# AN ODYSSEY INTO LOCAL REFINEMENT AND MULTILEVEL PRECONDITIONING III: IMPLEMENTATION AND NUMERICAL EXPERIMENTS *

BURAK AKSOYLU†, STEPHEN BOND‡, AND MICHAEL HOLST§

**Abstract.** In this paper, we examine a number of additive and multiplicative multilevel iterative methods and preconditioners in the setting of two-dimensional local mesh refinement. While standard multilevel methods are effective for uniform refinement-based discretizations of elliptic equations, they tend to be less effective for algebraic systems which arise from discretizations on locally refined meshes, losing their optimal behavior in both storage and computational complexity. Our primary focus here is on BPX-style additive and multiplicative multilevel preconditioners, and on various stabilizations of the additive and multiplicative hierarchical basis method (HB), and their use in the local mesh refinement setting. In the first two papers of this trilogy, it was shown that both BPX and wavelet stabilizations of HB have uniformly bounded condition numbers on several classes of locally refined 2D and 3D meshes based on fairly standard (and easily implementable) red and red-green mesh refinement algorithms. In this third article of the trilogy, we describe in detail the implementation of these types of algorithms, including detailed discussions of the datastructures and traversal algorithms we employ for obtaining optimal storage and computational complexity in our implementations. We show how each of the algorithms can be implemented using standard datatypes available in languages such as C and FORTRAN, so that the resulting algorithms have optimal (linear) storage requirements, thereby the resulting multilevel method or preconditioner can be applied with optimal (linear) computational costs. We have successfully used these datastructure ideas for both MATLAB and C implementations using the FEtk, an open source finite element software package. We finish the paper with a sequence of numerical experiments illustrating the effectiveness of a number of BPX and stabilized HB variants for several examples requiring local refinement.

**Key words.** finite element methods, local mesh refinement, multilevel preconditioning, BPX, red and red-green refinement, 2D and 3D, datastructures.

**AMS subject classifications.** 65M55, 65N55, 65N22, 65F10

## 1. Introduction.

While there are a number of effective (often optimal) multilevel methods for uniform refinement-based discretizations of elliptic equations, only a handful of these methods are effective for algebraic systems which arise from discretizations on locally refined meshes, and these remaining methods are typically suboptimal in both storage and computational complexity. In this paper, we examine a number of additive and multiplicative multilevel iterative methods and preconditioners, specifically for two-dimensional local mesh refinement scenarios. Our primary focus is on Bramble, Pasciak, and Xu (BPX)-style additive and multiplicative multilevel preconditioners, and on stabilizations of the additive and multiplicative hierarchical basis method (HB). In [1, 2, 3], it was shown that both BPX and wavelet

*The first author was supported in part by the Burroughs Wellcome Fund through the LJIS predoctoral training program at UC San Diego, in part by NSF (ACI-9721349, DMS-9872890), and in part by DOE (W-7405-ENG-48/B341492). Other support was provided by Intel, Microsoft, Alias|Wavefront, Pixar, and the Packard Foundation. The second author was supported in part by the Howard Hughes Medical Institute, and in part by NSF and NIH grants to J. A. McCammon. The third author was supported in part by NSF CAREER Award 9875856 and in part by a UCSD Hellman Fellowship.

† Department of Computer Science, California Institute of Technology, Pasadena, CA 91125, USA (baksoylu@cs.caltech.edu).

‡ Department of Chemistry and Biochemistry and Department of Mathematics, University of California at San Diego, La Jolla, CA 92093, USA (bond@ucsd.edu).

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

stabilizations of HB have uniformly bounded condition numbers on several classes of locally refined 2D and 3D meshes based on fairly standard (and easily implementable) red and red-green mesh refinement algorithms. In this article, we describe in detail the implementation of these types of algorithms, including detailed discussions of the datastructures and traversal algorithms we employ for obtaining optimal storage and computational complexity in our implementations. We show how each of the algorithms can be implemented using standard datatypes available in languages such as C and FORTRAN, so that the resulting algorithms have optimal (linear) storage requirements, and so that the resulting multilevel method or preconditioner can be applied with optimal (linear) computational costs. We have successfully utilized these datastructure ideas for both MATLAB and C implementations using the FEtk, an open source finite element software package. We also present a sequence of numerical experiments illustrating the effectiveness of a number of BPX and stabilized HB variants for examples requiring local refinement.

The problem class of interest for our purposes here is linear second order partial differential equations (PDE) of the form:

$$-\nabla \cdot (p \ \nabla u) + q \ u = f, \quad u \in H_0^1(\Omega). \tag{1.1}$$

Here, $f \in L_2(\Omega)$, $p$, $q \in L_\infty(\Omega)$ and $p : \Omega \to L(\mathbb{R}^d, \mathbb{R}^d)$, $q : \Omega \to \mathbb{R}$, where $p$ is a symmetric positive definite matrix, and $q$ is nonnegative. Let $\mathcal{T}_0$ be a shape regular and quasiuniform initial partition of $\Omega$ into a finite number of $d$-simplices, and generate $\mathcal{T}_1, \mathcal{T}_2, \ldots$ by refining the initial partition using either red-green or red local refinement strategies in $d = 2$ or $d = 3$ spatial dimensions. Let $\mathcal{S}_j$ be the simplicial linear $C^0$ finite element (FE) space corresponding to $\mathcal{T}_j$ equipped with zero boundary values. The set of nodal basis functions for $\mathcal{S}_j$ is denoted by $\{\phi_i^{(j)}\}_{i=1}^{N_j}$ where $N_j = \dim \mathcal{S}_j$ is equal to the number of interior nodes in $\mathcal{T}_j$. Successively refined FE spaces will form the following nested sequence:

$$\mathcal{S}_0 \subset \mathcal{S}_1 \subset \ldots \subset \mathcal{S}_j \subset \ldots \subset H_0^1(\Omega). \tag{1.2}$$

Although the mesh is nonconforming in the case of red refinement, $\mathcal{S}_j$ is used within the framework of conforming FE methods for discretizing (1.1).

Let the bilinear form and the linear functional representing the weak formulation of (1.1) be denoted as

$$a(u, v) = \int_\Omega p \ \nabla u \cdot \nabla v + q \ u \ v \ dx, \quad b(v) = \int_\Omega f \ v \ dx, \quad u, v \in H_0^1(\Omega),$$

and let us consider the following Galerkin formulation: Find $u \in \mathcal{S}_j$, such that

$$a(u, v) = b(v), \quad \forall v \in \mathcal{S}_j. \tag{1.3}$$

Employing the expansion $u = \sum_{i=1}^n u_i^{(j)} \phi_i^{(j)}$ in the nodal basis for $\mathcal{S}_j$, problem (1.3) reduces to an algebraic equation of the form:

$$A^{(j)} \mathbf{u}^{(j)} = \mathbf{b}^{(j)} \in \mathbb{R}^{N_j} \tag{1.4}$$

for the combination coefficients $\mathbf{u}^{(j)} \in \mathbb{R}^{N_j}$. The nodal discretization matrix and vector arise then as:

$$A_{rs}^{(j)} = a(\phi_s^{(j)}, \phi_r^{(j)}), \qquad \mathbf{b}_r^{(j)} = b(\phi_r^{(j)}), \quad 1 \le r, s \le N_j.$$

Solving the discretized form of (1.3), namely (1.4), by iterative methods, has been the subject of intensive research because of the enormous practical impact on a number of application areas in computational science. For quality approximation in physical simulation, one is required to use meshes containing very large numbers of simplices leading to approximation spaces $\mathcal{S}_j$ with very large dimension $N_j$. Only iterative methods which scale well with $N_j$ can be used effectively, which usually leads to the use of multilevel-type iterative methods and preconditioners. Even with the use of such optimal methods for (1.4), which means methods which scale linearly with $N_j$ in both memory and computational complexity, the approximation quality requirements on $\mathcal{S}_j$ often force $N_j$ to be so large that only parallel computing techniques can be used to solve (1.4).

