Spring 5-1-2003

An Efficient Block Variant of GMRES ; CU-CS-957-03

Baker
University of Colorado Boulder

E. Dennis
University of Colorado Boulder

Elizabeth R. Jessup
University of Colorado Boulder

AN EFFICIENT BLOCK VARIANT OF GMRES

A. H. BAKER *, J. M. DENNIS †, AND E. R. JESSUP ‡

Abstract. We present an alternative to the standard restarted GMRES algorithm for solving a single right-hand side linear system $Ax = b$ based on solving the block linear system $AX = B$. Additional starting vectors and right-hand sides are chosen to accelerate convergence. Algorithm performance, i.e. time to solution, is improved by using the matrix A in operations on groups of vectors, or "multivectors," thereby reducing the movement of A through memory. The efficient implementation of our method depends on a fast matrix-multivector multiply routine. We present numerical results that show that the time to solution of the new method is up to two and half times faster than that of restarted GMRES on preconditioned problems. We also demonstrate the impact of implementation choices on data movement and, as a result, algorithm performance.

Key words. GMRES, block GMRES, iterative methods, Krylov subspace techniques, restart, nonsymmetric linear systems, memory access costs

AMS subject classifications. 65F10, 65Y20

1. Introduction and motivation.

A wide range of physical processes are described by systems of linear and nonlinear partial differential equations (PDEs). Examples of such processes range from acoustic scattering [20] to structural analysis [21] to fluid flow [1]. Approximating the solutions of these systems of PDEs typically requires the solution of a large linear system, which frequently consumes a significant portion of the total application time [52, 21]. For this reason, reducing the time of the linear solve is of great interest.

Though the cost of a numerical linear algebra algorithm has traditionally been measured in terms of the floating-point operations required, a more memory-centric approach has long been advocated (e.g., see [24, 17, 33, 31]). It is well-known that matrix algorithm performance continues to be limited by the gap between microprocessor performance and memory access time (e.g., see [17, 59, 37, 35, 25, 1]). In fact, the discrepancy between DRAM (dynamic random access memory) access time and microprocessor speeds has increased by nearly 50% per year [46]. As a result, the percentage of overall application time spent waiting for data from main memory has increased [46]. The situation is compounded by advances in algorithm development that have decreased the total number of floating-point operations required by many algorithms [25]. Therefore, the number of floating-point operations required by an algorithm is not necessarily an accurate predictor of algorithm performance for a large matrix algebra problem [25]. In fact, performance bounds based on sustainable memory bandwidth (such as the STREAM [36] benchmark) are thought to predict algorithm performance most accurately [25, 1]. In particular, when solving a linear system, efficient data reuse (i.e., reducing data movement) is crucial to reducing an algorithm's memory costs [33, 17, 24].

For these reasons, we are interested in the problem of achieving a balance between

*Department of Applied Mathematics, University of Colorado, Boulder, CO 80309-0526, (allison.baker@colorado.edu). The work of this author was supported by the Department of Energy through a Computational Science Graduate Fellowship (DE-FG02-97ER25308).

†Scientific Computing Division, National Center for Atmospheric Research, Boulder, CO 80307-3000 (dennis@ucar.edu). The work of this author was supported by the National Science Foundation.

‡Department of Computer Science, University of Colorado, Boulder, CO 80309-0430 (jessup@cs.colorado.edu). The work of this author was supported by the National Science Foundation under grant no. ACI-0072119.

improving the efficiency of a sparse linear solver from a memory-usage standpoint and maintaining favorable numerical properties. Therefore, we investigate modifying a standard linear solver algorithm such that data movement is reduced while preserving accuracy and evaluate the impact of the changes on performance (time to solution). In particular, we consider the solution of the linear system $Ax = b$, where A is a sparse, nonsymmetric square matrix, b is a single right-hand side, and x is a single solution vector. We focus on the GMRES (generalized minimum residual) [51] method because it is commonly used to solve large, sparse, nonsymmetric linear systems resulting from partial differential equations. In this paper, we establish the feasibility of improving the performance of the restarted GMRES algorithm, and likewise other iterative solvers, via algorithmic changes that improve data reuse.

Many iterative methods such as GMRES, and Krylov methods in particular, are based on an iteration loop that accesses the coefficient matrix A once per loop and performs a matrix-vector multiply. To reduce data movement in the GMRES algorithm, our approach is to modify the algorithm such that more than one matrix-vector product occurs for a single memory access of A. To this end, we investigate an alternative for solving a single right-hand side system based on solving a corresponding block linear system $AX = B$, where X and B are both groups of vectors. In this work, capital letters indicate matrices or groups of vectors (*multivectors*), and lower case letters indicate single vectors. A block system allows multiple matrix-vector products for a single memory access of A. However, to use a block system to solve a single right-hand side system, we had to choose additional starting vectors and right-hand sides. Ideally these additional vectors accelerate convergence to the solution. Inspiration for such vectors came from the augmented Krylov subspace method LGMRES [3]. In fact, the block algorithm that we present in this paper is a theoretical extension of the LGMRES algorithm and has similarly appealing convergence properties.

Beyond the algorithmic modifications required for a block formulation of restarted GMRES, close attention to implementation techniques is essential for a memory-efficient, sparse linear solver. In our new block method, we reduce the cost of the additional matrix-vector operations in each iteration loop via an innovative matrix-multivector multiply and associated routines. We demonstrate the importance of certain implementation decisions by tracking data movement through parts of the memory hierarchy, which consists of registers, cache, and main memory. Furthermore, we show the correlation between time to solution and data movement. Of particular interest is the unexpected importance of optimizations that reduce movement of data between levels of cache.

In this paper, we report on our investigation into a more memory-efficient iterative linear solver and describe the resulting new block method. We emphasize the implementation of the new method, numerous numerical test results, and evaluation of performance based on the tracking of data movement. First, in Section 2, we review the standard GMRES and Block GMRES algorithms and discuss considerations for reducing an algorithm's memory access costs. Next, in Section 3, we begin with a description of the related LGMRES algorithm and then introduce our new block method. We present convergence results for a variety of problems and discuss implications of implementation decisions on data movement in Section 4. Finally, concluding remarks and recommendations are given in Section 5.

2. Background. In the previous section, we motivated our work, and now we provide additional relevant background information. First we briefly review the standard GMRES algorithm. We then describe the Block GMRES algorithm and several

of its variants. Finally we discuss specific implications of the growing gap between memory and CPU performance on an iterative solver code.

2.1. GMRES. We consider the solution of a large sparse system of linear equations:

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, and $x, b \in \mathbb{R}^n$. Iterative methods are often chosen to solve such linear systems, and, when A is nonsymmetric, the GMRES [51] algorithm is a common choice. GMRES is a Krylov subspace method, and, therefore, selects an approximate solution $x_m \in x_0 + \mathcal{K}_m(A, r_0)$, where $\mathcal{K}_m(A, r_0) \equiv Span\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ denotes an m-dimensional Krylov subspace, r_0 is the initial residual, x_0 is the initial guess, and $r_0 \equiv b - Ax_0$. The GMRES minimum residual property requires that $\|b - A(x_0 + z)\|_2$ be a minimum over all $z \in \mathcal{K}_m(A, r_0)$, or, equivalently, that the residual after m iterations satisfy $r_m \perp A\mathcal{K}_m(A, r_0)$.

In practice, the resources required by the standard GMRES algorithm may be impractical since storage and computational requirements increase with each iteration. In this case, the restarted version described in [51] is typically used. In restarted GMRES (GMRES(m)), the restart parameter m denotes a fixed maximum dimension for the Krylov subspace. Therefore, if convergence has not occurred at the end of m iterations, the algorithm restarts with $x_0 = x_m$. We refer to the group of m iterations between successive restarts as a cycle. We denote the restart number with a subscript: x_i is the approximate solution after i cycles or $m \cdot i$ total iterations, and r_i is the corresponding residual ($r_i = b - Ax_i$). After i cycles, the residual is a polynomial in A times the residual from the previous cycle: $r_i = p_i^m(A)r_{i-1}$, where $p_i^m(A)$ is the degree m residual polynomial.

2.2. Block GMRES. Applications often arise in which several linear systems with the same coefficient matrix A but different right-hand sides need to be solved (e.g., see [55, 8]). These linear systems can be written as a single block linear system

$$AX = B, \tag{2}$$

where $A \in \mathbb{R}^{n\times n}$, $X, B \in \mathbb{R}^{n\times s}$, and the columns of B are the different right-hand sides. O'Leary first introduced block iterative solvers for block linear systems with symmetric A with the Block Conjugate Gradient (BCG) and other related algorithms in [45]. For nonsymmetric A, a block version of the GMRES algorithm (BGMRES) is first described in [57]. Detailed descriptions can also be found in [53], [54], and [49].

BGMRES is essentially identical to standard GMRES, except that operations are performed with multivectors instead of single vectors. As with the standard GMRES algorithm, a restarted version of the BGMRES algorithm (BGMRES(m)) is commonly used in practice. For reference, one restart cycle (i) of BGMRES(m) is given in Figure 1. The Arnoldi iteration is now a block orthogonalization process, creating a basis for the block Krylov subspace $\mathcal{K}_m^s(A, R_i) = Span\{R_i, AR_i, \ldots, A^{m-1}R_i\}$, where s indicates the block size and R_i is given in line 1 of Figure 1. Because the block linear system (2) has s right-hand sides, the solution subspace is now of dimension $m \cdot s$.

In addition to being able to solve a multiple right-hand side system, BGMRES is desirable in terms of reducing memory access costs. By solving the block linear system (as opposed to solving s systems individually), A now operates on a multivector instead of a single vector at each iteration. Therefore, matrix A is accessed from memory fewer times than it would be if each system were solved individually.

1. $R_i = B - AX_i$
2. $R_i = V_1 \hat{R}$
3. for $j = 1 : m$
4. $\quad U_j = AV_j$
5. $\quad$ for $l = 1 : j$
6. $\quad\quad H_{l,j} = V_l^T U_j$
7. $\quad\quad U_j = U_j - V_l H_{l,j}$
8. $\quad$ end
9. $\quad U_j = V_{j+1} H_{j+1,j}$
10. end
11. $W_m = [V_1, V_2, \ldots, V_m]$, $H_m = \{H_{l,j}\}_{1 \le l \le j+1; 1 \le j \le m}$
12. find Y_m s.t. $\|E_1 \hat{R} - H_m Y_m\|_2$ is minimized
13. $X_{i+1} = X_i + W_m Y_m$

FIG. 1. *BGMRES(m) for restart cycle i.*

Several variants of the Block GMRES method have been developed for both multiple and single right-hand side systems. For multiple right-hand side systems, a hybrid block GMRES method is presented in [53]. This block method is analogous to a hybrid GMRES method for single right-hand side systems described in [41] and often solves the block system faster than solving each single system individually due in part to lower memory accesses. In addition, in [27], a block extension of Morgan's GMRES with eigenvectors routine [39] is described for multiple right-hand side systems, and Morgan himself has developed a block extension of GMRES-DR (GMRES with Deflated Restarting) [40, 38]. Methods such as Morgan's that augment the approximation space with eigenvector estimates are particularly useful for moderately non-normal matrices with a few particular eigenvalues (typically those small in magnitude) that inhibit convergence.

As explained in Section 1, of primary interest to us is the use of a block method on a block linear system to solve a single right-hand side system. This strategy is briefly noted as a possibility by O'Leary in [45] for block Conjugate Gradient algorithms. To our knowledge, the first mention of using Block GMRES to solve a single right-hand side system occurs in [9]. In this work, Chapman and Saad suggest that convergence may be improved for $Ax = b$ by using a block method with approximate eigenvectors or random vectors for the additional right-hand side vectors, particularly when convergence is hampered by a few small eigenvalues. In addition, motivated by a practical implementation, Li, in [34], presents a block variant of GMRES for single right-hand sides systems that is mathematically equivalent to standard GMRES. His method is implemented as an s-step method; matrix A performs s consecutive matrix-vector multiplies at each iteration. In addition, this blocked implementation allows for the use of level 3 BLAS (Basic Linear Algebra Subprograms) [14], the advantages of which are described in the next section. The primary drawback of Li's method is loss of accuracy with increasing block size.

