A Domain Decomposition for a Parallel Adaptive Meshing Algorithm

Randolph E. Bank*

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

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

1 Bank-Holst Algorithm

In [4, 5], we introduced a general approach to parallel adaptive meshing for systems of elliptic partial differential equations. This approach was motivated by the desire 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 discussion is framed in terms of continuous piecewise linear triangular finite element approximations used in PLTMG, although most ideas generalize to other approximation schemes.

Our original paradigm, called *Plan A* in this work, has three main components:

Step I: 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 II: 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

* 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.

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 is regularized such that the global mesh described in Step III is conforming.

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

With this paradigm, 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 III; and (3) the final solution phase, that requires communicating certain information about the interface system (see Section 2).

In [2], we considered a variant of the above approach in which the load balancing occurs on a much finer mesh. The motivation was to address some possible problems arising from the use of a coarse grid in computing the load balance. In particular, we assume in Plan A 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 II of the paradigm. Taking $N_p \gg N_c$ is important to marginalize the cost of redundant computations.

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 I and II of the paradigm as follows. This variant is called *Plan B* in this work.

Step I: 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 I of the original paradigm.

Step II: 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 remains N_p, but this adaptive

rezoning strategy concentrates the degrees of freedom in the processor's subregion. At the end of this step, the mesh is regularized such that the global mesh is conforming.
Step III: Global Solve. This step is the same as Plan A.

With Plan B, the initial mesh can be of any size. Indeed, our choice of N_p is mainly for convenience and to simplify notation; any combination of coarsening and refinement could be allowed in Step II. Allowing the mesh in Step I to be finer increases the cost of both the solution and the load balance in Step I, but it allows flexibility in overcoming potential deficiencies of a very course mesh in Plan A.

2 A Domain Decomposition Algorithm

In developing a domain decomposition solver appropriate for Step III, we follow a similar design philosophy. In particular, our DD solver has low communications costs, and recycles the sequential solvers employed in the Steps I and II. Furthermore, we use the existing partially refined global meshes distributed among the processors as the basis of local subdomain solves. This results in an overlapping DD algorithm in which the overlap is global, and provides a natural built-in coarse grid space on each processor. Thus no special coarse grid solve is necessary. Finally, a very good initial guess is provided by taking the fine grid parts of the solution on each processor.

The DD algorithm is described in detail in [7, 9]; some convergence analysis for a related algorithm in the symmetric, positive definite case can be found in [6]. To simplify the discussion, we initially consider the case of only two processors. We imagine the fine grid solutions for each of the two regions glued together using Lagrange multipliers to impose continuity along the interface. This leads to a block 5×5 system

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

Here U_1 and U_2 are the solutions for the interior of regions 1 and 2, while U_γ and U_ν are the solutions on the interface. R_* are the corresponding residuals. The blocks A_{11}, A_{22} correspond to interior mesh points for regions 1 and 2, while $A_{\gamma\gamma}$, $A_{\nu\nu}$ correspond to the interface. Λ is Lagrange multiplier; the identity matrix I appears because global mesh is conforming.

In a similar fashion, we can imaging the fine grid on processor 1 glued to the coarse grid on processor 1 using a similar strategy. This results in a similar block 5×5 system

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

where the barred quantities (e.g. $\bar{A}_{22}$) refer to the coarse mesh. The right hand side of (2) is a subset of (1), except that we have set $\bar{R}_2 \equiv 0$. If local solves in Step II of the procedure were done exactly, then the initial guess would produce zero residuals for all interior points in the global system (1). We thus assume $R_1 \approx 0$, $R_2 \approx 0$ at all steps. This approximation substantially cuts communication and calculation costs.

