SIAM J. SCI. COMPUT.
Vol. 27, No. 5, pp. 1608–1626

ON IMPROVING LINEAR SOLVER PERFORMANCE: A BLOCK VARIANT OF GMRES*

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

Abstract. The increasing gap between processor performance and memory access time warrants the re-examination of data movement in iterative linear solver algorithms. For this reason, we explore and establish the feasibility of modifying a standard iterative linear solver algorithm in a manner that reduces the movement of data through memory. In particular, we present an alternative to the restarted GMRES algorithm for solving a single right-hand side linear system $Ax = b$ based on solving the block linear system $AX = B$. Algorithm performance, i.e., time to solution, is improved by using the matrix A in operations on groups of vectors. Experimental results demonstrate the importance of implementation choices on data movement as well as the effectiveness of the new method on a variety of problems from different application areas.

Key words. GMRES, block GMRES, iterative methods, Krylov subspace, memory access costs

AMS subject classifications. 65F10, 65Y20

DOI. 10.1137/040608088

1. Introduction.

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 (generalized minimum residual) [38] method is a common choice. Because linear systems are ubiquitous in science and engineering applications, improving the robustness of linear solvers continues to be of interest.

The two main costs associated with an iterative linear solver algorithm are the floating-point operations and the cost of moving data through memory, and performance may be improved by reducing either of these costs. Our focus is on achieving a balance between improving the efficiency of an iterative linear solver from a memory-usage standpoint while maintaining favorable numerical properties. In particular, the goal of this work is to demonstrate the feasibility of improving the performance

*Received by the editors May 12, 2004; accepted for publication (in revised form) January 11, 2005; published electronically February 3, 2006.

http://www.siam.org/journals/sisc/27-5/60808.html

†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808 L-551, Livermore, CA 94551 (abaker@llnl.gov). The work of this author was primarily supported by the Department of Energy Computational Science Graduate Fellowship Program of the Office of Scientific Computing and Office of Defense Programs in the Department of Energy under contract DE-FG02-97ER25308. Portions of this work were performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract W-7405-Eng-48.

‡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 National Science Foundation grant ACI-0072119.

of a common iterative linear solver (restarted GMRES), and likewise other iterative solvers, via a combination of a memory-efficient implementation and algorithmic modifications that reduce data movement.

In this paper, we report on our investigation into a 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 through parts of the memory hierarchy. In section 2 we discuss related background information. We outline our approach for improving restarted GMRES performance in section 3. Section 4 introduces our new block method. We discuss implications of implementation decisions on data movement and present computational results for a variety of problems in section 5. Finally, we give concluding remarks in section 6.

2. Background. 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 [19, 16, 23]). 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 [16, 45, 28, 26, 20, 1]). The discrepancy between DRAM (dynamic random access memory) access time and microprocessor speeds has increased by nearly 50% per year [35]. As a result, the percentage of overall application time spent waiting for data from main memory has increased [35]. The situation is compounded by advances in algorithm development that have decreased the total number of floating-point operations required by many algorithms [20]. 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 [20]. In fact, performance bounds based on sustainable memory bandwidth (such as the STREAM [27] benchmark) are thought to predict algorithm performance most accurately [20, 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 [23, 16, 19].

Krylov subspace methods such as GMRES are based on an iteration loop that accesses the coefficient matrix A once per loop and performs a sparse matrix-vector multiply. Thus, a desirable modification in terms of reducing data movement is to perform more than one matrix-vector product for a single memory access of A. This optimization is natural when solving several linear systems with the same coefficient matrix A but different right-hand sides. Such 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. By solving the block linear system (as opposed to solving s systems individually), A now operates on a group of vectors (*multivector*) instead of a single vector at each iteration. The matrix A is accessed from memory fewer times than it would be if each system were solved individually.

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 [34]. For nonsymmetric A, a block version of the GMRES algorithm (BGMRES) is first described in [43], and detailed descriptions can also be found in [39, 40, 37]. BGMRES is essentially identical to GMRES, except that operations are performed with multivectors instead of single vectors. As with GMRES, the resources required by BGMRES

may be impractical since storage and computational requirements increase with each iteration. In this case, a restarted version of BGMRES (BGMRES(m)) is commonly used in practice.

Several variants of the BGMRES 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 [39] that often solves the block system faster than solving each single system individually due in part to fewer memory accesses. In addition, block methods that augment the approximation space with eigenvector approximations are given in [22, 29]. These methods are particularly useful for matrices that are sufficiently close to normal and have a few particular eigenvalues that inhibit convergence.

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 in [34] for BCG 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.

Also of note is a block variant of GMRES for single right-hand sides systems in [25]. This method is mathematically equivalent to standard GMRES and is implemented as an s-step method (matrix A performs s consecutive matrix-vector multiplies at each iteration). The primary drawback is loss of accuracy with increasing block size (s). As with most blocked algorithms, the implementation allows for the use of level 3 BLAS (basic linear algebra subprograms) [14], and codes based on these routines often achieve good performance because they minimize data movement (e.g., see [13, 14, 20]).

3. Approach. Our approach to reducing data movement in restarted GMRES (GMRES(m)) 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 as in (1) based on solving a corresponding block linear system as in (2). This algorithmic change requires the selection of additional starting vectors and right-hand sides.

Algorithmic changes that reduce memory access costs should not 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 existing Krylov subspace method acceleration techniques.

It is well known that subspace information is lost in GMRES(m) due to restarting, and a number of acceleration methods attempt to in some sense compensate for this loss (e.g., see [30, 31, 42, 12, 9, 4]). Our block method derives from the LGMRES method [4], an augmented Krylov subspace method with appealing convergence properties. In particular, the extra right-hand sides used in the block method are the same as used to augment the standard GMRES approximation space in LGMRES.

