Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Hierarchical Matrix Techniques for a Domain Decomposition Algorithm

by

Jeffrey Ovall

Preprint no.: 61 2006

Hierarchical Matrix Techniques for a Domain Decomposition Algorithm

Jeffrey S. Ovall*

July 11, 2006

Abstract

In this paper we investigate the effectiveness of hierarchical matrix techniques when used as the linear solver in a certain domain decomposition algorithm. In particular, we provide a direct performance comparison between an algebraic multigrid solver and a hierarchical matrix solver which is based on nested dissection clustering within the software package PLTMG.

AMS Subject Classifications: 65F10, 65F30, 65F50, 65N30, 65N55
Keywords: hierarchical matrices, domain decomposition, finite elements, nested dissection

1 Introduction

The purpose of this paper is see how the current state of the art in $\mathcal{H}$-matrix techniques compares with an established algebraic multigrid solver in the context of a domain decomposition algorithm. These comparisons are done within the software package PLTMG [1], which has been modified so that it can use linear solution techniques available in the $\mathcal{H}$-Matrix Library (http://www.hlib.org) in addition to its native multigrid/cg linear solver. The problems under consideration arise from piecewise linear finite elements for second order linear elliptic PDEs in two spatial dimensions. Because the relevant theory has been developed and presented elsewhere (we will give references), here we will give only brief descriptions of the aspects of these ideas which are important to the present discussion.

The remainder of this paper is structured as follows. In Section 2 we describe the Bank-Holst parallel adaptive meshing paradigm and a domain decomposition algorithm which was designed with this paradigm in mind. We describe the multigrid and hierarchical linear solvers which we compare within the domain decomposition algorithm in Section 3. Numerical comparisons are provided in Section 4, followed by a few concluding remarks in Section 5.

*Max Planck Institute for Mathematics in the Sciences, D-04103 Leipzig, Germany.

2 The Bank-Holst Parallel Paradigm and the Bank-Lu Domain Decomposition Algorithm

The Bank-Holst parallel adaptive meshing paradigm, presented in detail in [2, 3], was designed to keep communication costs low between nodes in a cluster and to take advantage of existing sequential adaptive software such as PLTMG without the need for extensive investment in recoding for use in a parallel environment. We describe it in brief:

> **Step 1: Load Balancing.** A "small" problem is solved on a coarse mesh, and *a posteriori* error estimates are used to partition the mesh into p subdomains, one for each processor. Each subdomain has approximately the same estimated error, although subdomains may vary considerably in terms of size and numbers of elements.
>
> **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 assigned subdomain. The target number of elements and grid points for each problem is the same. At the end of this step, the mesh on each processor is regularized at the boundary of its assigned subdomain (its "fine" region) so that a global fine mesh formed by piecing together the fine portions of the meshes on each processor would be conforming.
>
> **Step 3: Global Fine Solution.** An initial guess for the global fine solution is formed from the fine portions of solutions computed on each processor. We note that this initial guess is multi-valued along the interfaces between subdomains. The final finite element solution is computed using the domain decomposition solver described below.

The domain decomposition algorithm, developed by Bank and Lu with the same philosophy of keeping communication costs low and taking advantage of existing sequential software, is described in detail in [4]. There is a natural saddle point formulation of the global fine problem based on a mortar finite element discretization (formally an additive Schwarz method). Local versions of this system can be formulated on each processor, and then reduced "by hand" to systems involving the local stiffness matrices which are already computed on each processor. The right-hand side of the local system on a given processor involves residuals associated with the interior of its assigned subdomain and the global interface between all subdomains. Part of this interface residual is computed locally, but the rest of it is communicated via MPI from the other processors. An iteration of the domain decomposition procedure involves solving the local system, updating the global fine solution, and recomputing and exchanging the interface residuals. The key point for our discussion is that potentially several systems involving the same stiffness matrix must be solved on each processor.

3 The Linear Solvers

The algebraic multigrid solver under consideration is that of Bank and Smith [5]. Our comparisons are with the implementation of this linear solver in PLTMG. We will not discuss the details of the solver here, but merely state a few pertinent points helpful for those who already have a general familiarity with such solvers. The multigrid iteration is used as a preconditioner for a composite step conjugate gradient or biconjugate gragient iteration - depending on the whether the stiffness matrix is symmetric or not. The smoother used in each multigrid step is an ILU-factorization with drop tolerance. The various parameters have been tuned with an eye toward robustness. The multigrid iteration in combination with the CG iteration is what we refer to as the AMG solver in Section 4.

