SIAM J. MATRIX ANAL. Vol. 00, No. 0, pp. 000–000, May 0000



# PARALLEL COST ANALYSIS OF ADAPTIVE GMRES IMPLEMENTATIONS FOR HOMOTOPY METHODS*

MARIA SOSONKINA†, DONALD C. S. ALLISON†, AND LAYNE T. WATSON‡

**Abstract.** The success of homotopy methods in solving large-scale optimization problems and nonlinear systems of equations depends heavily on the solution of large sparse nonsymmetric linear systems on parallel architectures. Iterative solution techniques, such as GMRES($k$), favor parallel implementations. However, their straightforward parallelization usually leads to a poor parallel performance because of global communication incurred by processors. One variation of GMRES($k$) considered here is to adapt the restart value $k$ for any given problem and use Householder reflections in the orthogonalization phase to achieve high accuracy and to reduce the communication overhead. The Householder transformations can be performed without global communications and modified to utilize an arbitrary row distribution of the coefficient matrix. The effect of this modification on the GMRES($k$) performance is discussed here, as well as the abilities of parallel GMRES implementations using Householder reflections to maintain fixed efficiency with increase in problem size and number of processors. Theoretical communication cost and isoefficiency analyses are compared with experimental results on an Intel Paragon, Cray T3E, and IBM SP2.

**Key words.** GMRES method, Krylov subspace methods, nonsymmetric linear systems, homotopy methods, sparse matrix

**AMS(MOS) subject classifications.** 65F10, 65F50, 65H10

## 1. Introduction.

A typical scenario in high performance scientific computing is where an application code calls, in a loop, a nonlinear system of equations solver, which in turn, in tighter inner loops, calls function evaluation subroutines and a linear system solver. Efficiency of course matters, but a failure of either the linear system solver or the nonlinear system solver to return a solution to the outer client code is disastrous for the application code. In practice, robustness is far more important than efficiency or even accuracy (as long as the latter two are acceptable). This paper addresses the robustness issue for both types of solution "servers," by proposing globally convergent homotopy algorithms [21] for the nonlinear equations solver, and an adaptive GMRES($k$) algorithm [18] for the linear system solver. Locally convergent methods for nonlinear systems of equations, such as quasi-Newton methods, often fail whenever a good initial estimate of the solution to a given problem is unavailable. (Global convergence to a stationary point of some auxiliary function, which is not a zero of the original nonlinear system, does not suffice for a client needing a true zero.) On the other hand, probability-one homotopy methods converge to a true solution from an *arbitrary* starting point (outside of a set of measure zero). These methods have been successfully applied to solve Brouwer fixed point problems, certain classes

* Received by the editors ; accepted for publication (in revised form) .

† Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061-0106. The work of Sosonkina and Watson was supported in part by Department of Energy grant DE-FG05-88ER25068/A004, National Aeronautics and Space Administration grant NAG-1-1562, Air Force Office of Scientific Research grants F49620-96-1-0089 and F49620-96-1-0104, and National Science Foundation grant DMS-9625968.

‡ Departments of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061-0106.

of zero finding problems, polynomial systems of equations, nonlinear complementarity problems, constrained optimization problems, and discretizations of nonlinear two-point boundary value problems based on shooting, finite differences, collocation, and finite elements [20].

While homotopy zero curve tracking is an inherently serial process, the burden of computation in homotopy methods is in the numerical linear algebra and user-defined function evaluations [4], which can be parallelized. For large scale problems, the cost of this serial work may be negligible in comparison with linear system solving (and user-written function evaluation). As concluded in [1], a high degree of parallelism in linear system solutions contributes much to the overall parallel performance of homotopy methods. In evaluating the performance of a complex algorithm, parallel efficiency is a large concern. Efficiency is usually defined as the ratio of speedup to the number of processors. A quantity closely related to the efficiency metrics is the *isoefficiency scalability* proposed in [7]. This isoefficiency scalability is applied to the evaluation of the parallel algorithms in this paper.

In the context of homotopy methods, an efficient parallel implementation of a linear system solver has to satisfy the requirements set for a sequential linear solver—robustness and high accuracy in the solution. Under these requirements, a variation of the popular linear system solution tool GMRES [15] is considered in this paper. This variation [18] uses an adaptive strategy to deal with varying difficulties of linear systems which are to be solved by a homotopy method.

The purpose of this paper is to study both theoretically and experimentally the parallel performance of an adaptive GMRES algorithm, designed as an industrial strength linear system solution server for a homotopy method based nonlinear equation solver client. The main conclusions are that the parallel cost models agree very well with experiment, the behavior of the Householder reflection based adaptive GMRES($k$) algorithm on realistic problems is excellent, and the concept of isoefficiency may not be applicable to (or meaningful for) adaptive parallel algorithms. Section 2 gives an overview of the adaptive GMRES($k$) algorithm and its parallel implementations using Householder reflections. A theoretical communication cost analysis of these implementations constitutes Section 3. A theoretical derivation of the isoefficiency function for restarted GMRES is presented in §4 followed by comparison with numerical experiments in §5. Section 6 considers the applicability of isoefficiency analysis to adaptive GMRES($k$). Section 7 contains conclusions.

**2. Adaptive GMRES($k$) algorithm.** The GMRES algorithm [15] is used to solve a linear system $Ax = b$ with an $n \times n$ nonsymmetric invertible coefficient matrix $A$. Similar to the classical conjugate gradient method, GMRES produces approximate solutions $x_j$ which are characterized by a minimization property over the Krylov subspaces $K(j, A, r_0) \equiv \mathrm{span}\{r_0, Ar_0, A^2r_0, \cdots, A^{(j-1)}r_0\}$, where $r_0 = b - Ax_0$ and $j$ is the iteration number. However, unlike the conjugate gradient algorithm, the work and memory required by GMRES grow proportionately to the iteration number since GMRES needs all $j$ vectors to construct an orthonormal basis for $K(j, A, r_0)$. Restarting GMRES every $k$ iterations reduces the computational and storage costs at the expense of poorer convergence and a possibility of stagnation. An adaptive strategy may be designed [18] such that the restarted GMRES (called GMRES($k$)) is modified to adapt the parameter $k$ to the problem at hand. In particular, a test on near-stagnation is inserted after each restart cycle to determine whether to augment the current Krylov subspace with several new vectors or start building a new Krylov subspace (i.e. perform a restart). This test estimates the likelihood of convergence in a (weighted) number of the remaining iterations if GMRES maintains the rate of progress towards convergence achieved over the past restart cycle. With modern programming languages, which provide pointers and dynamic memory management, dealing with the variable storage requirements implied by varying $k$ is not difficult.

To balance the orthogonalization cost with numerical stability, the modified Gram-Schmidt process is often used in GMRES implementations. However, when a high accuracy solution is required, the GMRES orthogonalization phase may be a source of considerable roundoff errors. The affect of such errors has been observed, for example, in the postbuckling analysis of structures performed by homotopy methods (see [18] for details). As shown in [2], the orthogonalization with Householder reflections is more robust than the one with modified Gram-Schmidt. Although the Householder reflection orthogonalization is also more expensive, its use is justified by the improvement in the overall performance of the client algorithm. In detecting numerical instability due to orthogonalization errors, a condition number estimation of the GMRES least-squares problem was found to be helpful at each GMRES iteration [18]. The condition number can be estimated by an inexpensive procedure [3] requiring only $\mathcal{O}(n)$ floating point operations. For linear systems, which are not nearly singular, an estimated condition number being greater than the inverse of the machine unit roundoff implies a loss of orthogonality in the basis vectors, and adaptive GMRES($k$) aborts.