To overcome this difficulty one employs adaptive methods, which involves the use of *a posteriori* error estimation to drive *local mesh refinement* algorithms. This approach leads to approximation spaces $\mathcal{S}_j$ which are adapted to the particular target function $u$ of interest, and as a result can achieve a desired approximation quality with much smaller approximation space dimension $N_j$ than non-adaptive methods. One still must solve the algebraic system (1.4), but unfortunately most of the available multilevel methods and preconditioners are no longer optimal, in either memory or computational complexity. This is due to the fact that in the local refinement setting, the approximation spaces $S_j$ do not increase in dimension geometrically as they do in the uniform refinement setting. As a result, a single multilevel V-cycle no longer has linear complexity, and the same difficulty is encountered by other multilevel methods. Moreover, storage of the discretization matrices and vectors for each approximation space, required for assembling V-cycle and similar iterations, no longer has linear memory complexity.

A partial solution to the problem with multilevel methods in the local refinement setting is provided by the HB method [4, 5, 21]. This method is based on a direct or hierarchical decomposition of the approximation spaces $\mathcal{S}_j$ rather than the overlapping decomposition employed by the multigrid and BPX method, and therefore by construction has linear memory complexity as well as linear computational complexity for a single V-cycle-like iteration. Unfortunately, the HB condition number is not uniformly bounded, leading to worse than linear overall computational complexity. While the condition number growth is slow (logarithmic) in two dimensions, it is quite rapid (geometric) in three dimensions, making it ineffective in the 3D local refinement setting. Recent alternatives to the HB method, including both BPX-like methods [7, 8] and wavelet-like stabilizations of the HB methods [19], provide a final solution to the condition number growth problem. It was shown in [9] that the BPX preconditioner has uniformly bounded condition number for certain classes of locally refined meshes in two dimensions, and more recently in [2] it was shown that the condition number remains uniformly bounded for certain classes of locally refined meshes in three spatial dimensions. In [3], it was also shown that wavelet-stabilizations of the HB method give rise to uniformly bounded conditions numbers for certain classes of local mesh refinement in both the two- and three-dimensional settings.

In view of [2] and [3], our interest in this paper is to examine the practical implementation aspects of both BPX and stabilized HB iterative methods and preconditioners. The remainder of the paper is structured as follows. In §2, we review the algorithms presented in [2] and [3], giving a unified algorithm framework on which implementations can be based. The core of the paper is in some sense §3, which describes in detail the datastructures and key algorithms employed in the implemen-

tation of the algorithms. The focus is on practical realization of optimal (linear) complexity of the implementations, in both memory and operation complexity. The FEtk software package which is leveraged for our implementations is described briefly in §3.3. A sequence of numerical experiments with the implementations is presented in §4, illustrating the condition number growth properties of BPX and stabilized HB methods. Finally, we draw some conclusions in §5.

**2. Overview of the multilevel methods.** In the first article [2] of the trilogy, it was shown that the BPX preconditioner was optimal on the meshes under the local 2D and 3D red-green, as well as local 2D and 3D red, refinement procedures. The classical BPX preconditioner [8, 20] can be written as an action of the operator $X$ as follows:

$$Xu = \sum_{j=0}^{J} 2^{j(d-2)} \sum_{i=1}^{N_j} (u, \phi_i^{(j)}) \phi_i^{(j)}, \quad u \in \mathcal{S}_J. \tag{2.1}$$

Let the prolongation operator from level $j-1$ to $j$ be denoted by $P_{j-1}^{j}$, and also denote the prolongation operator from level $j$ to $J$ as:

$$P_j \equiv P_j^J = P_{J-1}^J \dots P_j^{j+1} \in \mathbb{R}^{N_J \times N_j},$$

where $P_J^J$ is defined to be the identity matrix $I \in \mathbb{R}^{N_J \times N_J}$. Then the matrix representation of (2.1) becomes [20]:

$$X = \sum_{j=0}^{J} 2^{j(d-2)} P_j P_j^t.$$

One can also introduce a version with a smoother $S_j$ (the smoother is a symmetric Gauss-Seidel iteration throughout the paper):

$$X = \sum_{j=0}^{J} 2^{j(d-2)} P_j S_j P_j^t.$$

The preconditioner (2.1) can be modified in the hierarchical sense;

$$X_{\text{HB}} u = \sum_{j=0}^{J} 2^{j(d-2)} \sum_{i=N_{j-1}+1}^{N_j} (u, \phi_i^{(j)}) \phi_i^{(j)}, \quad u \in \mathcal{S}_J. \tag{2.2}$$

The new preconditioner corresponds to the additive HB preconditioner in [21]. The matrix representation of (2.2) is formed from matrices $H_j$ which are simply the tails of the $P_j$ corresponding to newly introduced degrees of freedom (DOF) in the fine space. In other words, $H_j \in \mathbb{R}^{N_J \times (N_j - N_{j-1})}$ is given by only keeping the fine columns (the last $N_j - N_{j-1}$ columns of $P_j$). Hence, the matrix representation of (2.2) becomes:

$$X_{\text{HB}} = \sum_{j=0}^{J} 2^{j(d-2)} H_j H_j^t.$$

Only in the presence of a geometric increase in the number of DOF, the same assumption for optimality of a single classical multigrid or BPX iteration, does the cost

per iteration remain optimal. In the case of local refinement, the BPX preconditioner (2.1) (usually known as additive multigrid) can easily be suboptimal because of the suboptimal cost per iteration (see Figure 4.3). On the other hand, the HB preconditioner (2.2) suffers from a suboptimal iteration count. The above deficiencies of the preconditioners (2.1) and (2.2) can be overcome by restricting the sum over $i$ in (2.1) only to those nodal basis functions with supports that intersect the refinement region [6, 7, 9, 14]. We call this set *onering* of fine DOF, namely, the set which contains fine DOF and their immediate neighboring coarse DOF. The following is referred as the BPX preconditioner for local refinement.

$$Xu = \sum_{j=0}^{J} 2^{j(d-2)} \sum_{i \in \text{ONERING}^{(j)}} (u, \phi_i^{(j)}) \phi_i^{(j)}, \quad u \in \mathcal{S}_J, \tag{2.3}$$

where $\text{ONERING}^{(j)} = \{onering(ii) : \ ii = N_{j-1}+1, \ldots, N_j\}$.

The BPX decomposition gives rise to basis functions which are not locally supported, but they decay rapidly outside a local support region. This allows for locally supported *approximations* as illustrated in Figures 2.1, 2.2, and 2.3.

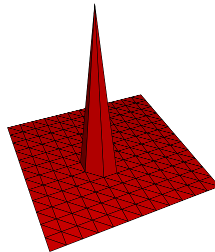

FIG. 2.1. *Hierarchical basis function without modification.*

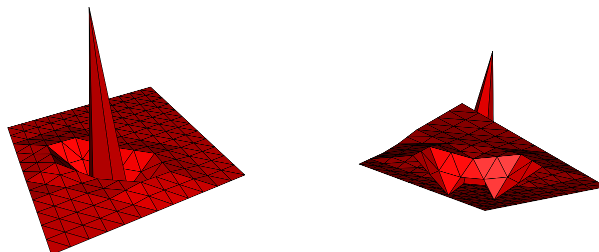

FIG. 2.2. *Wavelet modified hierarchical basis function with one iteration of symmetric Gauss-Seidel approximation, upper and lower view.*

The *wavelet modified hierarchical basis (WMHB) methods* [17, 18, 19] can be viewed as an approximation of the wavelet basis stemming from the BPX decomposition [13]. A similar wavelet-like multilevel decomposition approach was taken

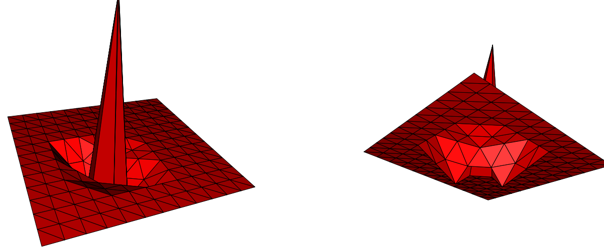

FIG. 2.3. *Wavelet modified hierarchical basis function with one iteration of Jacobi approximation, upper and lower view.*

in [16], where the orthogonal decomposition is formed by a discrete $L_2$-equivalent inner product. This approach utilizes the same BPX two-level decomposition [15, 16].