Because hierarchical matrix (or $\mathcal{H}$-matrix) techniques are not yet as widely known as multigrid techniques, we give some of the basic ideas before discussing how we specifically use them here. Some of the key papers which lay much of the theoretical and practical foundations are [6, 11, 12, 14, 13]. For a fairly recent systematic treatment of the motivations, theory, applications and technical details of algorithms we refer the reader to [7]. A hierarchical matrix is so-called because it possesses a hierarchical block structure. This hierarchy is determined by one of various techniques for generating a tree such that its root consists of indices for all of the degrees of freedom, and at each subsequent level of the tree the sons of a given node represent a splitting of the father's index set. Various strategies exist for splitting the index sets at each level of the tree, and these are referred to in the literature as clustering techniques. The tree is called a cluster tree and the nodes are called clusters. The actual hierarchical matrix consists of relatively small dense matrix blocks which are stored in full matrix format, and relatively large matrix blocks which are either sparse in the usual sense or data-sparse in the sense that they can be well-approximated by an outer product of low-rank matrices, $M = R_1 R_2^T$ where $R_1 \in \mathbb{R}^{m\times k}$, $R_2 \in \mathbb{R}^{n\times k}$, $\mathrm{rank}(R_1) = \mathrm{rank}(R_2) = k \ll \min(m, n)$. In fact, the large low-rank blocks are stored in precisely this format. Computations such as matrix inversion and triangular factorization are performed in a way that preserves this structure, with almost linear complexity.

The general strategy of nested dissection for producing an efficient ordering of a sparse stiffness matrix was introduced by Alan over 30 years ago [10] and has been investigated in the context of sparse or incomplete LU-factorizations (see [9] and the references therein, for example). The idea is to decouple the domain Ω into two subdomains Ω_1, Ω_2 by an interface Γ so that the basis functions associated with vertices in one subdomain are supported outside of the other. For piecewise linear elements in $\mathbb{R}^2$, the interface is a curve. Unknowns associated with Ω_1 are numbered first, then Ω_2 and finally Γ. This procedure is done recursively on the subdomains until there is some suitably small number of unknowns associated with each subdomain at that level of dissection. Le Borne, Grasedyck, and Kriemann [15, 8] have modified this strategy slightly for use in the $\mathcal{H}$-matrix framework, and have also investigated the performance of triangular factorizations in this context. The difference for the $\mathcal{H}$-matrix version is that interface clusters are also split, so that blocks of the stiffness

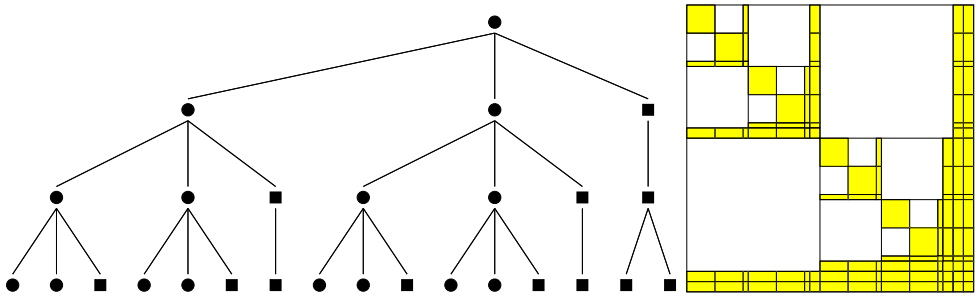

Figure 1: At left, the first four levels of a nested dissection cluster tree - circles represent (sub)domain clusters and squares represent interface clusters. At right, the structure of the corresponding hierarchical matrix - empty regions represent zero blocks and shaded regions represent blocks which are low rank blocks or full matrix blocks (if they are sufficiently small), or have additional hierarchical structure.

matrix associated with interface unknowns can be treated hierarchically - allowing for a more data-sparse representation. The first few levels of a cluster tree based on nested dissection are shown in Figure 1, together with the hierarchical matrix structure at that level of resolution.

In principle the method of splitting a (sub)domain is arbitrary, but storage requirements and performance of algorithms are reduced if, for example, the number of interface unknowns is large with respect to the number of unknowns in either of its associated subdomains. The strategy chosen for this paper is based on the fact that, on each processor, its assigned subdomain will be refined much more heavily than the rest of the domain. So our first partition separates this subdomain from the rest of the domain. Under the (perhaps fragile) assumption that the meshes within both subdomains are quasi-uniform, all subsequent partitions are done based on geometrically bisecting the subdomain in either the x- or y-direction, in whichever one the length of the subdomain is larger. In Section 4 the linear solution technique labelled HLU uses an $\mathcal{H}$-LU or Cholesky factorization under this clustering as a preconditioner for GMRES.