Next, on processor 1 we reorder the linear system (2) as

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

and formally eliminate the upper 2×2 block. The resulting local Schur complement system is given by

$$\begin{pmatrix} A_{11} & A_{1\gamma} & 0 \\ A_{\gamma 1} & A_{\gamma\gamma} + \bar{A}_{\nu\nu} & \bar{A}_{\gamma 2} \\ 0 & \bar{A}_{2\nu} & \bar{A}_{22} \end{pmatrix} \begin{pmatrix} \delta U_1 \\ \delta U_\gamma \\ \delta \bar{U}_2 \end{pmatrix} = \begin{pmatrix} R_1 \\ R_\gamma + R_\nu + \bar{A}_{\nu\nu}(U_\nu - U_\gamma) \\ 0 + \bar{A}_{2\nu}(U_\nu - U_\gamma) \end{pmatrix}. \tag{3}$$

The system matrix in (3) is just the stiffness matrix for the conforming mesh on processor 1. To solve this system, processor 1 must receive R_ν, and U_ν from processor 2 (and in turn send R_γ, and U_γ to processor 2). With this information, the right hand side can be computed and the system solved sequentially with no further communication. We use δU_1 and δU_γ to update U_1 and U_γ; we discard $\delta \bar{U}_2$. The update could be local ($U_1 \leftarrow U_1 + \delta U_1$, $U_\gamma \leftarrow U_\gamma + \delta U_\gamma$) or could require communication. In PLTMG, the update procedure is a Newton line search. Here is a summary of the calculation on processor 1.

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 the Schur complement system (3).
4. locally solve the linear system (3) via the multigraph iteration.
5. update U_1 and U_γ using δU_1 and δU_γ.

We now consider the case of the global saddle point system in the general case of p processors. Now the global system has the form

$$\begin{pmatrix} A_{ss} & A_{sm} & A_{si} & I \\ A_{ms} & A_{mm} & A_{mi} & -Z^t \\ A_{is} & A_{im} & A_{ii} & 0 \\ I & -Z & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta U_s \\ \delta U_m \\ \delta U_i \\ \Lambda \end{pmatrix} = \begin{pmatrix} R_s \\ R_m \\ R_i \\ ZU_m - U_s \end{pmatrix}. \tag{4}$$

Here U_i are the interior unknowns for all subregions, and A_{ii} is a block diagonal matrix corresponding to the interiors of all subregions; as before we expect $R_i \approx 0$. For the interface system, we (arbitrarily) designate one unknown at each interface point as the *master* unknown, and all others as *slave* unknowns; there will be more than one slave unknown at cross points (where more than 2 subregions share a single interface point). As before we impose continuity at interface points using Lagrange multipliers; $Z \neq I$ in general due to cross points. If we reorder (4) and eliminate the Lagrange multipliers and slave unknowns, the resulting Schur complement system is

$$\begin{pmatrix} A_{mm} + A_{ms}Z + Z^t A_{sm} + Z^t A_{ss} Z & A_{mi} + Z^t A_{si} \\ A_{im} + A_{is}Z & A_{ii} \end{pmatrix} \begin{pmatrix} \delta U_m \\ \delta U_i \end{pmatrix} = \begin{pmatrix} R_m + Z^t R_s - (A_{ms} + Z^t A_{ss})(ZU_m - U_s) \\ R_i - A_{is}(ZU_m - U_s) \end{pmatrix}. \tag{5}$$

The system matrix is just the stiffness matrix for the global conforming finite element space. The right hand side is the conforming global residual augmented by some "jump" terms arising from the Lagrange multipliers.

The situation on processor k is analogous; we imagine gluing the fine subregion on processor k to the $p-1$ coarse subregions on processor k. The resulting saddle point problem has the form

$$\begin{pmatrix} \bar{A}_{ss} & \bar{A}_{sm} & \bar{A}_{si} & I \\ \bar{A}_{ms} & \bar{A}_{mm} & \bar{A}_{mi} & -\bar{Z}^t \\ \bar{A}_{is} & \bar{A}_{im} & \bar{A}_{ii} & 0 \\ I & -\bar{Z} & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta \bar{U}_s \\ \delta \bar{U}_m \\ \delta \bar{U}_i \\ \Lambda \end{pmatrix} = \begin{pmatrix} \bar{R}_s \\ \bar{R}_m \\ \bar{R}_i \\ \bar{Z}\bar{U}_m - \bar{U}_s \end{pmatrix}. \tag{6}$$

