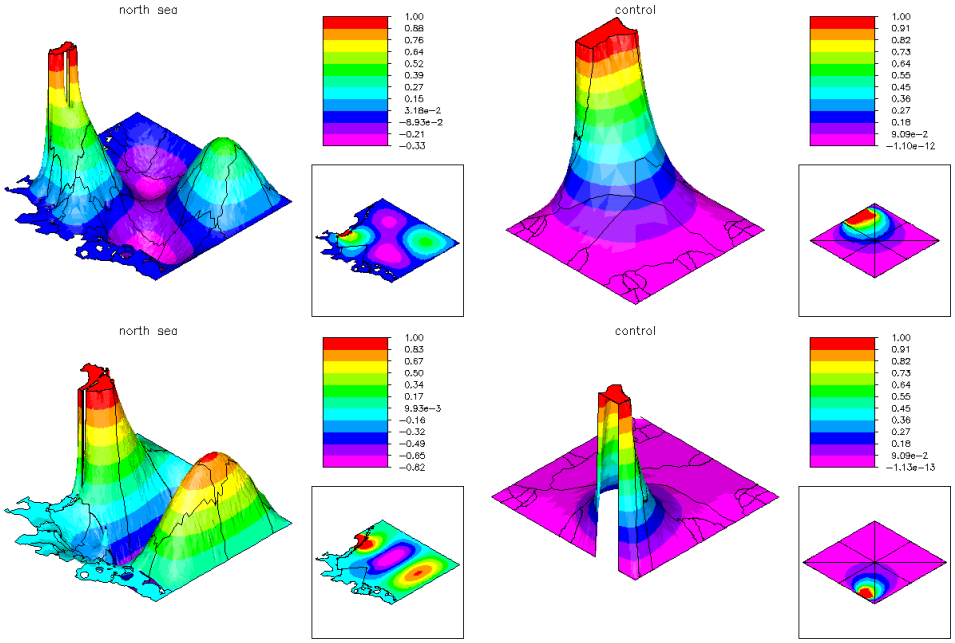

Fig. 5. Left: typical dual functions for the Helmholtz equation. Right: typical dual functions for the optimal control problem. In both cases, $p = 32$ and the original Bank-Holst paradigm was used.

We define ω_t by

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

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

In terms of linear algebra, problem (6) involves the solution of a linear system involving a submatrix of the stiffness matrix. Assume we use the standard nodal basis for $\mathcal{V}^i$. Then the stiffness matrix A has the block 2×2 form

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

where A_{ff} corresponds to mesh points in $\overline{\Omega}_i$ and A_{cc} corresponds to mesh points strictly outside of Ω_i. As the adaptive refinement proceeds, we expect that the order of A_{ff} will be much larger than that of A_{cc}. The linear system corresponding to (6) is

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

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

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

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

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

Finally for all elements with $d_t \le 1$ we set $\theta_t = 1$; otherwise, we define θ_t using (5). The goal of this weighting strategy is to produce a mesh where most of the refine-

ment occurs in Ω_i and most unrefinement occurs outside Ω_i, but at the same to time allow modest refinement as necessary outside of Ω_i to meet the multiple goals of the adaptive procedure itself, and the global domain decomposition solver used in Step 3 of the paradigm.

Acknowledgements. We thank Gabriel Wittum of the University of Heidelberg for providing that data defining the North Sea domain.

References

1. 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.
2. ———, *A new paradigm for parallel adaptive meshing algorithms*, SIAM Review, 45 (2003), pp. 292–323.
3. R. E. Bank and S. Lu, *A domain decomposition solver for a parallel adaptive meshing paradigm*, SIAM J. on Scientific Computing, (to appear).
4. R. E. Bank and J. Xu, *Asymptotically exact a posteriori error estimators, part I: Grids with superconvergence*, SIAM J. Numerical Analysis, (to appear).
5. ———, *Asymptotically exact a posteriori error estimators, part II: General unstructured grids*, SIAM J. Numerical Analysis, (to appear).
6. 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.
7. 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.
8. M. T. Jones and P. E. Plassmann, *Parallel algorithms for adaptive mesh refinement*, SIAM J. Sci. Comput., 18 (1997), pp. 686–708.
9. S. Lu, *Parallel Adaptive Multigrid Algorithms*, PhD thesis, Department of Mathematics, University of California at San Diego, 2004.
10. W. Mitchell, *The full domain partition approach to parallel adaptive refinement*, in Grid Generation and Adaptive Algorithms, IMA Volumes in Mathematics and its Applications, Springer-Verlag, Heidelberg, 1998, pp. 152–162.
11. 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.
12. C. Walshaw and M. Berzins, *Dynamic load balancing for pde solvers on adaptive unstructured meshes*, Concurrency: Practice and Experience, 7 (1995), pp. 17–28.