In summary, the described adaptive GMRES($k$) algorithm with Householder reflection orthogonalization can be outlined as follows (see [18] for more details):

1.**Start:** Choose initial guess, tolerance, and $k$.

2.**Construct basis:** Compute the first Householder reflection $P_1$.
$k_1 := 1$ and $k_2 := k$.
L: For $j := k_1, \ldots, k_2$
  Apply Householder reflections $w = P_j \ldots P_1 A P_1 \ldots P_j e_j$,
    where $e_j$ is the $j$th standard basis vector.
  Determine $P_{j+1}$ such that $P_{j+1} w$ has zero
    components after the $(j+1)$st component.
  Update residual norm.
  Estimate condition number.

3.**Test near stagnation:** If *insufficient residual norm reduction in the last restart cycle* then $k_1 := k_2$ and increment $k_2$; goto L.

4.**Solve GMRES least-squares problem.**

5.**Correct solution.**

**2.1 Parallel implementations of GMRES($k$) using Householder reflections.** In parallel environments, the choice of the orthogonalization process for the Krylov subspace basis vectors depends not only on the accuracy of the process but also on the amount and type of global communication it incurs. For some orthogonalization procedures, only one of the two requirements is satisfied. For example, in serial implementations of the GMRES method, the modified version of the Gram-Schmidt process is often used as being sufficiently accurate for a number of problems. However, in parallel GMRES implementations, other orthogonalization procedures are preferable since the modified Gram-Schmidt process exhibits a large communication overhead (see, e.g., [12]). Better efficiency is achieved with the classical Gram-Schmidt, which is unstable.

Here, an implementation of the Householder reflection orthogonalization in GMRES($k$) proposed in [19] is adapted to work in parallel. The parallel version employs an algorithm developed in [16] for generating and applying Householder reflections. This algorithm avoids dot-products and all-to-all communications. The degree of parallelism equal to the number of processors in parallel Householder reflection generation is achieved by letting each processor create its local portion of the Householder reflection vector independently. After that, local Householder reflection vectors are assembled into a global Householder reflection vector. To construct a global Householder reflection for a given vector $x$, the (first) single nonzero local component of $x$ is modified: it is changed to zero for all the processors except the first one in accordance with some sequential ranking of processors. Finally, the first processor updates its nonzero component based on the information obtained from the other processors. In order to preserve orthogonality of Householder reflections in the outlined modification, the orthogonalization procedure, a series of Givens rotations, updates the component of $x$ received from a neighbor and zeros out the local component. Then the updated value is transmitted to the other neighbor. Thus, to zero out all $p-1$ (where $p$ is the number of processors) components of $x$ and update the value located on the first processor, a ring topology of processors is the most suitable for the communication between processors. However, as will be shown later, different approaches in organizing communications on a ring affect partitioning of a problem. The approach taken in [16] assumes a *fixed* ring of processors, in which the communication always starts from a fixed (first) processor.

Pseudocode for the algorithm generating Householder reflections (call it HG) at the $j$th GMRES($k$) iteration on the processor *proc* (in a ring of $p$ processors) is given in Figure 1. It is followed by the pseudocode of the application of Householder reflections presented in Figure 2 for the same iteration and processor. Before applying a Householder reflection, a sequence of Givens rotations has to be applied. Thus, Householder reflection application can be viewed as the Householder reflection generation process reversed with respect to the order of using Givens rotations and Householder reflections.

In Figures 1 and 2, $H$ and $G$ denote the Householder transformation matrix and the Givens rotation matrix (as given in [6]), respectively; $v_{loc}$ denotes a portion of the Krylov subspace vector $A^j r_0$ located on a processor; $p$, $left$, $right$ are the processors with the highest rank, with the $proc-1$ rank, and the $proc+1$ rank, respectively. It is also assumed that the first processor has the $j$th row of the input matrix.

```
if (proc = 1) then s := j else s := 0 end if
determine H_{s+1} such that H_{s+1} v_loc ≡ w_loc has zeros
  after the (s+1)st component;
if (proc = 1) then
    send w_loc(s+1) to right;
else
    receive w from left;
    determine G_1 such that w_loc(1) = 0; update w;
    if (proc ≠ p) send w to right;
end if
```

FIG. 1. *Parallel Householder reflection generation.*

```
for i := j downto 1 step -1
    if (proc = 1) then
        send w_loc(i) to p;
        receive w_loc(i) from right;
        sc := i;
    else
        receive w from right;
        apply G_1 to (w, w_loc(1));
        send w to left;
        sc := 1;
    end if
    apply H_sc to w_loc(sc :);
end for
```

FIG. 2. *Parallel Householder reflection application.*

However, the design presented in Figures 1 and 2 admits only a special case of the matrix row distribution: assignment of a block of contiguous rows to each processor (call it block-striped distribution), which is rarely advantageous for an arbitrary unstructured matrix. For unstructured matrices a graph partitioning is used to minimize the communication to computation ratio of the distributed matrix-vector multiply. In the current implementation, a graph partitioning algorithm from the MeTiS package [11] is used and the parallel version of the matrix-vector multiply is performed as in [12]. The matrix-vector multiplication requires that the components of all vectors are distributed in accordance with the corresponding matrix rows and allows overlapping of computation and communication. To use the algorithms in Figures 1 and 2, the redistribution of a vector requires $\mathcal{O}(p^2)$ communications at each GMRES($k$) iteration, which is highly impractical and reduces the efficiency gained by the distributed matrix-vector product. Thus, it is beneficial to develop an extension (call it MHG) of the algorithms in Figures 1 and 2, which accepts an arbitrary row distribution among processors (Figures 3 and 4). The key idea of this extension is to allow a *flexible* ring in performing Givens rotations. The flexible ring is the ring of processors in which the communication may start from an arbitrary processor that (in a given vertex-based partitioning) holds the first component of the vector $x$. In practice, each processor in the flexible ring determines the current *ring_start* by consulting the global mapping array *mpart*, each entry of which is a processor number indexed by a matrix row number (where the row numbering refers to the original matrix before partitioning) located on that processor.

**if** $(j = 1)$ **then**
    $s := 1;$
**else**
    **if** $(proc$ has $(j-1)$st row$)$ **then** $s := s + 1;$
**end if**
determine $H_s$ such that $H_s v_{loc} \equiv w_{loc}$ has zeros
  after the $s$th component;
**if** $(proc$ has $j$th row$)$ **then**
    $ring\_end := left;$
    send $w_{loc}(s+1)$ and $ring\_end$ to $right;$
**else**
    receive $w$ and $ring\_end$ from $left;$
    determine $G_s$ such that $w_{loc}(s) = 0$; update $w;$
    **if** $(proc \neq ring\_end)$ **then** send $w$ to $right;$
**end if**

FIG. 3. *Modified parallel Householder reflection generation.*

$sc := s;$
**for** $i := j$ **downto** 1 **step** -1
    **if** $(proc$ has $i$th row$)$ **then**
        **if** $(sc \neq 1)$ **then** $sc := sc - 1;$
        send $w_{loc}(sc)$ to $left;$
        receive $w_{loc}(sc)$ from $right;$
    **else**
        receive $w$ from $right;$
        **if** $(G_{sc}$ exists$)$ **then**
          apply $G_{sc}$ to $(w, w_{loc}(sc));$
        **end if**
        send $w$ to $left;$
    **end if**
    apply $H_{sc}$ to $w_{loc}(sc:);$
**end for**

FIG. 4. *Modified Householder reflection application.*

Usually, the subspace dimension is much smaller than the matrix dimension and the graph partitioning algorithm produces a balanced workload by assigning an almost equal number of rows to each processor. Thus, the case when the index $s$ within $v_{loc}$ becomes equal to the size of a local partition (size of $v_{loc}$) occurs rarely for large matrices, unless the number of processors is very large.