In [57], Vital shows that the residual obtained from BGMRES is bounded by the maximum of any of the residuals obtained from running standard GMRES on each single right-hand side system. This result implies that augmenting a single right-hand side system with additional right-hand sides (such as with eigenvector approximations as in [9]) does not increase the number of times A is accessed from memory, provided the iteration terminates when the solution to the original single right-hand side system

converges. However, with restarting, convergence analysis is more complex, and a similar conclusion cannot be drawn.

2.3. Considerations for reducing memory costs. Algorithmic changes in a numerical code affect the manner in which data are moved through a memory hierarchy. Floating-point operations typically occur after data are moved from the slower part of the memory hierarchy (main memory) to the faster and more limited caches to the registers. Two levels of cache are typical for current microprocessor designs, and three levels are increasingly common. We are most interested in data movement between main memory and cache and between levels of cache as this movement is most affected by our algorithmic changes. L1 caches typically have access times of two to three clock cycles. Access times are generally 5-15 times slower for L2 caches and 20-30 times slower for L3 caches. Accessing data from main memory typically requires at least 100 times more cycles than does accessing data from L1 cache. Several complete hardware specifications are found in [58, 42, 6, 32].

Data are moved through a memory hierarchy in units called cache lines. Cache lines are used efficiently when data items located in a single cache line are accessed in close succession. A program with this property has good memory reference locality [28] which helps to minimize data movement. However, sparse matrix-vector multiply routines often have poor memory reference locality due to the use of compressed storage formats for sparse matrices. These storage schemes generally result in irregular patterns of memory referencing via indirect addressing (e.g., see [22, 16]).

Approaches to improving the performance of the sparse matrix-vector multiply for a given matrix problem include reordering the matrix and choosing matrix-specific data structures (e.g., see [56]). In particular, one approach is to optimize the sparse matrix-vector multiplication by representing the sparse matrix as a collection of small dense blocks (e.g., see [29, 47, 58]). These dense blocks are created through matrix reordering, the addition of non-zero elements, or a combination. This optimization technique reduces indirect addressing, which in turn reduces the impact of memory system latency on algorithm performance. Another well-known technique referred to as loop blocking involves writing matrix operations that divide the matrix into submatrices or blocks that are better suited for reuse in cache than a matrix row or column array (e.g., see [24, 7, 33]). For example, the dense level 2 and 3 BLAS [15, 14] employ loop blocking, and codes based on these routines often achieve good performance because they minimize data movement (e.g., see [24, 12, 14, 13, 25]).

Of particular interest to us is an approach in [25, 26] that improves the performance of *multiple* sparse matrix-vector multiplies. This approach allows the multiplication of a *multivector* of size four (i.e., a group of four vectors) by a matrix at about 1.5 times the cost of calculating a single matrix-vector product. The storage of the vectors (as opposed to the matrix) are manipulated to maximize cache line reuse in the following manner. Consider the size s multivector V, where $V \equiv [v_1, v_2, \ldots, v_s]$ for vectors v_1, v_2, ..., $v_s \in \mathbb{R}^{n\times 1}$. Instead of placing successive elements of a single vector next to each other in the multivector, multivector V is stored as a vector of length $n \cdot s$ in which the components of the s vectors are *interlaced*. Therefore multivector elements separated by stride s belong to a single vector: $v_1 = [V(0); V(s); V(2s); \ldots; V(ns - s)]$. This optimization effectively increases the number of floating point operations performed on data in cache by placing corresponding elements of each vector in the same cache line, reducing the required memory bandwidth. In Section 4, we demonstrate the advantages of this multivector implementation for our new method with numerical results and offer a more detailed

explanation of the specific areas of performance gains.

3. A new block algorithm: B-LGMRES. As noted in Section 1, our approach to improving data reuse for restarted GMRES is to reformulate the algorithm to solve a single right-hand side linear system as in (1) via a block linear system as in (2). However, to reduce memory access costs effectively with algorithmic changes, these changes cannot degrade the numerical properties of the original algorithm. Therefore, our goal in choosing appropriate families of additional right-hand sides and starting vectors was to select vectors that would accelerate convergence to the single right-hand side solution. For this reason, we looked to augmented Krylov subspace techniques and restarted Krylov method properties for inspiration in choosing these vectors.

Consider the generic case where vector c_i is chosen as an additional right-hand side vector with a corresponding zero initial guess for restart cycle $i+1$. This choice results in an approximation space consisting of Krylov spaces $\mathcal{K}_m(A, r_i)$ and $\mathcal{K}_m(A, c_i)$. In other words,

$$x_{i+1} = x_i + q_{i+1}^{m-1}(A)r_i + f_{i+1}^{m-1}(A)c_i, \tag{3}$$

where q_{i+1}^{m-1} and f_{i+1}^{m-1} are degree $m-1$ polynomials. As a result, the residual after $i+1$ restart cycles is

$$r_{i+1} = p_{i+1}^{m}(A)r_i + g_{i+1}^{m}(A)c_i,$$

where p_{i+1}^{m} and g_{i+1}^{m} are degree m polynomials with $p_{i+1}^{m}(0) = 1$ and $g_{i+1}^{m}(0) = 0$. The fundamental problem when converting a single right-hand side system into a block system is finding a readily available vector c_i such that $\|r_{i+1}\|_2$ is minimized through choices of p_{i+1}^{m} and g_{i+1}^{m}.

The LGMRES method [3] is a restarted augmented Krylov subspace technique. It is well-known that subspace information is lost due to restarting, and the LGMRES augmenting scheme in some sense compensates for this loss by augmenting with certain error approximations. We found that using these same error approximation vectors as additional right-hand side vectors, such as c_i in (3), is quite effective. In this section, we first briefly review the LGMRES method, to which our new method is closely related. We then present the new algorithm, discussing its implementation and some additional considerations.

3.1. A review of the LGMRES method. In [3], two of the authors of this work present the LGMRES ("Loose" GMRES) method, a technique for accelerating the convergence of restarted GMRES. The LGMRES method is an augmented Krylov subspace method that requires only minor changes to the GMRES(m) algorithm. At the end of each restart cycle of LGMRES(m, k), k vectors that approximate the error from previous restart cycles are appended to the standard Krylov approximation space.

Let x_i and x_{i-1} be the approximate solution vectors after i and $i-1$ restart cycles, respectively, of standard GMRES(m). Let $\hat{x}$ be the true solution to (1). We denote the error after the i-th restart cycle by e_i, where

$$e_i \equiv \hat{x} - x_i. \tag{4}$$

Because x_i is found such that $x_i \in x_{i-1} + \mathcal{K}_m(A, r_{i-1})$, then

$$z_i \equiv x_i - x_{i-1} \tag{5}$$

1. $r_i = b - Ax_i$, $\beta = \|r_i\|_2$, $v_1 = r_i/\beta$
2. for $j = 1 : m + k$
3. $\quad u = \begin{cases} Av_j & if\ \ j \le m \\ Az_{i-(j-m-1)} & otherwise \end{cases}$
4. $\quad$ for $l = 1 : j$
5. $\quad\quad h_{l,j} = \langle u, v_l \rangle$
6. $\quad\quad u = u - h_{l,j} v_l$
7. $\quad$ end
8. $\quad h_{j+1,j} = \|u\|_2$, $v_{j+1} = u/h_{j+1,j}$
10. end
11. $W_{m+k} = [v_1, \ldots, v_m, z_i, \ldots, z_{i-k+1}]$, $H_{m+k} = \{h_{l,j}\}_{1 \le l \le j+1; 1 \le j \le m+k}$
12. find y_{m+k} s.t. $\|\beta e_1 - H_{m+k} y_{m+k}\|_2$ is minimized
13. $z_{i+1} = W_{m+k} y_{m+k}$ (also $Az_{i+1} = V_{m+k+1} H_{m+k} y_{m+k}$)
14. $x_{i+1} = x_i + z_{i+1}$

FIG. 2. *LGMRES(m, k) for restart cycle i (Figure 1 in [3]).*

is an approximation to the error in the approximate solution after i restart cycles, where $z_i \equiv 0$ for $i < 1$. An error approximation is a natural choice of vector with which to augment our next approximation space $\mathcal{K}_m(A, r_i)$ since augmenting with the exact error would solve the problem exactly (e.g., see [19]). Furthermore, because $z_i \in \mathcal{K}_m(A, r_{i-1})$, it in some sense represents the space $\mathcal{K}_m(A, r_{i-1})$ generated in the previous cycle. Therefore, after $i + 1$ restart cycles, LGMRES(m, k) finds an approximate solution to (1) in the following way:

$$x_{i+1} = x_i + q_{i+1}^{m-1}(A) r_i + \sum_{j=i-k+1}^{i} \alpha_{ij} z_j, \tag{6}$$

where polynomial q_{i+1}^{m-1} and α_{ij} are chosen such that $\|r_{i+1}\|_2$ is minimized.

As shown in [3], equation (6) can be rewritten as

$$z_{i+1} = q_{i+1}^{m-1}(A) r_i + \sum_{j=i-k+1}^{i} \alpha_{ij} z_j,$$

and the error approximation vectors $z_j \equiv x_j - x_{j-1}$ with which we augment the Krylov space are A^*A-orthogonal. In this form, the LGMRES(m, k) algorithm resembles a truncated polynomial-preconditioned full conjugate gradient method where the polynomial preconditioner is a GMRES(m) iteration polynomial that varies with each step (i). The LGMRES(m, k) implementation requires minimal modifications to the standard GMRES(m) implementation and, in fact, is similar to that of Morgan's GMRES with eigenvectors (GMRES-E) method [39]. The LGMRES method is also closely related to the Flexible GMRES [48] method (e.g., see [50]). For reference, pseudo-code for one restart cycle (i) of LGMRES(m, k) is given in Figure 2. In the first cycle (i.e., $i = 0$) of LGMRES(m, k), no error approximation vectors are available since $z_0 \equiv 0$. In general, for cycles $i < k$, only i error approximations are appended to the space, irrespective of the value of k.

3.2. Idea. The augmentation scheme used by the LGMRES(m, k) algorithm is easily extended to a block method. We refer to this block extension as B-LGMRES(m, k), for "Block" LGMRES. The B-LGMRES method solves a single right-hand side

system (1) via the block system (2). In B-LGMRES(m, k), the k previous error approximations z_j are now used to build additional Krylov subspaces, as opposed to simply being appended to the approximation space as in LGMRES(m, k). Conceptually, we view one restart cycle (i) of B-LGMRES(m, k) as a cycle of standard BGMRES on the system $AX_{i+1} = B$, where B contains the right-hand side of (1) and the k most recent error approximations z_j, $j = (i-k+1) : i$. In other words, $B = [b,\ z_i,\ \ldots,\ z_{i-k+1}]$. The previous approximate solution, X_i, is then written as $X_i = [x_i,\ 0,\ \ldots,\ 0]$, where the solution to our single right-hand side system, x_i is placed in the first column and the remaining columns are set to zero. X and B are now size $n \times (k+1)$ or $n \times s$, where $s \equiv m + k$ indicates the block size.

After $i+1$ restart cycles, the B-LGMRES(m, k) approximation space consists of the traditional Krylov portion built by repeated application of A to the current residual r_i together with Krylov spaces resulting from the application of A to previous error approximations:

$$x_{i+1} \in x_i + \mathcal{K}_m(A, r_i) + \sum_{j=i-k+1}^{i} \mathcal{K}_m(A, z_j).$$

Similar to the formulation of LGMRES(m, k) given in (6), we now write the B-LGMRES(m, k) approximation as

$$x_{i+1} = x_i + q_{i+1}^{m-1}(A) r_i + \sum_{j=i-k+1}^{i} \alpha_{ij}^{m-1}(A) z_j, \tag{7}$$

where degree $m-1$ polynomials α_{ij}^{m-1} and q_{i+1}^{m-1} are chosen such that $\|r_{i+1}\|_2$ is minimized. The primary difference between the LGMRES approximation in (6) and the B-LGMRES approximation in (7) is that the coefficients α_{ij} of z_j in B-LGMRES(m, k) are now *polynomials* in A. In both (6) and (7), $k = 0$ corresponds to standard GMRES(m). Recall that after $i+1$ cycles, GMRES(m) finds

$$\|r_{i+1}\|_2 = \min_{q_{i+1} \in \mathcal{P}_{m-1}} \|(I - A q_{i+1}(A)) r_i\|_2,$$

where $\mathcal{P}_{m-1}$ is the space of polynomials of degree $\leq m+1$ (e.g., see [49]). Now a similar expression is given for B-LGMRES(m, k); in particular, for $k = 1$,

$$\|r_{i+1}\|_2 = \min_{q_{i+1}, \alpha_{ii} \in \mathcal{P}_{m-1}} \|(I - A q_{i+1}(A)) r_i + \alpha_{ii}(A)(r_i - r_{i-1})\|_2.$$

Note that polynomial $\alpha_{ii}(A)$ is defined in (7), and from (5), $Az_i = r_{i-1} - r_i$.