In addition, beyond the algorithmic modifications required for a block formulation of GMRES(m), close attention to implementation techniques is essential for a memory-efficient, sparse linear solver. The compressed storage formats used for sparse matrices generally result in irregular patterns of memory referencing (e.g., see [18, 15]), and many approaches to improving the performance of the sparse matrix-vector multiply have been investigated (e.g., see [41, 36, 44, 19, 8, 23]). In our new

block method, we reduce the cost of the additional matrix-vector operations in each iteration loop via an efficient matrix-multivector multiply routine discussed in [20, 21] 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. We discuss this type of matrix-multivector multiply routine and associated routines in section 5.1.

4. A new block algorithm. In this section, we first briefly review the LGMRES method, of which the new algorithm is an extension. We then describe the new algorithm and its implementation and discuss additional considerations.

4.1. The LGMRES method. Consider GMRES(m) when solving (1). Restart parameter m denotes a fixed maximum dimension for the Krylov subspace. If convergence has not occurred at the end of m iterations, the algorithm restarts. 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 ($r_i = b - Ax_i$). The minimum residual property requires that GMRES(m) find $x_{i+1} \in x_i + K_m(A, r_i)$ such that $r_{i+1} \perp AK_m(A, r_i)$ (e.g., see [37]), where $K_i(A, r_i) \equiv span\{r_i, Ar_i, \ldots, A^{i-1}r_i\}$ denotes an i-dimensional Krylov subspace.

Let $\hat{x}$ be the true solution to (1). We denote the true error after the ith restart cycle by e_i, where $e_i \equiv \hat{x} - x_i$. Then the vector

$$z_i \equiv x_i - x_{i-1} \tag{3}$$

is an approximation to that error 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 the next approximation space $\mathcal{K}_m(A, r_i)$, since augmenting with the true error would solve the problem exactly (e.g., see [17]). 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 and discarded in the previous cycle. Therefore, the LGMRES method accelerates convergence by appending k vectors that approximate the error from previous restart cycles to the standard Krylov approximation space at the end of each restart cycle. In particular, after $i+1$ restart cycles, LGMRES(m, k) finds an approximate solution to (1) in the following way:

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

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

4.2. The new algorithm: B-LGMRES. The augmentation scheme used by LGMRES(m, k) is easily extended to a block method by using the k previous error approximations z_j to build additional Krylov subspaces. We refer to this block extension as B-LGMRES(m, k), for "block" LGMRES. Conceptually, we view one restart cycle (i) of B-LGMRES(m, k) as a cycle of block GMRES 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 initial guess at restart cycle i, X_i, is then written as $X_i = [x_i,\ 0,\ \ldots,\ 0]$, where the approximate 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 size $n \times (k+1)$ or $n \times s$, where $s \equiv k+1$ 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. Therefore, we write the B-LGMRES(m, k) approximation as

$$x_{i+1} = x_i + q_{i+1}^{m-1}(A)r_i + \sum_{j=i-k+1}^{i} \alpha_{ij}^{m-1}(A)z_j, \tag{4}$$

where the degree $m-1$ polynomials α_{ij}^{m-1} and q_{i+1}^{m-1} are chosen such that $\|r_{i+1}\|_2$ is minimized. Recall that $z_i \in \mathcal{K}_m(A, r_{i-1})$ and that $Az_i = r_{i-1} - r_i$ from (3). Therefore, by choosing z_i as an additional right-hand side, the standard approximation space is augmented with part of the information generated in the previous approximation space.

Using (3), we now rewrite B-LGMRES(m, k) in (4) in the following way:

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

In the above form, as shown for the LGMRES method in [4], 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 iteration (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 (α_{ij}) instead of scalars. Furthermore, as in [4], it is easily shown that these direction vectors z_j are A^TA-orthogonal.

THEOREM 1 (orthogonality of the error approximations). *The error approximation vectors $z_j \equiv x_j - x_{j-1}$ in B-LGMRES are A^TA-orthogonal.*

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

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

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

By construction, $r_i \perp A\mathcal{M}_i$ and $r_{i+1} \perp A\mathcal{M}_{i+1}$. From (3), $r_i - r_{i+1} = Az_{i+1} \Rightarrow Az_{i+1} \perp A(\mathcal{M}_i \cap \mathcal{M}_{i+1})$. Because $\{z_j\}_{j=(i-k+1):i} \subset \mathcal{M}_i \cap \mathcal{M}_{i+1}$, $z_{i+1} \perp_{A^TA} \{z_j\}_{j=(i-k+1):i}$. ∎

To provide additional insight into the B-LGMRES augmentation scheme, we briefly describe 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

$$\mathcal{M} = \mathcal{K} + \mathcal{W}, \tag{5}$$

where $\mathcal{K}$ is the 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

1. $r_i = b - Ax_i$, $\beta = \|r_i\|_2$, $r_i = r_i/\beta$
2. $R_i = [r_i, z_i, \ldots, z_{i-k+1}]$
3. $R_i = V_1\hat{R}$ (QR factorization)
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. 1. *B-LGMRES(m, k) for restart cycle i.*

process. Following the discussion in [9], the minimization process over the augmented approximation space (5) 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

$$\|\tilde{r}\|_2 \le \|(I - P_{AW})r_d\|_2, \tag{6}$$

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 (6) 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 $\mathcal{K}_m(A, e_i - e_{i-1})$.