**3. Comparative communication cost analysis.** In the context of homotopy methods, which are used for a wide range of different problems, an optimal partitioning scheme for sparse matrices cannot be predetermined. Thus, graph theory results are often used to obtain a matrix row partitioning that achieves good load balancing and a small communication to computation ratio.

**3.1. Definition of a communication cost function.** Let $A$ be an $n \times n$ matrix with a symmetric nonzero structure ($a_{ij} \neq 0 \iff a_{ji} \neq 0$). Let $G(A)$ be an undirected graph with $n$ vertices such that there is an edge between the $i$th and $j$th vertices if and only if $a_{ij} \neq 0$. Each vertex of $G(A)$ corresponds to a row of $A$. The vertices of $G(A)$ are partitioned into $p$ disjoint (nonempty) subsets (subdomains) $V_1, \ldots, V_p$, such that each subset belongs to a processor. The edges of $G(A)$ describe the connectivity of vertices within each subdomain and between a pair of subdomains. Figure 5 shows a $12 \times 12$ matrix $A$ and its graph $G(A)$ partitioned into four subdomains.

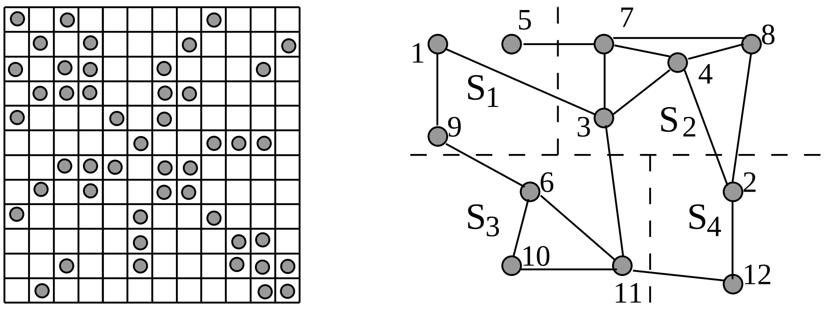


FIG. 5. *Partitioning of* $12 \times 12$ *matrix into 4 subdomains.*

Given a partition $\mathcal{V} = \{V_1, \ldots, V_p\}$, call an *internal edge* an edge of $G(A)$ that has both vertices incident to it in the same $V_i$. Conversely, an *external edge* is an edge incident with two vertices from two different subsets. Following the definition in [22], let *contracting* an edge $e$ of $G(A)$ mean replacing both endpoints of $e$ by a single vertex with edges to those vertices that were adjacent to either of the endpoints of $e$.

Consider a procedure (call it CONTRM) that produces a multigraph corresponding to the communication pattern among processors given a partitioning of a matrix into subdomains.

Step 1. Perform contractions on all internal edges of $G(A)$.

Step 2. Let $W_i$ be the set of vertices left from $V_i$ after step 1. Replace all the vertices in $W_i$ by a single vertex $u_i$ such that the edges incident with it are all the external edges that have one endpoint in $V_i$.

This procedure results in a multigraph $G'(U, E)$, where $U = \{u_1, \ldots u_p\}$ is a set of vertices and $E = \{(l_{ij}, u_i, u_j) \mid l_{ij} = 1, \ldots, m_{ij},\ i \neq j, \text{ and } i, j = 1, \ldots, p\}$ is a set of labeled edges, where $m_{ij}$ is the number of edges between $u_i$ and $u_j$. Figure 6 shows a multigraph $G'(U, E)$ obtained from the graph in Figure 5 by the procedure CONTRM.

When minimizing the communication to computation ratio, a criterion for measuring the quality of the partition $\mathcal{V}$ can be taken as the sum (over all subdomains) of the costs of computation and communication (see, e.g., [14]). The communication mechanism together with the interconnection network and the type of messages affect the communication cost greatly. Communication start-up requires much more time than transmission of data on homogeneous distributed-memory architectures. For example, on the Intel Paragon, the start-up time becomes negligible compared with the transmission time only when the size of a message nears a hundred megabytes. Furthermore, in sparse numerical linear algebra, say in sparse matrix-vector multiplication, an array of values of the same type (floating point numbers) is exchanged by a single message between subdomains. Thus, under some reasonable restrictions on the number of data values sent between any two subdomains, consider the cost function $f : U \times U \to \{0, 1\}$ such that $f(u_i, u_j) = 1$ if *at least one* edge connects $u_i$ and $u_j$, and $f(u_i, u_j) = 0$ otherwise. Then an edge in $E$ incident with both $u_i$ and $u_j$ ($i \neq j$ and $i, j = 1, \ldots, p$) has a cost $1/m_{ij}$, where $m_{ij}$ is the number of edges incident with $u_j$ and $u_j$. For the multigraph shown in Figure 6, the total cost is equal to 5.

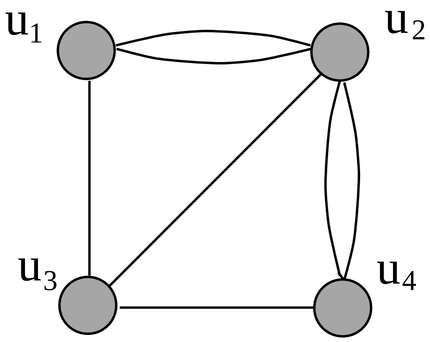


FIG. 6. *Contraction of $G(A)$ to a multigraph using the procedure CONTRM.*

Since establishing communication overwhelms the other phases of communication and since a single message can contain all the data to be exchanged between adjacent vertices in $G'(U, E)$, minimizing the sum of edge costs would minimize of the communication cost for the partition $\mathcal{V}$. Note that this cost function may not be valid for other application domains, when each edge in $E$ corresponds to some unique piece of information, and adopting some other cost function may be appropriate (see, e.g, [13]). A number of graph-theoretical heuristics exist to produce partitioning (call it *vertex-based*) into subdomains with the load balanced and a minimum number of external edges (minimum number of edges that connect different $V_i$ in $\mathcal{V}$). For parallel implementations considered here, a recently developed graph partitioning algorithm, Multilevel Recursive Bisection [11], is used. This algorithm follows a multilevel graph partitioning scheme that consists of three phases: reducing the size of a graph (coarsening), partitioning a smaller graph, restoring the graph to its original size (uncoarsening) [9], [10]. In general, this algorithm produces a partition in which all external edges are treated uniformly; that is, all edges in $G'(U, E)$ have the same cost (say, 1). Thus, the total communication cost of this partition is the total number of edges. For the multigraph shown in Figure 6 this cost equals 7, which is clearly not minimum, for the type of communication considered here.

However, for very large problems, when communication start-up time is small compared with the data transmission time, minimizing the number of external edges also achieves a minimum of the communication cost. Since the graph partitioning algorithm is designed to balance the load among subdomains, the communication to computation ratio can be smaller compared with a partitioning that has less communication cost but also a less balanced load.

**3.2. Experimental results.** To compare the Householder reflections restricted to the block-striped partitioning with the proposed extension for the vertex-based partitioning, GMRES was instrumented to collect the timing information relevant only to the parallel Householder reflection generation and application. The time spent in HG and MHG for generating and applying a global Householder reflection vector is independent of the type of partitioning and thus presents a good measure of the relative computational complexity of these two implementations. As a test problem, the matrix BCSSTK13 ($2003 \times 2003$) was taken from the Harwell-Boeing collection [5]. This matrix is the stiffness matrix for a fluid flow eigenvalue problem from a dynamic analysis in structural engineering. BCSSTK13 has 83883 nonzeros and a symmetric sparsity pattern [5].