For adaptive regimes, the other primary method of interest is the WMHB method. The WMHB methods can be described as additive or multiplicative Schwarz methods. In one of the previous papers [3] of this trilogy, it was shown that the additive version of the WMHB method is optimal under certain types of red-green mesh refinement. Following the notational framework in [3, 19], this method is defined recursively as follows:

DEFINITION 2.1. *The additive WMHB method $D^{(j)}$ is defined for $j = 1, \ldots, J$ as*

$$D^{(j)} \equiv \begin{bmatrix} D^{(j-1)} & 0 \\ 0 & B_{22}^{(j)} \end{bmatrix},$$

*with $D^{(0)} = A^{(0)}$.*

With smooth PDE coefficients, optimal results were also established for the multiplicative version of the WMHB method in [3]. Our numerical experiments demonstrate such optimal results. This method can be written recursively as:

DEFINITION 2.2. *The multiplicative WMHB method $B^{(j)}$ is defined as*

$$B^{(j)} \equiv \begin{bmatrix} B^{(j-1)} & A_{12}^{(j)} \\ 0 & B_{22}^{(j)} \end{bmatrix} \begin{bmatrix} I & 0 \\ {B_{22}^{(j)}}^{-1} A_{21}^{(j)} & I \end{bmatrix} = \begin{bmatrix} B^{(j-1)} + A_{12}^{(j)} {B_{22}^{(j)}}^{-1} A_{21}^{(j)} & A_{12}^{(j)} \\ A_{21}^{(j)} & B_{22}^{(j)} \end{bmatrix},$$

*with $B^{(0)} = A^{(0)}$.*

$A_{12}^{(j)}, A_{21}^{(j)}, A_{22}^{(j)}$ represent subblocks of $A^{(j)}$ and they correspond to coarse-fine, fine-coarse, and fine-fine interactions of DOF at level $j$, respectively. $B_{22}^{(j)}$ denotes an approximation of $A_{22}^{(j)}$, e.g. Gauss-Seidel or Jacobi approximation. For a more complete description of these and related algorithms, see [2, 3].

**3. Implementation.** The overall utility of any finite element code depends strongly on efficient implementation of its core algorithms and data structures. Theoretical results involving complexity are of little practical importance if the methods cannot be implemented. For algorithms involving data structures, this usually means striking a balance between storage costs and computational complexity. Finding a minimal representation for a data set is only useful if the information can be accessed efficiently.

**3.1. Sparse Matrix Structures.** Our implementation relies on a total of four distinct sparse matrix data structures: compressed column (COL), compressed row (ROW), diagonal-row-column (DRC), and orthogonal-linked list (XLN). Each of these storage schemes attempts to record the location and value of the nonzeros using a minimal amount of information. The schemes differ in the exact representation which effects the speed and manner with which the data can be retrieved. To illustrate how each of these data structures works in practice, we consider storing the following sparse matrix:

$$\begin{bmatrix} 1 & 2 & & & \\ 3 & 4 & 5 & & 6 \\ & 7 & 8 & & \\ & & & 9 & 10 \\ & 11 & & 12 & 13 \end{bmatrix} \tag{3.1}$$

• COL: The compressed column format is the most commonly used sparse matrix type in the literature. It is the format chosen for the Harwell-Boeing matrix collection [11], and is used in production codes such as SuperLU [10]. In this data structure, the nonzeros are arranged by column in a single double-precision array:

$$\mathrm{A}_{\mathrm{COL}} = [1,\ 3,\ 2,\ 4,\ 7,\ 11,\ 5,\ 8,\ 9,\ 12,\ 6,\ 10,\ 13]\,.$$

The indices of A (often referred to as pointers) corresponding to the first entry in each column is then stored in an integer array:

$$\mathrm{IA}_{\mathrm{COL}} = [1,\ 3,\ 7,\ 9,\ 11,\ 14]\,.$$

The length of the array IA is always one greater than the number of columns, with the last entry is equal to the number of nonzeros plus one. The difference in successive entries in the IA array reflects the number of nonzeros in each column. If a column has no nonzeros, the index from the next column is repeated. To determine the location of each nonzero within its column, the row index of each entry is stored in an integer array:

$$\mathrm{JA}_{\mathrm{COL}} = [1,\ 2,\ 1,\ 2,\ 3,\ 5,\ 2,\ 3,\ 4,\ 5,\ 2,\ 4,\ 5]\,.$$

There is no restriction that the entries are ordered within each column, only that the columns are ordered. The memory required to store this datastructure is: $(nZ+nC+1) * \mathrm{size}\,(\mathrm{int}) + nZ * \mathrm{size}\,(\mathrm{double})$, where $nZ$ and $nC$ are the number of nonzeros and columns respectively.

• ROW: The compressed row data structure is just the transpose of the compressed column data structure, where the nonzero entries, row pointers, and column indices are stored in A, IA, and JA respectively:

$$\mathrm{A}_{\mathrm{ROW}} = [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13]\,,$$

$$\mathrm{IA}_{\mathrm{ROW}} = [1,\ 3,\ 7,\ 9,\ 11,\ 14]\,, \quad \mathrm{JA}_{\mathrm{ROW}} = [1,\ 2,\ 1,\ 2,\ 3,\ 5,\ 2,\ 3,\ 4,\ 5,\ 2,\ 4,\ 5]\,.$$

One should note that since in our example the matrix is *structurally symmetric*, the IA and JA arrays are identical in both the ROW and COL cases. The memory required to store this datastructure is: $(nZ + nR + 1) * \mathrm{size}\,(\mathrm{int}) + nZ * \mathrm{size}\,(\mathrm{double})$, where $nR$ is the number of rows.

• DRC: The diagonal-row-column format is a structurally symmetric data structure, which is only valid for square matrices. In this format, the diagonal is stored in its own full vector, while the strictly upper and lower triangular portions are stored in ROW and COL formats respectively. Leveraging the symmetry in the nonzero structure, the same IA and JA arrays can be used for the upper and lower triangular parts:

$$\mathrm{AD}_{\mathrm{DRC}} = [1,\, 4,\, 8,\, 9,\, 13]\,,$$

$$\mathrm{AU}_{\mathrm{DRC}} = [2,\, 5,\, 6,\, 10]\,, \quad \mathrm{AL}_{\mathrm{DRC}} = [3,\, 7,\, 11,\, 12]$$

$$\mathrm{IA}_{\mathrm{DRC}} = [1,\, 2,\, 4,\, 4,\, 5,\, 5]\,, \quad \mathrm{JA}_{\mathrm{DRC}} = [2,\, 3,\, 5,\, 5]\,.$$

The memory required to store this datastructure is less than ROW or COL if the diagonal is full, and the matrix is structurally symmetric.