4.3. Algorithm details. The implementation of B-LGMRES(m, k) is similar to that of BGMRES. For reference, one restart cycle (i) of B-LGMRES(m, k) is given in Figure 1. As with BGMRES, B-LGMRES(m, k) requires the application of s^2 rotations at each iteration to transform H_m into an upper triangular matrix (e.g., see [37]). Because B-LGMRES(m, k) solves $Ax = b$, as opposed to a block system, the least-squares solution step (line 13) varies from that of standard BGMRES; the triangular matrix $\hat{R}$ from the QR decomposition in line 3 does not need to be saved since only $\beta = \|r_i\|_2$ is needed, and only s rotations must be applied to the least-squares problem's right-hand side (βe_1) at each step.

Though we think of the error approximations z_j, $j = (i - k + 1) : i$ as additional right-hand side vectors, we do not form 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 1. 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. The m 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 $(m + 1)s \times m \cdot s$ band-Hessenberg matrix with s subdiagonals, and the following standard relationship holds:

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

TABLE 1
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

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.

For the B-LGMRES(m, k) implementation given in Figure 1, the multivectors are of size s, where $s = k + 1$. For example, the orthogonal block matrices V_j in lines 3, 5, 8, 10, and 12 of Figure 1 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 demonstrated in section 5.1.

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. For comparison, 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 GMRES(m), LGMRES(m, k), and B-LGMRES(m, k) in Table 1.

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 1 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 1, note that, with right preconditioning, line 15 is equivalent to $x_{i+1} = x_i + M^{-1}\hat{z}_{i+1}$.

We note that in our implementation, no reorthogonalization was performed, and residual norms are computed recursively at each step within a restart cycle. Additionally, although deflation is often an important issue for block methods, it is not required for B-LGMRES because we are not solving a block linear system. The possibility of a breakdown due to rank deficiency for the initial residual block does exist, but we have not had any breakdowns in practice (a random vector may be substituted for z_i if a breakdown is detected).

5. Numerical experiments. In this section, we first give the specific details of our B-LGMRES implementation and demonstrate the advantages of some performance programming techniques. We then show the importance of reducing data movement to achieve an efficient implementation. Finally, we present promising experimental results from our efficient implementation of B-LGMRES on a variety of problems.

We implemented the B-LGMRES algorithm in C using a locally modified version

TABLE 2
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

of PETSc 2.1.5 (Argonne National Laboratory's Portable, Extensible Toolkit for Scientific Computation) [6, 5]. All results provided in this paper were run on a single processor of a 16-processor Sun Enterprise-6500 server with 16 Gbytes RAM. This system consists of 400 Mhz Sun Ultra II processors, each with a 16 Kbyte L1 cache and a 4 Mbyte L2 cache. For reference, Table 2 lists multiple test problems from the University of Florida Sparse Matrix Collection [11], the Matrix Market Collection [33], and the PETSc test collection.

5.1. An efficient implementation. To specifically demonstrate the benefit of an efficient implementation, we implemented B-LGMRES in PETSc 2.1.5 both with and without what we refer to as a *multivector optimization*. For the multivector optimization implementation, the MV implementation, we followed the previously mentioned approach in [20, 21] for improving the performance of a matrix-multivector multiply routine by grouping computations on the same data. Consider the size s multivector V, where $V \equiv [v_1, v_2, \ldots, v_s]$ for vectors $v_1, v_2, \ldots, v_s \in \mathbb{R}^{n\times 1}$. By processing the s individual vectors as a group, the matrix-multivector multiply routine performs a greater number of floating-point operations for each access of a nonzero element of the coefficient matrix A and has the potential to reduce the amount of data moved through parts of the memory subsystem by a factor of s. The approach of grouping computations together affects more than just the matrix multiply routine; it pervades the implementation of the entire B-LGMRES algorithm. The second implementation, referred to as the *non-MV* implementation, does not group multivector computations together and represents the best implementation possible with the tools available in PETSc. Both implementations were written so as to eliminate any copying of data from one data structure to another and represent best coding efforts.

Three primary sections of the B-LGMRES code are affected by the multivector optimization: the matrix-vector multiply (MatMult), the modified Gram–Schmidt orthogonalization (MGS), and the application of the preconditioner (Prec), if required. For the MV implementation for our set of test problems, the percentage of time spent in each of the three primary sections varies from 18% to 83% for MatMult, 6% to 55%

for MGS, and 31% to 53% for Prec. Because the Prec section of code shows similar characteristics to the MatMult section, we only discuss the MatMult and MGS sections of code. In addition, note that we concentrate on a multivector of size $s = 2$, which corresponds to B-LGMRES(m, k) with $k = 1$. We show in section 5.3 that size $k = 1$ is generally optimal for the algorithm. However, because our results are applicable to improving the performance of general block linear solvers as well, we also discuss the advantages of a larger block size.

For the remainder of this section, we detail the MV implementation, explain how it differs from the non-MV implementation in terms of data movement, and compare the performance of the two implementations. Because algorithmic changes in a numerical code affect data movement between main memory and cache and between levels of cache, the size of the data structures determines the part of the memory most affected by the multivector optimizations. Therefore, for each we discuss the *working set size*, which is the size in Mbytes of data loaded through the memory hierarchy for each operation. Note that for a typical memory hierarchy, L1 caches typically have access times of two to three clock cycles, L2 cache access times are generally 5–15 times slower, and main memory access times are generally at least 100 times slower still (e.g., see [44, 32, 7]).