Alternatively, it is natural to think of restart cycle $i+1$ of GMRES(m) and B-LGMRES(m, k) in terms of minimizing $\|e_{i+1}\|_{A^*A}$ since this minimization is equivalent to minimizing $\|r_{i+1}\|_2$. Recall that e_{i+1} is the true error (not an approximation like z_{i+1}) as defined in (4). To simplify the following discussion, we consider the particular case for B-LGMRES(m, k) where $k = 1$. After $i+1$ restart cycles, GMRES(m) finds an approximate solution $x_{i+1} \in x_i + q_{i+1}^{m-1}(A) r_i$ such that

$$e_{i+1} \perp_{A^*A} \mathcal{K}_m(A, r_i).$$

The error e_{i+1} is written as

$$e_{i+1} = [I - A q_{i+1}^{m-1}(A)] e_i. \tag{8}$$

In contrast, B-LGMRES(m, 1) finds x_{i+1} as in (7) such that

$$e_{i+1} \perp_{A^*A} \mathcal{K}_m(A, r_i) + \mathcal{K}_m(A, e_i - e_{i-1}),$$

resulting in

$$e_{i+1} = [I - Aq_{i+1}^{m-1}(A) + \alpha_{ii}^{m-1}(A)]]e_i - \alpha_{ii}^{m-1}(A)e_{i-1}. \tag{9}$$

Comparing (8) and (9), we note that for B-LGMRES(m, 1), the new error e_{i+1} now depends on the error from *two* previous restart cycles, e_i and e_{i-1}.

Now we re-write B-LGMRES(m, k) in (7) in the following way:

$$z_{i+1} = q_{i+1}^{m-1}(A)r_i + \sum_{j=i-k+1}^{i} \alpha_{ij}^{m-1}(A)z_j. \tag{10}$$

In the above form, as shown for the LGMRES method, this block method resembles a truncated polynomial-preconditioned full conjugate gradient method with variable preconditioning. In this analogy, the polynomial preconditioner is q_{i+1}^{m-1}, and it changes at each step (i). In addition, the error approximation vectors z_j are analogous to conjugate gradient direction vectors (e.g., see [2]), and now these direction vectors are weighted by polynomials in A instead of scalars. As in Theorem 3 of [3], it is easily shown that these direction vectors z_j are A^*A-orthogonal.

THEOREM 1 (Orthogonality of the error approximations). *The error approximation vectors $z_j \equiv x_j - x_{j-1}$ with which we augment the Krylov space in B-LGMRES are A^*A-orthogonal.*

Proof. We define subspaces $\mathcal{M}_{i+1}$ and $\mathcal{M}_i$ as

$$\mathcal{M}_{i+1} \equiv \mathcal{K}_m(A, r_i) + \sum_{j=i-k+1}^{i} \mathcal{K}_m(A, z_j)$$

and

$$\mathcal{M}_i \equiv \mathcal{K}_m(A, r_{i-1}) + \sum_{j=i-k}^{i-1} \mathcal{K}_m(A, z_j),$$

respectively. Finding a new $z_{i+1} \in \mathcal{M}_{i+1}$ such that $\|r_{i+1}\|_2$ is minimized as in (10) is equivalent to requiring that

$$e_{i+1} \perp_{A^*A} \mathcal{M}_{i+1}. \tag{11}$$

By recursion,

$$e_i \perp_{A^*A} \mathcal{M}_i.$$

Because

$$e_i - e_{i+1} = z_{i+1},$$

then

$$z_{i+1} \perp_{A^*A} \mathcal{M}_i \cap \mathcal{M}_{i+1}.$$

Recalling the definitions of $\mathcal{M}_{i+1}$ and $\mathcal{M}_i$ and that $z_i \in \mathcal{M}_i$,

$$\{z_j\}_{j=(i-k+1):i} \subset \mathcal{M}_i \cap \mathcal{M}_{i+1}.$$

As a result,

$$z_{i+1} \perp_{A^*A} \{z_j\}_{j=(i-k+1):i}.$$

□

To provide additional insight into the B-LGMRES augmentation scheme, we now examine the B-LGMRES method in the framework presented in [9]. In [9], Chapman and Saad consider the addition of an arbitrary subspace, $\mathcal{W}$, to the standard Krylov approximation space of a minimum-residual method, such as GMRES. This addition results in the augmented approximation space

$$\mathcal{M} = \mathcal{K} + \mathcal{W}, \tag{12}$$

where $\mathcal{K}$ is a standard Krylov subspace. A minimal residual method finds an approximate solution $\tilde{x} \in x_0 + \mathcal{M}$ such that the residual $\tilde{r}$ satisfies $\tilde{r} \perp A\mathcal{M}$, removing components of the initial residual r_0 in subspace $A\mathcal{M}$ via an orthogonal projection process. Following the discussion in [9], the minimization process over the augmented approximation space (12) is

$$\|\tilde{r}\|_2 = \min_{d,w} \|b - Ad - Aw\|_2,$$

where $d \in x_0 + \mathcal{K}$ and $w \in \mathcal{W}$. Chapman and Saad then show that if r_d results from minimizing $\|b - Ad\|_2$, where again $d \in x_0 + \mathcal{K}$, then

$$\|\tilde{r}\|_2 \leq \|(I - P_{AW})r_d\|_2, \tag{13}$$

where P_{AW} is an orthogonal projector onto subspace $A\mathcal{W}$. In particular, for a cycle of B-LGMRES(m, k) with $k = 1$, we have that $\mathcal{K} = \mathcal{K}_m(A, r_i)$ and $\mathcal{W} = \mathcal{K}_m(A, z_i) = \mathcal{K}_m(A, e_i - e_{i-1})$. Therefore from (13), we see that the addition of $\mathcal{W}$ in the B-LGMRES method results in the removal of components of r_d (the residual from the standard Krylov approximation space) from the subspace $A\mathcal{K}_m(A, e_i - e_{i-1})$, or equivalently, the removal of components of the error from subspace $\mathcal{K}_m(A, e_i - e_{i-1})$.

3.3. Implementation. The implementation of B-LGMRES(m, k) is similar to that of BGMRES given in Figure 1. The primary difference is that B-LGMRES(m, k) finds approximations x_i at each restart cycle to the solution of the single right-hand side system (1), as opposed to the block system (2). One restart cycle (i) of B-LGMRES(m, k) is given in Figure 3.

Though we think of the error approximations z_j, $j = (i - k + 1) : i$ as additional right-hand side vectors, we are only solving a single right-hand side system and do not need to save the block approximate solutions X_i. Instead we append the k most recent error approximations to the initial residual to form a block residual R_i, as seen in line 2 of Figure 3. We normalize the error approximations ($z_j/\|z_j\|_2$) so that each column of the initial residual block R_i is of unit length. During each restart cycle, an orthogonal basis for the block Krylov space $\mathcal{K}_m^s(A, R_i)$ is generated. Using the same notation as in Section 2.2, the size $n \times s$ orthogonal block matrices V_j form the orthogonal $n \times m \cdot s$ matrix W_m, where $W_m = [V_1, V_2, \ldots, V_m]$. H_m is a size

1. $r_i = b - Ax_i$, $\beta = \|r_i\|_2$
2. $R_i = [r_i, z_i, \ldots, z_{i-k+1}]$
3. $R_i = V_1 \hat{R}$
4. for $j = 1 : m$
5. $\quad U_j = AV_j$
6. $\quad$ for $l = 1 : j$
7. $\quad\quad H_{l,j} = V_l^T U_j$
8. $\quad\quad U_j = U_j - V_l H_{l,j}$
9. $\quad$ end
10. $\quad U_j = V_{j+1} H_{j+1,j}$
11. end
12. $W_m = [V_1, V_2, \ldots, V_m]$, $H_m = \{H_{l,j}\}_{1 \le l \le j+1; 1 \le j \le m}$
13. find y_m s.t. $\|\beta e_1 - H_m y_m\|_2$ is minimized
14. $z_{i+1} = W_m y_m$
15. $x_{i+1} = x_i + z_{i+1}$

FIG. 3. *B-LGMRES(m, k) for restart cycle i.*

$(m+1)s \times m \cdot s$ band-Hessenberg matrix with s sub-diagonals, and the following standard relationship holds:

$$AW_m = W_{m+1} H_m.$$

Because B-LGMRES(m, k) finds the minimum residual approximate solution x_{i+1} to $Ax = b$, the least squares solution step (line 13) varies from that of standard BGMRES (which minimizes a block residual). B-LGMRES(m, k) finds y_m such that $x_{i+1} = x_i + W_m y_m$, where y_m is size $m \cdot s \times 1$. As a result, triangular matrix $\hat{R}$ from the QR decomposition in line 3 does not need to be saved since the least-squares solve only requires $\beta = \|r_i\|_2$. As with BGMRES, B-LGMRES(m, k) still requires the application of s^2 rotations at each iteration to transform H_m into an upper triangular matrix (see [49] for more details). However, now only s rotations must be applied to the least-squares problem's right-hand side (βe_1) at each step.

As with LGMRES, only i error approximations are available at the beginning of restart cycles with $i < k$. As a result, the creation of the initial residual block in line 2 of the B-LGMRES(m, k) algorithm must be modified for the first k cycles where $i < k$. (the first cycle is $i = 0$). Recall that $z_j \equiv 0$ for $j < 1$, and when $z_j \equiv 0$, we say that z_j is an error approximation that is not available. In our implementation, we replace any $z_i, \ldots, z_{i-k+1}$ that is not available with a randomly generated vector of length n. This choice is justified in Section 3.4.

We refer to a group of vectors as a multivector. For the B-LGMRES(m, k) implementation given in Figure 3, the multivectors are size s, where $s = k + 1$. For example, the orthogonal block matrices V_j in lines 3, 5, 8, 10, and 12 of Figure 3 are size s multivectors consisting of s vectors of length n. In addition, R_i in lines 2 and 3 and U_j in lines 5, 7, 8 and 10 are also multivectors. The matrix-multivector multiply occurs in line 5, and the importance of this aspect of the implementation is shown in Section 4.2.

In Figure 3, observe that one restart cycle, or m iterations, of B-LGMRES(m, k) requires m matrix-*multivector* multiplies, irrespective of the value of k. Therefore, matrix A is accessed from memory m times per cycle. The B-LGMRES(m, k) algorithm requires storage for the following vectors of length n: $(m+1)s$ orthogonal basis

vectors $(V_1, V_2, \ldots\ V_{m+1})$, k error approximation vectors (z_j), the approximate solution (x_i), and the right-hand side (b). Therefore, B-LGMRES$(m,\ k)$ requires storage for $(m+2)s+1$, or equivalently $(m+2)k+m+3$, vectors of length n.

B-LGMRES$(m,\ k)$ is compatible with both left and right preconditioning. We denote the preconditioner by M^{-1}. For left preconditioning, the initial residual in line 1 of Figure 3 must be preconditioned as usual: $r_i = M^{-1}(b - Ax_i)$. Then we replace A with $M^{-1}A$ in line 5. To incorporate right preconditioning, we replace A with AM^{-1} in line 5. We define $\hat{z}_j \equiv M(x_j - x_{j-1}) = Mz_j$ and replace z with $\hat{z}$ everywhere in lines 2 and 14. While no explicit change is required for line 15 as given in Figure 3, note that, with right preconditioning, line 15 is equivalent to $x_{i+1} = x_i + M^{-1}\hat{z}_{i+1}$.