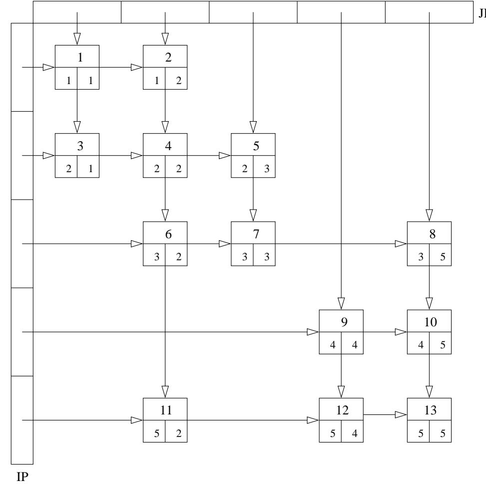


FIG. 3.1. *An illustration of the XLN datastructure.*

• XLN: The orthogonal-linked list format is the only dynamically "fillable" datastructure used by our methods. By using variable length linked lists, rather than a fixed length array, it is suitable for situations where the total number of nonzeros is not known *a priori*. The XLN datastructure is illustrated graphically in Figure 3.1. For each nonzero, there is a link containing the value, row index, column index, and pointers to the next in the row and column. To keep track of the first link in each row and column, there are two additional pointer arrays, IP and JP. As long as there are "order-one" nonzeros per row, accessing any entry can be accomplished in "order-one" time. The structure can be traversed both rowwise, and columnwise depending on the situation. If the matrix is symmetric, only the lower triangular portion is stored. The total storage overhead for this structure is: $nZ * (\mathrm{size}\,(\mathrm{double}) + 2 * \mathrm{size}\,(\mathrm{int})) + (nC + nR + 2\, nZ) * \mathrm{size}\,(\mathrm{ptr})$. Although this is considerably more than the other three datastructures, one should note that the asymptotic complexity is still linear in the number of nonzeros.

**3.2. Sparse Matrix Products.** The key preprocessing step in the hierarchical basis methods, is converting the "nodal" matrices and vectors into the hierarchical basis. This operation involves sparse matrix-vector and matrix-matrix products for each level of refinement. To ensure that this entire operation has linear cost, with respect to the number of unknowns, the per-level change of basis operations must have a cost of $\mathcal{O}(n_j)$, where $n_j := N_j - N_{j-1}$ is the number of "new" nodes on level $j$. For the traditional multigrid algorithm this is not possible, since enforcing the variational conditions operates on *all* the nodes on each level, not just the newly introduced nodes.

The linear operator which converts from the nodal to the hierarchical basis can be written in terms of a change of basis matrix:

$$G = \begin{bmatrix} I & K_{12} \\ K_{21} & I + K_{22} \end{bmatrix},$$

where $G \in \mathbb{R}^{N_j \times N_j}$, $K_{12} \in \mathbb{R}^{N_{j-1} \times n_j}$, $K_{21} \in \mathbb{R}^{n_j \times N_{j-1}}$, and $K_{22} \in \mathbb{R}^{n_j \times n_j}$. In this representation, we have assumed that the nodes are ordered with the nodes $N_{j-1}$ inherited from the previous level listed first, and the $n_j$ new DOF listed second. For both wavelet modified (WMHB) and unmodified hierarchical basis (HB), the $K_{21}$ block represents the last $n_j$ rows of the prolongation matrix, $P_{j-1}^{j}$. In the HB case, the $K_{12}$ and $K_{22}$ blocks are zero resulting in a very simple form:

$$G_{\mathrm{hb}} = \begin{bmatrix} I & 0 \\ K_{21} & I \end{bmatrix} \tag{3.2}$$

For WMHB, the $K_{12}$ and $K_{22}$ blocks are computed using the mass matrix, which results in the following formula:

$$G_{\mathrm{wmhb}} = \begin{bmatrix} I & -\mathrm{inv}\left[M_{11}^{\mathrm{hb}}\right] M_{12}^{\mathrm{hb}} \\ K_{21} & I - K_{21}\mathrm{inv}\left[M_{11}^{\mathrm{hb}}\right] M_{12}^{\mathrm{hb}} \end{bmatrix}, \tag{3.3}$$

where the $\mathrm{inv}[\cdot]$ is some approximation to the inverse which preserves the complexity. For example, it could be as simple as the inverse of the diagonal, or a low-order matrix polynomial approximation. The $M^{\mathrm{hb}}$ blocks are taken from the mass matrix in the HB basis:

$$M^{\mathrm{hb}} = G_{\mathrm{hb}}^T M^{\mathrm{nodal}} G_{\mathrm{hb}}. \tag{3.4}$$

For the remainder of this section, we restrict our attention to the WMHB case. The HB case follows trivially with the two additional subblocks of $K$ set to zero.

To reformulate the nodal matrix representation of the bilinear form in terms of the hierarchical basis, we must perform a triple matrix product of the form:

$$\begin{aligned} A_{(j)}^{\mathrm{wmhb}} &= G_{(j)}^T A_{(j)}^{\mathrm{nodal}} G_{(j)} \\ &= \left(I + K_{(j)}^T\right) A_{(j)}^{\mathrm{nodal}} \left(I + K_{(j)}\right). \end{aligned}$$

In order to keep linear complexity, we can only copy $A^{\mathrm{nodal}}$ a fixed number of times, i.e. it cannot be copied on every level. Fixed size data-structures are unsuitable for storing the product, since predicting the nonzero structure of $A_{(j)}^{\mathrm{wmhb}}$ is just as difficult as actually computing it. It is for these reasons that we have chosen the following strategy: First, copy $A^{\mathrm{nodal}}$ on the finest level, storing the result in an XLN which

will eventually become $A^{\text{wmhb}}$. Second, form the product pairwise, contributing the result to the XLN. Third, the last $n_j$ columns and rows of $A^{\text{wmhb}}$ are stripped off, stored in fixed size blocks, and the operation is repeated on the next level, using the $A_{11}$ block as the new $A^{\text{nodal}}$:

ALGORITHM 3.1. *(Wavelet Modified Hierarchical Change of Basis)*

- *Copy* $A_J^{\text{nodal}} \to A^{\text{wmhb}}$ *in XLN format.*
- *While* $j > 0$
  1. *Multiply* $A^{\text{wmhb}} = A^{\text{wmhb}} G$ *as*

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} + = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} 0 & K_{12} \\ K_{21} & K_{22} \end{bmatrix}$$

  2. *Multiply* $A^{\text{wmhb}} = G^T A^{\text{wmhb}}$ *as*

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} + = \begin{bmatrix} 0 & K_{21}^T \\ K_{12}^T & K_{22}^T \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

  3. *Remove* $A_{21}^{(j)}$, $A_{12}^{(j)}$, $A_{22}^{(j)}$ *blocks of* $A^{\text{wmhb}}$ *storing in ROW, COL, and DRC formats respectively.*
  4. *After the removal, all that remains of* $A^{\text{wmhb}}$ *is its* $A_{11}^{(j)}$ *block.*
  5. *Let j = j - 1, descending to level* $j - 1$.
- *End While.*
- *Store the last* $A^{\text{wmhb}}$ *as* $A_{\text{coarse}}$

We should note that in order to preserve the complexity of the overall algorithm, all of the matrix-matrix algorithms must be carefully implemented. For example, the change of basis involves computing the products of $A_{11}$ with $K_{12}$ and $K_{12}^T$. To preserve storage complexity, $K_{12}$ must be kept in compressed column format, COL. For the actual product, the loop over the columns of $K_{12}$ must be ordered first, then a loop over the nonzeros in each column, then a loop over the corresponding row or column in $A_{11}$. It is exactly for this reason, that one must be able to traverse $A_{11}$ both by row and by column, which is why we have chosen an orthogonal-linked matrix structure for $A$ during the change of basis (and hence $A_{11}$).