The MatMult section of the code corresponds to the matrix-vector multiply in line 5 of Figure 1. For the non-MV implementation, successive calls are made to a matrix-vector multiply routine for each individual vector $v_1, v_2, \ldots, v_s$ in multivector V_j. In contrast, for the MV implementation, we store the multivectors in a format referred to as *multicomponent vectors* in the PETSc 2.1.5 manual [5] and use the associated PETSc matrix-multivector multiply routine. This multicomponent format is a row-wise storage format that stores V as a vector of length $n \cdot s$ in which the components of the s vectors are interlaced. In other words, multivector elements separated by stride s belong to a single vector: $v_1 = [V(0); V(s); V(2s); \ldots ; V(ns - s)]$. The working set size for the MatMult section, $WS_{MatMult}$, is approximately equal to the storage required for matrix A. Therefore, for compressed sparse row storage, $WS_{MatMult} = sizeof(double) * nnz + sizeof(int) * (n + nnz)$, where the function $sizeof()$ returns the size of its argument in bytes. If $WS_{MatMult}$ 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 for successive matrix-vector multiplies.

The MGS section (lines 6–10 in Figure 1) often contributes significantly to the overall cost of the B-LGMRES algorithm and is dominated by the PETSc routines *VecDot* and *VecAXPY*, which are the vector dot product and axpy routines, respectively. For the MV implementation, we wrote multivector versions of these two routines, referred to as *VecStrideDot* and *VecStrideAXPY*, in a manner that limits movement of data. For example, in the non-MV implementation, determining the dot product of two size s multivectors, which are themselves $n \times s$ matrices, results in an $s \times s$ matrix. Computing that matrix requires s^2 successive calls to VecDot and $2 \cdot n \cdot s^2$ data values to be read from the memory hierarchy. In contrast, one call to VecStrideDot provides the same functionality, but only $2 \cdot n \cdot s$ data values are read because computations on related data are fused together. Therefore, if $WS_{MGS} \equiv sizeof(double) \cdot 2 \cdot n \cdot s$ is greater than either the L1 or L2 cache size, the multivector optimization affects data movement in the MGS section of code.

Both VecStrideAXPY and VecStrideDot were written using loop temporaries and loop unrolling to aid compiler optimization. The use of loop temporaries allows a

TABLE 3
Execution times in μsec for a single call to AXPY for the non-MV and MV implementations as well as for the standard and optimized BLAS 3 DGEMM routines with a block size of $s = 2$.

	Problem	n	non-MV	MV	DGEMM (src)	DGEMM (opt)
13	ex40	7740	2.00	1.25	3.96	2.96
3	memplus	17758	4.55	2.85	9.18	6.73
11	bcircuit	68902	17.43	10.35	35.32	25.73
14	epb3	84617	19.54	12.58	43.49	31.88

TABLE 4
Execution times in μsec for a single call to a dot product routine for the non-MV and MV implementations as well as for the standard and optimized DGEMM routines with a block size of $s = 2$.

	Problem	n	non-MV	MV	DGEMM (src)	DGEMM (opt)
13	ex40	7740	1.20	.78	3.94	2.89
3	memplus	17758	2.60	1.69	9.16	6.66
11	bcircuit	68902	10.91	6.29	35.32	24.98
14	epb3	84617	12.64	7.89	44.01	30.96

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 our MV implementation we unroll the inner loop s times, where s is the block size. We chose to write our own VecStrideAXPY and VecStrideDot routines instead of using the level 3 BLAS DGEMM routine [14] due to the performance advantage of a hand-coded implementation. The timings to perform a multivector AXPY and dot product are given in Tables 3 and 4, respectively, for $s = 2$. The version of DGEMM labeled "src" is compiled from source code and the version labeled "opt" is the vendor supplied optimized library. Timings are given in μsec for a subset of the test problems with a range of matrix orders. For both routines, the MV and non-MV implementations are superior to DGEMM, and the hand-tuned MV implementations are the clear winners. Our MV implementations outperform DGEMM because the loops are unrolled to the specific block sizes. Because the MGS section can consume a large percentage of total execution time, the simple optimizations described can have as significant an impact on overall execution time of the solver as the use of the optimized matrix-multivector multiply routine.

As previously explained, we store the multivectors row-wise in our MV implementation. Alternatively, one might wonder whether column-wise ordering is a viable option as column-wise ordering better facilitates algorithmic needs such as deflation and requires less reorganization of data. Therefore, in Table 5, we give results for the MatMult routine on block sizes $s = 2$ and $s = 4$. MV-row and MV-col indicate row-wise and column-wise variations of the MV implementation. In addition to four problems from our test set, row 5 in Table 5 lists an additional problem, poisson3Db, from the UF collection (n= 85623 and nnz = 2374949) that has a large bandwidth. Our experiments indicate that the use of the column-wise storage format for multivectors on large problems with a large matrix bandwidth causes TLB misses, as suggested in [24]. (The TLB is a cache of translations between the real and virtual addresses of recently referenced pages of memory. A TLB miss incurs two loads to memory: one to load the address translation and a second to load the actual operand.) To illustrate the impact of TLB misses, line 6 of Table 5 lists the results for the poisson3Db problem with reverse Cuthill–McKee (RCM) [10] reordering. The

TABLE 5
Execution times in μsec for the non-MV and MV implementations of the MatMult routine for block sizes $s = 2$ and $s = 4$.