The matrix $\bar{A}_{ii}$ and the vector $\bar{U}_i$ are fine for region k and coarse for the $p-1$ other regions. The residual $\bar{R}_i$ corresponds to R_i on region k, and is zero for the coarse subregions. Master interface variables are chosen from region k if possible; this part of the local interface system on processor k corresponds exactly to the global interface system. For other parts of the local interface system, master unknowns can be chosen arbitrarily; in PLTMG, they are actually defined using arithmetic averages, but that detail complicates the notation and explanation here. The vectors $\bar{R}_m$ and $\bar{R}_s$ are subsets of R_m and R_s, respectively.

A local Schur complement system on processor k is computed analogously to (5). This system has the form

$$\begin{pmatrix} \bar{A}_{mm} + \bar{A}_{ms}\bar{Z} + \bar{Z}^t\bar{A}_{sm} + \bar{Z}^t\bar{A}_{ss}\bar{Z} & \bar{A}_{mi} + \bar{Z}^t\bar{A}_{si} \\ \bar{A}_{im} + \bar{A}_{is}\bar{Z} & \bar{A}_{ii} \end{pmatrix} \begin{pmatrix} \delta\bar{U}_m \\ \delta\bar{U}_i \end{pmatrix} = \begin{pmatrix} \bar{R}_m + \bar{Z}^t\bar{R}_s - (\bar{A}_{ms} + \bar{Z}^t\bar{A}_{ss})(\bar{Z}\bar{U}_m - \bar{U}_s) \\ \bar{R}_i - \bar{A}_{is}(\bar{Z}\bar{U}_m - \bar{U}_s) \end{pmatrix}. \quad (7)$$

As in the 2 processor case, the system matrix is just the conforming finite element stiffness matrix for the partially refined global mesh on processor k. To compute the right hand side of (7), processor k requires interface solution values and residuals for the global interface system. Once this is known, the remainder of the solution can be carried out with no further communication. To summarize, on processor k, one step of the DD algorithm consists of the following.

1. locally compute $\bar{R}_i$ and parts of R_s and R_m from subregion k.
2. exchange boundary data, obtaining the complete fine mesh interface vectors R_m, R_s, U_m and U_s.
3. locally compute the right-hand-side of (7) (using averages).
4. locally solve the linear system (7) via the multigraph iteration.
5. update the fine grid solution for subregion k using subsets of $\delta\bar{U}_i$, $\delta\bar{U}_m$.

3 Numerical Experiments

We now present several numerical illustrations; the details of the example problems are summarized below.

Example 1: Our first example is the Poisson equation

$$\begin{aligned} -\Delta u &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \quad (8)$$

where Ω is the domain shown in Figure 1.

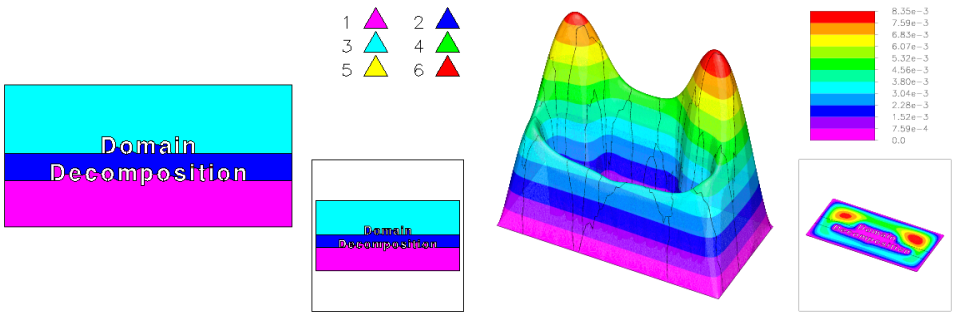

Fig. 1. The domain (left) and solution (right) for the Poisson equation (8).

Example 2: Our second example is the convection-diffusion equation

$$\begin{aligned} -\Delta u + \beta u_y &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \\ \beta &= 10^5, \end{aligned} \tag{9}$$

where Ω is the domain shown in Figure 2.

