To appear in Linear Algebra and its Applications 2009

MULTILEVEL SPARSE APPROXIMATE INVERSE PRECONDITIONERS FOR ADAPTIVE MESH REFINEMENT[*]

SHUN WANG[†] AND ERIC DE STURLER[‡]

We dedicate this paper to Henk van der Vorst, for his friendship, inspiration, and leadership in numerical linear algebra and numerical algorithm development, on the occasion of his 65th birthday.

Abstract. We present an efficient and effective preconditioning method for time-dependent simulations with dynamic, adaptive mesh refinement and implicit time integration. Adaptive mesh refinement greatly improves the efficiency of simulations where the solution develops steep gradients in small regions of the computational domain that change over time. Unfortunately, adaptive mesh refinement also introduces a number of problems for preconditioning in (parallel) iterative linear solvers, as the changes in the mesh lead to structural changes in the linear systems we must solve. Hence, we may need to compute a new preconditioner at every time-step. Since this would be expensive, we propose preconditioners that are cheap to adapt for dynamic changes to the mesh; more specifically, we propose preconditioners that require only *localized changes to the preconditioner* for *localized changes in the mesh*.

Our preconditioners combine sparse approximate inverses and multilevel techniques. We demonstrate significant improvements in convergence rates of Krylov subspace methods and significant reductions of the overall runtime. Furthermore, we demonstrate experimentally level-independent convergence rates for various problems.

Key words. sparse approximate inverse, multilevel, adaptive mesh refinement, preconditioning, Krylov subspace methods.

AMS subject classifications. 65F10

1. Introduction.

Adaptive Mesh Refinement (AMR) was proposed by Berger and Oliger in 1984 to solve hyperbolic partial differential equations (PDEs) with discontinuous coefficients, shocks, and steep gradients [9, 8]. The method uses finer meshes to represent areas where the solution changes rapidly, while it uses coarser meshes to represent areas where the solution changes more slowly. In this way the method achieves high accuracy while keeping the computational cost low by limiting the number of mesh cells or elements. AMR has become increasingly popular for a wide range of problems beyond the solution of hyperbolic PDEs, such as parabolic PDEs, problems on domains that change shape over time, and optimization problems in which some property changes that requires accurate spatial resolution, for example the anomaly to be resolved in a tomography problem [1]. To make AMR more flexible and efficient, especially for parallel machines, the computational domain is usually partitioned into many small mesh blocks, each of which consists of a fixed small number of mesh cells or elements representing a uniform mesh. The refinement and derefinement of the mesh lead to structural changes in the system matrix; new rows and columns may be introduced and existing rows and columns may be removed. Moreover, in parallel implementations the mesh blocks are redistributed over the processors after (each) mesh refinement or derefinement to maintain a good load balance; see Figure 1.1. Load balancing may require the redistribution of mesh blocks even on

[*]This work was supported in part under a Round III Grand Challege Cooperative Agreement with NASA's Computational Technologies Project, grant NASA NCC 5-615, through the IBEAM Project and in part by the Materials Computation Center (UIUC) through grant NSF-DMR 0325939.

[†]Department of Computer Science, University of Illinois at Urbana-Champaign.

[‡]Department of Mathematics, Virginia Tech.

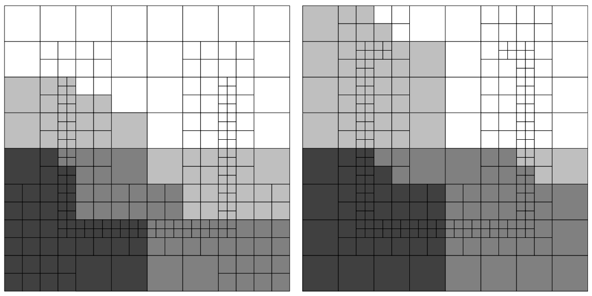

Fig. 1.1. *Typical block distribution before and after mesh refinement. The gray scales indicate on which processor a grid block resides.*

processors where the mesh did not change. So, we need preconditioners that accommodate the frequent changes in the mesh and the data redistributions. Unfortunately, these consequences of AMR make many popular preconditioners unfavorable.

First, preconditioners that depend explicitly on the matrix and the matrix ordering, such as incomplete factorizations like ILU and IC [40, 41, 49], are hard to update for structural changes to the matrix that result from mesh refinement and derefinement. Moreover, the factorization is typically defined with respect to a chosen ordering of the unknowns; every row and column in the (factors of) the preconditioner depends on previous rows and columns and choices with respect to fill-in. Even if the mesh changes only locally, the localized changes in the system matrix generally affect the factorization of many rows and columns. In addition, in a parallel implementation, the forward and backward substitutions in preconditioners based on incomplete factorizations lead to synchronization problems among the processors. Although various methods exist to limit such dependencies for each processor to neighboring processors only while maintaining reasonable convergence [50, 24, 43, 44, 33, 29], a redistribution of mesh blocks for load balancing following mesh adaptation will destroy such localized dependence and introduce global synchronization problems. Hence, these preconditioners seem unfavorable for problems with dynamic mesh refinement.

Second, domain decomposition preconditioners [26, 27, 10, 25] also appear less suitable for AMR, if the frequency of mesh adaptation is relatively high and regions with high mesh resolution traverse the computational domain. In this context, the decomposition of the domain into subdomains and, hence, the boundaries of the subdomains change frequently, which means that local factorizations need to be recomputed often, and coarse grid solvers or Schur complement preconditioners need to be computed frequently. This would be expensive.

Evaluating the drawbacks of popular preconditioners, we see that a good preconditioner for AMR should have the following properties. Computing or updating the preconditioner should require only local information from the mesh and the discretization (method). In addition, local updates of the preconditioner should be sufficient to maintain quality, and such local changes should be cheap. So, a local change in the mesh should not lead to a cascade of global changes in the preconditioner. Finally, for efficient parallel implementation, the redistribution of mesh cells should not greatly increase the cost of multiplying a (distributed) vector by the preconditioner. Considering these criteria, various explicit sparse approximate inverses make good candidates. These are sparse matrices that approximate the inverse of a sparse matrix A directly; they were first introduced in [2, 28, 3], and more recently have been greatly popularized by [6, 31, 23, 20, 22, 21, 7]. There are several ways to construct such sparse approximate inverses. A popular method is to minimize the Frobenius norm of $AM - I$, subject to some sparsity pattern for the preconditioner M, which may or may not be fixed a priori, [2, 28, 3, 31, 22, 20]. This gives an explicit representation of the approximate matrix inverse. We will refer to preconditioners of this type generically as SAI (sparse approximate inverse). An important alternative class of preconditioners constructs an approximate factorization of $A^{-1} \approx ZD^{-1}W^T$, where Z and W are unit upper triangular and D is diagonal. This includes FSAI [35] and AINV [6] and variants [5]. In many cases, these latter preconditioners yield more rapid convergence, but unfortunately they suffer from the same problems as ILU.

It turns out that SAI overcomes the difficulties mentioned above associated with AMR. Each column of the preconditioner depends only on the mesh in the immediate neighborhood of the mesh cell with which the column is associated. After mesh refinement, only those columns of the preconditioner that are associated with the changed mesh blocks (refined or removed) need computing or updating. Hence, we need to update only a few columns of the approximate inverse. Indeed, as we describe later, *a judicious use of ghost cells for each mesh block restricts updating of the preconditioner to only computing columns associated with new mesh cells (for linear PDEs).* This makes updating the approximate inverse very cheap. The approximate inverse is represented in explicit matrix form and applied to vectors by distributed matrix-vector multiplication. There is no forward or backward substitution for SAI. Therefore, the redistribution of mesh blocks does not seriously affect the cost of the multiplication by the sparse approximate inverse.

Unfortunately, the fact that SAI depends only on local information from the mesh and discretization also has drawbacks. In general, SAI does not approximate the inverse well for the smooth, global components of the solution that are often important in elliptic (like) problems, and this in turn often leads to slow convergence relative to other preconditioners. To approximate these components better, for many applications, a large sparsity pattern (a sparsity pattern with many nonzeros) is required for the approximate inverse. Obviously, a large sparsity pattern leads to high computational cost in computing the approximate inverse and in applying it at every iteration. Moreover, the desirable property that each column of the approximate inverse depend only on local information from the mesh and discretization is then lost. However, we propose to remedy this problem at low cost by combining the SAI preconditioner with multilevel corrections using sparse approximate inverses at coarser meshes. We can do this efficiently by exploiting the hierarchical nature of AMR meshes. This leads to a relatively simple approach that is efficient in computing and updating the preconditioner and highly effective in reducing the number of iterations with a Krylov subspace method. Various other approaches for combining sparse approximate inverses and multilevel techniques have been proposed, depending on the underlying problem. In [51, 16, 15, 42] sparse approximate inverses are used as smoothers for multigrid methods. In [14], sparse approximate inverses are computed in combination with wavelet-based transforms to derive a hierarchical structure. In [12, 55], algebraic information is explored to construct multilevel sparse approximate inverse preconditioners. We discuss these approaches and their relation to the present paper in some more depth in section 2.