MHG and HG generated the first Householder vector in 0.26s and 0.25s, respectively, whereas MHG was slightly faster (0.23s and 0.25s, respectively) than HG in applying a Householder reflection. (The times quoted here are the averaged parallel times over 3 experiments performed on 8 processors of an Intel Paragon.) Thus, the proposed MHG implementation exhibits performance comparable to HG.

Table 1
*Total and matrix-vector multiplication parallel times for GMRES(15) with MHG and HG orthogonalizations.*

| | | 4 | 8 | 16 | 32 | 64 | 100 |
|---|---|---|---|---|---|---|---|
| MHG | $T_p$ | 1.52 | 2.21 | 3.23 | 5.30 | 8.84 | 12.49 |
| | $T^{MV}$ | 0.36 | 0.29 | 0.34 | 0.42 | 0.59 | 0.76 |
| | $np$ | 3 | 7 | 12 | 17 | 24 | 25 |
| | $Comm$ | 6 | 22 | 74 | 177 | 457 | 808 |
| HG | $T_p$ | 1.66 | 1.91 | 3.37 | 5.25 | 8.85 | 12.62 |
| | $T^{MV}$ | 0.34 | 0.23 | 0.29 | 0.42 | 0.76 | 1.09 |
| | $np$ | 3 | 6 | 13 | 13 | 22 | 30 |
| | $Comm$ | 6 | 17 | 55 | 135 | 388 | 699 |

For the test matrix BCSSTK13, two restart cycles of GMRES(15) were executed to determine the dependence of the overall parallel time on the type of partitioning (and thus on HG and MHG) with an increase in the number of processors. Table 1 shows the averaged results over three experiments (largest standard deviation was 0.02) for 4, 8, 16, 32, 64, and 100 processors, where $T_p$ denotes the overall parallel time for GMRES(15), $T^{MV}$ denotes the parallel time for matrix-vector multiplication, $np$ is the maximum number of neighboring processors, and $Comm$ is the total number of communication channels among processors. Comparison of the last two columns of Table 1 shows that with increasing numbers of processors the performance of GMRES with the MHG orthogonalization deteriorates slower than that of GMRES with the HG orthogonalization. On 100 processors, GMRES with MHG organizes communications among processors more uniformly by having a smaller maximum number of neighbors. Thus, with more neighbors HG performs worse than MHG for large numbers of processors.

**4. Comparative scalability analysis.** An algorithm-architecture scalability analysis estimates, in a single expression, characteristics of the algorithm as well as parameters of the architecture on which the algorithm is implemented. Thus, testing an algorithm on a few processors allows one to predict its performance on a larger number of processors. The following terminology is needed to support the proposed scalability analysis. A *parallel system* is a parallel algorithm and machine combination. The *useful work* $W$ is the total number of basic floating point operations required to solve the problem. $W$ depends on the *problem size* $\bar{N}$, which is a vector of problem-specific parameters such as problem dimensions and the number of nonzero entries in a matrix. For numerical linear algebra problems solved by iterative methods, $\bar{N}$ may also include an indication of problem difficulty, such as the Krylov subspace dimension $k$ used in GMRES($k$) or the condition number. In general, choosing $\bar{N}$ requires a detailed investigation of a given problem and of the way scaling of problem dimensions affects the increase in $W$ [17]. The *sequential execution time* $T_1$ characterizes the time interval needed to solve a given problem on a uniprocessor. If the time of executing an integer operation is negligible compared with the time $t_c$ of performing a floating point operation and if $T_1$ is spent in useful work only, then $T_1 = t_c W$. The *parallel execution time* $T_p$ is the time taken by $p$ processors to solve the problem. The *total parallel overhead* $V$ is the sum of all overheads incurred by all the processors during parallel execution of the algorithm. For a given parallel system, $V$ is a function of the useful work $W$ and number of processors $p$: $V = pT_p(W, p) - T_1(W)$. The *efficiency* $E$ is the ratio of the speedup $S(p)$ to $p$, where the speedup $S(p) = T_1/T_p$. Hence, $E = \frac{T_1}{pT_p} = \frac{1}{(1+V/T_1)}$. A parallel system is called *scalable* if $V = \mathcal{O}(W)$ and unscalable otherwise. For scalable systems, it is possible to keep the efficiency fixed and to monitor the rate of increase in $W$ and $p$ with the *isoefficiency function* $f_E(p)$, as proposed by Kumar [7]. The isoefficiency function is defined implicitly by the following relation between $W$ and the parallel overhead $V$:

$$W(\bar{N}) = eV(W(\bar{N}), p, \bar{h}), \tag{1}$$

where $e = \frac{E}{t_c(1-E)}$ is a constant and $\bar{h}$ is a vector of machine-dependent parameters affecting the amount of the parallel overhead. Usually, the communication cost of the total parallel overhead incorporates these parameters, which are defined in accordance with a communication model supported by a given architecture.

**4.1. Operation count for the useful work.** Let $N$ be the scaled matrix dimension and $N_z$ be the number of nonzeros in the scaled matrix. To obtain an operation count for the useful work in GMRES($k$) at the $j$th iteration the following parts of the pseudocode for AGMRES($k$) can be distinguished:

*1.* matrix-vector multiplication, which takes $N_z$ operations;

*2.* Householder reflection generation, which takes $2(N - j + 1)$ operations;

*3.* Householder reflection application, which requires $4\sum_{i=1}^{j}(N - i + 1)$ operations.

The remaining work is accomplished in $\mathcal{O}(k)$ operations. For a single restart cycle, the useful work is

$$\begin{aligned} W &\approx (k+1)N_z + 2\sum_{j=1}^{k+1}(N - j + 1) + N + \sum_{j=1}^{k}\left(4\sum_{i=1}^{j}(N - i + 1) + (N - j + 1)\right), \\ &\approx (k+1)N_z + N(2k^2 + 5k + 3) + C_k, \end{aligned} \tag{2}$$

where $C_k$ is a term depending only on $k$.

**4.2. Description of the test problem.** The scalability analysis is performed on a real-world problem representing a commercial circuit design at AT&T Bell Laboratories. The dimension of this problem is $n = 125$ and the number of nonzeros $n_z = 782$. Here scaling the problem $K$ times means assembling $K$ replicas of the $n \times n$ matrix of coefficients in an $N \times N$ matrix, where $N = K \times n$ and the number of nonzeros $N_z = K \times n_z$, as shown in Figure 7 for $K = 5$. Both the initial solution vector and right-hand side are $e_N^t = (0, \ldots, 0, 1)^t$. A $K$-block diagonal structure of the matrix was purposely avoided to distinguish between block-striped assignment of rows to processors (used in HG) and vertex-based assignment (used in MHG). However, since the matrix does possess a specific (block) structure, the effect of using graph partitioning algorithms to distribute rows may not be pronounced compared with the block-striped partitioning.