3.4. Additional algorithmic considerations. Deflation is often an important issue for block methods since solutions that correspond to different right-hand sides typically converge at different rates. Deflating the converged system from the block system, i.e., modifying the block size, or avoiding the need for deflation through algorithmic changes have been addressed in various ways (e.g., see [45, 44, 23, 10, 18]). For the B-LGMRES method, we find that deflation is not required. Because we are not solving a block linear system, the problem of varying convergence rates for multiple solutions does not apply. However, we now discuss the possibility of rank deficiency for the initial residual block.

Consider the initial block residual for restart cycle i: $R_i = [r_i,\ z_i,\ \ldots,\ z_{i-k+1}]$. The initial residual block is rank deficient only if any of the vectors (r_i or z_j, $j = i : (i-k+1)$) are linearly dependent. Recall from the previous section that random vectors are used in place of unavailable error approximation vectors in cycles $i < k$. Therefore, all z_j are random for the initial cycle. Statistically these random vectors will be linearly independent of the other vectors in the initial residual block (cf. [30]).

For simplicity, consider the $k = 1$ case for restart cycle $i > 1$. The initial residual block is $R_i = [r_i,\ z_i]$. If r_i is the zero vector, then the solution has been found. If $z_i = 0$, stalling has occurred. Now assume that both r_i and z_i are not zero, and R_i is rank deficient. Then r_i and z_i are linearly dependent and $r_i \approx \alpha z_i$ for some $\alpha \neq 0$. Using the definitions in the proof of Theorem 1, we have that $z_i \in \mathcal{M}_i$. As in (11), the minimal residual property gives $r_i \perp A\mathcal{M}_i$, and, therefore,

$$\langle r_i,\ Az_i \rangle = 0,$$

where $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product. So for linearly dependent r_i and z_i we have

$$\alpha \langle z_i,\ Az_i \rangle = 0. \tag{14}$$

Because $z_i \neq 0$ implies stalling has not occurred, then $r_i \neq r_{i-1}$. By definition $r_i = r_{i-1} - Az_i$, and, as a result, $Az_i \neq 0$. Therefore since r_i, z_i, Az_i and α are all nonzero, (14) can only hold if A is not positive definite and $z_i \perp Az_i$.

As a result, the initial residual block is rank deficient if any of the following three cases is true: $r_i = 0$, $z_i = 0$, or A is not positive definite and $z_i \perp Az_i$. The first case is not an issue since $r_i = 0$ indicates that a solution has been found. In the second case, stalling has occurred, and including z_i in the initial residual block at the start of the cycle does not make sense. However, at worst $z_i \approx 0$ in practice since this is a finite precision computation. Similarly, the third case is unlikely, particularly in finite precision. In fact, we have not had any breakdowns due to rank deficiency in R_i in practice.

4. Numerical Experiments. In this section, we present promising experimental results from our implementation of the B-LGMRES method on problems from a variety of application areas. First, in Section 4.1, we compare our B-LGMRES implementation to the standard GMRES implementation available in PETSc 2.1.5 (Argonne National Laboratory's Portable, Extensible Toolkit for Scientific Computation) [5, 4]. We also compare the B-LGMRES method to the PETSc implementation of the LGMRES method described in [3]. In Section 4.2, we demonstrate the advantage of the matrix-multivector approach discussed in the previous section. We provide results from monitoring the movement of data through the memory hierarchy during the B-LGMRES solve to explain how performance gains occur.

We implemented the B-LGMRES algorithm in C using a locally modified version of PETSc 2.1.5. The PETSc 2.1.5 libraries contain the tools for storing a multivector in the interlaced format described in Section 2.3, referred to as *multi-component vectors* in the PETSc manual [4]. We use the PETSc matrix-vector multiply routine for multiplying a matrix by a multi-component vector in B-LGMRES. Additionally, we modified our local installation of PETSc to include multivector versions of the PETSc routines *VecDot*, *VecAXPY*, *VecMAXPY*, and *MatSolve*. VecDot and VecAXPY are the vector dot product and axpy routines, respectively, required in the modified Gram-Schmidt steps. VecMAXPY adds a scaled sum of vectors to a vector, which is used when solving the least-squares problem. MatSolve performs a forward and back solve for use with the ILU preconditioner. Our modifications to several of these routines are discussed in more detail in Section 4.2.

We obtained test problems from the University of Florida Sparse Matrix Collection [11], the Matrix Market Collection [43], and the PETSc test collection. For reference, all test problems used in this section are listed in Table 1. If a right-hand side was not provided, we generated a random right-hand side. For all problems, the initial guess was a zero vector. All results provided were run on a single processor of a 16-processor Sun Enterprise-6500 server with 16G RAM. This system contains 400 Mhz Sun Ultra II processors, each with a 16 Kbyte L1 cache and a 4 Mbyte L2 cache. It is running Solaris 5.9 with the Forte Developer 7 version 5.4 C complier.

TABLE 1
List of test problems together with the matrix order (n), number of nonzeros (nnz), preconditioner, and a description of the application area (if known).

	Problem	n	nnz	Preconditioner	Application Area
1	pesa	11738	79566	none	
2	epb1	14734	95053	none	heat exchanger simulation
3	memplus	17758	126150	none	digital circuit simulation
4	zhao2	33861	166453	none	electromagnetic systems
5	epb2	25288	175027	none	heat exchanger simulation
6	ohsumi	8140	1456140	none	
7	aft01	8202	125567	ILU(0)	acoustic radiation, FEM
8	memplus	17758	126150	ILU(0)	digital circuit simulation
9	arco5	35388	154166	ILU(0)	multiphase flow: oil reservoir
10	arco3	38194	241066	ILU(1)	multiphase flow: oil reservoir
11	bcircuit	68902	375558	ILUTP(.01, 5, 10)	digital circuit simulation
12	garon2	13535	390607	ILUTP(.01, 1, 10)	fluid flow, 2-D FEM
13	ex40	7740	458012	ILU(0)	3-D fluid flow (die swell problem)
14	epb3	84617	463625	ILU(1)	heat exchanger simulation
15	e40r3000	17281	553956	ILU(2)	2-D fluid flow in a driven cavity
16	scircuit	170998	958936	ILUTP(.01, .5, 10)	digital circuit simulation
17	venkat50	62424	1717792	ILU(0)	2-D fluid flow

4.1. Comparison with other methods. We demonstrate the potential of the B-LGMRES method both with and without preconditioning by presenting experimental results for multiple test problems. We first compare the performance of B-LGMRES(m, k) to that of restarted methods GMRES(m) and LGMRES(m, k). We then examine the correlation between the time to solution and number of matrix accesses for these restarted methods. Finally we compare the time to solution of B-LGMRES to that of full (non-restarted) GMRES. Because the focus of this work is the solution of linear systems with single right-hand sides, we do not compare B-LGMRES performance to that of BGMRES or any other method designed to solve a system with multiple right-hand sides.

For each problem we report wall clock time for the linear solve only; we do not time any I/O or the setup of the preconditioner. For the preconditioned problems, we use the either the ILU(p) preconditioner, where p indicates the level of fill, or the ILUTP(*droptol*, *permtol*, $lfil$) preconditioner, where *droptol* indicates the drop tolerance, *permtol* indicates the column pivot tolerance, and $lfil$ is the fill-in parameter (e.g., see [49]). The timings reported are averages from five runs and have standard deviations of at most two percent. All tests are run until the relative residual norm is less than the convergence tolerance $\zeta = 10^{-9}$, i.e., when $\|r_i\|_2/\|r_0\|_2 \leq \zeta$. However, if a method does not converge in 1000 restart cycles, then the execution time reported reflects the time for 1000 cycles and we say that the method does not converge.

We evaluate the performance of B-LGMRES(m, k) for a particular problem by comparing its time to converge to that of GMRES(m) and LGMRES(m, k) with equal-sized approximation spaces. For reference, in Table 2, we list the approximation space size, the number of vectors of length n stored, and the number of matrix accesses per restart cycle in terms of parameters m and k for each method. Values of m and k may be different for each method. For GMRES(m), we chose restart parameter $m = 30$ because it is a common choice for GMRES(m) and is the default in PETSc. Therefore for this comparison, we required that the approximation spaces for both LGMRES(m, k) and B-LGMRES(m, k) be of size 30 as well. Furthermore, we wanted to evaluate the performance of LGMRES(m, k) and B-LGMRES(m, k) using the same number of error approximation vectors, i.e., the same k, for each. In choosing a value of k, we note that for LGMRES(m, k), $k \leq 3$ is typically optimal, and variations in algorithm performance are small for $k \leq 3$ (see [3]). Therefore, the performance of B-LGMRES(m, k) determined our choice of $k = 1$, which we justify in the next paragraph. The choice of $k = 1$ together with the constraint of a size 30 approximation space determined parameter m for each method. The three algorithms for which we report the time required for convergence are given in Table 3 for clarification. Notice that m and k were chosen for each method such that the three have equal approximation space sizes and similar storage requirements for each restart cycle.

As shown in Table 3, the approximation space for B-LGMRES(15, 1) at restart cycle $i + 1$ is given by $\mathcal{K}_{15}(A, r_i) + \mathcal{K}_{15}(A, z_i)$. The B-LGMRES method with $k = 2$ and an approximation space of size 30 is denoted by B-LGMRES(10, 2). The B-LGMRES(10, 2) approximation space at restart cycle $i + 1$ is given by $\mathcal{K}_{10}(A, r_i) + \mathcal{K}_{10}(A, z_i) + \mathcal{K}_{10}(A, z_{i-1})$. Therefore, only 10 vectors form the traditional Krylov space $\mathcal{K}_{10}(A, r_i)$ associated with the current residual. When keeping the total approximation space constant at size 30, using $k > 2$ decreases the dimension of the standard Krylov space portion even further. For this set of test problems, preliminary testing showed that using $k > 2$ was typically not beneficial. In fact, we found that for these test problems with an approximation space of size 30, B-LGMRES(m, k) with $k = 1$,

or B-LGMRES(15, 1), is optimal. B-LGMRES(10, 2) converges in less time than B-LGMRES(15, 1) for only one problem. For several test problems, convergence was similar, but in general B-LGMRES(15, 1) is the clear winner. For an approximation space larger than 30, we expect that using $k > 1$ would be an advantage more often. However, note that, with increasing k, the amount of modified Gram-Schmidt work per access of A increases.

TABLE 2
Algorithm specifications per restart cycle.

Method	Approx. Space Size	Length n Vector Storage	Accesses of A
GMRES(m)	m	$m+3$	m
LGMRES(m, k)	$m+k$	$m+3k+3$	m
B-LGMRES(m, k)	$m(k+1)$	$(m+2)k+m+3$	m

TABLE 3
Specifications for one restart cycle of the three algorithms used in subsequent numerical tests.

Method	Approx. Space Size	Length n Vector Storage	Accesses of A	Approx. Space at Restart Cycle $i+1$
GMRES(30)	30	33	30	$\mathcal{K}_{30}(A, r_i)$
LGMRES(29, 1)	30	35	29	$\mathcal{K}_{29}(A, r_i) + z_i$
B-LGMRES(15, 1)	30	35	15	$\mathcal{K}_{15}(A, r_i) + \mathcal{K}_{15}(A, z_i)$

We first present results for the non-preconditioned test problems, problems 1-6 in Table 1. Figure 4 compares the time required for convergence for B-LGMRES(15, 1) to both LGMRES(29, 1) and GMRES(30). The x-axis corresponds to the numbered test problems given in Table 1. These six non-preconditioned problems are listed in order of increasing number of nonzeros. The y-axis is the time required for convergence for either LGMRES(29, 1) or GMRES(30) (the left and right bars, respectively) divided by the time required for convergence for B-LGMRES(15, 1). Therefore, bars extending above one indicate faster convergence for B-LGMRES(15, 1). For all of these problems, B-LGMRES(15, 1) converges. However, arrows above the bars in Figure 4 indicate that GMRES(30) did not converge in 1000 restart cycles for problem 1 and both LGMRES(29, 1) and GMRES(30) did not converge for problem 4. As previously mentioned, the time for 1000 restart cycles is reported for methods that do not converge, resulting in an understated ratio of improvement for B-LGMRES(15, 1) for the bars with arrows. B-LGMRES(15, 1) converges in less time than GMRES(30) for all problems. However, improvements of B-LGMRES(15, 1) over LGMRES(29, 1) are more modest since LGMRES is typically also an improvement over GMRES(m) [3].