Multigrid methods form an alternative for the present approach, and indeed the multilevel structure of the proposed preconditioners follows the multigrid concept of coarse grid corrections [52]. In this context, one can consider the sparse approximate inverses as smoothers, and, in many cases, sparse approximate inverses have been shown to be effective smoothers [16, 15]. However, in the context of parallel AMR simulations, multigrid has some disadvantages, in particular for problems with strong anisotropy. The traditional solutions for poor error smoothing in weakly coupled directions, semicoarsening and line smoothers, are not well-suited to parallel AMR simulations; see [19] and the references there. Note that [19] also proposes an alternative within the multigrid framework. Hence, we consider multigrid an important alternative, and one might view the solution approach in this paper as a Krylov acceleration of a multigrid method (or multigrid as a preconditioner) [47, 48]. However, we do not follow some of the key ingredients of multigrid. The preconditioner at each level is applied once, irrespective of whether high frequency components are sufficiently damped, and we do not assume an accurate solve at the coarsest grid. Hence, it would be hard to analyze the algorithm within the multigrid framework. We view our approach as generating a sufficiently good preconditioner at each level to affect strong clustering of the eigenvalues. The latter generally leads to fast convergence. This will also be the setting to analyze the proposed preconditioners theoretically, but we will not do this in the present paper. A brief overview of various multilevel solver approaches related to preconditioners can be found in [4].

The remainder of this paper is organized as follows. In section 2, we discuss the motivation for multilevel sparse approximate inverse preconditioners, and we outline some previous approaches. In section 3, we introduce our proposed multilevel sparse approximate inverse preconditioners for adaptively refined meshes. In section 4, we give numerical experiments with convergence and timing results. Finally, in section 5, we provide conclusions and discuss future extensions of this work. Although parallelization is part of the motivation for our choice of preconditioner, we will not test parallel implementations in this paper. In this paper, we focus on the underlying principles and numerical experiments to analyze convergence and sequential runtimes.

2. Multilevel sparse approximate inverse preconditioners.

In this paper, we consider the solution of linear time-dependent diffusion and convection-diffusion problems on adaptive meshes of the form

$$u_t = (a_1 u_x)_x + (a_2 u_y)_y + b_1 u_x + b_2 u_y + cu + f,$$

where the coefficient functions a_1, a_2, ..., c depend only on space, and f may depend on space and time. We discretize these partial differential equations using finite differences in space and time, and we use implicit time integration by either the backward Euler method or the Crank-Nicolson method. This results in systems of linear equations of the form

$$Au^{(n+1)} = Bu^{(n)} + g, \tag{2.1}$$

which must be solved for A. Our purpose is to propose efficient preconditioners that are cheap to compute and update and that lead to fast convergence for these linear systems. If the coefficient functions do not change with time, the linear system (2.1) depends only on the time step and the mesh. In many situations, the time step is fixed (or varies in a minor way), but mesh refinement and derefinement locally change the discretization. So, we need preconditioners that are cheap to update for such local changes of the mesh. For nonlinear problems, the coefficient functions will depend on u as well, leading to a system of nonlinear equations. In addition to structural changes in the matrix (Jacobian) due to mesh adaptation, there will also be changes in coefficients due to the nonlinear nature of the problem. However, in many problems, the time step will be such that, even in this case, large changes in the coefficients will occur at only a few places and the preconditioner can still be updated in a cheap localized fashion [56].

2.1. Sparse approximate inverse preconditioners. We consider the linear system $Ax = b$ and a right preconditioner M leading to the preconditioned system $AMy = b$ with $x = My$. We want to choose M such that AM is a good approximation to the identity matrix and M is cheap to compute, update, and apply. A popular way to compute M is to minimize the Frobenius norm of $AM - I$ [31, 22, 20]. Since

$$\|AM - I\|_F^2 = \sum_j \|Am_j - e_j\|_2^2, \tag{2.2}$$

where the m_j are the columns of M, we can compute each column of M independently by minimizing $\|Am_j - e_j\|_2$ for a given sparsity pattern (with a few nonzeros per column). So, M can be computed in parallel, solving small least squares problems, and stored in explicit matrix form.

For our time-dependent (convection-)diffusion problems, the exact inverse of a system matrix is full. This can easily be seen by considering the Green's functions for a 1D diffusion problem. The columns of the exact inverse of the system matrix are the discrete analogues of Green's functions. However, the largest coefficients in the inverse correspond to the mesh points around the point source. Therefore, one typically chooses a sparsity pattern for M that contains only a few neighboring mesh points. This also makes it cheap to compute and apply M [31, 22, 20].

The choice of the sparsity pattern is usually the key issue for an effective sparse approximate inverse preconditioner. A small sparsity pattern yields a cheap preconditioner, but generally leads to slow convergence. A larger sparsity pattern for M would lead to including more global information per iteration, which alleviates this slow convergence problem. However, for fast convergence, a large number of nonzeros per column are required in the approximate inverse, which results in high computational cost in both constructing and applying the sparse approximate inverse. This makes the preconditioner too expensive.

Before introducing our approach to improve SAI in the next section, we review this problem further to provide the basic ideas for our proposed preconditioners. We point out that the ideas presented in this subsection have been discussed also in one form or another in the papers cited below and in [15, 16]. In [11], Bollhöfer and Mehrmann observe for the Laplace equation that, although most eigenvalues of the residual matrix for the right preconditioned system, $E = I - AM$, are small, there are a number of eigenvalues very close to 1, and even a significantly larger stencil (more nonzero coefficients) of the approximate inverse M does not cure this. The eigenvalues of E that are close to 1 correspond to smooth eigenvectors. We demonstrate this for the standard implicit (in time) discretization of the 1D diffusion problem $u_t = u_{xx}, x \in (0, 1)$ with homogeneous boundary conditions, and with $\Delta x = 1/128$ and $\Delta t = 1/128$. Figure 2.1 shows the eigenvalues of the matrix $E = I - AM$ and the eigenvectors corresponding to the largest three eigenvalues of E. This example demonstrates again that sparse approximate inverses are not good at handling low frequency modes.

To understand the importance of the smooth global modes for a sufficiently accurate approximation of the matrix inverse, we now consider the Green's function

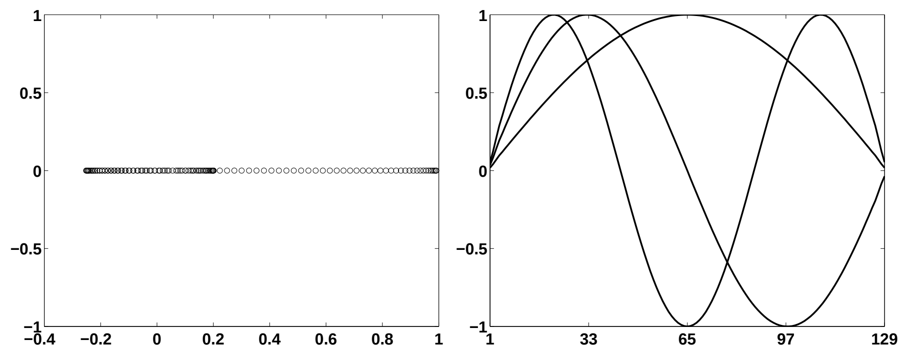

Fig. 2.1. *Eigenvalues and eigenvectors for the residual $E = I - AM$ arising from an implicit time step of the discretized diffusion equation $u_t = u_{xx}, x \in (0,1)$, with $\Delta x = \Delta t = 1/128$. Left: eigenvalues of $E = I - AM$; right: eigenvectors corresponding to the largest 3 eigenvalues of E.*