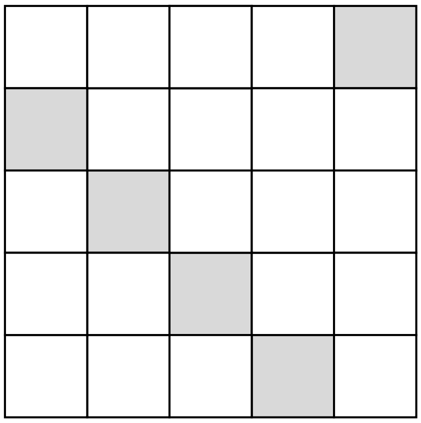

FIG. 7. *Scaling of problem size.*

**4.3. Derivation of the isoefficiency function.** Equation (1) takes the following form when GMRES($k$) is used for solving the test problem:

$$W(Kn, k) = eV(W(Kn, k), p, \bar{h}), \tag{3}$$

where $\bar{h} = (t_s, t_w)^t$. Parallel GMRES($k$) with Householder reflection orthogonalization incurs an overhead in matrix-vector multiplication, Householder reflection generation and application, and in the residual norm update, which is performed on a single processor.

The isoefficiency function is derived under the following assumptions for the serial time calculation, graph partitioning of a matrix, and communication handling: (a) all the work in the sequential algorithm is considered useful; (b) graph partitioning algorithm produces balanced partitions; (c) for large scale problems, partitions produced by graph partitioning keep the computation to communication ratio no smaller than block-striped partitioning; (d) each processor holds $Kn/p$ rows of the matrix (Figure 7), where $p \leq K$; (e) if $p \leq K$, then each $n \times n$ matrix block is partitioned between no more than two processors; (f) in matrix-vector multiplication, communication is performed by asynchronous sends and synchronous receives; (g) the time complexity of the MPI broadcast operation implemented on the Intel Paragon, IBM SP2, and Cray T3E does not grow substantially as the number of processors increases. Note that, since for the majority of realistic applications the amount of computation grows superlinearly in the number of processors, assumption (d) is not very constraining.

The parallel overhead due to a matrix-vector multiplication can be predicted by considering the nonzero structure of a given coefficient matrix. For the type of matrices shown in Figure 7, a processor receives at most $n$ vector components from no more than $\lceil p/K \rceil$ processors and sends its $Kn/p$ vector components to at most $\lceil p/K \rceil$ processors. Let $V_{r_j}^{MV}$ and $V_{s_j}^{MV}$ be the total overheads incurred by the processors at data receiving and sending stages of a matrix-vector multiplication, respectively. Then the total parallel overhead of the matrix-vector multiplication at the $j$th iteration is $V_j^{MV} = V_{r_j}^{MV} + V_{s_j}^{MV}$ with

$$V_{r_j}^{MV} \approx \left( \left\lceil \frac{p}{K} \right\rceil t_s + n t_w + C_0 \right) \times p \quad \text{and} \quad V_{s_j}^{MV} \approx \left( \left\lceil \frac{p}{K} \right\rceil t_s + \frac{Kn}{p} t_w - C_1 \right) \times p,$$

where the constants $C_0$ and $C_1$ describe the waiting and communication-computation overlapping times for asynchronous communications, respectively. For the problem sizes considered here, $Kn/p \geq n$, and thus $\lceil p/K \rceil t_s \leq t_s$. Combining $V_{r_j}^{MV}$ and $V_{s_j}^{MV}$ yields

$$V_j^{MV} \approx \left( 2t_s + \left( n + \frac{Kn}{p} \right) t_w + C_0 - C_1 \right) \times p. \tag{4}$$

Observe that $C_0 - C_1 \approx 0$ since the waiting time of each processor during asynchronous matrix vector multiplication is compensated by the time gain in communication-computation overlapping while sending the information. Thus, $C_0 - C_1$ will be dropped from the expression for $V_j^{MV}$.

Householder reflection generation and application cause a noticeable communication and nonessential work overhead. At the $j$th GMRES($k$) iteration, the overhead $V_j^H$ due to Householder reflection generation and application comprises the overheads $V_{a_j}^H$ and $V_{c_j}^H$ caused by applying and generating Householder reflections, respectively, such that $V_j^H = 2V_{a_j}^H + V_{c_j}^H$ with

$$V_{a_j}^H \approx \Big[ jp\big(2(t_s + t_w) + g_a\big) \Big] \times p \quad \text{and} \quad V_{c_j}^H \approx \Big[ p\big(2(t_s + t_w) + g_c\big) \Big] \times p, \tag{5}$$

where $g_c$ is the number of operations needed to create a Givens rotation and $g_a$ is the number of operations needed to perform a Givens rotation to zero out one vector component. Since Householder reflections are applied twice per GMRES iteration, $V_{a_j}^H$ has a coefficient of two.

Another source of the parallel overhead appears in estimating the condition number of the GMRES least squares problem, which is done on a single processor. For a typical case, when the Krylov subspace dimension is $j, j = 1, \ldots, k$, gathering $\mathcal{O}(j)$ vector components on a processor, estimating the condition number incrementally, and updating the residual norm are relatively inexpensive, since the subspace dimension is much smaller than the matrix dimension $N$ for large scale problems. Global all-to-all communication would be required to perform the condition number estimation in parallel. At the $j$th GMRES($k$) iteration, the parallel overhead $V_j^{IC}$ is caused by the time spent to exchange $\mathcal{O}(j)$ values, to update the residual norm by application of $j$

previous Givens rotations and by generation of a new Givens rotation, and to perform approximately $(j-1)$ operations of the incremental condition number estimation. Thus,

$$V_j^{IC} \approx \Big[2j(t_s+t_w)+jg_a+g_c+(j-1)\Big]\times p. \tag{6}$$

The total parallel overhead $V_j$ ($V_j = V_j^{MV}+V_j^H+V_j^{IC}$) incurred at the $j$th iteration of the GMRES($k$) algorithm with Householder reflections in its orthogonalization stage is

$$\begin{aligned} V_j \approx & \Big[2t_s+\Big(n+\frac{Kn}{p}\Big)t_w\Big]\times p \\ & + \Big[2jp\Big(2(t_s+t_w)+g_a\Big)+p\Big(2(t_s+t_w)+g_c\Big)\Big]\times p \\ & + \Big[2j(t_s+t_w)+jg_a+g_c+(j-1)\Big]\times p. \end{aligned}$$

When a restart cycle is finished, i.e., $j=k$, the GMRES($k$) algorithm has performed $k$ matrix-vector multiplications, $k+1$ Householder reflection generations, $k$ residual norm updates along with $k$ incremental condition number estimates, and $2k$ Householder reflection applications. At the end of a restart, GMRES($k$) calculates the true residual norm using one more matrix-vector multiplication and one more Householder reflection application to correct the current solution. Combining all the overhead terms incurred during $k$ iterations, the expression for the total overhead is

$$V=\sum_{j=1}^{k+1}\Big(V_j^{MV}+V_{c_j}^H\Big)+\sum_{j=1}^{k}\Big(2V_{a_j}^H+V_j^{IC}\Big)+V_{a_k}^H. \tag{7}$$

Substituting equations (4), (5), and (6) into this equation results in

$$V\approx C_g p^2+(\bar{h}^t\bar{c}+C_k')p+Kn(k+1)t_w,$$

where

$$C_g = k(k+2)\Big(2(t_s+t_w)+g_a\Big)+(k+1)\Big(2(t_s+t_w)+g_c\Big),$$

$$C_k' = k\left(\frac{(k+1)}{2}g_a+\frac{(k-1)}{2}+g_c\right),$$

and $\bar{h}=(t_s,t_w)^t$, $\bar{c}=(c_1,c_2)^t$ with $c_1=k^2+3k+2$ and $c_2=k^2+(n+1)k+n$.

The expression for $V$ is a quadratic polynomial in $p$. Thus for a fixed $k$, the fastest growing term is the leading term. The leading term in $V$ comes from creating and applying Givens rotations on processors logically connected in a ring.