To derive optimal complexity algorithms for the other products, it is enough to ensure that the outer loop is always over a dimension of size $n_j$. Due to the limited ways in which a sparse matrix can be traversed, the ordering of the remaining loops will usually be completely determined. Further gains can be obtained in the symmetric case, since only the upper or lower portion of the matrix needs to be explicitly computed and stored.

**3.3. The Finite Element ToolKit (FEtk).** A number of variations of the methods described above have been implemented using the Finite Element ToolKit (FEtk) [12]. FEtk is an open source finite element modeling package which has been developed by the Holst research group over several years at Caltech and UC San Diego, with generous contributions from a number of colleagues. FEtk consists of a low-level portability library called MALOC (Minimal Abstraction Layer for Object-oriented C), on top of which is built a general finite element modeling kernel called MC (Manifold Code). Most of the images appearing later in this paper were produced using another component of FEtk call SG (Socket Graphics), which is also built on top of MALOC. FEtk also includes a fully functional MATLAB version of MC called MCLite, which shares with MC its datastructures, *a posteriori* error estimation and mesh refinement algorithms, and iterative solution methods. All of the preconditioners employed in

this paper have been implemented by the authors as ANSI-C class library extensions to MC, and as MATLAB toolkit-like extensions to MCLite. The two implementations are mathematically equivalent, although the MCLite implementation is restricted to two spatial dimensions. (The MC-based implementation is both two- and three-dimensional.) The extensions to MC are distributed as the *MCX* library, and as MATLAB extensions to MCLite are distributed as *MCLiteX*.

MALOC, SG, MC, and MCLite are freely redistributable under the GNU General Public License (GPL). More information about FEtk can be found at:

`http://www.fetk.org`

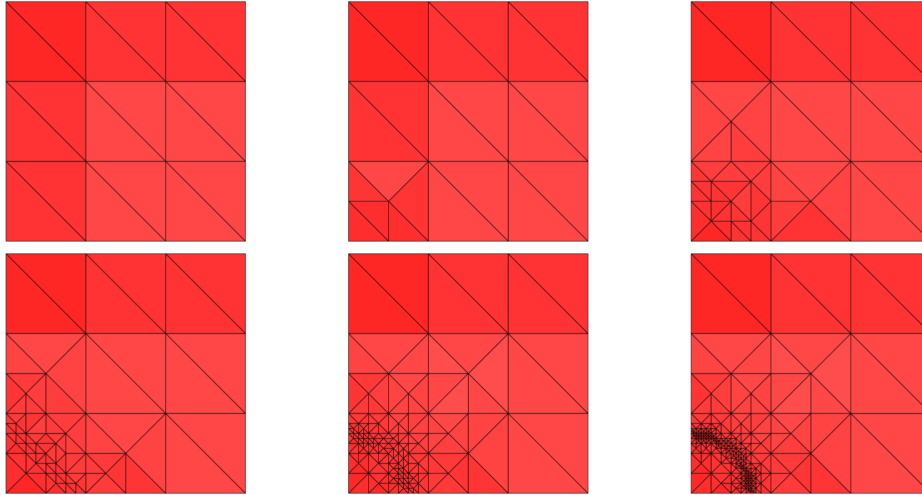

FIG. 4.1. *Adaptive mesh, experiment set I.*

**4. Numerical Experiments.** The test problem is as follows:

$$\begin{aligned} -\nabla \cdot (p\ \nabla u) + q\ u &= f, \quad x \in \Omega \subset \mathbb{R}^2, \\ n \cdot (p\ \nabla u) &= g, \quad \text{on } \Gamma_N, \\ u &= 0, \quad \text{on } \Gamma_D, \end{aligned}$$

where $\Omega = [0,1] \times [0,1]$ and

$$p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } q = 1.$$

The source term $f$ is constructed so that the true solution is $u = \sin \pi x \sin \pi y$. We present two experiment sets in which adaptivity is driven by a geometric criterion. Namely, the simplices which intersect with the quarter circle centered at the origin with radius 0.25 and 0.05, in experiment sets I and II respectively, are repeatedly marked for further refinement.

• Boundary conditions for the domain in experiment set I:

$$\begin{aligned} \Gamma_N &= \{(x,y) : x = 0, 0 < y < 1\} \cup \{(x,y) : x = 1, 0 < y < 1\} \\ \Gamma_D &= \{(x,y) : 0 \le x \le 1, y = 0\} \cup \{(x,y) : 0 \le x \le 1, y = 1\}. \end{aligned}$$

• Boundary conditions for the domain in experiment set II:

$$\Gamma_N = \{(x,y) : 0 \le x \le 1, y = 0\} \cup \{(x,y) : 0 \le x \le 1, y = 1\}$$
$$\cup\{(x,y) : x = 0, 0 \le y \le 1\} \cup \{(x,y) : x = 1, 0 \le y \le 1\}.$$

Stopping criterion: $\|\text{error}\|_A < 10^{-7}$.

In experiment set I, red-green refinement subdivides simplices intersecting an arc of radius 0.25 which gives rise to a rapid increase in the number of degrees of freedom (DOF). Although we have an adaptive refinement strategy, this indeed creates a geometric increase in the number of DOF, see Figure 4.1. Experiment set II is designed so that a small number of DOF is introduced at each level. In order to do this, green refinement subdivides simplices intersecting a smaller arc with radius 0.05.

Table 4.1
*MCLite iteration counts for various methods, red-green refinement driven by geometric refinement, experiment set I.*

| Levels | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| MG | 1 | 4 | 7 | 7 | 7 | 6 | 6 | 6 |
| M.BPX | 1 | 4 | 7 | 7 | 7 | 7 | 6 | 6 |
| HBMG | 1 | 10 | 19 | 28 | 32 | 37 | 45 | 56 |
| WMHBMG | 1 | 6 | 12 | 13 | 16 | 17 | 17 | 17 |
| PCG-MG | 1 | 3 | 4 | 5 | 5 | 5 | 5 | 5 |
| PCG-M.BPX | 1 | 3 | 5 | 5 | 5 | 5 | 5 | 5 |
| PCG-HBMG | 1 | 3 | 7 | 10 | 12 | 14 | 15 | 16 |
| PCG-WMHBMG | 1 | 3 | 7 | 7 | 9 | 9 | 9 | 9 |
| PCG-A.MG | 1 | 8 | 13 | 17 | 20 | 21 | 23 | 24 |
| PCG-BPX | 1 | 6 | 12 | 14 | 17 | 17 | 18 | 18 |
| PCG-HB | 1 | 5 | 14 | 21 | 26 | 32 | 38 | 41 |
| PCG-WMHB | 1 | 5 | 12 | 15 | 19 | 20 | 21 | 21 |
| Nodes | 16 | 19 | 31 | 55 | 117 | 219 | 429 | 835 |
| DOF | 8 | 10 | 21 | 43 | 102 | 202 | 410 | 814 |

In all the experiments, we utilize a direct coarsest level solve and the smoother is a symmetric Gauss-Seidel iteration. Set of DOF on which the smoother acts is the fundamental difference between the methods. Classical multigrid methods smooth on all DOF, whereas HB-like methods smooth only on fine DOF. WMHB style methods smooth as HB methods do, but in a different basis. BPX methods smooth on the onering of the fine DOF, which is more than HB methods but less than classical multigrid.