		Block size = 2 (μsec)			Block size = 4 (μsec)		
	Problem	non-MV	MV-row	MV-col	non-MV	MV-row	MV-col
13	ex40	30.7	24.7	25.1	47.1	27.4	27.4
3	memplus	17.8	13.4	13.4	32.4	15.7	16.2
11	bcircuit	65.3	46.2	46.5	116.4	61.9	63.1
14	epb3	84.6	57.8	60.2	134.2	71.0	75.6
–	poisson3Db	424.2	263.5	369.2	831.7	367.6	927.2
–	poisson3Db (RCM)	339.2	152.7	153.6	679.9	184.9	183.5

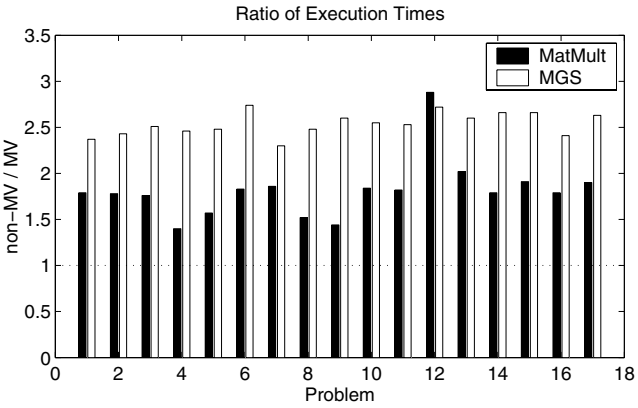

FIG. 2. *A comparison of execution time for the MatMult and MGS sections with the non-MV implementation of B-LGMRES(*15,1*) versus the MV implementation for ten restart cycles for the* 17 *test problems.*

RCM reordering reduces the matrix bandwidth and significantly decreases execution time for the column-wise multivector storage format. A large bandwidth does not impact the row-wise storage as much because consecutive accesses of the multivector elements correspond to data items that are stored contiguously. All further numerical experiments in this manuscript use the row-wise MV implementation.

We now compare the execution times for the non-MV and MV implementations on both the MatMult and MGS sections of code in Figure 2. The x-axis corresponds to the numbered test problems in Table 2, and the y-axis is the execution time for the non-MV implementation divided by the execution time for the MV implementation. A value greater than one indicates that the execution time for MV is less than that for non-MV. For the MatMult section, the MV implementation reduces the execution time by a factor of 1.4 to 2.7 over the non-MV implementation, and it is an open question what impact matrix density (or even nonzero structure) has on the effectiveness of the multivector optimizations. For the MGS section, the execution time shows an even greater improvement on average: the MV implementation of MGS reduces execution time by a factor of 2.3 to 2.7 over the non-MV implementation. As for the execution time for the entire B-LGMRES code, the MV implementation is close to twice as fast on average as the non-MV implementation with multivectors of size $s = 2$, and the performance improvement is independent of problem size (n) and number of nonzeros (nnz) for our test set in Table 2.

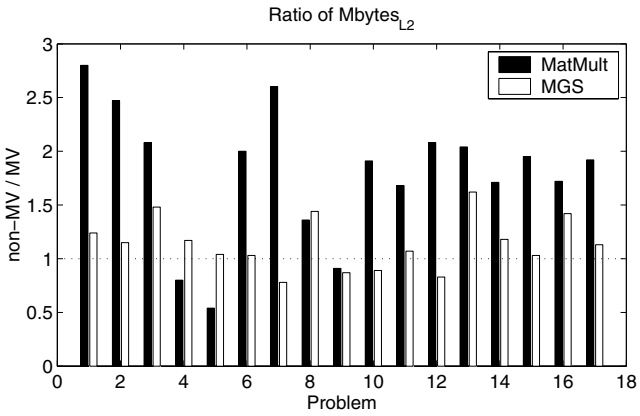

FIG. 3. *A comparison of data movement from main memory to L2 cache in the MatMult and MGS sections for the non-MV implementation of B-LGMRES(*15,1*) versus the MV implementation for ten restart cycles for the* 17 *test problems.*

Additionally, we remark that we modified our local installation of PETSc to include multivector versions of the PETSc routines *VecMAXPY* and *MatSolve*. 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.

5.2. Impact of data movement on execution time. We now explain reductions in execution time due to the multivector optimization using data from hardware performance counters that monitor data movement. We focus on two counters that measure the megabytes of data moved between main memory and L2 cache, denoted by $Mbytes_{L2}$, and between the L2 and L1 caches, denoted by $Mbytes_{L1}$.

We first examine data movement between main memory and the L2 cache in Figure 3. The y-axis in this figure indicates the ratio of $Mbytes_{L2}$ for non-MV to MV for both the MatMult and MGS sections. Because $s = 2$, we expected a factor of two reduction in $Mbytes_{L2}$ for the MV implementations for test problems with a working set size significantly larger than the L2 cache. For the MatMult section, problems 6 and 11–17 have $WS_{MatMult}$ larger than the 4 Mbytes L2 cache size. These problems all have ratios from 1.75 to 2.0, which correlate well with the factor of two reductions in execution time for those problems seen in Figure 2. For the MGS section, only problem 16 has WS_{MGS} greater than the L2 cache size, and this problem shows a ratio of 1.4, which does not correlate well with the factor of 2.4 improvement in execution time shown in Figure 2. Furthermore, the results in Figure 3 are inconsistent with those in Figure 2 in other ways: several problems appear to show an increase in $Mbytes_{L2}$ for the MGS and MatMult sections for the MV implementation. In fact, for these problems the amounts of data moved are quite small for both implementations and the apparent increases are likely the result of measurement error. Nonetheless, the MV implementation is faster than the non-MV one. These inconsistencies indicate that a reduction in $Mbytes_{L2}$ does not accurately predict a reduction in execution time for MGS for most of the test problems.