We now examine the performance of B-LGMRES(m, k) on preconditioned problems, problems 7-17 in Table 1. We use ILU preconditioning because it is a common choice. We use left preconditioning which minimizes the preconditioned residual norm ($\|M^{-1}r\|_2$), and, therefore, the determination of convergence is based on this preconditioned residual norm. A comparison of the time required for convergence for B-LGMRES(15, 1) to both LGMRES(29, 1) and GMRES(30) is given in Figure 5. As in Figure 4, the x-axis corresponds to the numbered test problems given in Table 1, and the y-axis indicates the ratio of the time required for convergence for either LGMRES(29, 1) or GMRES(30) to that of B-LGMRES(15, 1). Bars extending above

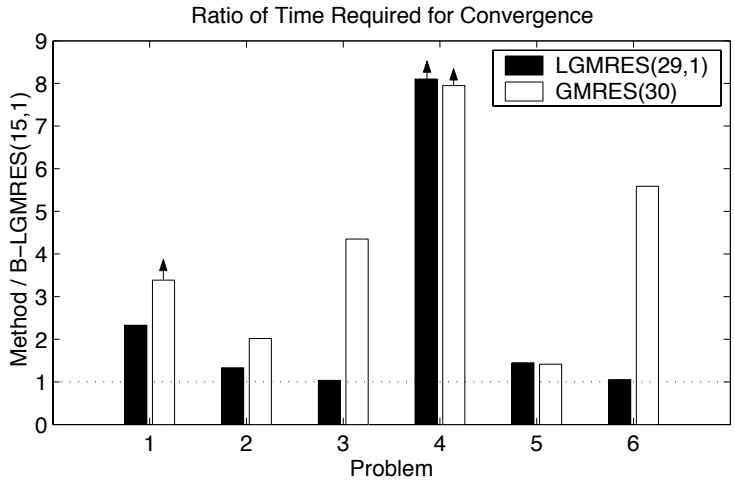

FIG. 4. *A comparison of the time required for convergence for non-preconditioned test problems 1-6 with GMRES(*30*) and LGMRES(*29, 1*) versus B-LGMRES(*15, 1*). All methods use an approximation space of size* 30*, and problems are listed in order of increasing number of nonzeros.*

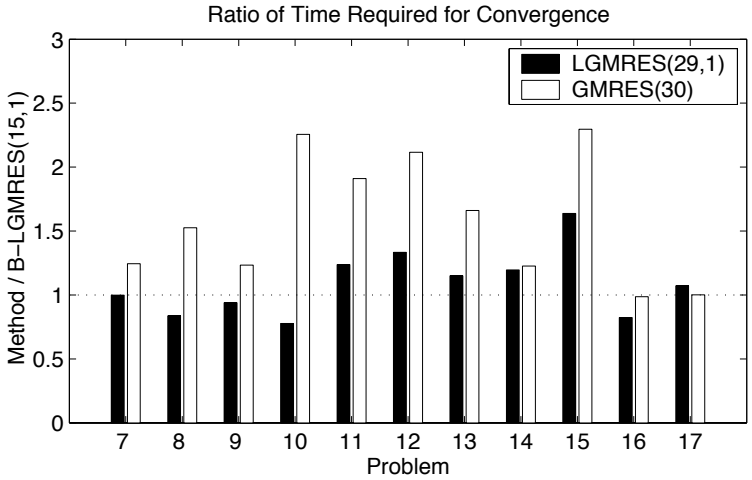

FIG. 5. *A comparison of the time required for convergence for preconditioned test problems 7-17 with GMRES(*30*) and LGMRES(*29, 1*) versus B-LGMRES(*15, 1*). All methods use an approximation space of size* 30*, and problems are listed in order of increasing number of nonzeros.*

one favor B-LGMRES(15, 1).

Figure 5 shows that the time required for convergence for B-LGMRES(15, 1) is less than that for GMRES(30) for problems 7-15 and about the same as GMRES(30) for problems 16 and 17. However, because iteration counts are generally lower with preconditioning, performance gains of B-LGMRES(15, 1) over GMRES(30) with preconditioning are not as dramatic as those without preconditioning. For large problems, though, even a small improvement in iteration count translates into a non-trivial time savings. The comparison of B-LGMRES(15, 1) to LGMRES(29, 1) is not as straightforward. The LGMRES method is quite effective for these test problems, and, as a result, LGMRES(29, 1) converges in less time than B-LGMRES(15, 1) for four

problems. Predicting which algorithm will "win" for a particular test problem is an open question. As an example, we tested problem memplus both without and with preconditioning, problems 3 and 8, respectively. For this problem, B-LGMRES(15, 1) converges in slightly less time than LGMRES(29, 1) when no preconditioner was used, but LGMRES(29, 1) is faster with preconditioning. In addition, it is well-known that GMRES convergence behavior can depend on the right-hand side vector. Therefore, we compared the results of B-LGMRES(15, 1), LGMRES(29, 1), and GMRES(30) using four alternate right-hand sides for each test problem in Table 1. The qualitative results from the additional four runs for each problem were the same as those shown in Figures 4 and 5 with three exceptions: B-LGMRES(15, 1) converged in less time for problems 9 and 16, and LGMRES(29, 1) converged in less time for problem 15.

TABLE 4
Matrix garon2, with $n = 13535$, $nnz = 390607$, and ILUTP(.01, 1, 10) preconditioning. Times are in seconds and include mean and standard deviations of times for five runs.

Method	Matrix-vector Multiplies	Matrix Accesses	Execution Time
GMRES(30)	1960	1960	165.9 ± .6
LGMRES(29,1)	1292	1292	104.6 ± .6
B-LGMRES(15, 1)	2008	1004	78.5 ± .2

For all of the test problems in Table 1, we found that the time to solution of the restarted methods generally correlates well with the number of accesses of A, as opposed to the number of matrix-vector multiplies. For example, in Table 4 we give the timing results for medium-sized problem garon2 (problem 12) where this correlation is easily seen. A similar comparison for all 17 test problems is given in Figure 6. In the top panel, the y-axis indicates the ratio of time required for convergence to matrix accesses for the three algorithms: GMRES (30), LGMRES(29, 1), and B-LGMRES(15, 1). The x-axis of each plot corresponds to the numbered test problems as usual. In the bottom panel, the y-axis indicates the ratio of the time required for convergence to matrix-vector multiplies. The near constant ratio for all three methods for each problem in the top panel shows that the number accesses of A largely determines execution time. However, the ratios in the bottom panel are not constant for each problem, indicating that the number of matrix-vector multiplies does not correlate well with execution time for B-LGMRES(15, 1). In the next section, the relative importance of individual sections of the algorithm, including preconditioning and modified Gram-Schmidt orthogonalization, are discussed in detail.

Because our machine has sufficient resources, full GMRES is a viable option. Therefore, we compare the time to solution of B-LGMRES(15, 1) to full GMRES for each of the 17 test problems. In Figure 7, the y-axis indicates the log of the time required for convergence for GMRES divided by the time for B-LGMRES(15, 1). Bars extending above the x-axis favor B-LGMRES(15, 1), and bars extending below favor GMRES. B-LGMRES converges in less time than GMRES on all but four problems: problems 7, 10, 13, and 15. There is no correlation between problem size and method in terms of time required for convergence for these test problems.

The experimental results presented in this section demonstrate that solving a single right-hand side system via a block system can be an effective means of improving the performance of an iterative method for sparse linear systems such as GMRES(m). For the problems presented here, the time to solution of B-LGMRES(m, k) is typically an improvement over that of GMRES(m), as shown in Figures 4 and 5, and is often

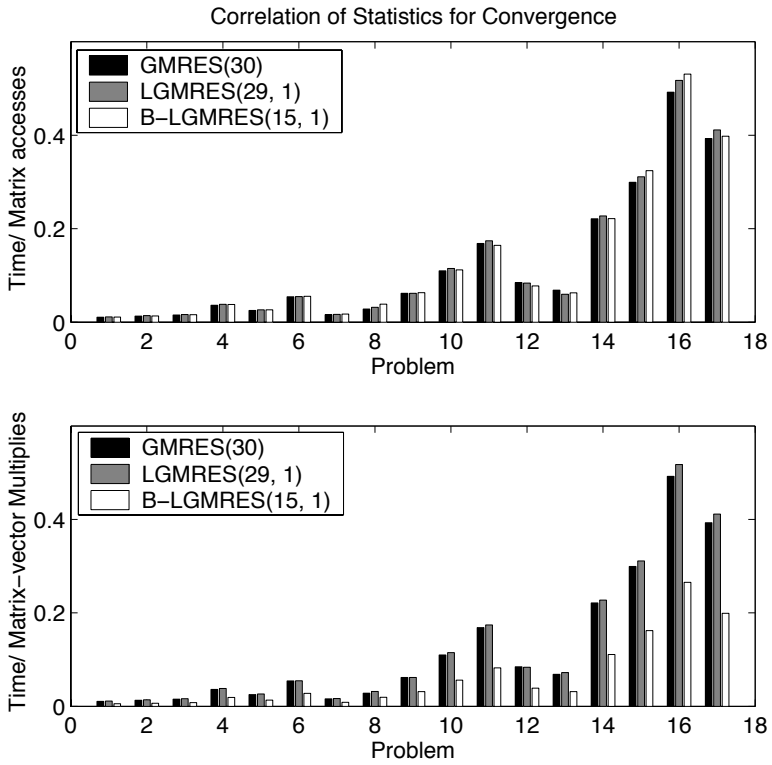

FIG. 6. *The upper panel shows the ratios of time required for convergence to number of accesses of A for GMRES (*30*), LGMRES(*29, 1*), and B-LGMRES(*15, 1*) on the 17 test problems. The lower panel shows the ratios of time required for convergence to number of matrix-vector multiplies.*

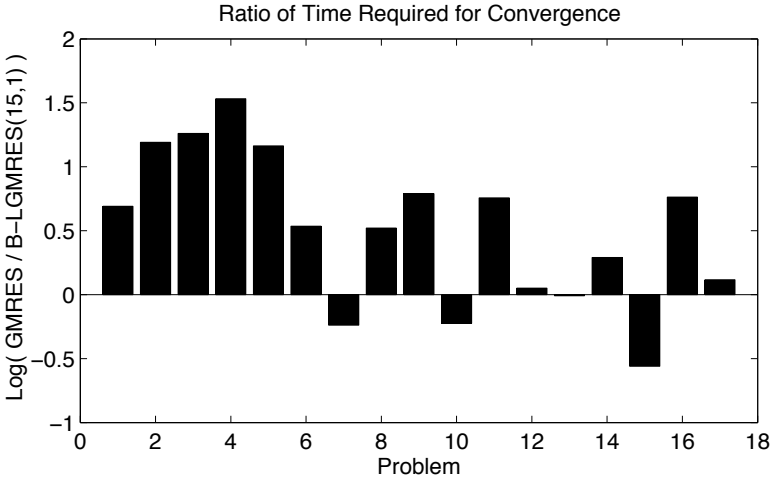

FIG. 7. *A comparison of the time required for convergence with B-LGMRES(*15, 1*) versus full GMRES on the 17 test problems.*

faster than full GMRES as well (Figure 7). The additional right-hand side vector(s) in B-LGMRES inhibit the tendency of restarted GMRES to form similar polynomials at every other restart cycle (see [3]), thus improving convergence. As with the LGMRES method, B-LGMRES acts as an accelerator, but in general does not help with stalling. In other words, B-LGMRES(m, k) has a total approximation space size of $m \cdot (k+1)$ and typically does not help problems that stall for GMRES($m \cdot (k+1)$), though we have found a few exceptions. For problems that stall, full GMRES or Morgan's GMRES-E method [39] can be good options. Furthermore, B-LGMRES is not particularly helpful when GMRES(m) converges in a small number of iterations. All of the test problems in this section required at least ten restart cycles of GMRES(30) to converge.