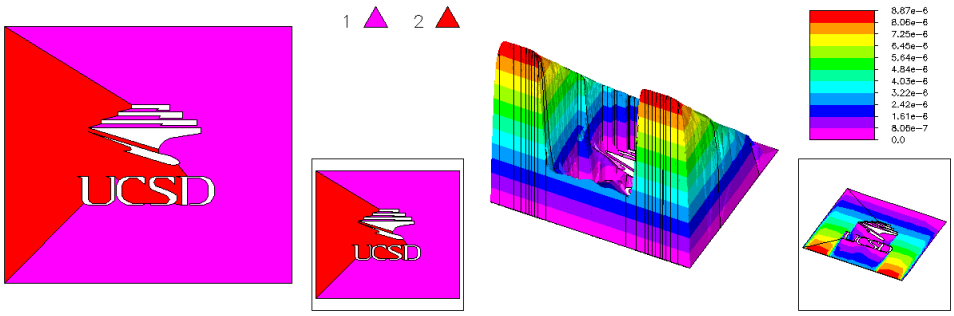

Fig. 2. The domain (left) and solution (right) for the convection-diffusion equation (9).
.

Example 3: Our third example is the anisotropic equation

$$\begin{aligned} -a_1 u_{xx} - a_2 u_{yy} - f &= 0 \quad \text{in } \Omega, \\ (a_1 u_x, a_2 u_y) \cdot \mathbf{n} &= c - \alpha u \quad \text{on } \partial\Omega, \end{aligned} \tag{10}$$

where Ω is the domain shown in Figure 3. Values of the coefficient functions are given in Table 1.

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

Table 1. Coefficient values for equation (10). Region numbers refer to Figure 3.

Example 4: Our fourth example is the optimal control problem

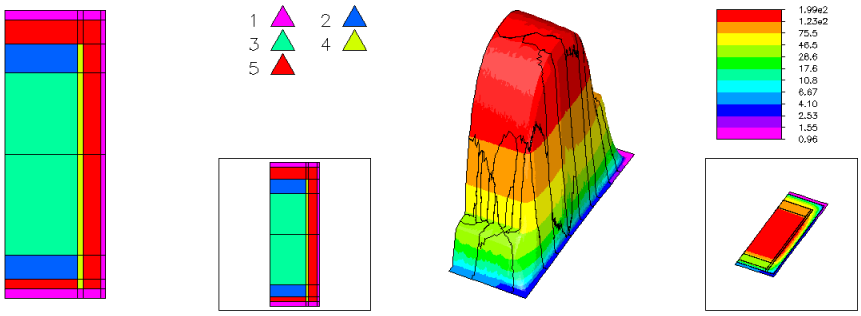

Fig. 3. The domain (left) and solution (right) for the anisotropic equation (10).
.

$$\begin{aligned} &\min \int_\Omega (u-u_0)^2 + \gamma\lambda^2\, dx \ \text{ such that} \\ -\Delta u &= \lambda \quad \text{in } \Omega \equiv (0,1)\times(0,1), \\ u &= 0 \quad \text{on } \partial\Omega, \\ 1 \le \lambda &\le 10, \quad \gamma = 10^{-4}, \\ u_0 &= \sin(3\pi x)\sin(3\pi y). \end{aligned} \tag{11}$$

This problem is solved by an interior point method described in [3, 1]. Three finite element functions are computed; the state variable u, the Lagrange multiplier v, and the optimal control λ.

Our Linux cluster consists of 20 dual 1800 Athlon-CPU nodes with 2GB of memory each, with a dual Athlon 1800 file server, also with 2GB of memory. Communication is provided via a 100Mbit CISCO 2950G Ethernet switch. The cluster runs the NPACI Rocks version of Linux, using Mpich.