In the test problem considered here, $N_z\approx 6Kn$. Thus $W\approx Kn(2k^2+11k+9)+C_k$. Substituting the expressions for V and W into equation (3) gives

$$Kn\Big[(2k^2+11k+9)-e(k+1)t_w\Big]+C_k\approx\ e\times\Big[C_g p^2+(\bar{h}^t\bar{c}+C_k')p\Big]. \tag{8}$$

The following relation can be derived for $K$:

$$K\approx\ e\times\Big[\frac{C_g}{n\,(2k^2+11k+9-e(k+1)t_w)}p^2+\frac{\bar{h}^t\bar{c}+C_k'}{n\,(2k^2+11k+9-e(k+1)t_w)}p\Big]. \tag{9}$$

Note that since the term containing $C_k$ is small and has no dependence on either $K$ or $p$, it does not appear in equation (9).

If one considers the GMRES($k$) solution process as consisting of $l$ restart cycles and including the computation of the initial residual $r_0$ (which requires one matrix-vector multiplication and one subtraction), then equation (4) can be rewritten as

$$l\times Kn\Big[(2k^2+11k+9)+C_k-e(k+1)t_w\Big]+Kn(7-et_w) \approx l\times e\Big[C_gp^2+(\bar{h}^t\bar{c}+C_k')p\Big]. \quad (10)$$

Comparison of equations (8) and (10) suggests that the isoefficiency function value is smaller if the initial residual computation is taken into consideration, since the useful work becomes larger and makes the leading term coefficient of equation (9) smaller.

**5. Numerical experiments.** The behavior of six different parallel systems has been studied. An algorithm component of each parallel system is either GMRES($k$) with HG orthogonalization or GMRES($k$) with MHG orthogonalization and an architecture component is one of three parallel computers: IBM SP2, Cray T3E, or Intel Paragon. For convenient reference, the following abbreviations denote these parallel systems:

MHG_P — GMRES($k$) with MHG executed on the Intel Paragon;
MHG_S — GMRES($k$) with MHG executed on the IBM SP2;
MHG_T — GMRES($k$) with MHG executed on the Cray T3E;
HG_P — GMRES($k$) with HG orthogonalization on the Intel Paragon;
HG_S — GMRES($k$) with HG orthogonalization on the IBM SP2;
HG_T — GMRES($k$) with HG orthogonalization on the Cray T3E.

Both the algorithm and the architecture were subject to scaling in the experiments, where scaling an algorithm means increasing the problem size and scaling an architecture is a mapping of an application to increasing numbers of processors. To examine the isoefficiency scalability of these parallel systems, the (processor number, problem size) pairs with the same efficiency were selected. For the efficiency computation, the parallel time needed to perform two GMRES($k$) restart cycles and the initial residual $r_0$ computation by a parallel system was recorded as well as the time for executing the same algorithm on a uniprocessor. Three repetitions of each experiment were performed (the plots show the average of these three values). For each parallel architecture, the maximum coefficient of variation for the measured data (parallel time) in three repetitions is determined as 0.09, 0.14, and 0.08, for the IBM SP2, Intel Paragon, and Cray T3E, respectively.

**5.1. Architectures and machine-dependent constants.** Numerical experiments were performed on three parallel computing platforms: an IBM SP2 (at Virginia Tech and NASA Langley Research Center), an Intel Paragon (at Virginia Tech and the Center for Computational Sciences at Oak Ridge National Laboratory), and a Cray T3E (at the National Energy Research Scientific Computing Center). To calibrate the expression of the isoefficiency function, the constants $t_c$, $t_s$, and $t_w$ are determined for the target parallel computers. Their numerical values are obtained using simple appropriate models, for which a sufficient amount of empirical data can be collected to estimate accurately these values. The time $t_c$ needed to perform a floating point operation is obtained as the parameter of a linear regression model $T_1 = t_c W'$, where $W'$ includes an operation count for $W$ and additional terms capturing the effects of memory hierarchy operations when solving large scale problems by a sequential algorithm. Clearly, the value of $W'$ differs from one parallel architecture to another depending on such factors as the memory size and the interface mechanism among memory layers and processor. The amount of time that accounts for the memory hierarchy operations can often be modeled only by studying particular cases of a problem solved on a given architecture. On supercomputers with small cache size and a centralized I/O mechanism, such as the Intel Paragon (16KB of level-one on-board data cache), slow memory (paging) operations affect the overall computation time considerably. In a given sequential algorithm, this effect is noticeable already for medium-size ($N \approx 5000$) problems. Thus the cost of memory accesses (the cost of load/store operations) to compute a floating point value is added to the overall number of operations performed by the sequential algorithm. To perform a floating point operation, two loads from slow memory and one store operation are needed in the worst case.

Observation of the performance of the sequential algorithm on the IBM SP2 also suggests that a portion of its execution time is spent on fetching data from a slower memory. Although an IBM SP2 processor has a significant amount of data cache (64KB and 256KB for Models 390 and 590, respectively), this cache is not placed on chip, thus the interface with the processor presents a bandwidth bottleneck. Also, neither Model 390 nor Model 590 has level-two cache, which plays an important role in floating point computations. Solution of large scientific problems on the IBM SP2 causes a high cache miss rate and, subsequently, frequent references to main memory. Similarly to the Intel Paragon, the useful work $W$ on the IBM SP2 was augmented by the cost of the load and store operations. The total work of a sequential algorithm $W'$ is approximately equal to $3\Big[(k+1)N_z + N(2k^2+5k+3) + C_k\Big]$ on the Intel Paragon and IBM SP2.

On the other hand, each processor of the Cray T3E is coupled with 8KB of data cache and a 96KB level-two data cache. In addition, there are another 4MB of on-board data cache. This is enough cache capacity to hold all the floating point data of the problem sizes considered here with a high cache hit rate. Thus, $W' \approx W$ is acceptable in this case.

The communication start-up time and the transmission time of a double precision number are determined from a communication cost model $T_{comm} = t_s + t_w L$, where $T_{comm}$ is the response variable and $L$ is the predictor variable. This model estimates the communication time $T_{comm}$ between two processors under the assumption that the time required to send a message from one processor to another is independent of processor location and the number of other processors that might be communicating at the same time. The experimental data were gathered from measuring the time $T'_{comm}$ needed to exchange a message between two processors ($T'_{comm} = 2T_{comm}$).

For the IBM SP2, the linear regression $T_1 = 0.024W'$ models the uniprocessor CPU time with the standard deviation of errors equal to 1.84. The regression explains 81% of the variation in $T_1$, because the coefficient of determination of this model is 0.81. For the linear regression $T_{comm} = 69 + 0.45L$, the standard deviation of errors is 9.6 and the coefficient of determination is 0.97.

For the Intel Paragon, the linear model is $T_1 = 0.33W'$ with the standard deviation of errors equal to 2.25. The regression explains 99% of the variation in $T_1$. The standard deviation of values $T_{comm}$ observed in each repetition of the experiment was very large. However, if the message length is known or lies within a certain range, some constant value of $T_{comm}$ can be estimated by the linear regression within that specific range. In particular, if a small array of double precision constants is transmitted, the communication latency can be approximated by the start-up time, i.e., $T_{comm} = 605\mu$s.

For the Cray T3E, the linear model is $T_1 = 0.019W'$ with the standard deviation of errors equal to 0.07. The regression explains 99% of the variation in $T_1$. To estimate the communication time $T_{comm}$, the linear regression $T_{comm} = 50 + 0.16L$ is obtained. For this regression, the standard deviation of errors is 6.3 and the coefficient of determination is 0.91.