4.2. Relationship between data movement and execution time. Reformulating an iterative linear solver algorithm to use matrices in block operations requires a balance between maintaining or improving the algorithm's numerical properties and reducing data movement. The reduction in execution time of B-LGMRES(m, k) over GMRES(m) shown in the previous section indicates that such a balance is possible. In this section, we further quantify how B-LGMRES performance gains occur from the reduction of data movement. We show that using the multivector implementation described at the end of Section 2.3 is of key importance. We also show that a reduction in execution time for our test problems in Table 1 correlates more strongly to a reduction in data movement between levels of cache than to a reduction between main memory and cache.

Recall from Section 2.3 that the interlaced storage scheme for multivectors places corresponding elements of its constituent vectors in the same cache line. As a result, the matrix multiply routine, for example, uses a higher fraction of elements from each cache line for each access of a nonzero element of the coefficient matrix A. Therefore, the number of floating-point operations performed per byte of data read from memory is increased. The advantages of interlacing of data items in general are explained in detail in [26]. To demonstrate the benefit of the multivector optimization specifically, we implemented B-LGMRES in PETSc 2.1.5 both with and without multivectors. We refer to our implementation of B-LGMRES with multivectors as the *MV* implementation. This implementation was described at the beginning of Section 4 and produced the results given in Section 4.1. The B-LGMRES implementation without multivectors, referred to as *non-MV*, represents the best non-multivector implementation possible with the tools available in PETSc. Both implementations were written so as to eliminate any copy of data from one data structure to another and represent a best coding effort.

Table 5 shows the impact of the multivector optimization on execution time by comparing the MV and non-MV implementations of B-LGMRES for ten restart cycles. The problems in Table 5 are a subset of the test problems in Table 1 with a range of number of nonzeros (nnz). The percentage improvement of the MV over the non-MV implementation is given in the right-most column of Table 5, and, as expected, the MV implementation has the lowest execution time for each problem. In fact, the MV implementation, which has multivectors of size $s = 2$, is about twice as fast as the non-MV implementation for B-LGMRES(15, 1), regardless of the value of nnz. For the remainder of this section, we detail how the multivector optimization impacts various sections of the code, and we explore the relationship between data movement through the memory hierarchy and execution time.

The multivector optimization impacts more than just the matrix multiply in line 5 but rather pervades the entire algorithm. For example, because U and V in the B-

TABLE 5
A comparison of execution times (in seconds) for the MV and non-MV implementations of B-LGMRES(15, 1) for ten restart cycles. The matrix order (n), number of nonzeros (nnz), and percentage improvement of the MV over the non-MV implementation are also listed.

	Problem	n	nnz	Execution Time MV	Execution Time non-MV	Relative improvement
1	pesa	11738	79566	1.7	3.5	51%
3	memplus	17758	126150	2.5	5.1	51%
10	arco3	38194	241066	17.2	30.5	44%
11	bcircuit	68902	375558	26.1	48.4	46%
13	ex40	7740	458012	9.0	20.4	51%
14	epb3	84617	463625	32.5	63.5	49%
16	scircuit	170998	958936	80.7	151.6	47%
17	venkat50	62424	1717792	61.6	113.6	46%

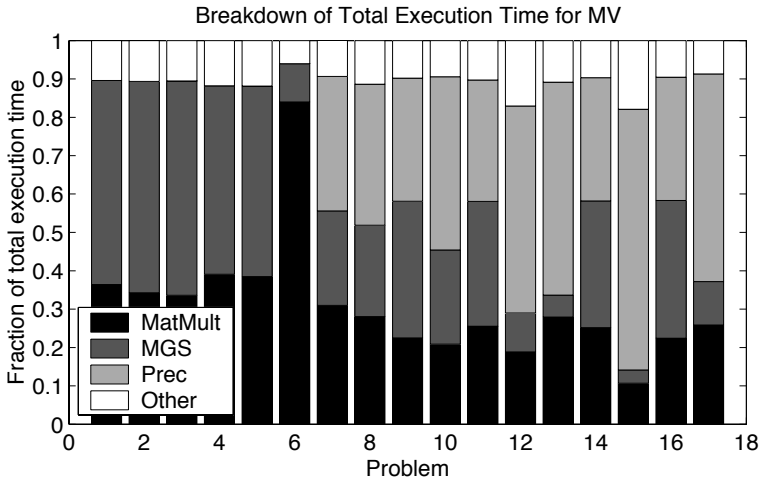

FIG. 8. *Percentage of time for each section of code of the MV implementation of B-LGMRES(15,1) for the 17 test problems.*

LGMRES algorithm given in Figure 3 are stored as multivectors, the orthogonalization routines in lines 6 - 9 require modification as well. In fact, three primary sections of the B-LGMRES code are impacted by the multivector optimization: the matrix-vector multiply (MatMult), the modified Gram-Schmidt orthogonalization (MGS), and the application of the preconditioner (Prec), if required. Because the Prec section of code shows similar characteristics to the MatMult section, we only discuss the MatMult and MGS sections of code. For reference, Figure 8 gives the percentage of time spent in each of the three primary sections for the MV implementation of the B-LGMRES algorithm. The Other category represents the difference between the total time and the sum of times for the three sections shown. For our 17 test problems, execution time is not consistently dominated by a single section of the B-LGMRES code.

The MatMult section of the code is the matrix-vector multiply in line 5 of Figure 3. For the non-MV implementation, successive calls are made to a matrix-vector multiply routine for each individual vector in V_j. In contrast, the MV implementation utilizes a single call to the PETSc matrix-*multivector* multiply routine, as described in the beginning of Section 4. The matrix-multivector multiply groups computations on the same data, which allows more floating-point operations per byte of data moved

through the memory hierarchy. In fact, the use of the matrix-multivector multiply has the potential to reduce the amount of data moved through parts memory subsystem by a factor of s, where s is the number of vectors in the multivector. Following the experimental results in the previous section, we concentrate on a multivector of size $s = 2$ which corresponds to B-LGMRES(m, k) with $k = 1$.

The size of the data structures determines the part of the memory most affected by the multivector optimizations for the MatMult section of code. In particular, we denote the size of the data structures accessed by the MatMult section by $WS_{MatMult}$ which is the *working set size*. $WS_{MatMult}$ is approximately equal to the storage required for matrix A. The matrix, which is stored in AIJ format, requires an array of double precision floating point values of length nnz, and two arrays of integer values of length nnz and n which represent the column and row pointers, respectively. Note that PETSc AIJ matrix storage format is equivalent to compressed sparse row storage (e.g., see [49]). Therefore, the size of the matrix A is calculated by

$$sizeof(A) = sizeof(double) * nnz + sizeof(int) * (n + nnz), \tag{15}$$

where *double* is a double precision value, *int* is an integer value, and function $sizeof()$ returns the size of its argument in bytes. If $WS_{MatMult} \approx sizeof(A)$ is significantly larger than the L2 cache size, then successive calls to a matrix-vector multiply routine repeatedly read matrix A from main memory through both levels of cache. For $WS_{MatMult}$ smaller than the L2 cache size but larger than L1, portions of matrix A may remain in L2 cache and allow some L2 cache reuse for the successive calls to matrix-vector multiply in the non-MV implementation.

As seen in Figure 8, the MGS section of code (lines 6-10 in Figure 3) often contributes significantly to the overall cost of the B-LGMRES algorithm. In fact, this section of code consumes more than 50% of the time to solution for several of the test problems. This section of code required the creation of multivector versions of the PETSc routines *VecDot* and *VecAXPY*. Following the PETSc use of *Stride* to denote multivector versions of common functions, we refer to the new versions of these routines as *VecStrideDot* and *VecStrideAXPY*, respectively. These routines represent the majority of the total time for the MGS section. Therefore, we wrote the routines in a manner that limits movement of data by fusing together computations on related data. For example, the functionality provided by VecStrideDot for a multivector of size s is equivalent to s^2 successive calls to VecDot (the approach used in the non-MV implementation). With successive calls to VecDot, $2 \cdot n \cdot s^2$ data values must be read from the memory hierarchy. VecStrideDot reduces the number of data values read to $2 \cdot n \cdot s$. Therefore if $WS_{MGS} \equiv sizeof(double) \cdot 2 \cdot n \cdot s$ is greater than either the L1 or L2 cache size, use of the multivectors impacts data movement in the MGS section of code.

Both VecStrideDot and VecStrideAXPY were written using loop temporaries and loop unrolling to aid compiler optimization. The use of loop temporaries allows a compiler to identify data reuse more easily at the register level. Loop unrolling further helps register reuse and allows different iterations of the loop to occur simultaneously. In Figure 9, we illustrate these optimization techniques applied to the inner loop of VecStrideAXPY. Recall that each multivector consists of two vectors of length n. Version A in Figure 9 is a 'naive' implementation of the loop (without loop temporaries and unrolling). In Version B, we use loop temporaries, which means that references to the *alpha* and x arrays are replaced with scalars. Version C incorporates loop unrolling: the inner loop in Version B is unrolled by a factor of two. Version D is not

```
/*------------ version A -----------------------------*/
stride=2; m=stride*n;
for (i=0; i<m; i+=stride) {
        y[i]   = y[i]   + alpha[0]*x[i] + alpha[1]*x[i+1];
        y[i+1] = y[i+1] + alpha[2]*x[i] + alpha[3]*x[i+1];}

/*------------ version B -----------------------------*/
stride=2; m=stride*n;
a0=alpha[0]; a1=alpha[1]; a2=alpha[2]; a3=alpha[3];
for (i=0; i<m; i+=stride) {
        x0=x[i]; x1=x[i+1];
        y[i]   = y[i]   + a0*x0 + a1*x1;
        y[i+1] = y[i+1] + a2*x0 + a3*x1;}

/*------------- version C -----------------------------*/
stride=2; m=stride*n;
unroll=2; step=unroll*stride; mm=m/step; rem=m%stride
a0=alpha[0]; a1=alpha[1]; a2=alpha[2]; a3=alpha[3];
for (i=0; i<(mm*step); i+=step) {
        x0=x[i]; x1=x[i+1]; x2=x[i+2]; x3=x[i+3];
        y[i]   = y[i]   + a0*x0 + a1*x1;
        y[i+1] = y[i+1] + a2*x0 + a3*x1;
        y[i+2] = y[i+2] + a0*x2 + a1*x3;
        y[i+3] = y[i+3] + a2*x2 + a3*x3;}
if (rem)
   for (i=mm*step; i<m; i+=stride) {
        y[i]   = y[i]   + a0*x[i] + a1*x[i+1];
        y[i+1] = y[i+1] + a2*x[i] + a3*x[i+1];}

/*------------- version D -----------------------------*/
/*      Unrolled version of C where unroll=4           */
```

FIG. 9. *Code to perform a multivector AXPY operation. Successive versions add additional optimization techniques.*

shown in Figure 9 but is the version B loop unrolled by four. In Table 6, the timings to perform each version of the loops in Figure 9 as well as the functional equivalent for the non-MV implementation are given in μsec for a subset of the test problems with a range of matrix orders. Note how successive optimizations either have no impact or reduce the execution time in each case. The combination of optimizations in version D of the loop results in a 40% reduction on average in execution time for the VecStrideAXPY routine over its non-MV equivalent. Therefore, loop version D was used in all subsequent timings in this section. Because the MGS section can consume a large percentage of total execution time, simple optimizations such as those in *VecStrideAXPY* have as significant an impact on overall execution time of the solver as the use of the matrix-multivector multiply routine.

Having described the multivector modifications to the MatMult and MGS sections, we now determine the effects of these changes by comparing the execution times for both sections of code in the non-MV and MV implementations. In Figure 10, the y-axis indicates the execution time for the non-MV implementation divided by the execution time for the MV implementation for both the MatMult and MGS sections of code. The x-axis contains the 17 test problems. A value greater than one indicates that the execution time for MV is less than that for non-MV. The MV implementation reduces the execution time of the MatMult section by a factor 1.4 to 2.7 over the non-MV implementation. The least improvement in execution time for the MatMult section occurs for problems 4 and 9. These problems have neither the

TABLE 6
Execution times in μsec for a single call to VecStrideAXPY for the non-MV implementation and MV implementations with loop versions A - D in Figure 9. Relative improvement of version D versus non-MV is also listed. Problems are listed in increasing order of matrix size (n).