In the case of the original paradigm, Plan A, in Step I for each problem we created an adaptive mesh with $N \approx 10000$ vertices. This mesh was then partitioned for $p = 8, 16, 32, 64, 128$ processors, and the coarse problem was broadcast to all processors[2]. In Step II of the paradigm, we adaptively created a mesh with $N \approx 100000$ vertices. In particular, we first adaptively refined to $N \approx 40000$, solved that problem, adaptively refined to $N \approx 100000$, and then regularized the mesh. In Step III, PLTMG first solved the local problem with $N \approx 100000$, in order to insure that interior residuals were small and validate the assumption that coarse interior residuals could be set to zero in the DD solver. This local solve was followed by several iterations of the DD solver.

In the case of the variant paradigm, Plan B, in Step I we created an adaptive mesh with $N \approx 100000$. As in Plan A, this mesh was then partitioned for $p = 8, 16, 32, 64, 128$ processors, and broadcast to all processors. In Step II, through a process of adaptive unrefinement/refinement, each processor

[2] Since our cluster had only 20 nodes, the results are simulated using Mpich for the larger values of p.

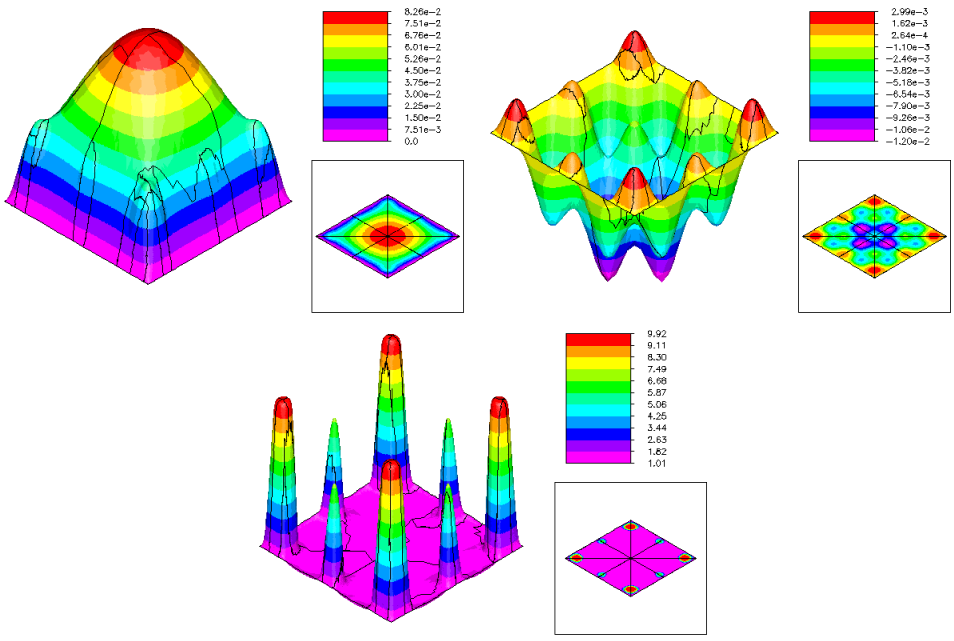

Fig. 4. The state variable (left,top), Lagrange multiplier (right, top) and optimal control (bottom) for equation (11).
.

transferred approximately 50000 vertices from outside its subregion to inside, so that the total number of vertices remained $N \approx 100000$. This mesh was them made conforming as in Step II of Plan A. In Step III, the local problem was solved, followed by several iterations of the DD solver.

For both Plan A and Plan B, the convergence criteria for the DD iteration was

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

$$\frac{\|\mathcal{R}^k\|_{G^{-1}}}{\|\mathcal{R}^0\|_{G^{-1}}} \leq 10^{-2}$$

Here G is the diagonal of the finite element mass matrix, introduced to account for nonuniformity of the global finite element mesh. u_h and e_h are the finite element solution and a posteriori error estimate, respectively, introduced to include the approximation error in the convergence criteria. The norms in various terms are different, but we have not observed any difficulties arising as a result. For the multigraph iteration on each processor, the convergence criteria was

$$\frac{\|\overline{\mathcal{R}}^j\|_{\ell^2}}{\|\overline{\mathcal{R}}^0\|_{\ell^2}} \leq 10^{-3}.$$