$g(x,\tau)$ for the equation

$$-au_{xx} + u = f \quad \text{on } (0,1), \quad u(0) = u(1) = 0. \tag{2.3}$$

We can represent the solution $u(x)$ as

$$u(x) = \int_0^1 g(x,\tau) f(\tau)\, \mathrm{d}\tau, \tag{2.4}$$

where the Green's function is (see [13, p. 95])

$$g(x,\tau) = \sum_{k=1}^{\infty} \frac{2\sin k\pi x}{1 + ak^2\pi^2} \sin k\pi\tau. \tag{2.5}$$

Analogously, we can represent the discrete solution of $Au = f$ by

$$u = \sum_{j=1}^{n} (A^{-1})_j f_j,$$

where $(A^{-1})_j$ indicates the j-th column of A^{-1}. Hence, A^{-1} is the discrete analogue of the Green's function $g(x,\tau)$, each column $(A^{-1})_j$ representing the approximate solution for a point source at a grid point, $f(\tau) = \delta(\tau - \tau_0)$. It is clear from (2.5) that the lower frequency components (small k) have much larger weights than the higher frequency components (large k). Analogously, substituting $f(\tau)$ above and $g(x,\tau)$ from (2.5) in (2.4) immediately shows that the columns of A^{-1} have relatively large low frequency components and small high frequency components. In fact, (2.4) and (2.5) indicate that, unless f has very large high frequency components, the solution is largely determined by the low frequency components. Therefore, we cannot expect to approximate A^{-1} accurately, unless we represent the low frequency components reasonably accurately. The issue then is to do this at low cost.

Unfortunately, accurately representing low frequency components with respect to the basis imposed by the mesh requires the approximation to the inverse, M, to be fairly dense (even if many of the coefficients are relatively small). This makes the construction of M and the matrix-vector product with M very expensive, especially on a parallel computer. So, for the purpose of efficiency, we require a practical sparsity

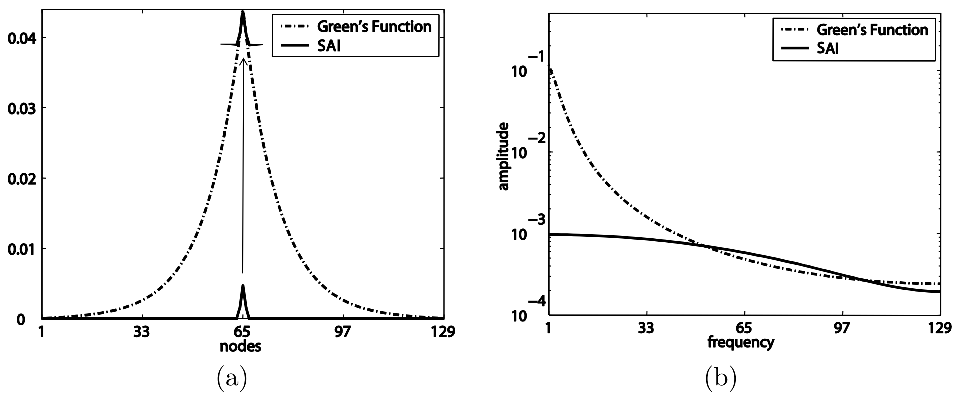

FIG. 2.2. *Left: The Green's function for (2.3) with a point source in the middle of the domain and its SAI approximation. To emphasize that SAI approximates the tip of the Green's function, we also superimpose the computed approximation on the Green's function. Right: The Green's function and its SAI approximation in the frequency domain. Notice the poor approximation for low frequencies.*

pattern for an approximate inverse to have a small local stencil, often the same as that of the matrix A itself. In that case, the Frobenius norm minimization (2.2) gives the columns of the approximate inverse a small wedge shape (see Figure 2.2(a)), which approximates the tip of the Green's function. In Figure 2.2(b) we plot the frequency decomposition of this wedge shaped function. We see that SAI approximates the Green's function well for high frequency modes, but has large errors for low frequency modes. Due to the local support property, SAI is unable to capture the low frequency components well.

This problem has been recognized by several people, and various methods have been proposed as a remedy [14, 12, 18, 17]. The common underlying idea for these approaches is to construct a new basis, such that the discrete representation of the Green's function with respect to this basis is nearly a diagonal matrix; that is, outside a narrow band the coefficients of the matrix inverse are nearly zero. This allows an accurate approximation of the inverse by a sparse approximate inverse with very few nonzeros. In these approaches one has to construct the new basis, the basis transformation and its inverse, and the representation of the approximate inverse with respect to this new basis. This procedure is not cheap, but for hard problems may pay off in a greatly reduced number of iterations. In [14, 18, 17] the authors discuss various approaches using hierarchical wavelet bases. In particular, in [14] two hierarchical wavelet bases are constructed using second generation wavelets, so that unstructured meshes can be handled well. The idea is that smooth regions in the Green's function lead to small wavelet coefficients that are accurately approximated by zeros. On the other hand, in [12] strictly algebraic information is used to find a multilevel basis. To be specific, the coarsening process is based on the construction of the sparse approximate inverse.

Although we also aim to represent better approximations to the smooth components of the inverse in an efficient way using alternative bases, our approach is quite different. Instead of using new bases (explicitly or implicitly), we exploit the hierarchy of meshes already present for the AMR representation. This avoids the need for any new bases, and hence keeps the computational cost of our approach very low (see the numerical experiments). This is all the more important for us since we aim at

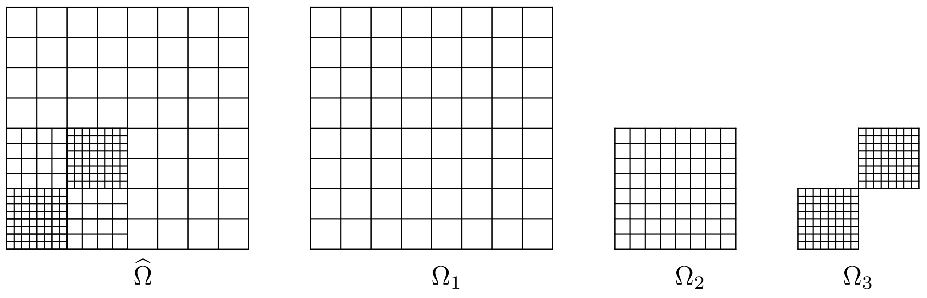

Fig. 3.1. *A hierarchy of uniform meshes.*

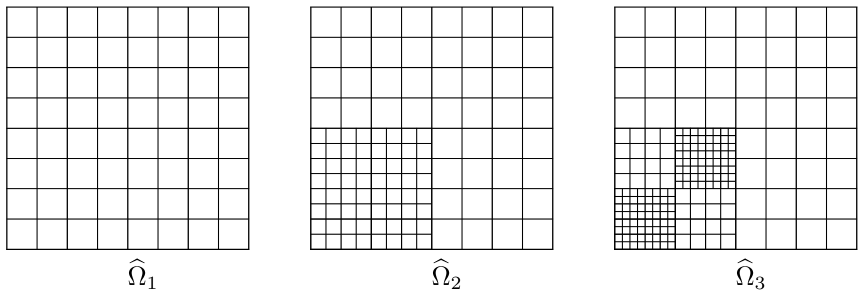

Fig. 3.2. *Composite meshes corresponding to a hierarchy of uniform meshes.*

problems where the mesh possibly changes every time step. The smooth components of the Green's functions can be represented cheaply and reasonably accurately using only a few nonzero coefficients at coarse level meshes. Hence, we do not aim for a basis in which many or most coefficients of the approximate inverse can be approximated accurately by zeros. Rather, we exploit the hierarchy of meshes to approximate the components of the Green's function as economically as possible at the appropriate level. The key observation is that representing 'most' of M at the coarse levels leads to efficient storage of M and makes the multiplication by M very cheap. This leads to an efficient hierarchical preconditioner that is cheap to update for changes in the mesh. We will introduce our method in detail in the next section.

3. Multilevel Sparse Approximate Inverse Preconditioners for Adaptively Refined Meshes.

First, we introduce some useful notation for adaptive (AMR) meshes and the matrices represented on these meshes.