We now consider data movement between the L1 and L2 caches. Figure 4 shows the ratio of $Mbytes_{L1}$ for the non-MV to MV implementation. For the MatMult section, we expected test problems with $WS_{MatMult} \gg sizeof(L1\ cache)$ to show a

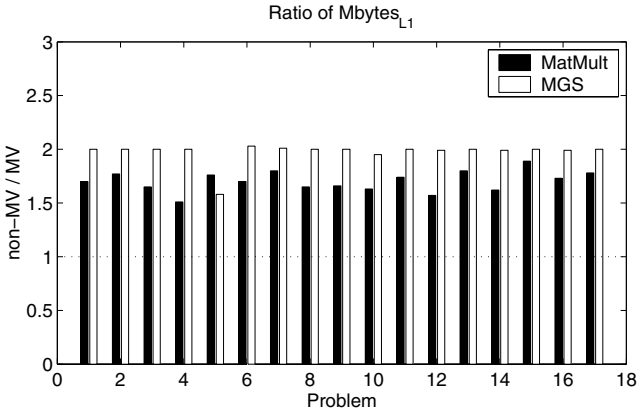

FIG. 4. *A comparison of data movement from L2 cache to L1 cache in the MatMult and MGS sections for the non-MV implementation of B-LGMRES(*15,1*) versus the MV implementation for ten restart cycles for the* 17 *test problems.*

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. Thus, Figure 4 shows that for all of the test problems the ratios of data moved do in fact range from 1.5 to 1.9. Similarly for the MGS section, 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 that range from 1.6 to 2.0. The reduction in $Mbytes_{L1}$ is thus consistent with the reduction in execution time seen in Figure 2 for both sections of code.

Therefore, from Figures 3 and 4 one can infer that the reduction in *total* execution time due to the MV implementation correlates to the reduction in $Mbytes_{L1}$ for our test problems (see [3] for more details). The importance of optimizations that reduce movement of data between levels of cache was unexpected in that emphasis is traditionally placed on reducing data movement between cache and main memory, which is of most concern to large problems. For much larger test problems, where both $WS_{MatMult}$ and WS_{MGS} are much larger than the L2 cache size, for example, we expect that the reduction $Mbytes_{L2}$ would more strongly correlate to execution time. However, our results indicate that data movement is not just an issue for very large problems. In fact, the results presented in this section demonstrate that reformulating an iterative solver to use multivectors enables an efficient implementation that can reduce data movement for problems of any size. Furthermore, these multivector optimizations would benefit larger block sizes and other types of block methods, including those that solve systems with multiple right-hand sides.

5.3. Comparison with other methods. In this section, we demonstrate the potential of the B-LGMRES method both with and without preconditioning by presenting experimental results for the test problems in Table 2. 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. 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 either ILU(p) or ILUTP($droptol$, $permtol$, $lfil$) (e.g., see [37]). 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

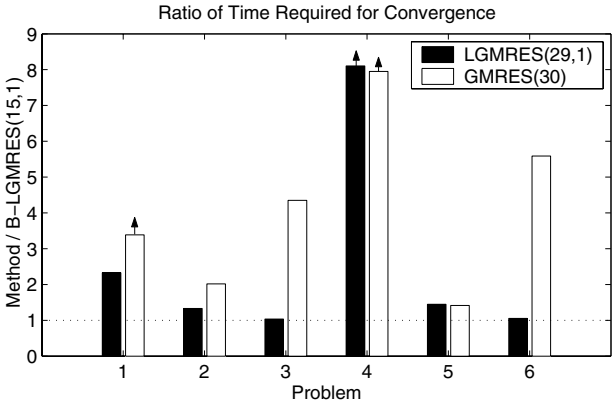

FIG. 5. *A comparison of the time required for convergence for nonpreconditioned test problems* 1–6 *with GMRES(*30*) and LGMRES(*29, 1*) versus B-LGMRES(*15, 1*).*

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. We use the GMRES(m) implementation available in PETSc 2.1.5 and the PETSc implementation of LGMRES(m, k) described in [4]. For GMRES(m), we chose restart parameter $m = 30$ because it is a common choice for GMRES(m) and is the default in PETSc. 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 [4]). Note that for a fixed approximation space size, increasing k for B-LGMRES(m, k) decreases the powers of A represented in the approximating subspace. For this set of test problems, preliminary testing showed that using $k > 2$ was typically not beneficial and $k = 1$ was generally optimal. (For an approximation space larger than 30, we expect that using $k > 1$ would be an advantage more often.) As a result, we chose $k = 1$ for the experiments presented here, and this choice together with the constraint of a size 30 approximation space determined parameter m for each method as in Table 1. Note the approximation space for B-LGMRES(15, 1) at restart cycle $i+1$ is given by $\mathcal{K}_{15}(A, r_i)+\mathcal{K}_{15}(A, z_i)$ and that for LGMRES(29, 1) is given by $\mathcal{K}_{29}(A, r_i)+span\{z_i\}$.

We first present results for the nonpreconditioned test problems. Figure 5 compares the time required for convergence for B-LGMRES(15, 1) to both LGMRES(29, 1) and GMRES(30) for the test problems given in Table 2. The y-axis is the time required for convergence for either LGMRES(29, 1) or GMRES(30) 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 5 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. Again, the time

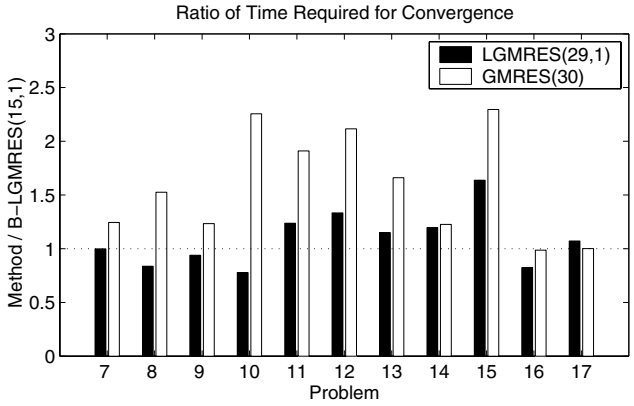

FIG. 6. *A comparison of the time required for convergence for preconditioned test problems* 7–17 *with GMRES(*30*) and LGMRES(*29, 1*) versus B-LGMRES(*15, 1*).*

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, and improvements over LGMRES(29, 1) are more modest since LGMRES is typically also an improvement over GMRES(m) [4].