As a result, the times spent for computing a million floating point operations are 0.024s, 0.33s, and 0.019s for the IBM SP2, Intel Paragon, and Cray T3E, respectively. The reciprocal of $t_c$ defines uniprocessor computing rates of 41.6 Mflop/s, 3.0 Mflop/s, and 53 Mflop/s, correspondingly. Start-up–transmission time pairs (in microseconds) are (69, 0.45) and (50, 0.16) for the IBM SP2 and Cray T3E, respectively.

**5.2. Isoefficiency of GMRES($k$) on the IBM SP2.** Least squares fits $f_E$ of a quadratic polynomial to HG_S data in Figure 8 for efficiencies .14, .22, .34, and .42 with $k = 15$, respectively, are

$$\begin{aligned} &0.17p^2 - 1.11p, \qquad 0.31p^2 - 4.15p; \\ &0.48p^2 - 6.06p, \qquad 0.74p^2 - 7.42p. \end{aligned}$$

Least squares fits to MHG_S data in Figure 9 for efficiencies .14, .22, .34, and .42 with $k = 15$, respectively, are

$$\begin{aligned} &0.10p^2 + 5.15p, \qquad 0.17p^2 + 3.14p; \\ &0.40p^2 - 1.17p, \qquad 0.56p^2 - 4.31p. \end{aligned}$$

Predicted isoefficiency functions for the same efficiences and the same value of $k$, respectively, are

$$\begin{aligned} &0.13p^2 + 0.09p, \qquad 0.22p^2 + 0.12p; \\ &0.41p^2 + 0.22p, \qquad 0.58p^2 + 0.30p. \end{aligned}$$

Least squares fits to MHG_S data in Figure 10 for efficiencies .22 and .42 and $k = 25$, respectively, are

$$0.24p^2 + 2.03p \qquad \text{and} \qquad 0.75p^2 - 5.06p.$$

Predicted isoefficiency functions for efficiences .22 and .42 with $k = 25$, respectively, are

$$0.26p^2 + 0.14p \qquad \text{and} \qquad 0.66p^2 + 0.36p.$$

Each least squares approximation grows in accordance with the corresponding predicted isoefficiency. Thus, the same function $f_E$ can be used to estimate the isoefficiency scalabilities of MHG_S and HG_S. However, the leading term coefficients for

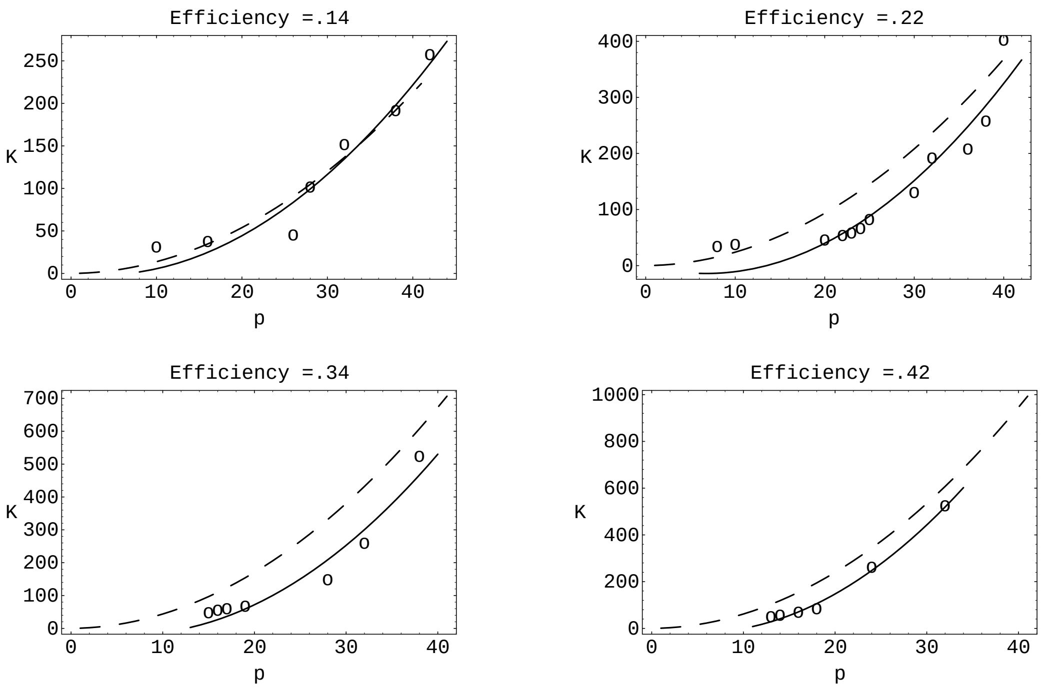


FIG. 8. *Isoefficiency curves for HG_S with $k = 15$. Dashed line—theoretical; solid line—fit to data.*

MHG_S are slightly smaller than the leading term coefficients for HG_S, which implies that MHG_S may be more scalable. As reflected in the form of the isoefficiency function, the larger the efficiency to be maintained, the larger the increase in the problem size that is required with scaling of an architecture component. In particular, the leading term coefficient in $f_E$ grows with the fixed efficiency parameter. Consider, for example, $f_E$ with $E = .14$ and $E = .42$, where the coefficient of the quadratic term increases by a factor of 4.5.

When the restart parameter $k$ increases, the leading term coefficient also increases. This variation in the isoefficiency function value is in agreement with consideration of the whole restart cycle ($k$ iterations) in the isoefficiency analysis of GMRES($k$), which differs in this sense from the analysis of the preconditioned conjugate gradient method conducted by Gupta and Kumar [8] for a single iteration of the method.

**5.3. Isoefficiency of GMRES($k$) on the Intel Paragon.** Least squares fits to HG_P data in Figure 11 (top) for efficiencies .28, and .46 with $k = 15$, respectively, are

$$0.03p^2 + 1.98p \qquad \text{and} \qquad 0.13p^2 + 0.05p.$$

Least squares fits to MHG_P data in Figure 11 (bottom) for efficiencies .28, and .46 with $k = 15$, respectively, are

$$0.05p^2 + 1.58p \qquad \text{and} \qquad 0.13p^2 + 0.54p.$$

Predicted isoefficiency functions for the same efficiences and value of $k$, respectively, are

$$0.07p^2 + 0.04p \qquad \text{and} \qquad 0.15p^2 + 0.08p.$$

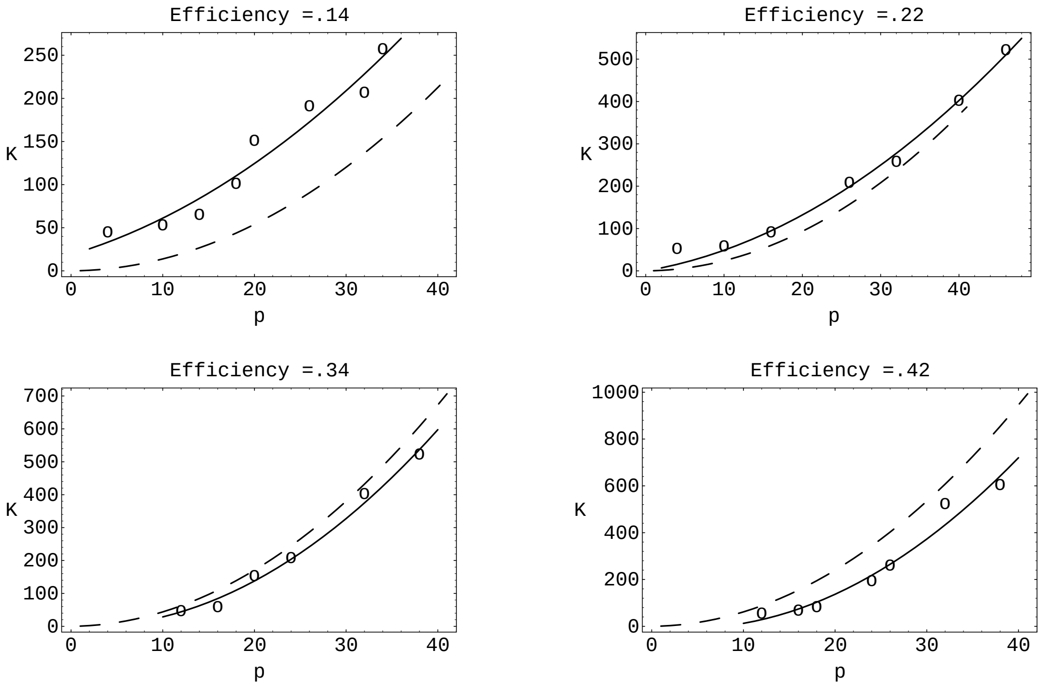


FIG. 9. *Isoefficiency curves for MHG_S with $k = 15$. Dashed line—theoretical; solid line—fit to data.*