Adaptive mesh refinement yields a hierarchy of uniform meshes; we denote the uniform mesh at level ℓ by Ω_ℓ (see the 2D example in Figure 3.1). Higher level meshes have increasingly finer resolution, typically cover smaller and smaller subdomains, and are restricted to parts of the domain covered by the next lower level (forming a hierarchy of meshes). Note that higher level meshes need not be contiguous. In addition to these uniform meshes, we consider compositions of uniform meshes. As shown in Figure 3.2, we recursively define $\widehat{\Omega}_\ell$ as the composite mesh that results from combining meshes Ω_ℓ and $\widehat{\Omega}_{\ell-1}$, excluding those mesh components (points, faces, cells) from $\widehat{\Omega}_{\ell-1}$ that are *covered* by Ω_ℓ. The initial composite mesh is $\widehat{\Omega}_1 = \Omega_1$. In many applications, we have a minimum level of refinement ℓ^*, i.e., Ω_{ℓ^*} covers the whole domain. So, $\widehat{\Omega}_\ell = \Omega_\ell$ for $\ell \leq \ell^*$. For mesh levels above ℓ^*, we have adaptive local refinements as required by solution accuracy.

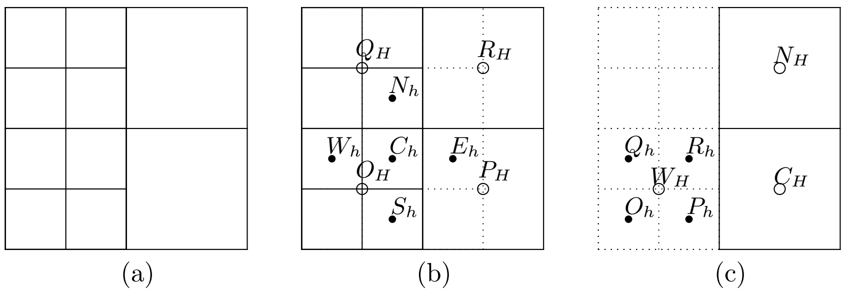

Fig. 3.3. *Matrix operations on a nonuniform mesh. (a) The boundary between two mesh levels. (b) The fine grid mesh cell C_h has a ghost cell E_h as its east neighbor. Since there is no real mesh cell corresponding to E_h, the ghost cell value is computed by an interpolation using coarse grid mesh cells O_H, P_H, Q_H and R_H. (c) The coarse grid mesh cell C_H has a ghost cell W_H as its west neighbor. Since the corresponding real mesh cell has been refined, the ghost cell value is computed by a restriction using fine grid mesh cells O_h, P_h, Q_h and R_h.*

Although the resulting composite meshes are nonuniform and the uniform meshes at a particular refinement level are noncontiguous, we can define operations on these meshes as if they were uniform meshes by the use of *ghost cells*. We define one or more layers of ghost cells around each mesh block, which forms a small part of the uniform mesh at a particular level. Any operation on a (local) group of cells can now be implemented as if the mesh is uniform, using the ghost cell values for references to mesh cells in neighboring mesh blocks. Of course, these ghost cell values must be computed (or copied) before carrying out this operation; see below. This use of ghost cells has two major advantages: (1) computations on mesh cells can be defined in a simple, consistent way, and (2) changes to the mesh do not change the matrix coefficients except for the creation and removal of matrix rows corresponding to the creation and removal of mesh blocks. For example, we do not need to change the matrix row for a particular variable associated with a mesh block if neighboring mesh blocks are removed by mesh derefinement.

The values at ghost cells are determined in one of three ways. If the corresponding neighboring mesh block has the same refinement level, the ghost cell takes its value from the neighboring real mesh cell (a copy). If the neighboring mesh block has a lower refinement level (coarser), the ghost cell value is computed by an interpolation from the coarse level (existing) real mesh cells. If the neighboring block has a higher refinement level (finer), the ghost cell value is computed by a restriction from the higher level real mesh cells (often trivial restriction or averaging). See Figure 3.3. So, using ghost cells, the coefficients for a matrix defined on a nonuniform mesh are the same as those for a matrix defined on a uniform mesh. The matrix-vector product is implemented in two steps. First, the ghost cells are assigned a value by a copy, an interpolation, or a restriction. Then, using the ghost cells, the matrix-vector product is carried out as for a matrix defined on a uniform mesh. In this way, local changes to mesh blocks do not affect the rows of the matrix corresponding to cells in other mesh blocks.

For more details, especially on implementation, we refer to the documentation of the PARAMESH package [38, 39, 46, 45], which we use to implement hierarchical AMR meshes and related data structures, and to [56]. PARAMESH is a FORTRAN90 parallel package for developing multidimensional AMR simulations. It builds a hierarchy of meshes with varying spatial resolution. The meshes consist of (small) mesh

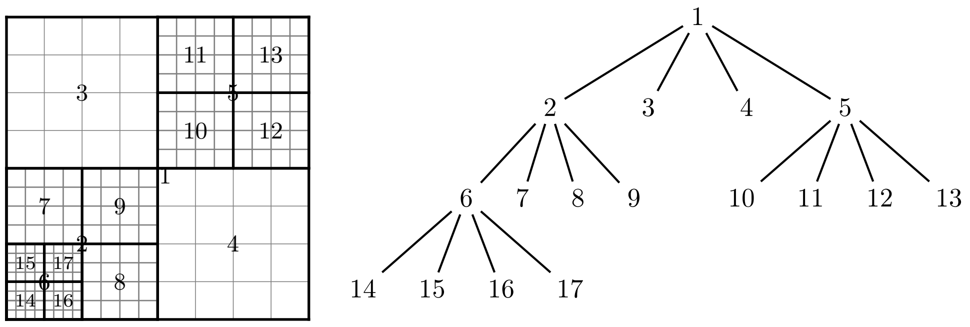

Fig. 3.4. *Mesh structure of PARAMESH.*

blocks arranged in a tree data structure (see Figure 3.4). The mesh blocks are, typically, of small size, e.g. 4×4, to enable easy load balancing and to make AMR more flexible. Many other packages that support adaptive mesh refinement exist, such as `libMesh` [34], SAMRAI [32], and CLAWPACK [37, 36].

To make the matrix-vector product cheap and convenient, especially on parallel machines, we store our sparse matrices row by row with the corresponding mesh blocks. Let A_ℓ be the matrix defined on composite mesh $\widehat{\Omega}_\ell$. For the experiments in this paper, we choose the sparsity pattern of A_ℓ as the sparsity pattern for M_ℓ (the sparse approximation to A_ℓ defined on $\widehat{\Omega}_\ell$), but this is easily generalized to other (local) patterns, such as the sparsity pattern of A_ℓ^k [22, 20]. Since we store the matrix row by row, it is convenient to use left preconditioning, solving the preconditioned system $MAx = Mb$ (with residual matrix $E = I - MA$). Hence, on each composite mesh $\widehat{\Omega}_\ell$, we define M_ℓ as the matrix that minimizes

$$\|M_\ell A_\ell - I\|_F = \|A_\ell^T M_\ell^T - I\|_F \tag{3.1}$$

given the chosen sparsity pattern, instead of the matrix that minimizes $\|A_\ell M_\ell - I\|_F$, which is more commonly used for SAI preconditioning; see also [4]. However, our multilevel preconditioning methods introduced below can be used for both left and right preconditioning. Due to the locality of the sparsity pattern, each row of M_ℓ associated with a given mesh cell depends only on the rows of A_ℓ that correspond to neighboring mesh cells (and hence potentially neighboring mesh blocks). We can consider the definition of the matrices M_ℓ based on a sequence of adaptive refinements generating a given locally refined mesh. In practice, of course the matrices will be the result of refinements and derefinements. We start by defining M_1 on $\widehat{\Omega}_1 = \Omega_1$ based on (3.1). Next we compute M_2 on $\widehat{\Omega}_2$ by computing only the rows for new mesh cells generated by mesh refinements on Ω_1. The use of ghost cells, discussed above, ensures that we do not need to make any changes to rows corresponding to mesh blocks that are not refined. We continue this process until we have defined the finest composite mesh. Our use of ghost cells also means that the rows of any M_ℓ remain unchanged if the corresponding cells themselves are not refined or derefined in mesh adaptations. Hence, for mesh refinements we need only compute new rows for the matrix and preconditioner for newly created mesh blocks (and their mesh cells). For derefinements we simply remove the rows corresponding to blocks (and their cells) that are removed. *No further computations are required.*

Using the definitions of composite meshes and matrices above, we now define our multilevel preconditioner in terms of the matrices A_ℓ and their sparse approximate inverses, M_ℓ, for each composite mesh $\widehat{\Omega}_\ell$. We first define a two-level preconditioner, and then define the multilevel version recursively. Algorithm 1 describes the multiplication of a vector z defined on the fine, composite mesh $\widehat{\Omega}_h$ by the two-level preconditioner P_2 for $\widehat{\Omega}_h$ using a coarse, composite mesh $\widehat{\Omega}_H$. The coarse level mesh, $\widehat{\Omega}_H$, does not need to be the next coarser mesh of $\widehat{\Omega}_h$. It can be an arbitrary coarser mesh.