			VecStrideAXPY time (μsec)					Relative
	Problem	n	non-MV	A	B	C	D	Improvement
13	ex40	7740	1.8	1.3	1.3	1.2	1.1	39%
3	memplus	17758	4.4	2.9	2.8	2.7	2.5	43%
11	bcircuit	68902	17.2	11.3	10.3	10.3	9.4	45%
14	epb3	84617	20.4	14.8	12.7	12.7	12.7	38%

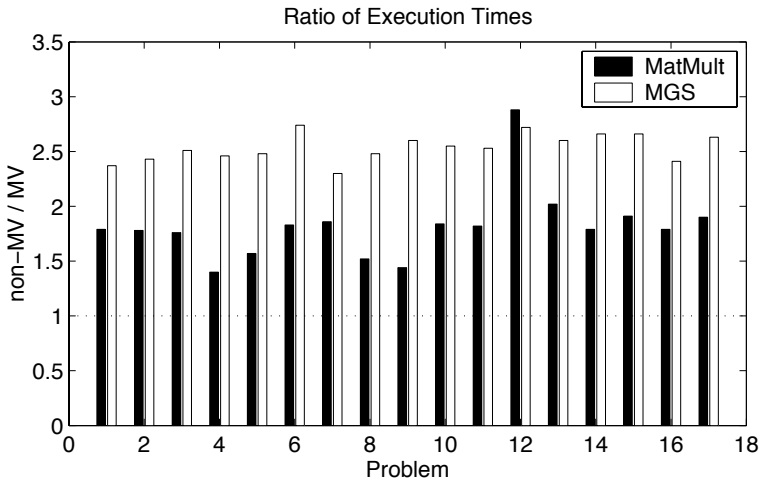

FIG. 10. *A comparison of execution time for the MatMult and MGS sections with the non-MV implementation of B-LGMRES(15,1) versus the MV implementation for ten restart cycles for the 17 test problems.*

largest nor smallest n or nnz, but they do have the lowest average number of nonzeros per row: 4.9 and 4.3, respectively. However, problems 6 and 13 have the highest average number of nonzeros per row at 178.9 and 59.2, respectively, and do not show the most improvement in execution time. It is therefore unclear what impact matrix density (or even nonzero structure) has on the effectiveness of the multivector optimizations in general. The execution time for the MGS section shows an even greater improvement of MV over non-MV on average. In fact, the MV implementation of MGS reduces execution time by a factor 2.3 to 2.7 over the non-MV implementation.

For the remainder of this section, we explain reductions in execution time due to the multivector optimization with data from hardware performance counters that monitor data movement through the memory hierarchy. Our experimental platform described at the beginning of Section 4 allows direct access to hardware counters. We focus on two counters that measure the number of cache lines read from main memory to the L2 cache and the number of cache lines read from the L2 to the L1 cache since their measurements correlate strongly with execution time. The raw counts from the counters are accumulated and converted into Mbytes using a tool we wrote. We refer to the amount of data moved between main memory and L2 cache as $Mbytes_{L2}$, and between L2 and the L1 caches as $Mbytes_{L1}$.

To determine the specific source of performance gains for the MV implementation,

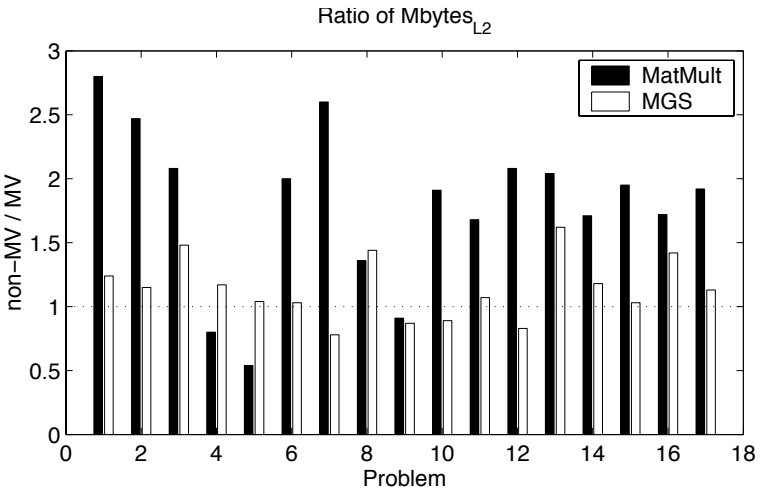

FIG. 11. *A comparison of data movement from main memory to L2 cache in the MatMult and MGS sections for the non-MV implementation of B-LGMRES(15,1) versus the MV implementation for ten restart cycles for the 17 test problems.*

we first examine data movement between main memory and the L2 cache for the non-MV and MV implementations. In Figure 11, the y-axis indicates the ratio of $Mbytes_{L2}$ for non-MV to MV for both the MatMult and MGS sections. Bars extending above one indicate that the non-MV implementation requires greater data movement than does the MV implementation. In general, the MV implementation has a smaller $Mbytes_{L2}$ than the non-MV implementation.

For the MatMult section, we expected a factor of two reduction in $Mbytes_{L2}$ for test problems with a $WS_{MatMult}$ significantly larger than the L2 cache. Using (15), we determined that eight test problems have $WS_{MatMult}$ larger than the 4 Mbytes L2 cache on our test system, and these problems (6 and 11-17) all have ratios from 1.75 to 2.0 in Figure 11. These ratios of reduction in $Mbytes_{L2}$ for the MV implementation correlate well with factor of two reductions in execution time for those problems seen in Figure 10. In other words, if a reduction in $Mbytes_{L2}$ is responsible for an improvement in execution time, then Figures 10 and 11 should show similar ratios for each problem.

For the MGS section of the algorithm, we expected $Mbytes_{L2}$ to be impacted only if the working set size, WS_{MGS}, is significantly greater than the L2 cache size. The four problems with the largest matrix order are 11, 14, 16 and 17, but only problem 16 has a WS_{MGS} (= 5.2) greater than the 4 Mbyte L2 cache size.

However, problem 16 shows a ratio of $Mbytes_{L2}$ for non-MV to MV of 1.4 which is not the largest ratio in Figure 11 and does not correlate with the improvement in execution time in Figure 10. In fact, other inconsistencies exist in Figure 11 with respect to Figure 10; several problems show an increase in $Mbytes_{L2}$ for the MGS and MatMult sections for the MV implementation even though that implementation is faster. These inconsistencies indicate that a reduction in $Mbytes_{L2}$ does not accurately predict a reduction in execution time for most of the test problems. We show subsequently that the multivector optimization impacts a different part of the memory hierarchy for those problems.

We now consider data movement between the L1 and L2 caches. Analogously to

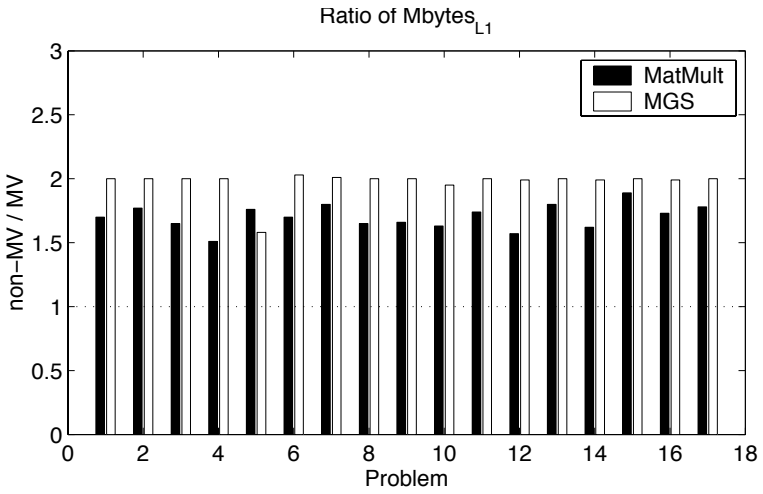

FIG. 12. *A comparison of data movement from L2 cache to L1 cache in the MatMult and MGS sections for the non-MV implementation of B-LGMRES(15,1) versus the MV implementation for ten restart cycles for the 17 test problems.*

the plot in Figure 11, Figure 12 shows the ratio of $Mbytes_{L1}$ for the non-MV to MV implementations for both the MatMult and MGS sections of code. For the MatMult section of code, we expected test problems with $WS_{MatMult} >> sizeof(L1\ \ cache)$ to show a reduction in $Mbytes_{L1}$. Test problem 1 has the smallest $WS_{MatMult}$ at 978 Kbytes, which is significantly larger than the 16 Kbyte L1 cache. Therefore, because all of the test problems have a $WS_{MatMult}$ larger than the L1 cache size, Figure 12 shows ratios of non-MV to MV that do in fact range from 1.5 to 1.9. Similarly for the MGS section of code, test problems with WS_{MGS} significantly greater than the size of the L1 cache should also show a reduction in $Mbytes_{L1}$. Test problem 12 has the smallest WS_{MGS} at 241 Kbytes, which is also significantly larger than the L1 cache. Consequently, all of the test problems have ratios of non-MV to MV that range from 1.6 to 2.0. Furthermore, the reduction in $Mbytes_{L1}$ is consistent with the reduction in execution time seen in Figure 10.

Now we compare reductions in *total* execution time and data movement for B-LGMRES. The top panel in Figure 13 clearly illustrates the correlation between reduction in execution time and *total* reduction in $Mbytes_{L1}$. The y-axis in the top panel in Figure 13 shows the execution time and $Mbytes_{L1}$ (the left and right bars, respectively) for the non-MV implementation divided by that for the MV implementation. The closer the two bars are in value for each problem, the stronger the correlation between the reduction in $Mbytes_{L1}$ and execution time for that problem. While the relationship between execution time and reduction in data movement between the L2 and L1 cache is noticeable in the top panel of Figure 13, a similar plot in the bottom panel of Figure 13 for $Mbytes_{L2}$ does not show such a correlation for most of the test problems. Therefore, we conclude that a reduction in $Mbytes_{L1}$ is a better predictor than $Mbytes_{L2}$ for the reduction in execution time achieved by the multivector optimizations for our test problems. For much larger test problems, where both $WS_{MatMult}$ and WS_{MGS} are larger than the L2 cache size, for example, we expect that the reduction $Mbytes_{L2}$ would more strongly correlate to execution time.

The results presented in this section demonstrate that reformulating an iterative

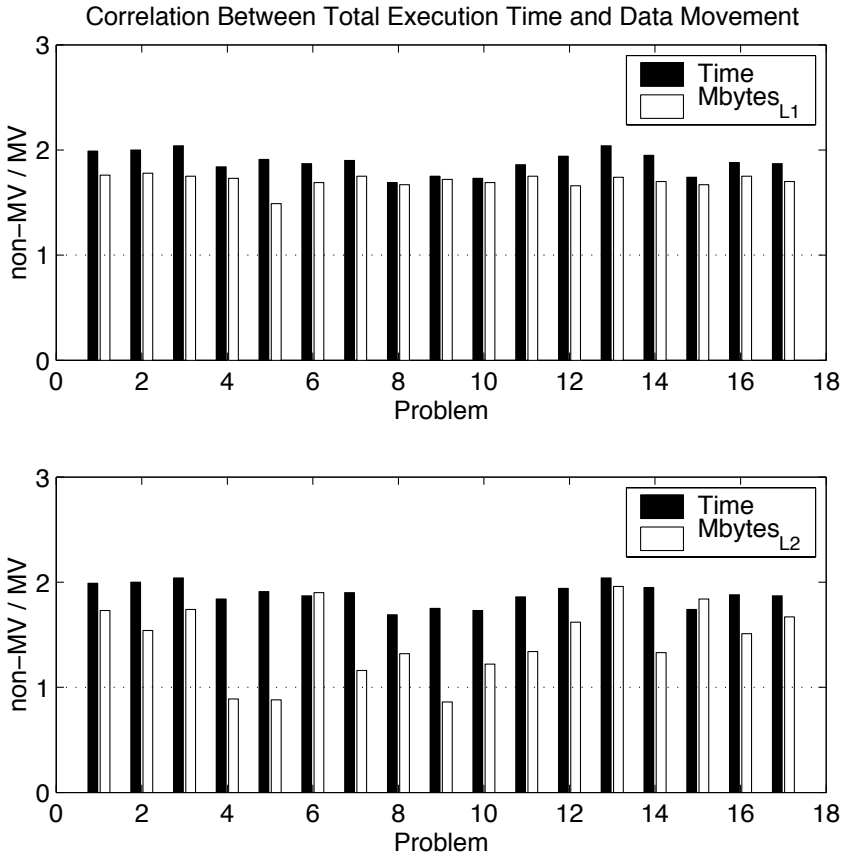

FIG. 13. *The upper panel is a comparison of data movement from L2 to L1 cache in the MatMult and MGS sections for the non-MV implementation of B-LGMRES(15,1) versus the MV implementation for ten restart cycles for the 17 test problems. The lower panel compares data movement from main memory to L2 cache.*

solver to use multivectors enables an efficient implementation that reduces data movement. An efficient implementation of a block version of an iterative method may be only marginally more expensive per access of coefficient matrix A than a non-block version. In other words, our results in this section combined with the results in Figure 6 support the hypothesis that the time to solution is largely dependent on the number of accesses of A from memory as opposed to the total number of matrix-vector multiplies.