There are four multiplicative methods under consideration: MG, M.BPX, HBMG, and WMHBMG. The following is a guide to the tables and figures below. MG will refer to classical multigrid, in particular corresponds to the standard V-cycle implementation. HBMG corresponds exactly to the MG algorithm, but where pre- and post-smoothing are restricted to fine DOF. M.BPX refers to multiplicative version of BPX with the smoother is restricted to fine DOF and their immediate coarse neighbors which are often called as the *onering* neighbors. The onering neighbors of the fine nodes can be directly determined by the sparsity pattern of the fine-fine subblock $A_{22}$ of the stiffness matrix. The set of DOF over which the BPX method smooths is simply the union of the column locations of nonzero entries corresponding to fine

Table 4.2
*MCLite iteration counts for various methods, green refinement driven by geometric refinement, experiment set II.*

| Levels | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| MG | 1 | 3 | 4 | 3 | 4 | 4 | 3 |
| | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| M.BPX | 1 | 4 | 4 | 4 | 4 | 4 | 4 |
| | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| HBMG | 1 | 13 | 14 | 16 | 22 | 25 | 26 |
| | 30 | 32 | 32 | 36 | 38 | 42 | 44 |
| WMHBMG | 1 | 8 | 11 | 11 | 12 | 12 | 12 |
| | 13 | 15 | 15 | 15 | 15 | 15 | 15 |
| PCG-MG | 1 | 2 | 3 | 3 | 3 | 4 | 3 |
| | 3 | 4 | 3 | 4 | 3 | 3 | 3 |
| PCG-M.BPX | 1 | 2 | 3 | 4 | 4 | 3 | 4 |
| | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| PCG-HBMG | 1 | 2 | 5 | 7 | 8 | 9 | 10 |
| | 10 | 11 | 12 | 11 | 12 | 13 | 13 |
| PCG-WMHBMG | 1 | 2 | 5 | 6 | 6 | 7 | 7 |
| | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| PCG-A.MG | 1 | 10 | 13 | 15 | 18 | 20 | 21 |
| | 23 | 25 | 26 | 28 | 28 | 28 | 29 |
| PCG-BPX | 1 | 6 | 10 | 11 | 13 | 14 | 15 |
| | 16 | 18 | 19 | 19 | 20 | 20 | 21 |
| PCG-HB | 1 | 3 | 9 | 11 | 14 | 18 | 20 |
| | 22 | 24 | 27 | 30 | 32 | 34 | 36 |
| PCG-WMHB | 1 | 3 | 9 | 12 | 14 | 16 | 17 |
| | 19 | 20 | 20 | 22 | 23 | 23 | 23 |
| Nodes=DOF | 289 | 290 | 296 | 299 | 309 | 319 | 331 |
| | 349 | 388 | 423 | 489 | 567 | 679 | 837 |

DOF. Using this observation, HBMG smoother can easily be modified to be a BPX smoother. WMHBMG is similar to HBMG, in that both are multiplicative methods in the sense of Definition 2.2, but the difference is in the basis used. In particular, the change of basis matrices are different as a result of the wavelet stabilization, where the $L_2$-projection to coarser finite element spaces is approximated by two Jacobi iterations.

PCG stands for the preconditioned conjugate gradient method. PCG-A.MG, PCG-BPX, PCG-HB, and PCG-WMHB involve the use of additive MG, PBX, HB, and WMHB as preconditioners for CG, respectively. In the sense of Definition 2.1, HB and WMHB are additive versions of HBMG and WMHBMG respectively. Each preconditioner is implemented in a manner similar to that described in [17, 19].

Finally, note that *Nodes* denotes the total number of nodes in the simplicial mesh, including Dirichlet and Neumann nodes. The iterative methods view DOF as the union of the unknowns corresponding to interior and Neumann/Robin boundary DOF, and these are denoted as such.

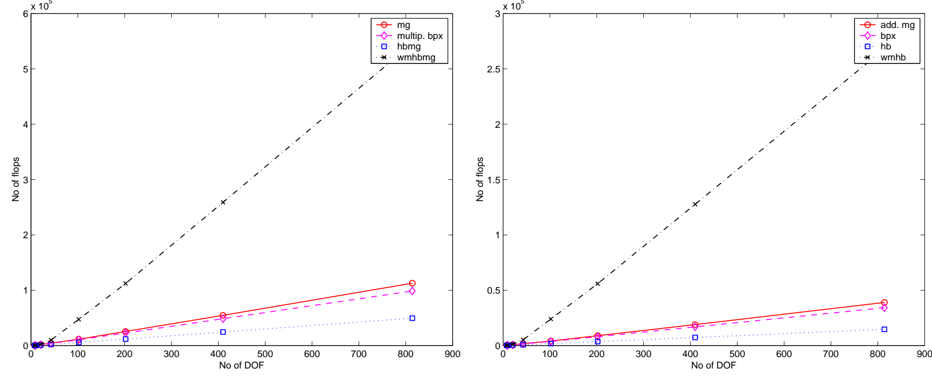


FIG. 4.2. *Flop counts for single iteration of multiplicative (left) and additive (right) methods, experiment set I.*

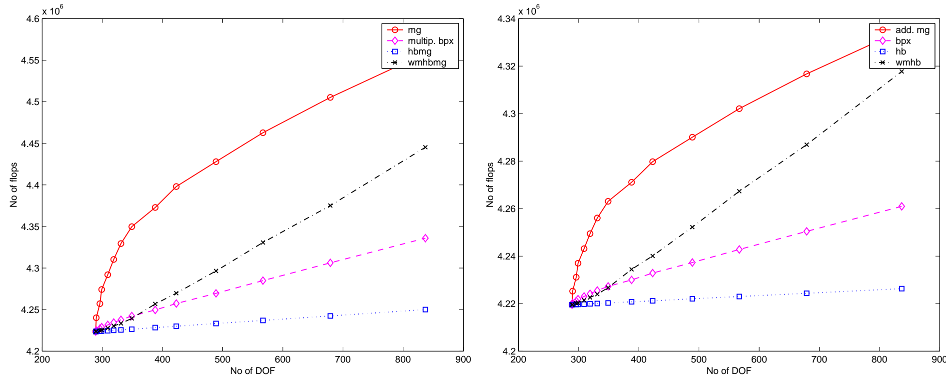


FIG. 4.3. *Flop counts for single iteration of multiplicative (left) and additive (right) methods, experiment set II.*

The refinement procedure utilized in the experiments is fundamentally the same as the 2D red-green described in [2, 3]. We, however, remove the restrictive conditions that the simplices for level $j+1$ have to be created from the simplices at level $j$ and the bisected (green refined) simplices cannot be further refined. Even in this case the claimed results seem to hold. Experiments are done in the MCLite module of the FEtk package. Several key routines from this implementation, used to produce most of the numerical results in this paper, are given in the appendix.

Iteration counts are reported in Tables 4.1 and 4.2. The optimality of M.BPX, BPX, WMHBMG and WMHB is evidenced in each of the experiments. We observed a constant number of iterations independent of the number of DOF in each case. HB and HBMG methods suffer from a logarithmic increase in the number of iterations. Among all the methods tested, the M.BPX is the closest to MG in terms of low iteration counts.