The stronger criteria was to insure that the approximation on coarse interior residuals by zero remained valid.

p	N	DD	Breakpoints		
			Step I	Step II	Step III
Poisson Equation: Plan A					
8	657464	2	2.8	20.1 (17.2-21.3)	61.6 (58.2-65.4)
16	1240805	2	3.0	19.6 (16.4-21.3)	62.4 (56.7-67.0)
32	2329953	2	3.2	20.4 (17.4-22.5)	67.8 (58.1-72.7)
64	4361844	2	3.3	20.0 (15.5-21.8)	68.5 (56.1-76.9)
128	8057638	3	3.5	20.0 (15.2-22.2)	77.0 (59.8-88.7)
Poisson Equation: Plan B					
8	478398	1	75.0	92.7 (90.2-95.2)	123.5 (121.2-126.2)
16	827827	1	82.4	98.8 (97.4-103.0)	130.9 (126.2-136.0)
32	1472509	1	87.2	106.1 (103.0-109.2)	139.9 (133.0-145.0)
64	2626624	1	90.1	109.1 (105.0-112.1)	143.8 (136.6-150.9)
128	4641395	1	97.9	116.7 (113.4-119.6)	151.6 (143.7-157.9)

Table 2. Numerical results for problem (8).

p	N	DD	Breakpoints		
			Step I	Step II	Step III
Convection Diffusion Equation: Plan A					
8	698859	2	3.3	18.7 (17.2-19.7)	47.4 (45.7-49.7)
16	1345203	2	3.5	18.9 (16.6-21.1)	47.5 (45.4-50.1)
32	2543913	2	3.8	18.9 (16.6-21.0)	49.3 (43.5-54.8)
64	4665741	2	4.0	19.0 (16.6-21.3)	51.1 (46.0-63.1)
128	8547289	2	4.2	19.3 (16.0-22.4)	53.1 (44.6-65.0)
Convection Diffusion Equation: Plan B					
8	493182	2	66.7	79.5 (77.2-82.4)	105.1 (100.5-111.3)
16	872605	2	76.4	89.0 (86.6-91.7)	115.8 (111.6-119.1)
32	1598504	1	81.4	94.3 (91.6-97.6)	118.8 (113.7-123.7)
64	2941956	1	84.8	97.6 (95.3-100.6)	122.6 (118.9-126.5)
128	5305634	2	83.8	96.6 (94.3-100.3)	128.0 (122.0-132.6)

Table 3. Numerical results for problem (9).

In Tables 2-5 we summarize the results of our computations. In these tables, p is the number of processors, N is the number of vertices on the final global mesh, and DD is the number of domain decomposition iterations used in Step III. Execution times, in seconds, at the end of Steps I, II, and III are also reported. Step I is done on a single processor. For Steps II and III, average

p	N	DD	Breakpoints		
			Step I	Step II	Step III
Anisotropic Equation: Plan A					
8	646293	1	3.0	20.7 (19.3-22.0)	49.7 (46.0-54.5)
16	1169837	1	3.3	20.7 (19.0-22.6)	49.8 (44.9-54.8)
32	2038184	2	3.6	21.6 (18.7-23.2)	59.9 (48.0-68.5)
64	3500678	2	3.8	21.9 (19.4-24.6)	61.4 (51.6-71.8)
128	5729057	2	4.0	22.1 (19.5-24.9)	62.2 (53.8-75.6)
Anisotropic Equation: Plan B					
8	484972	1	56.5	73.7 (71.4-77.0)	100.7 (97.1-106.7)
16	832360	1	62.5	79.2 (76.6-82.2)	105.8 (101.4-111.1)
32	1460881	1	68.0	86.3 (83.0-89.3)	115.1 (109.9-119.8)
64	2512231	1	74.8	92.9 (89.7-95.4)	122.7 (116.4-130.5)
128	4102960	1	81.4	99.3 (96.2-103.1)	129.8 (123.7-140.4)

Table 4. Numerical results for problem (10).