4 Numerical Experiments

In this section we present numerical experiments on three test problems, thereby demonstrating the general usefulness of the HLU-solvers in the domain decomposition algorithm. Experiments are done on a cluster of Dual AMD Opteron 250s with 2.4 GHz CPU clock rate, 4GB RAM and a 2x Gigabit Ethernet interconnect. For each of the three problems two series of experiments are done. The average processor times (wall time, in seconds) for the global domain decomposition solver are reported in Table 1. In the first series of experiments the problem is solved on a "coarse" mesh of 10 thousand vertices and error estimates are used to partition the domain into

$p = 2$, 4, 8, 16 and 32 subdomains. Then each of the p processors continues to solve and adaptively refine (primarily in its assigned subdomain) until a final mesh size N of 100 thousand vertices are reached (before mesh regularization at the interfaces). In the second series of experiments, the number of processors is held fixed at 16, and the size of the coarse problem varies between 5 and 25 thousand; then each processor continues until the problem size is increased by a factor of 10. We are interested in how the solve times of both approaches compare with each other and how they scale with respect to p and N.

We now describe the three test problems. The **Simple Problem** is to solve $-\Delta u = f$, on the unit square with zero Dirichlet conditions on all four edges, and right-hand side chosen so that the solution is $u = x(1-x)y(1-y)$. The **Convection-Reaction-Diffusion Problem** is to solve $-2(u_{xx} + u_{xy} + u_{yy}) + u_x + u_y + 3u = f$ on the square $(0, \pi) \times (0, \pi)$, with zero Dirichlet conditions on the top and bottom of the square and Neumann conditions on the left and right. The right-hand side and Neumann conditions are chosen to match the solution $u = e^x \sin 2y$. The **Lake Superior Problem** is to solve $-\Delta u = 1$, with zero Dirichlet conditions on a domain shaped like Lake Superior. The domain has six holes corresponding to six islands in the lake. The three domains are pictured in Figure 2 together with their partitions into 16 subdomains based on error estimates computed on a coarse mesh of 10 thousand unknowns.

For a fixed number of processors, there is roughly a linear relation between the target problem size N and the actual problem size after interface regularization on each processor. This is due to the fact that the change (increase) in the total number of unknowns on a given processor due to the interface regularization is small compared to the target number of unknowns - in fact, we expect the number of interface unknowns on a given processor before regularization to be on the order of $\sqrt{N}$ and that this relation should also hold upon regularization. Therefore, the approximate square root scaling of solution time with respect to p and the approximate linear scaling with respect to N seen in Table 1 for both AMG and HLU are optimal. Additionally, we note that the solve times for HLU tend to be slightly better.

5 Conclusions

We have demonstrated that $\mathcal{H}$-matrix techniques based on a nested dissection dissection clustering can be competitive with algebraic multigrid methods in the context of the domain decomposition solver described in Section 2. The original motivation for the comparison was that, on each processor, the stiffness matrix remains the same at each iteration of the DD algorithm for linear problems. So although the set-up cost for the $\mathcal{H} - LU$ preconditioner is larger than that for the AMG preconditioner, the solve time for each right-hand side is less. Therefore, the set-up cost might be ameliorated over several DD iterations. In fact, for the problems considered here, no more than three DD iterations were ever necessary for convergence. This bodes well for problems where more iterations are necessary - perhaps in cases where subdomains which are well-separated are nevertheless more strongly coupled due to convection.