However, it should be clearly noted that in the experiments we present below, the cost per iteration of the various methods can differ substantially. We report flop counts of a single iteration of the above methods, see Figures 4.2 and 4.3. In experiment set I, the cost per iteration is linear for all the methods. The WMHB and WMHBMG methods are the most expensive ones. We would like to emphasize that the refinement

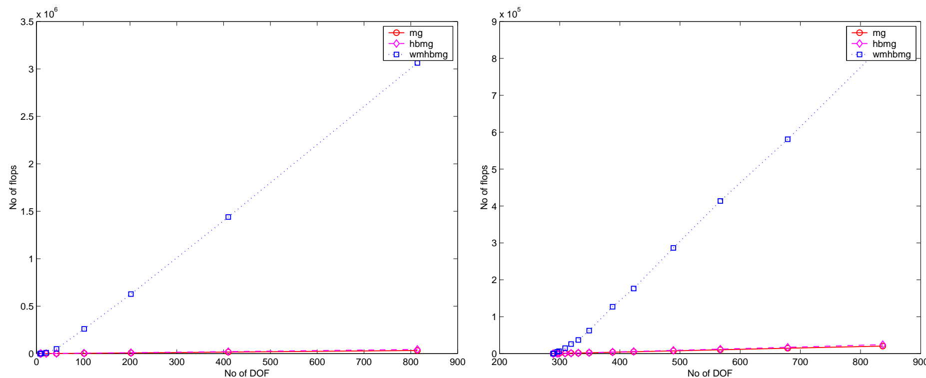


FIG. 4.4. *Flop counts for variational conditions for experiment set I (left) and experiment set II (right).*

in experiment set I cannot be a good example for adaptive refinement given the geometric increase in the number of DOF. MG exploits this geometric increase and enjoys a linear computational complexity. Experiment set II is more realistic in the sense that the refinement is highly adaptive and introduces a small number of DOF at each level. One can now observe a suboptimal (logarithmic) computational complexity for MG-like methods in such realistic scenarios. In accordance with the theoretical justification, under highly adaptive refinement MG methods will asymptotically be suboptimal. Moreover, storage complexity severely prevents MG-like methods from being a viable tool for large and highly adaptive settings.

Coarser representations of the finest level system (1.4) are algebraically formed by enforcing variational conditions. Some methods require further stabilizations in the from of matrix-matrix products. These form the so-called *preprocessing* step in multilevel methods. The computational cost of variational conditions is the same regardless of having a multiplicative or an additive version of the same method. This computational cost is orders of magnitude cheaper than the cost of a single iteration. However, this is the step where the storage complexity can dominate the overall complexity. Due to memory bandwidth problems on conventional machines, one should be very careful with the choice of datastructures. Since only the $A_{11} = A_{coarse}$ subblock of $A$ is formed for the next coarser level, the cost of variational conditions for MG, M.BPX, A.MG, and BPX is the cheapest among all the methods. On the other hand, HBMG and HB require stabilizations of $A_{12}$ and $A_{21}$ using the hierarchical basis. The WMHBMG and WMHB methods are more demanding by requiring stabilizations of $A_{12}$, $A_{21}$, and $A_{22}$ using the wavelet modified hierarchical basis. Wavelet structure creates denser change of basis matrix than that of the hierarchical basis. Therefore, preprocessing in the WMHB and WMHBMG methods is the most expensive among all the methods.

## 5. Conclusion.

In this paper, we examined a number of additive and multiplicative multilevel iterative methods and preconditioners in the setting of two-dimensional local mesh refinements. While standard multilevel methods are effective for uniform refinement-based discretizations of elliptic equations, they tend to be less effective for algebraic systems which arise from discretizations on locally refined meshes, losing their optimal behavior in both storage and computational complexity. Our primary focus here was on BPX-style additive and multiplicative multilevel preconditioners, and

on various stabilizations of the additive and multiplicative hierarchical basis method, and their use in the local mesh refinement setting. In the first two papers of this trilogy, it was shown that both BPX and wavelet stabilizations of HB have uniformly bounded condition numbers on several classes of locally refined 2D and 3D meshes based on fairly standard (and easily implementable) red and red-green mesh refinement algorithms. In this third article of the trilogy, we described in detail the implementation of these types of algorithms, including detailed discussions of the datastructures and traversal algorithms we employ for obtaining optimal storage and computational complexity in our implementations. We showed how each of the algorithms can be implemented using standard datatypes available in languages such as C and FORTRAN, so that the resulting algorithms have optimal (linear) storage requirements, thereby the resulting multilevel method or preconditioner can be applied with optimal (linear) computational costs.

We presented a sequence of numerical experiments illustrating the effectiveness of the BPX and stabilized HB methods in adaptive regimes. As expected, multigrid methods are most effective in terms of iteration counts (remaining a small constant as the DOF increase), but the suboptimal complexity per iteration in the local refinement setting makes the BPX methods the most attractive. Furthermore, storage complexity prohibits MG methods from being a viable tool for large and highly adaptive settings. In addition, both the additive and multiplicative WMHB-based methods and preconditioners demonstrated similar constant iteration requirements with increasing DOF, yet the cost per iteration remains optimal (linear) even in the local refinement setting. Consequently in highly adaptive regimes, the BPX methods prove to be the most effective, and the WMHB methods become the second most effective. The superiority of the BPX and WMHB methods would be more striking in large three-dimensional problems.

**Acknowledgments.** The authors thank R. Bank for many enlightening discussions.

## 6. Appendix: Highlights from The MCLite Implementation.

```
function [u]=multiplicative(b,lev,hb);
%%% Multiplicative methods: MG, M.BPX, HBMG, WMHBMG

%%% prolongation, stiffness, change of basis, one-ring
global P_12 P_23 P_34 P_45 A_1 A_2 A_3 A_4 A_5;
global level S_2 S_3 S_4 S_5 ONER_2 ONER_3 ONER_4 ONER_5;
global A_hb smthKey exactC bpx;

%%% get the stiffness matrix on this level
A = eval(['A_' num2str(lev)]);

if (lev == 1)
   if (exactC) u = A \ b; else u = b; end;
else
   ONER = eval(['ONER_' num2str(lev)]);

   %%% recover the dimensions
   P = eval(['P_' num2str(lev-1) num2str(lev)]);
   [r c] = size(P);

   %%% shorthand for the top and tail of vectors/matrices
```