p	N	DD	Breakpoints		
			Step I	Step II	Step III
Optimal Control Problem: Plan A					
8	677913	1	6.0	31.8 (29.8-35.6)	109.9 (101.2-114.3)
16	1297918	1	6.3	32.1 (29.0-37.7)	119.3 (106.1-131.3)
32	2421235	1	6.5	33.3 (29.0-38.3)	131.3 (110.4-141.9)
64	4514511	1	6.6	33.8 (30.4-38.3)	137.3 (105.9-157.3)
128	8324507	2	7.0	34.2 (29.9-42.2)	173.6 (135.5-212.5)
Optimal Control Problem: Plan B					
8	481309	1	139.0	156.7 (155.0-159.0)	215.6 (205.5-229.5)
16	830641	1	143.0	160.2 (157.0-163.1)	221.0 (210.6-236.7)
32	1482322	1	150.8	168.2 (164.0-171.1)	235.9 (220.1-248.4)
64	2600084	2	154.9	172.5 (169.0-175.6)	256.7 (242.0-275.6)
128	4507853	2	143.0	160.7 (156.5-166.0)	249.4 (230.3-268.3)

Table 5. Numerical results for problem (11).

times across all processors are reported; the range of times is also included in parentheses.

- The times for Step I are much larger for Plan B than Plan A due to the larger size of the problem. The increase in time with increasing p is due mostly to eigenvalue problems that are solved are part of the spectral bisection load balancing scheme.
- The distribution of times in Steps II and III is due mainly to differences in the local sequential algorithms, for example using one instead of two multigraph V-cycles in a local solve.

- The DD algorithm in [6] is shown to converge independently of N, which was empirically verified in [7] for the version implemented here. There is some slight, empirically logarithmic, dependence on p.
- For the convection-diffusion problem a multigraph preconditioned Bi-CG algorithm was used, while for the Poisson equation and the anisotropic equation regular preconditioned CG was used. Details of the multigraph solver are given in [8].
- For the optimal control problem, the block linear systems were of order $3N$, and each iteration required the solution of four linear systems with the $N \times N$ finite element stiffness matrix, and one system with an $N \times N$ matrix similar to the finite element mass matrix. See [1] for details.

In viewing the results as a whole, both paradigms scale reasonably well as a function of p; since Step III is a very costly part of the calculation, it is clearly worthwhile to try to make the convergence rate independent of p as well as N, or at least to reduce the dependence on p. This is a topic of current research interest.

References

1. R. E. Bank, *PLTMG: A software package for solving elliptic partial differential equations, users' guide 9.0*, tech. report, Department of Mathematics, University of California at San Diego, 2004.
2. ———, *Some variants of the Bank-Holst parallel adaptive meshing paradigm*, Computing and Visualization in Science, (accepted).
3. R. E. Bank, P. E. Gill, and R. F. Marcia, *Interior methods for a class of elliptic variational inequalities*, in Large-scale PDE-constrained Optimization, L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders, eds., vol. 30 of Lecture Notes in Computational Science and Engineering, Berlin, Heidelberg and New York, 2003, Springer-Verlag, pp. 218–235.
4. 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.
5. ———, *A new paradigm for parallel adaptive meshing algorithms*, SIAM Review, 45 (2003), pp. 292–323.
6. R. E. Bank, P. K. Jimack, S. A. Nadeem, and S. V. Nepomnyaschikh, *A weakly overlapping domain decomposition preconditioner for the finite element solution of elliptic partial differential equations*, SIAM J. on Scientific Computing, 23 (2002), pp. 1817–1841.
7. R. E. Bank and S. Lu, *A domain decomposition solver for a parallel adaptive meshing paradigm*, SIAM J. on Scientific Computing, 26 (2004), pp. 105–127 (electronic).
8. R. E. Bank and R. K. Smith, *An algebraic multilevel multigraph algorithm*, SIAM J. on Scientific Computing, 25 (2002), pp. 1572–1592.
9. S. Lu, *Parallel Adaptive Multigrid Algorithms*, PhD thesis, Department of Mathematics, University of California at San Diego, 2004.