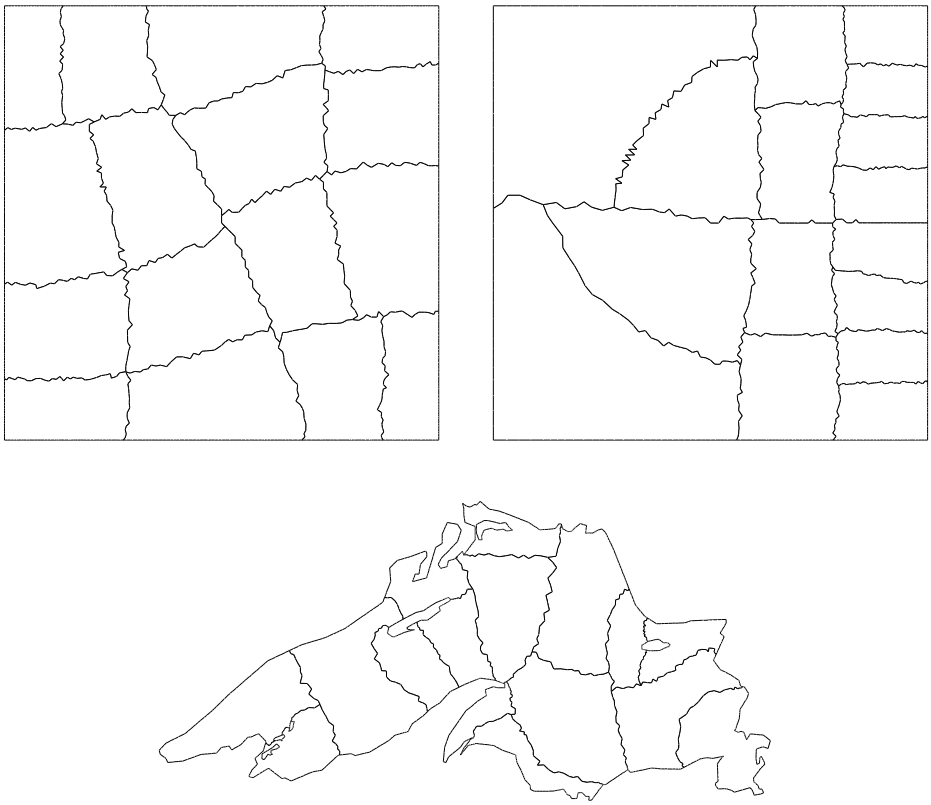

Figure 2: The three test problems each partitioned into 16 subdomains based on error estimates computed on a coarse mesh of 10 thousand unknowns. Clockwise: Simple, Convection-Reaction-Diffusion, Lake Superior.

Table 1: Average processor time of DD solve (in seconds) with respect to number of processors/subdomains.

	p	2	4	8	16	32
Simple	AMG	13.36	15.08	15.61	17.61	18.06
	HLU	11.74	12.37	14.44	15.98	15.39
Conv-React-Diff	AMG	17.42	18.49	18.42	20.10	20.65
	HLU	14.06	15.37	16.44	18.21	19.19
Lake Superior	AMG	8.78	9.25	11.56	12.91	14.10
	HLU	8.95	9.03	11.07	12.06	12.60

	N	$50k$	$100k$	$150k$	$200k$	$250k$
Simple	AMG	7.69	17.61	29.86	42.55	50.44
	HLU	7.54	15.98	24.61	32.57	41.50
Conv-React-Diff	AMG	8.38	19.28	32.22	47.95	57.83
	HLU	8.75	18.21	28.92	38.12	51.33
Lake Superior	AMG	5.42	12.71	22.11	29.39	38.97
	HLU	6.27	12.06	18.53	25.71	32.17

We thank Lars Grasedyck and Ronald Kriemann for very helpful discussions concerning nested dissection in the context of $\mathcal{H}$-matrices, and for assistance with the use of the $\mathcal{H}$-matrix Library. We also thank Wolfgang Hackbusch for the suggestion of developing an $\mathcal{H}$-matrix based domain decomposition algorithm for use within the Bank-Holst paradigm. This paper introduces one realization of this goal - one which requires little adjustment of existing software.

References

[1] R. E. Bank. PLTMG: A software package for solving elliptic partial differential equations, users' guide 9.0. Technical report, University of California, San Diego, 2004.

[2] R. E. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM J. Sci. Comput.*, 22(4):1411–1443 (electronic), 2000.

[3] R. E. Bank and M. J. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Review*, 45(2):291–323, 2003.

[4] R. E. Bank and S. Lu. A domain decomposition solver for a parallel adaptive meshing paradigm. *SIAM J. Sci. Comput.*, 26(1):105–127 (electronic), 2004.

[5] R. E. Bank and R. K. Smith. Mesh smoothing using a posteriori error estimates. *SIAM J. Numerical Analysis*, 34:979–997, 1997.

[6] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numer. Math.*, 95(1):1–28, 2003.

[7] S. Börm, L. Grasedyck, and W. Hackbusch. Hierarchical Matrices. Lecture Note 21 of the Max Planck Institute for Mathematics in the Sciences, 2003.

[8] S. L. Borne, L. Grasedyck, and R. Kriemann. Parallel black box domain decomposition based $\mathcal{H}$-LU preconditioning. *Math. Comp.*, submitted Jan. 2005.

[9] I. Brainman and S. Toledo. Nested-dissection orderings for sparse *LU* with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 23(4):998–1012 (electronic), 2002.

[10] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973. Collection of articles dedicated to the memory of George E. Forsythe.

[11] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[12] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[13] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates. *J. Comput. Appl. Math.*, 125(1-2):479–501, 2000. Numerical analysis 2000, Vol. VI, Ordinary differential equations and integral equations.

[14] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.

[15] S. LeBorne, L. Grasedyck, and R. Kriemann. Domain-decomposition based $\mathcal{H}$-lu preconditioners. In *16th International Conference on Domain Decomposition Methods*, Lecture Notes in Computational Science and Engineering, New York, to appear. Springer Verlag.