Algorithm 1 Compute $y \leftarrow P_2 z$.

```
1: Multiply $z$ by the fine mesh sparse approximate inverse: $\tilde{y} \leftarrow M_h z$
2: Compute fine mesh defect: $d_h \leftarrow z - A_h \tilde{y}$
3: Restrict $d_h$ to the coarse mesh: $d_H \leftarrow I_h^H d_h$
4: Multiply $d_H$ by the coarse mesh sparse approximate inverse: $e_H \leftarrow M_H d_H$
5: Compute prolongation of $e_H$ to the fine mesh: $e_h \leftarrow I_H^h e_H$
6: Add coarse mesh correction to the preconditioned vector: $y \leftarrow \tilde{y} + e_h$
```

The final result $y = P_2 z$ consists of an initial approximation $M_h z$ to $A_h^{-1} z$ (step 1), defined by the usual sparse approximate inverse preconditioner M_h, and a coarse mesh correction using M_H (steps 2-6). We denote this two-level method by SAI2. In SAI2, $\widehat{\Omega}_h$ is $\widehat{\Omega}_{\ell_{\max}}$, the finest composite mesh, whereas, $\widehat{\Omega}_H$ need not be the next coarser grid, $\widehat{\Omega}_{\ell_{\max}-1}$. A good choice for $\widehat{\Omega}_H$ is $\widehat{\Omega}_{\ell^*}$, since $\widehat{\Omega}_{\ell^*}$ is invariant. Although we do not try this in the present paper, it seems worthwhile to construct a more accurate sparse approximate inverse at this invariant level, as it has to be computed only once. The preconditioning operator for SAI2 is

$$P_2 = M_h + I_H^h M_H I_h^H (I - A_h M_h), \tag{3.2}$$

and its residual matrix is

$$E_2 = I - P_2 A_h = (I - I_H^h M_H I_h^H A_h)(I - M_h A_h).$$

The matrix $I - I_H^h M_H I_h^H A_h$ dampens low frequency modes (of the error) much better than $I - M_h A_h$.

To turn this two-level preconditioner into a multilevel preconditioner, we replace $e_H = M_H d_H$ in step 5 of Algorithm 1 by a multilevel correction based on a recursive application of Algorithm 1. On the coarsest mesh $\widehat{\Omega}_1$ we define the preconditioner $P^{(1)} = M_1$. On the next level, $\widehat{\Omega}_2$, we define

$$P^{(2)} = M_2 + I_1^2 P^{(1)} I_2^1 (I - A_2 M_2), \tag{3.3}$$

and more generally, we define

$$P^{(\ell)} = M_\ell + I_{\ell-1}^\ell P^{(\ell-1)} I_\ell^{\ell-1} (I - A_\ell M_\ell). \tag{3.4}$$

The sequence of coarse(r) level corrections combines more global information. We refer to our *sparse approximate inverse preconditioner with multilevel corrections* as SAI-MC. In most cases, we apply the coarse mesh correction recursively to level 1 (coarsest level). However, for certain problems, the mesh width on the coarsest levels may be too large for the coarse mesh operator to capture rapid changes in the local equations, for example, rapid changes in coefficient functions of a partial differential equation. At that point, corrections on those coarsest levels are often not effective, and the recursive scheme should stop at the lowest level that has sufficient resolution

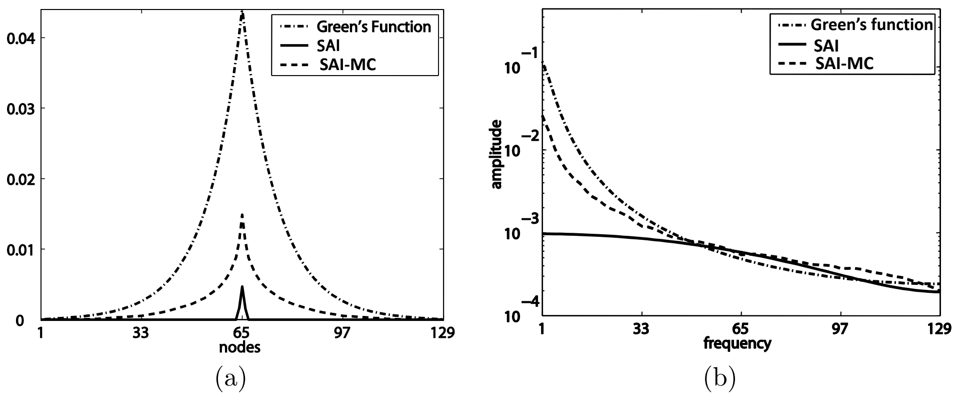

Fig. 3.5. *Left: The Green's function for (2.3) with a point source in the middle of the domain and its approximation by SAI and by SAI-MC. Right: The Green's function and its approximation by SAI and by SAI-MC in the frequency domain. Notice the much better approximation by SAI-MC for low frequencies.*

to capture the relevant physics. Alternatively, we can use special schemes to derive proper coarse mesh operators for those mesh levels [52].

In Figure 3.5, we show the improvement of SAI-MC over SAI for the 1D problem given in (2.3), cf. Figure 2.2. SAI-MC leads to a much better approximation of the Green's function, especially for the low frequency modes. By incorporating global information through multilevel corrections, we obtain a very good approximation to the exact inverse at a fairly low cost. We discuss this issue further in section 4.

4. Numerical results.

We present results for three two-dimensional model problems. The first two model problems are variations of a model problem in [53]. All spatial derivatives are approximated by standard, central, finite differences, except at the boundaries between levels, where the discretization is adapted as described in the previous section. For the time derivative, we use either backward Euler or the trapezoid rule (Crank-Nicolson). However, all experiments in this section were done using the backward Euler method. The discretization of a partial differential equation (PDE) on a nonuniform mesh using ghost cells leads to slight nonsymmetry of the resulting linear systems for the coefficients defined on or near the interface between (sub)meshes with different levels of refinement, even if the PDE itself is self-adjoint. Hence we use BiCGStab [53] as the solver in all cases, as it is appropriate for nonsymmetric systems and does not require a matrix transpose. In the discussion of our experiments we use 'NONE' to denote no preconditioning, 'SAI' to denote the standard sparse approximate inverse preconditioner (with the same sparsity pattern as the system matrix), 'SAI2' to denote the two-level sparse approximate inverse preconditioner (3.2) described in Algorithm 1, with $\widehat{\Omega}_{\ell^*}$ as the uniform coarse mesh (the mesh at the minimum refinement level ℓ^*, discussed in the previous section), and 'SAI-MC' to denote the full multilevel sparse approximate inverse (with coarse mesh correction on all levels). In our experiments $\ell^* = 4$ corresponding to a 32×32 uniform mesh. Moreover, we use M, P_2, and P_m to denote the preconditioning matrices for SAI, SAI2 and SAI-MC respectively.

To compare the various preconditioners, we discuss the clustering of eigenvalues, generally a good indicator for the effectiveness of a preconditioner to reduce the number of iterations, the convergence of preconditioned BiCGStab for multiple time

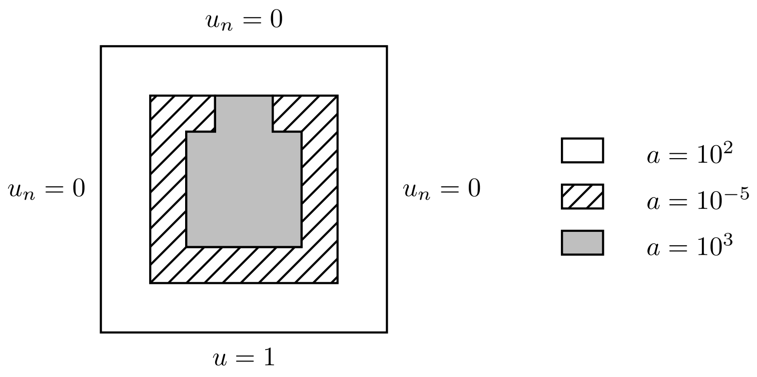

FIG. 4.1. *DIFF: diffusion problem* $u_t = \nabla \cdot (a\nabla u)$ *in the unit square* $[0,1] \times [0,1]$.

steps as measured by the number of iterations, and the CPU time of preconditioned BiCGStab for multiple time steps. In addition, we give the CPU time for the *local updates* to the preconditioner (at each level where refinements or derefinements occur) to emphasize how cheap updating our proposed multilevel preconditioner is. All timings were carried out on a Dell desktop computer with a 32-bit Intel processor running at 2GHz (under Linux).