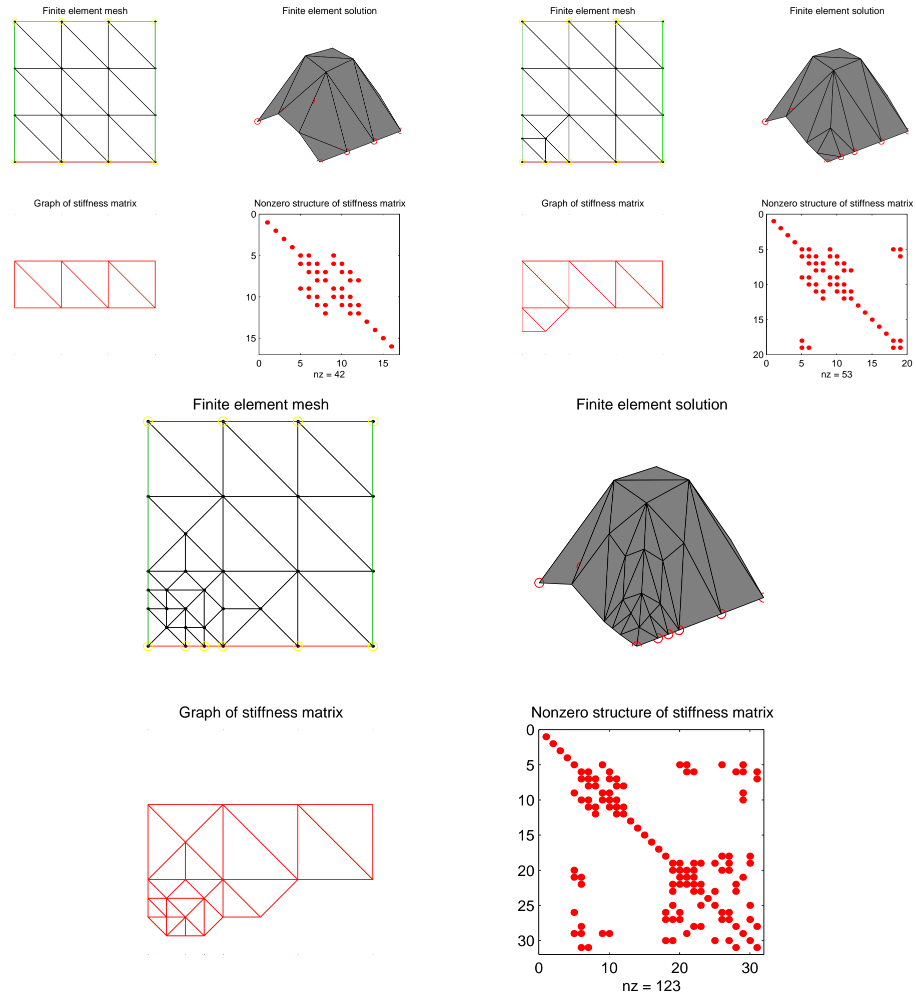


FIG. 5.1. *Dialog box from MCLite for experiment set I.*

```
top_  = 1:c;
tail_ = (c+1):r;

if (hb)
   u = zeros(r,1);
   f = b;

   %%% Get the change of basis matrix for this level
   S = eval(['S_' num2str(lev)]);
   %%% Transform f into the HB basis
   f = f + S'*f;
   %%% pre-smoothing by symmetric Gauss-Seidel
   u = smooth_point(A_hb,u,f,smthKey,2,lev);
   %%% correct f using smoother result
   f(top_,1) = f(top_,1) - A_hb(top_,tail_)*u(tail_,1);
else %%% mg/bpx
```

```
      u = zeros(c,1);
      d = zeros(r,1);
      if (bpx)
         d = smooth_point(A,d,b,smthKey,3,lev);
      else %%% mg
         d = smooth_point(A,d,b,smthKey,1,lev);
      end;

      %%% coarse grid defect restriction: f = P'*(b - A*d);
      if (bpx)
         f = b - A(:,ONER)*d(ONER);
      else %%% mg
         f = b - A*d;
      end

      f(top_,1) = f(top_,1) + P(tail_,:)'*f(tail_,1);
   end;

   %%% Recursion
   u(top_,1) = multiplicative(f(top_,1),lev-1,hb);

   if (hb)
      %%% correct f using the coarse solve result
      f(tail_,1) = f(tail_,1) - A_hb(tail_,top_)*u(top_,1);

      %%% post-smoothing by symmetric Gauss-Seidel
      u = smooth_point(A_hb,u,f,smthKey,2,lev);

      %%% transform u back into the nodal basis
      u = u + S*u;
   else %%% mg/bpx
      %%% interpolate result: u = P*u;
      u(tail_,1) =  P(tail_,:)*u(top_,1);

      if (bpx)
         u(ONER) = u(ONER) + d(ONER);
         u = smooth_point(A,u,b,smthKey,3,lev);
      else %%% mg
         u = u + d;
         u = smooth_point(A,u,b,smthKey,1,lev);
      end
end;


function [u]=additive(f,lev,hb);
%%% Additive methods: A.MG, BPX, HB, WMHB

%%% prolongation, stiffness, change of basis, one-ring
global P_12 P_23 P_34 P_45 A_1 A_2 A_3 A_4 A_5;
global S_2 S_3 S_4 S_5 ONER_2 ONER_3 ONER_4 ONER_5;
global A_hb smthKey exactC bpx;

%%% get the stiffness matrix on this level
A = eval(['A_' num2str(lev) ]);
```

```
if (lev == 1)
   if (exactC) u = A \ f; else u = f; end
else
   ONER = eval(['ONER_' num2str(lev)]);

   %%% recover the dimensions
   P = eval(['P_' num2str(lev-1) num2str(lev)]);
   [r c] = size(P);

   %%% shorthand for the top and tail of vectors/matrices
   top_  = 1:c;
   tail_ = (c+1):r;

   if (hb)
      u = zeros(r,1);

      %%% Get the change of basis matrix for this level
      S = eval(['S_' num2str(lev)]);
      %%% Transform f into the HB basis
      f = f + S'*f;
      %%% fine smoothing by symmetric Gauss-Seidel
      u = smooth_point(A_hb,u,f,smthKey,2,lev);
   else %%% additive MG
      u = zeros(c,1);
      d = zeros(r,1);
      %%% smoothing by symmetric Gauss-Seidel
      if (bpx)
         d = smooth_point(A,d,f,smthKey,3,lev);
      else %%% mg
         d = smooth_point(A,d,f,smthKey,1,lev);
      end;

      %%% coarse grid restriction: f = P'*f;
      f(top_,1) = f(top_,1) + P(tail_,:)'*f(tail_,1);
   end;

   %%% Recursion
   u(top_,1) = additive(f(top_,1),lev-1,hb);

   if (hb)
      %%% Transform u into the HB basis
      u = u + S*u;
   else
      %%% interpolate result: u = P*u;
      u(tail_,1) = P(tail_,:)*u(top_,1);
      if (bpx)
         u(ONER) = u(ONER) + d(ONER);
      else %%% mg
         u = u + d;
      end;
   end;
end;
```

REFERENCES

[1] B. Aksoylu, *Adaptive Multilevel Numerical Methods with Applications in Diffusive Biomolecular Reactions*, PhD thesis, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2001.

[2] B. Aksoylu and M. Holst, *An odyssey into local refinement and multilevel preconditioning I: Optimality of the BPX preconditioner*, SIAM J. Numer. Anal., (2002). in review.

[3] ———, *An odyssey into local refinement and multilevel preconditioning II: Stabilizing hierarchical basis methods*, SIAM J. Numer. Anal., (2002). in review.

[4] R. E. Bank and T. Dupont, *Analysis of a two-level scheme for solving finite element equations*, tech. rep., Center for Numerical Analysis, University of Texas at Austin, 1980. CNA–159.

[5] R. E. Bank, T. Dupont, and H. Yserentant, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[6] F. Bornemann and H. Yserentant, *A basic norm equivalence for the theory of multilevel methods*, Numer. Math., 64 (1993), pp. 455–476.

[7] J. H. Bramble and J. E. Pasciak, *New estimates for multilevel algorithms including the V-cycle*, Math. Comp., 60 (1993), pp. 447–471.

[8] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[9] W. Dahmen and A. Kunoth, *Multilevel preconditioning*, Numer. Math., 63 (1992), pp. 315–344.

[10] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[11] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15 (1989), pp. 1–14.

[12] M. Holst, *Adaptive numerical treatment of elliptic systems on manifolds*, Advances in Computational Mathematics, 15 (2001), pp. 139–191.

[13] S. Jaffard, *Wavelet methods for fast resolution of elliptic problems*, SIAM J. Numer. Anal., 29 (1992), pp. 965–986.

[14] P. Oswald, *Multilevel Finite Element Approximation Theory and Applications*, Teubner Skripten zur Numerik, B. G. Teubner, Stuttgart, 1994.

[15] R. Stevenson, *Robustness of the additive multiplicative frequency decomposition multi-level method*, Computing, 54 (1995), pp. 331–346.

[16] ———, *A robust hierarchical basis preconditioner on general meshes*, Numer. Math., 78 (1997), pp. 269–303.

[17] P. S. Vassilevski and J. Wang, *Stabilizing the hierarchical basis by approximate wavelets, I: Theory*, Numer. Linear Alg. Appl., 4 Number 2 (1997), pp. 103–126.

[18] ———, *Wavelet-like methods in the design of efficient multilevel preconditioners for elliptic PDEs*, in Multiscale Wavelet Methods For Partial Differential Equations, W. Dahmen, A. Kurdila, and P. Oswald, eds., Academic Press, 1997, ch. 1, pp. 59–105.

[19] ———, *Stabilizing the hierarchical basis by approximate wavelets, II: Implementation and numerical experiments*, SIAM J. Sci. Comput., 20 Number 2 (1998), pp. 490–514.

[20] J. Xu and J. Qin, *Some remarks on a multigrid preconditioner*, SIAM J. Sci. Comput., 15 (1994), pp. 172–184.

[21] H. Yserentant, *On the multilevel splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.