We now do the same performance evaluation for B-LGMRES(m, k) on preconditioned problems, problems 7–17, in Figure 6. We use left preconditioning, and determination of convergence is based on the preconditioned residual norm as usual. As in Figure 5, the bars extending above one favor B-LGMRES(15, 1). 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, performance gains are not as dramatic as those without preconditioning due to lower iteration counts. For large problems, though, even a small improvement in iteration count translates into a nontrivial 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, 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 is used, but LGMRES(29, 1) is faster with preconditioning.

For all of the test problems in Table 2, we found that the time to solution of the restarted methods correlates well with the number of accesses of A, as opposed to the number of matrix-vector multiplies. In particular, Figure 7 illustrates that the number of accesses of A largely determines execution time. Therefore, the method that converges in the least amount of time for a particular problem in Figures 5 and 6 is generally that with the smallest number of iterations. However, it is important to acknowledge that the efficient implementation of B-LGMRES(15, 1) makes the preceding statement possible because each iteration of B-LGMRES(15, 1) requires two matrix-vector multiplies; a poor implementation of B-LGMRES(15, 1) could potentially cost twice as much per iteration as GMRES(30).

Because our machine has sufficient resources, full GMRES is a viable option. Therefore, we also compared the time to solution of B-LGMRES(15, 1) to full GMRES for each of the 17 test problems. B-LGMRES converges in significantly less time than

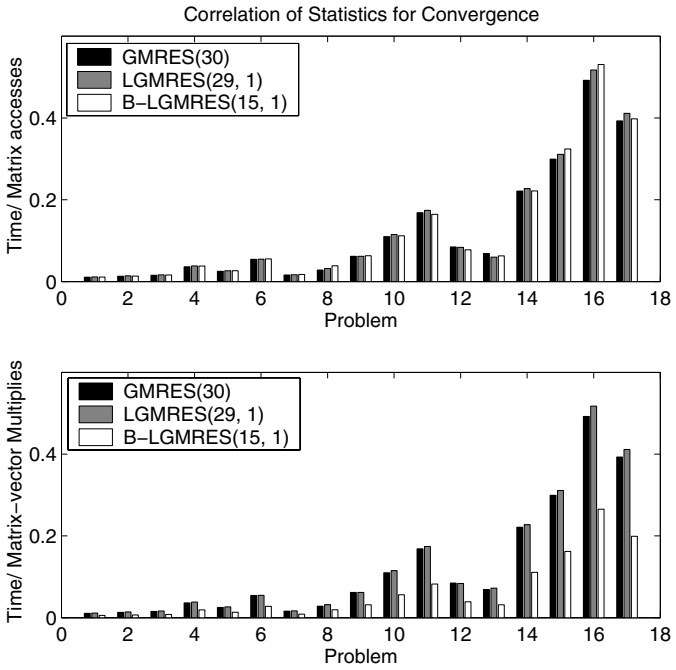

FIG. 7. *The upper panel shows the ratios of time required for convergence to number of accesses of A for GMRES (*30*), LGMRES(*29, 1*), and B-LGMRES(*15, 1*) on the* 17 *test problems. The lower panel shows the ratios of time required for convergence to number of matrix-vector multiplies.*

GMRES on all but four problems: 7, 10, 13, and 15. There was no correlation between problem size and method in terms of time required for convergence for these test problems. See [3] for more details.

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 5 and 6. 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 [4]), 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-DR method [31] can be good options. Although a thorough understanding of the convergence behavior of B-LGMRES is an open question that warrants further investigation, the experimental results presented in this section demonstrate that solving a single right-hand side system via a block system is a viable means of improving the time to solution of an iterative method such as GMRES(m). In particular, achieving a balance between maintaining or improving an iterative linear solver algorithm's numerical properties and reducing data movement is possible.

6. Concluding remarks. In this paper, we explore the feasibility of modifying restarted GMRES to reduce data movement. We show that for iterative linear solvers, 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. Therefore, mod-

ifications that reduce data movement are a particularly effective means of improving performance. Furthermore, our results indicate that using available performance programming techniques to drive algorithm development can be effective and that careful attention to implementation details can be quite beneficial. For example, hand-tuned routines can outperform subroutines such as DGEMM for certain matrix sizes, and 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 nonblock version. A unique aspect of this study is the thorough investigation of data movement through the memory hierarchy. Typically the impact of memory performance on an iterative linear solver is not studied in this detail.

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, results from choosing error approximation vectors as additional right-hand side vectors. This choice mimics a truncated polynomial-preconditioned conjugate gradient method. In our experiments, B-LGMRES performed best with a block size of two for an approximation space of dimension 30. For larger approximation space sizes and larger problems, a larger block size may prove more beneficial. We find that predicting the algorithm's performance is nontrivial 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. Furthermore, other right-hand side vectors may be more appropriate for particular problem classes. Given the potential performance gains from larger block sizes described in section 5.1, our implementation of B-LGMRES can serve as a template for other block methods.

Nevertheless, the substantial improvement in performance due to reduced data movement is sufficient enticement toward pursuing block methods for single right-hand side problems. Given the increasing gap between processor performance and memory access time, re-examining popular linear solver algorithms is particularly important to achieving respectable performance on modern architectures.

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. E. Saylor, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[3] A. H. Baker, *On Improving the Performance of the Linear Solver Restarted GMRES*, Ph.D. thesis, University of Colorado, Boulder, CO, 2003.