Problem 1. The first problem is the diffusion equation $u_t = \nabla \cdot (a\nabla u)$ in the unit square, with a discontinuous coefficient function $a(x)$ and boundary conditions as shown in Figure 4.1. The initial value is $u = 0$ in the whole domain. We denote this problem as DIFF.

Before we discuss the spectrum of the preconditioned matrices and the performance of the various preconditioners, we provide some insight in the structure of the preconditioners as a (single) matrix defined at the finest composite mesh and the actual cost of multiplying a vector by each preconditioner in Table 4.1. The *algebraic sparsity* of each preconditioner is measured by the average number of nonzeros per column of the preconditioner given as a single matrix on the finest composite mesh. It indicates how much information the preconditioner uses for a single mesh cell. The *actual work* for the matrix-vector product with each preconditioner is measured by the average number of floating point operations per column in that matrix-vector product. We give this comparison for the first five time steps, with the maximum refinement level increasing from level 5 to level 8. In our numerical experiments, to demonstrate the efficiency of our preconditioning approach, we update the mesh at each timestep using a simple gradient based refinement criterion. The maximum refinement level ($\ell_{\max}$) and the number of unknowns (n) give an indication of the number of refinements (derefinements happen as well). The table illustrates the significant increase of the 'stencil' of SAI2 relative to SAI (note that the coarse level for the two-level preconditioner is level 5 independent of the highest refinement level), which typically results in a significant reduction in the number of iterations (see below). The table also shows that the SAI-MC preconditioner is essentially a dense matrix, but for a matrix-vector product requires only about 4 times the number of floating point operations that the standard 5-point SAI takes. Hence the cost of a matrix-vector product with the SAI-MC preconditioner scales linearly with the mesh size.

In Figure 4.2, we compare the spectra of the preconditioned systems, A, MA,

TABLE 4.1
Comparison of algebraic sparsity pattern and actual computational work for SAI (M), SAI2 (P_2) and SAI-MC (P_m) for the first five timesteps. The value of $\ell_{\max}$ indicates the highest level of refinement; n indicates the total number of unknowns in the finest composite mesh; the algebraic sparsity pattern is the average number of nonzeros per column of the preconditioner as a single matrix on the finest composite mesh; and the actual work gives the average number of floating point operations per unknown for the matrix-vector product.

time step	$l_{\max}$	n	algebraic sparsity pattern (nnz per column)			actual work (#flop/n)		
			M	P_2	P_m	SAI	SAI2	SAI-MC
1	5	4096	4.9	24.1	4022	10	30	26.6
2	6	6112	5.5	55.8	6027	10	26.8	38.0
3	7	12448	5.7	204.3	12420	10	23.2	41.6
4	8	23488	5.9	601.0	23421	10	21.8	44.2
5	8	27136	5.9	636.4	27110	10	21.6	42.6

TABLE 4.2
Convergence results (number of iterations) for DIFF.

time step	1	2	3	4	5
$\ell_{\max}$	5	6	7	8	8
n	4096	6112	12448	23488	27136
NONE	864	681	1296	2309	2159
SAI	118	111	164	235	265
SAI2	82	72	83	85	85
SAI-MC	17	18	16	17	19

P_2A, and P_mA for a typical time step. Note the differences in scale for the real axis (which are unavoidable in this case). The small imaginary part of a few eigenvalues is caused by the nonsymmetry of the matrix due to a nonuniform mesh. Standard SAI already gives a significantly better clustering of the spectrum compared with no preconditioning. However, it leaves many eigenvalues close to the origin. The two-level preconditioner SAI2 clusters more eigenvalues near 1 and, more importantly, leaves fewer eigenvalues close to the origin. The full multilevel preconditioner, SAI-MC, clusters nearly all eigenvalues in a small disk centered at 1 with a radius of about 0.15. A few eigenvalues lie outside the disk, but they are well-separated from the origin compared with the eigenvalues of the other preconditioners. Moreover, Krylov subspace methods converge rapidly for linear systems with a clustered spectrum and a few outlying eigenvalues [54, 30].

The convergence results for DIFF are listed in Table 4.2. The convergence criterion is

$$\frac{\|b - Ax_k\|_2}{\|b\|_2} < 10^{-12}.$$

We use this stringent tolerance, which is relatively small compared with the truncation error, to carefully evaluate the convergence and runtimes of the solver for the proposed preconditioners. SAI-MC significantly reduces the number of iterations compared with standard SAI and SAI2. Furthermore, the convergence rate appears to be level-independent as the mesh is (locally) refined from level 5 to level 8 in time steps 1 through 5 (same for SAI2). However, we solve a different system with a different

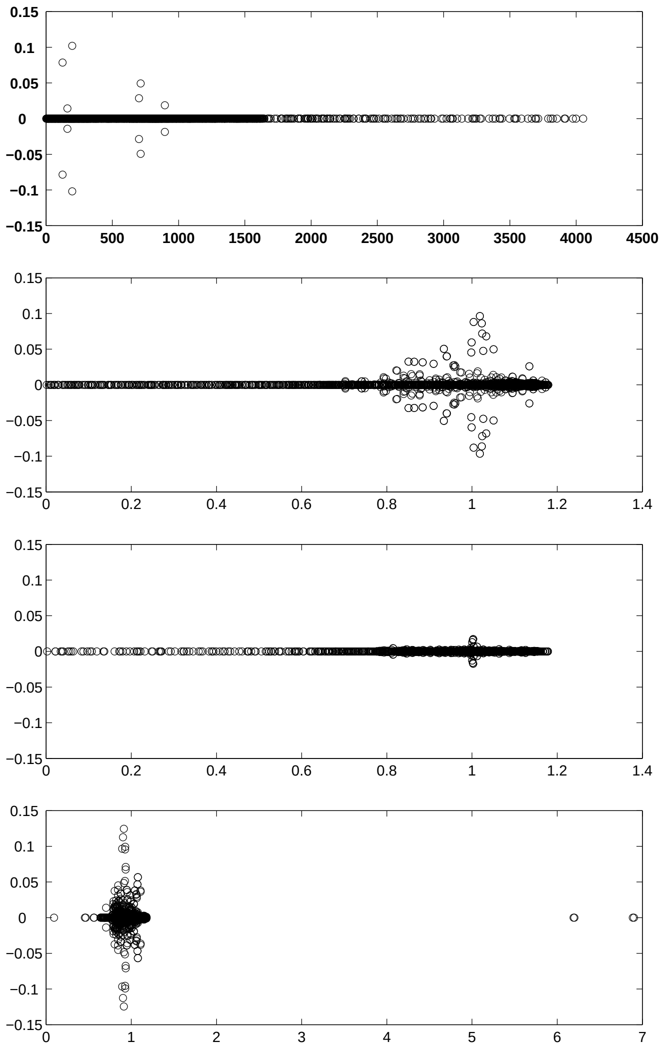

FIG. 4.2. *Spectra of A, MA, P_2A and P_mA in the complex plane for DIFF.*

right hand side at each time step. Therefore, we also check the number of iterations for the same (second) time step on meshes with different maximum refinement levels; the results are given in Table 4.3. Note the slight reduction of iterations for finer and finer meshes. This is often observed for multigrid methods and optimal multilevel preconditioners. Also note the roughly linear behavior of CPU time versus the number of unknowns (n). The multigrid method is known to have h-independent convergence for diffusion problems, but it has difficulties handling problems with discontinuous coefficients, strong convection, and strong anisotropy. We demonstrate experimentally in this and subsequent examples that we obtain level-independent convergence for exactly those problems with our multilevel preconditioner at very low cost (see CPU times below).

Table 4.4 gives the timing results for the DIFF problem. Although SAI-MC requires a small amount of extra work for the multilevel corrections compared with

TABLE 4.3
Convergence results of SAI-MC for different meshes at the 2nd time step of DIFF.