5. Concluding remarks. In this paper, we explore the feasibility of modifying a common iterative linear solver method, restarted GMRES, to reduce data movement. Our investigation led to a block variant of the GMRES method that solves a linear system with a single right-hand side. This new method, B-LGMRES, lends itself to a more memory-efficient implementation. In particular, the B-LGMRES algorithm results from choosing an error approximation vector $z_i \equiv x_i - x_{i-1}$ as an additional right-hand side vector c_i, as explained in the beginning of Section 3. This choice mimics a truncated polynomial-preconditioned conjugate gradient method. However, we recognize that other right-hand side vectors may be more appropriate for particular

problem classes and could be implemented in the same manner as the B-LGMRES algorithm.

Although our experimental results show that the B-LGMRES method typically converges in less time than standard GMRES(m), a thorough understanding of the convergence behavior of B-LGMRES is an open question. In most cases, B-LGMRES is effective on problems that also benefit from the augmentation scheme of LGMRES. Though we do not recommend B-LGMRES as a substitute for preconditioning, it can be effective both with or without a preconditioner. We find that predicting the algorithm's performance is non-trivial due to its dependence on many factors: problem size (number of nonzeros and matrix order), restart parameter, block size, matrix properties, preconditioner choices, and machine characteristics.

A unique aspect of this study is the thorough investigation of data movement through the memory hierarchy during the execution of B-LGMRES. Typically the hardware performance of an iterative linear solver is not studied in this detail. With B-LGMRES, however, we evaluated the effectiveness of implementation decisions based on the measurement of data movement. We found that the use of interlaced data structures allows a significant reduction in data movement between different levels of cache. This reduction correlates well with the reduction in execution time, particularly for data movement between the L2 and L1 cache. In addition, the multivector optimization that benefits B-LGMRES could improve the performance of standard block GMRES and other block algorithms that solve systems with multiple right-hand sides.

Because of the gap between processor performance and memory access time, re-examining traditional matrix algorithms is important to achieving respectable performance on modern architectures. In this paper, we demonstrate the feasibility of improving performance for a standard algorithm via algorithmic changes together with an implementation that utilizes innovative programming techniques.

REFERENCES

[1] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *Achieving high sustained performance in an unstructured mesh CFD application*, in Proceedings of Supercomputing '99, 1999. Also published as Mathematics and Computer Science Division, Argonne National Laboratory, Technical Report ANL/MCS-P776-0899.

[2] S. F. Ashby, T. A. Manteuffel, and P. F. Saylor, *A taxonomy for conjugate gradient methods*, SIAM Journal on Numerical Analysis, 27 (1990), pp. 1542–1568.

[3] A. H. Baker, E. R. Jessup, and T. Manteuffel, *A technique for accelerating the convergence of restarted GMRES*, Tech. Report CU-CS-945-03, University of Colorado, Department of Computer Science, January 2003. Submitted for publication.

[4] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual*, Tech. Report ANL-95/11 - Revision 2.1.5, Mathematics and Computer Science Division, Argonne National Laboratory, 2003.

[5] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, L. C. McInnes, and B. F. Smith, *PETSc home page*. http://www.mcs.anl.gov/petsc, 2001.

[6] S. Behling, R. Bell, P. Farrell, H. Holthoff, F. O'Connell, and W. Weir, *The POWER4 Processor Introduction and Tuning Guide*, IBM Redbooks, November 2001.

[7] S. Carr and K. Kennedy, *Blocking linear algebra codes for memory hierarchies*, in Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, 1989, pp. 400–405.

[8] T. F. Chan and W. L. Wan, *Analysis of projection methods for solving linear systems with multiple righthand sides*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1698–1721.

[9] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numerical Linear Algebra with Applications, 4 (1997), pp. 43–66.

[10] H. Dai, *Block bidiagonalization methods for solving nonsymmetric linear systems with multiple right-hand sides*, Tech. Report TR/PA/98/35, CERFACS, Toulouse, France, 1998.
[11] T. Davis, *University of Florida sparse matrix collection, http://www.cise.ufl.edu/research/sparse/matrices*, 2002.
[12] J. W. Demmel, N. J. Higham, and R. S. Schreiber, *Block LU factorization*, Numerical Linear Algebra with Applications, 2 (1995), pp. 173–190.
[13] J. Dongarra, J. DuCroz, S. Hammarling, and I. Duff, *Algorithm 679: A set of level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 18–28.
[14] ———, *A set of level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.
[15] J. Dongarra, J. DuCroz, S. Hammarling, and R. J. Hanson, *An extended set of Fortran Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14 (1988), pp. 1–17.
[16] J. Dongarra and V. Eijkhout, *Self-adapting numerical software for next generation applications*, tech. report, LAPACK Working Note 157, ICU-UT-02-07, August 2002.
[17] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling, *Block reduction of matrices to condensed forms for eigenvalue computations*, Journal of Computational and Applied Mathematics, 27 (1989), pp. 215–227.
[18] A. A. Dubrulle, *Retooling the method of conjugate gradients*, Electronic Transactions on Numerical Analysis, 12 (2001), pp. 216–233.
[19] M. Eiermann, O. G. Ernst, and O. Schneider, *Analysis of acceleration strategies for restarted minimum residual methods*, Journal of Computational and Applied Mathematics, 123 (2000), pp. 261–292.
[20] C. Farhat, A. Macedo, and M. Lesoinne, *A two-level domain decomposition method for the iterative solution of high frequency exterior Helmholtz problems*, Numerische Mathematik, 85 (2000), pp. 283–308.
[21] M. Field, *Optimizing a parallel conjugate gradient solver*, SIAM Journal on Scientific Computing, 19 (1998), pp. 27–37.
[22] B. B. Fraguela, R. Doalla, and E. L. Zapata, *Cache misses prediction for high performance sparse algorithms*, in Proceedings of the Fourth International Euro-Par Conference (Euro-Par '98), 1998, pp. 224–233. Also published as University of Malaga, Department of Computer Architecture, Technical Report UNMA-DAC-98/22.
[23] R. W. Freund and M. Malhotra, *A block-QMR algorithm for non-Hermitian linear systems with multiple right-hand sides*, Linear Algebra and its Applications, 254 (1997), pp. 119–157.
[24] K. Gallivan, W. Jalby, U. Meier, and A. H. Sameh, *Impact of hierarchical memory systems on linear algebra algorithm design*, The International Journal of Supercomputing Applications, 2 (1988), pp. 12–48.
[25] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *Toward realistic performance bounds for implicit CFD codes*, in Proceedings of Parallel CFD'99, D. Keyes, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, eds., Elsevier, 1999, pp. 233–240.
[26] ———, *High-performance parallel implicit CFD*, Parallel Computing, 27 (2001), pp. 337–362.
[27] G. Gu and Z. Cao, *A block GMRES method augmented with eigenvectors*, Applied Mathematics and Computation, 121 (2001), pp. 278–289.
[28] J. Hennessey and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2nd ed., 1996.
[29] E.-J. Im and K. Yelick, *Model-based memory hierarchy optimizations for sparse matrices*, in Workshop on Profile and Feedback-Directed Compilation, 1998.
[30] E. R. Jessup and I. C. F. Ipsen, *Improving the accuracy of inverse iteration*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 550–571.
[31] A. H. Karp, *Bit reversal on uniprocessors*, SIAM Review, 38 (1996), pp. 1–26.
[32] R. E. Kessler, E. J. McLellan, and D. A. Webb, *The alpha 21264 microprocessor architecture*, in Proceedings of the 1998 IEEE International Conference on Computer Design, October 1998, pp. 90–95.
[33] M. S. Lam, E. E. Rothberg, and M. E. Wolf, *The cache performance and optimizations of blocked algorithms*, in Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, 1991.
[34] G. Li, *A block variant of the GMRES method on massively parallel processors*, Parallel Computing, 23 (1997), pp. 1005–1019.
[35] J. D. McCalpin, *Memory bandwidth and machine balance in current high performance computers*, IEEE Computer Society Technical Committee on Computer Architecture Newsletter,

(1995). http://www.cs.virginia.edu/stream.

[36] ———, *STREAM: Sustainable memory bandwidth in high performance computers.* http://www.cs.virginia.edu/stream, 2003.

[37] S. McKee and W. Wulf, *Access order and memory-conscious cache utilization*, in First Symposium on High Performance Computer Architecture (HPCA1), January 1995.

[38] R. Morgan, *GMRES with deflated restarting and multiple right-hand sides.* Presentation at the Seventh Copper Mountain Conference on Iterative Methods, March 2002.

[39] R. B. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 1154–1171.

[40] R. B. Morgan, *GMRES with deflated restarting*, SIAM Journal on Scientific Computing, 24 (2002), pp. 20–37.

[41] N. M. Nachitgal, L. Reichel, and L. N. Trefethen, *A hybrid GMRES algorithm for nonsymmetric linear systems*, SIAM Journal of Matrix Analysis and Applications, 13 (1992), pp. 796–825.

[42] S. Naffziger and G. Hammond, *The implementation of the next generation 64b Itanium microprocessor*, in Proceedings of the IEEE International Solid-State Circuits Conference, vol. 2, 2002, pp. 276–504.

[43] National Institute of Standards and Technology, Mathematical and Computational Sciences Division, *Matrix Market.* http://math.nist.gov/MatrixMarket, 2002.

[44] A. A. Nikishin and A. Y. Yeremin, *Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: general iterative scheme*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 1135–1153.

[45] D. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra and its Applications, 29 (1980), pp. 293–322.

[46] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yellick, *A case for intelligent RAM*, IEEE Micro, (1997), pp. 34–44.

[47] A. Pinar and M. T. Heath, *Improving performance of sparse matrix-vector multiplication*, in Proceedings of Supercomputing '99, November 1999.

[48] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM Journal on Scientific Computing, 14 (1993), pp. 461–469.

[49] ———, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1996.

[50] ———, *Analysis of augmented Krylov subspace methods*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 435–449.

[51] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[52] H. Simon and A. Yeremin, *A new approach to construction of efficient iterative schemes for massively parallel applications: variable block CG and BiCG methods and variable block Arnoldi procedure*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., vol. 1, SIAM, 1993, pp. 57–60.

[53] V. Simoncini and E. Gallopoulos, *An iterative method for nonsymmetric systems with multiple right-hand sides*, SIAM Journal on Scientific Computing, 16 (1995), pp. 917–933.

[54] ———, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra and its Applications, 247 (1996), pp. 97–119.

[55] C. F. Smith, A. F. Peterson, and R. Mittra, *A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields*, IEEE Transactions on Antennas and Propagation, 37 (1989), pp. 1490–1493.

[56] S. Toledo, *Improving the memory-system performance of sparse-matrix vector multiplication*, IBM Journal of Research and Development, 41 (1997), pp. 711–725.

[57] B. Vital, *Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multi-processeur*, PhD thesis, Université de Rennes I, Rennes, 1990.

[58] R. Vudoc, J. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee, *Performance optimizations and bounds for sparse matrix-vector multiply*, in Proceedings of Supercomputing '02, 2002.

[59] W. A. Wulf and S. A. McKee, *Hitting the wall: Implications of the obvious*, Tech. Report CS-94-48, University of Virginia, Department of Computer Science, 1994.