[4] A. H. Baker, E. R. Jessup, and T. Manteuffel, *A technique for accelerating the convergence of restarted GMRES*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 962–984.

[5] 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, Argonne, IL, 2003.

[6] 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).

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

[8] 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, J. Dongarra, P. Messina, D. C. Sorensen, and R. G. Voigt, eds., SIAM, 1990, pp. 400–405.

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

[10] E. H. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings 24th Nat. Conf. Assoc. Comp. Mach., ACM Publications, New York, 1969, pp. 157–172.
[11] T. Davis, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices (2002).
[12] E. de Sturler, *Truncation strategies for optimal Krylov subspace methods*, SIAM J. Numer. Anal., 36 (1999), pp. 864–889.
[13] J. W. Demmel, N. J. Higham, and R. S. Schreiber, *Block LU factorization*, Numer. Linear Algebra Appl., 2 (1995), pp. 173–190.
[14] J. Dongarra, J. DuCroz, S. Hammarling, and I. Duff, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.
[15] J. Dongarra and V. Eijkhout, *Self-Adapting Numerical Software for Next Generation Applications*, Tech. Report ICU-UT-02-07, LAPACK Working Note 157, 2002.
[16] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling, *Block reduction of matrices to condensed forms for eigenvalue computations*, J. Comput. Appl. Math., 27 (1989), pp. 215–227.
[17] M. Eiermann, O. G. Ernst, and O. Schneider, *Analysis of acceleration strategies for restarted minimal residual methods*, J. Comput. Appl. Math., 123 (2000), pp. 261–292.
[18] 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.
[19] K. Gallivan, W. Jalby, U. Meier, and A. H. Sameh, *Impact of hierarchical memory systems on linear algebra algorithm design*, Internat. J. Supercomput. Appl., 2 (1988), pp. 12–48.
[20] 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, New York, 1999, pp. 233–240.
[21] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *High-performance parallel implicit CFD*, Parallel Comput., 27 (2001), pp. 337–362.
[22] G. Gu and Z. Cao, *A block GMRES method augmented with eigenvectors*, Appl. Math. Computation, 121 (2001), pp. 278–289.
[23] M. S. Lam, E. E. Rothberg, and M. E. Wolf, *The cache performance and optimizations of blocked algorithms*, in Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 1991.
[24] B. C. Lee, R. W. Vuduc, J. W. Demmel, K. A. Yelick, M. de Lorimier, and L. Zhong, *Performance Optimizations and Bounds for Sparse Symmetric Matrix-Multiple Vector Multiply*, Tech. Report UCB-CSD-03-1297, Computer Science Division, University of California, Berkeley, 2003.
[25] G. Li, *A block variant of the GMRES method on massively parallel processors*, Parallel Comput., 23 (1997), pp. 1005–1019.
[26] J. D. McCalpin, *Memory bandwidth and machine balance in current high performance computers*, IEEE Computer Society Technical Committee on Computer Architecture Newsletter, (1995).
[27] J. D. McCalpin, *STREAM: Sustainable Memory Bandwidth in High Performance Computers*, http://www.cs.virginia.edu/stream (2003).
[28] S. McKee and W. Wulf, *Access order and memory-conscious cache utilization*, in First Symposium on High Performance Computer Architecture (HPCA1), Raleigh, NC, 1995.
[29] R. Morgan, *GMRES with deflated restarting and multiple right-hand sides*, presentation at the Seventh Copper Mountain Conference on Iterative Methods, 2002.
[30] R. B. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1154–1171.
[31] R. B. Morgan, *GMRES with deflated restarting*, SIAM J. Sci. Comput., 24 (2002), pp. 20–37.
[32] S. Naffziger and G. Hammond, *The implementation of the next generation* 64*b Itanium microprocessor*, in Proceedings of the IEEE International Solid-State Circuits Conference, Vol. 2, 2002, pp. 276–504.
[33] National Institute of Standards and Technology, Mathematical and Computational Sciences Division, *Matrix Market*, http://math.nist.gov/MatrixMarket (2002).
[34] D. O'Leary, *The block conjugate gradient algorithm and related methods*, Lin. Algebra Appl., 29 (1980), pp. 293–322.
[35] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, *A case for intelligent RAM*, IEEE Micro, 17 (1997), pp. 34–44.

[36] A. Pinar and M. T. Heath, *Improving performance of sparse matrix-vector multiplication*, in Proceedings of Supercomputing '99, 1999.
[37] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
[38] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statistical Comput., 7 (1986), pp. 856–869.
[39] V. Simoncini and E. Gallopoulos, *An iterative method for nonsymmetric systems with multiple right-hand sides*, SIAM J. Sci. Comput., 16 (1995), pp. 917–933.
[40] V. Simoncini and E. Gallopoulos, *Convergence properties of block GMRES and matrix polynomials*, Lin. Algebra Appl., 247 (1996), pp. 97–119.
[41] S. Toledo, *Improving the memory-system performance of sparse-matrix vector multiplication*, IBM J. Research Development, 41 (1997), pp. 711–725.
[42] H. A. van der Vorst and C. Vuik, *GMRESR: A family of nested GMRES methods*, Numer. Linear Algebra Appl., 1 (1994), pp. 369–386.
[43] B. Vital, *Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multi-processeur*, Ph.D. thesis, Université de Rennes I, Rennes, France, 1990.
[44] R. Vuduc, 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, Baltimore, MD, 2002.
[45] W. A. Wulf and S. A. McKee, *Hitting the Wall: Implications of the Obvious*, Tech. Report CS-94-48, University of Virginia, Charlottesville, VA, 1994.