$\ell_{\max}$	n	niters	solver time
5	4096	16	0.92
6	6112	18	1.71
7	16096	14	3.26
8	36448	14	7.05

TABLE 4.4
Timing results (seconds) for DIFF.

time step	1	2	3	4	5	total
NONE	9.85	11.64	43.71	146.28	157.79	369.27
SAI	2.23	3.17	9.20	25.08	32.17	73.19
SAI2	2.82	3.55	7.42	13.75	15.84	44.72
SAI-MC	0.96	1.71	2.97	6.06	7.54	20.58
Update all M_ℓ	0.12	0.58	0.21	0.32	0.11	1.34

SAI and SAI2, it reduces the overall solver time by about a factor four compared with SAI and by about a factor two compared with SAI2. The CPU time for SAI-MC varies roughly linearly with the number of unknowns per system due to the level independent convergence rate. The cost of updating the preconditioner is about six percent of the solver time for SAI-MC (with BiCGStab) and less than two percent of the solver time for SAI. Note that the relative cost of updating the preconditioner goes down in further iterations as (generally) fewer changes are needed; in the first few time steps all columns must be computed.

Problem 2. Our next problem, CONVECT, is a convection-diffusion problem (with strong convection) $u_t = \nabla \cdot (a\nabla u) + bu_x$ on the unit square $[0,1] \times [0,1]$ with b shown in Figure 4.3(a) and the same diffusion coefficient a, boundary conditions, and initial solution as in DIFF. We use central differences for the convective terms. However, on the coarsest composite mesh that covers the whole domain ($\ell^* = 4$), the mesh Péclet number is less than two, so that a discrete maximum principle always holds.

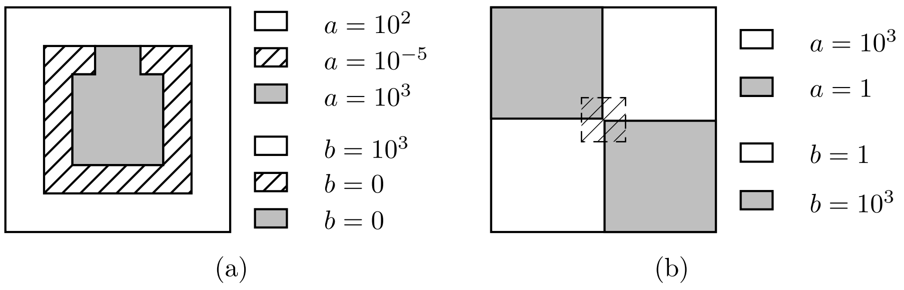

FIG. 4.3. *Left: coefficients of CONVECT; right: coefficients of ANISO.*

The spectra of the preconditioned matrices for the convection-diffusion problem on a typical adaptive mesh are given in Figure 4.4. The results are similar to those for the problem with only diffusion. The preconditioners SAI2 and SAI-MC show similar improvements over NONE and SAI here as for the diffusion problem, although the

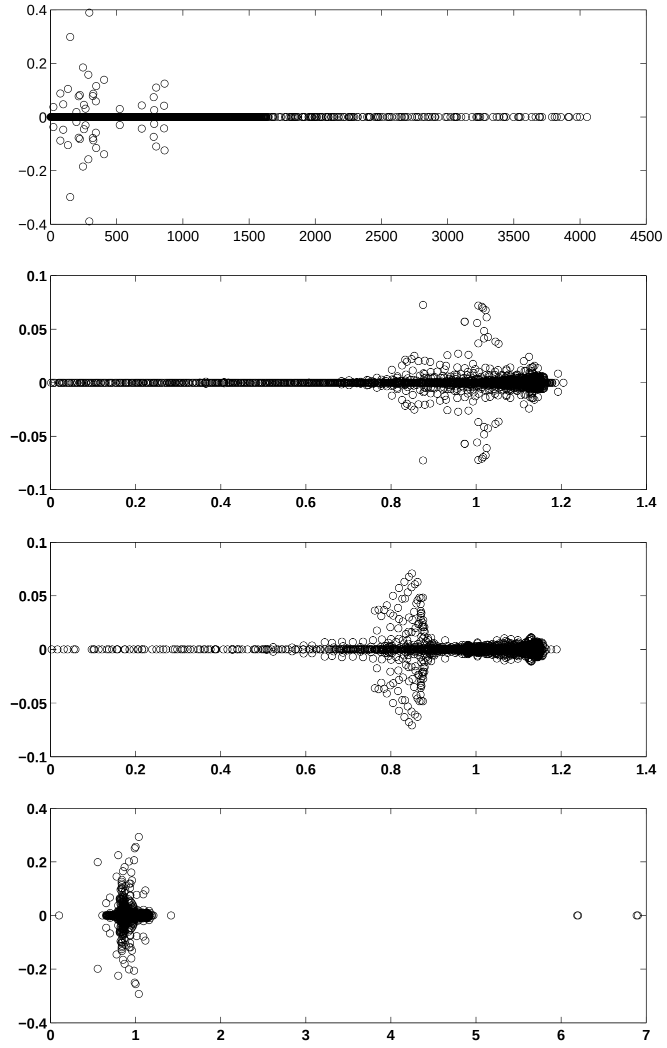

FIG. 4.4. *Spectra of A, MA, P_2A and P_mA for CONVECT.*

magnitudes of the imaginary parts of the eigenvalues are slightly larger, since the problem is not self-adjoint.

We give the convergence and timing results for CONVECT in Table 4.5. Again we see significant improvements in terms of both iteration count and solution time for the multilevel versions. In addition, both for SAI2 and SAI-MC we see that the number of iterations remains roughly constant for increasingly refined meshes. Notice that for the first two time steps the solution time with the SAI2 preconditioner is slightly longer than with the SAI preconditioner (also for DIFF). The solution time with the SAI-MC preconditioner is the shortest for each time step.

Problem 3. ANISO is a diffusion problem, $u_t = au_{xx} + bu_{yy} + f$, with strongly anisotropic, discontinuous diffusion coefficients, defined on the unit square with homogeneous boundary conditions. The coefficient functions $a(x, y)$ and $b(x, y)$ are given

Table 4.5
Convergence and timing results for CONVECT.

time step	1	2	3	4	5
$\ell_{\max}$	5	6	7	8	8
n	4096	6208	12064	23056	32848
	convergence (niters)				
NONE	832	692	1270	3985	10051
SAI	140	125	169	249	342
SAI2	90	80	84	98	92
SAI-MC	22	18	19	21	21
	timing (secs)				
NONE	9.47	12.00	41.88	248.78	882.01
SAI	2.65	3.62	9.23	25.83	49.64
SAI2	3.08	3.97	7.30	15.55	19.95
SAI-MC	1.25	1.72	3.46	7.35	9.83
Update all M_ℓ	0.14	0.69	0.22	0.36	0.32

Table 4.6
Convergence and timing results for ANISO.

timestep	1	2	3	4	5
$\ell_{\max}$	4	5	6	7	8
n	4096	12544	28672	42496	48640
	convergence (niters)				
NONE	140	312	558	1271	2563
SAI	54	127	291	306	430
SAI2	31	44	77	156	156
SAI-MC	28	40	34	36	37
	timing (secs)				
NONE	1.17	7.30	30.07	100.25	227.71
SAI	0.79	5.14	26.58	41.51	65.65
SAI2	0.89	2.86	9.91	29.33	33.49
SAI-MC	1.09	3.89	7.25	13.19	19.20
Update all M_ℓ	0.385	0.793	1.516	1.298	0.793

in Figure 4.3(b). The source term f is given by $f = 10$ in the small central square $[0.4, 0.6] \times [0.4, 0.6]$ and $f = 0$ elsewhere. The initial solution of ANISO equals $u = 0$.

We give the convergence results and the timing results for ANISO in Table 4.6. The multilevel sparse approximate inverse preconditioner, SAI-MC, significantly reduces the number of the iterations and the overall computational time. Moreover, it again achieves a level-independent convergence rate. Note that for this problem the use of the SAI2 preconditioner does not lead to level-independent convergence. We also see that for the first two time steps the full multilevel preconditioner is not faster (in time) than the 2-level preconditioner. However, as we further refine the mesh in subsequent time steps, SAI-MC is the fastest in time.

5. Conclusions and Future Work.

We have introduced a class of multilevel preconditioners that can be efficiently updated for rapid changes in AMR meshes. The cost in CPU time of updating the preconditioner is only a few percent of the (fastest) preconditioned linear solver time. The preconditioner exploits the hierarchical nature of AMR meshes to represent smooth global components of an approximate inverse efficiently on coarse meshes. We have shown experimentally that the proposed multilevel preconditioners yield level-independent convergence rates for a number of challenging problems. In future work, we will extend this to efficient preconditioners for structural optimization problems involving the equations of elasticity in three dimensions [56, 57, 58].

6. Acknowledgements. The PARAMESH software used in this work was developed at the NASA Goddard Space Flight Center and Drexel University under NASA's HPCC and ESTO/CT projects and under grant NNG04GP79G from the NASA/AISR project.

REFERENCES

[1] W. Bangerth and A. Joshi, *Adaptive finite element methods for the solution of inverse problems in optical tomography*, Inverse Problems, 24 (2008), pp. 1–22.

[2] M. W. Benson, *Iterative solution of large scale linear systems*, master's thesis, Lakehead University, Thunder Bay, Ontario, 1973.

[3] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Mathematica, 22 (1982), pp. 127–140.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, Journal of Computational Physics, 182 (2002), pp. 418–477. doi:10.1006/jcph.2002.7176.

[5] M. Benzi, J. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1318–1332.

[6] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM Journal on Scientific Computing, 17 (1996), pp. 1135–1149.

[7] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM Journal on Scientific Computing, 19 (1998), pp. 968–994.

[8] M. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, Journal of Computational Physics, 82 (1989), pp. 64–84.

[9] M. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, Journal of Computational Physics, 53 (1984), pp. 484–512.

[10] P. E. Bjørstad, W. Gropp, and B. F. Smith, *Domain Decomposition: Parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, Cambridge, UK, 1996.

[11] M. Bollhöfer and V. Mehrmann, *A new approach to algebraic multilevel methods based on sparse approximate inverses*, tech. report, Preprint SFB393/99-22, Department of Mathematics, TU Chemnitz, Germany, 1999.

[12] ———, *Algebraic multilevel methods and sparse approximate inverses*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 191–218.

[13] R. Bracewell, *The Fourier Transform and Its Applications*, McGraw-Hill, New York, 3rd edition ed., 1999.

[14] R. Bridson and W.-P. Tang, *Multiresolution approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 23 (2002), pp. 463–479.

[15] O. Bröker and M. Grote, *Sparse approximate inverse smoothers for geometric and algebraic multigrid*, Applied Numerical Mathematics, 41 (2002), pp. 61–80.

[16] O. Bröker, M. Grote, C. Mayer, and A. Reusken, *Robust parallel smoothing for multigrid via sparse approximate inverses*, sisc, 23 (2001), pp. 1395–1416.

[17] T. F. Chan and K. Chen, *Two-stage preconditioners using wavelet band splitting and sparse approximation*, Tech. Report CAM 00-26, Department of Mathematics, UCLA, 2000.

[18] T. F. Chan, W.-P. Tang, and W.-L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT Numerical Mathematics, 37 (1997), pp. 644–660.

[19] Z. Cheng, *Krylov solvers for error smoothing for strongly anisotropic problems on structured AMR meshes*, master's thesis, University of Illinois at Urbana-Champaign, 2005.

[20] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1804–1822.

[21] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1657–1675.

[22] ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[23] J. Cosgrove, J. C. Díaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems*, International Journal of Computer Mathematics, 44 (1992), pp. 91–110.

[24] E. de Sturler, *Incomplete Block LU preconditioners on slightly overlapping subdomains for a massively parallel computer*, Applied Numerical Mathematics (IMACS), 19 (1995), pp. 129–146.

[25] C. Farhat, K. Pierson, and M. Lesoinne, *The second generation of FETI methods and their application to the parallel solution of large-scale linear and geometrically nonlinear structural analysis problems*, Computer Methods in Applied Mechanics and Engineering, 184 (2000), pp. 333–374.

[26] C. Farhat and F.-X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, International Journal for Numerical Methods in Engineering, 32 (1991), pp. 1205–1227.

[27] ———, *Implicit parallel processing in structural mechanics*, Computational mechanics advances, 2 (1994), pp. 1–124.

[28] P. O. Frederickson, *Fast approximate inversion of large sparse linear systems*, Tech. Report Math. Report 7, Lakehead University, Thunder Bay, Ontario, Thunder Bay, Ontario, 1975.

[29] M. J. Gander, *Optimized Schwarz methods*, SIAM Journal on Numerical Analysis, 44 (2006), pp. 699–731.

[30] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[31] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.

[32] R. D. Hornung, A. M. Wissink, and S. R. Kohn, *Managing complex data and geometry in parallel structured AMR applications*, Engineering with Computers, 22 (2006), pp. 181–195.

[33] C. Japhet, F. Nataf, and F. Rogier, *The optimized order 2 method. Application to convection-diffusion problems*, Future Generation Computer Systems, 18 (2001), pp. 17–30.

[34] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, `libMesh`*: a C++ library for parallel adaptive mesh refinement/coarsening simulations*, Engineering with Computers, 22 (2006), pp. 237–254.

[35] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 45–58.

[36] R. J. LeVeque, *Clawpack web page*. http://www.amath.washington.edu/ claw/.

[37] ———, *CLAWPACK Version 4.3, User's Guide*, University of Washington, Department of Applied Mathematics, Box 352420, Seattle, Washington 98195-2420, 4.3 ed., 2006.

[38] P. MacNeice and K. M. Olson, *PARAMESH V4.1 User's Manual*.

[39] P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, and C. Packer, *PARAMESH: A parallel adaptive mesh refinement community toolkit*, Computer Physics Communications, 126 (2000), pp. 330–354.

[40] J. Meijerink and H. A. van der Vorst, *An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[41] J. A. Meijerink and H. A. van der Vorst, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comput. Phys., 44 (1981), pp. 134–155.

[42] G. A. Meurant, *A multilevel AINV preconditioner*, Numerical Algorithms, 29 (2002), pp. 107–129.

[43] F. Nataf, F. Rogier, and E. de Sturler, *Optimal interface conditions for domain decomposition methods*, Tech. Report CMAP-301, Centre de Mathematiques Appliquées, CNRS URA-756, Ecole Polytechnique, 1994.

[44] ———, *Domain decomposition methods for fluid dynamics*, in Navier-Stokes Equations and Related Nonlinear Problems, A. Sequeira, ed., New York, 1995, Plenum Press, pp. 367–376.

[45] K. M. Olson, *PARAMESH: A parallel adaptive grid tool*, in Parallel Computational Fluid Dynamics 2005: Theory and Applications: Proceedings of the Parallel CFD Conference, College Park, MD, U.S.A., A. Deane, A. Ecer, G. Brenner, D. Emerson, J. McDonough,

J. Periaux, N. Satofuka, and D. Tromeur-Dervout, eds., Elsevier, 2006.

[46] K. M. Olson and P. MacNeice, *Adaptive Mesh Refinement-Theory and Applications*, vol. 41 of Lecture Notes in Computational Science and Engineering, Springer, Berlin Heidelberg, 2005, ch. An Overview of the PARAMESH AMR Software package and Some of Its Applications, pp. 315–330.

[47] C. W. Oosterlee and T. Washio, *An evaluation of parallel multigrid as a solver and as a preconditioner for singularly perturbed problems*, SIAM Journal on Scientific Computing, 19 (1998), pp. 87–110.

[48] ———, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flow*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1670–1690.

[49] Y. Saad, *ILUT: a dual threshold incomplete ILU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.

[50] W.-P. Tang, *Generalized Schwarz splittings*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 573–595.

[51] W.-P. Tang and W.-L. Wan, *Sparse approximate inverse smoother for multigrid*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1236–1252.

[52] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.

[53] H. A. van der Vorst, *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.

[54] ———, *Iterative Krylov Methods for Large Linear systems*, Cambridge University Press, Cambridge, April 2003.

[55] K. Wang, J. Zhang, and C. Shen, *Parallel multilevel sparse approximate inverse preconditioners in large sparse matrix computations*, Supercomputing '03, Phoenix, Arizona, (2003).

[56] S. Wang, *Krylov subspace methods for topology optimization on adaptive meshes*, PhD thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, September 2007. Advisor: Eric de Sturler, Co-Advisor: Glaucio H. Paulino.

[57] S. Wang, E. de Sturler, and G. H. Paulino, *Large-scale topology optimization using preconditioned Krylov subspace methods with recycling*, Internat. J. Numer. Methods Engrg., 69 (2007), pp. 2441–2468.

[58] S. Wang, E. de Sturler, and G. H. Paulino, *Dynamic adaptive mesh refinement for topology optimization*, Internat. J. Numer. Methods Engrg., (2008). Draft NME-Mar-08-0166, submitted.