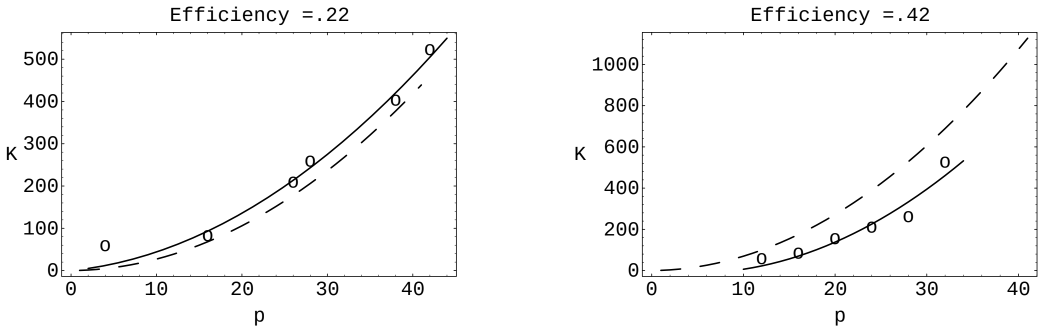


FIG. 10. *Isoefficiency curves for MHG_S with $k = 25$. Dashed line—theoretical; solid line—fit to data.*

Least squares fits to HG_P data in Figure 12 for efficiencies .28, and .46 with $k = 35$, respectively, are

$$0.08p^2 + 0.28p \qquad \text{and} \qquad 0.18p^2 - 1.14p.$$

Predicted isoefficiency functions for the same efficiences and $k = 35$, respectively, are

$$0.08p^2 + 0.04p \qquad \text{and} \qquad 0.18p^2 + 0.09p.$$

For the Intel Paragon parallel systems, each least squares approximation grows similarly to the corresponding predicted isoefficiency.

**5.4. Isoefficiency of GMRES($k$) on the Cray T3E.** Least squares fits to HG_T data in Figure 13 (top) for efficiencies .24 and .36, respectively, are

$$0.45p^2 - 1.87p \qquad \text{and} \qquad 0.70p^2 - 5.36p.$$

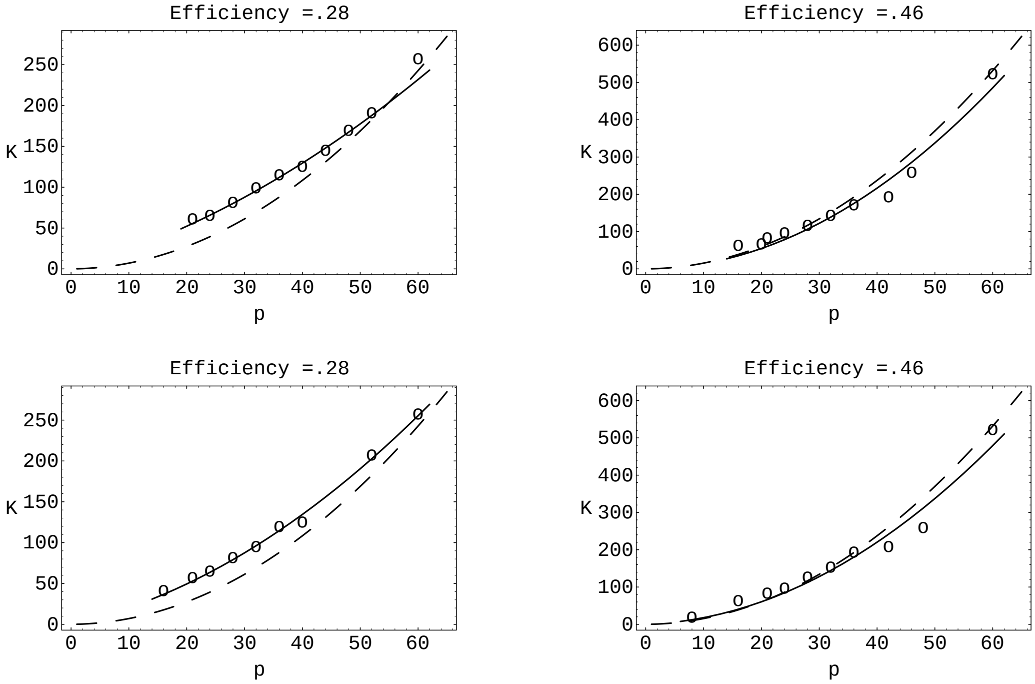


FIG. 11. *Isoefficiency curves for HG_P ($k$ = 15, top) and MHG_P ($k$ = 15, bottom). Dashed line—theoretical; solid line—fit to data.*

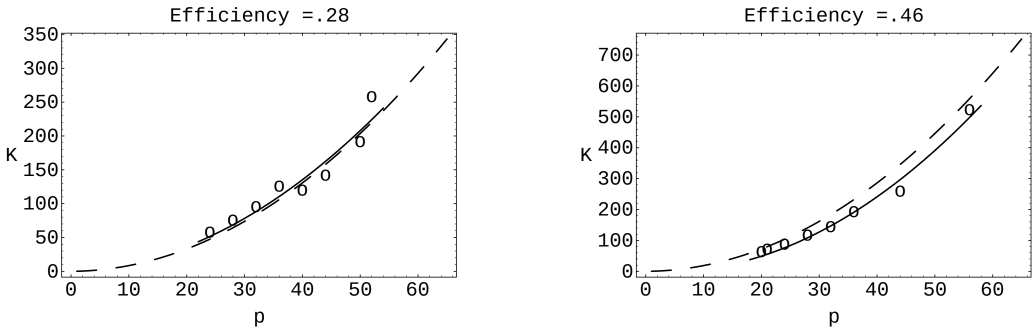


FIG. 12. *Isoefficiency curves for HG_P with $k$ = 35. Dashed line—theoretical; solid line—fit to data.*

Least squares fits to MHG_T data in Figure 13 (bottom) for efficiencies .24 and .36, respectively, are

$$0.32p^2 + 1.87p \qquad \text{and} \qquad 0.58p^2 - 2.71p.$$

Predicted isoefficiency functions for the same efficiences, respectively, are

$$0.33p^2 + 0.18p \qquad \text{and} \qquad 0.59p^2 + 0.32p.$$

For the Cray T3E parallel systems, each least squares fit grows similarly to the corresponding predicted isoefficiency.

Since the architectures considered here have different values of machine-dependent constants, the corresponding isoefficiency functions have different coefficients, even though they include terms of the same order. For example, the isoefficiency function

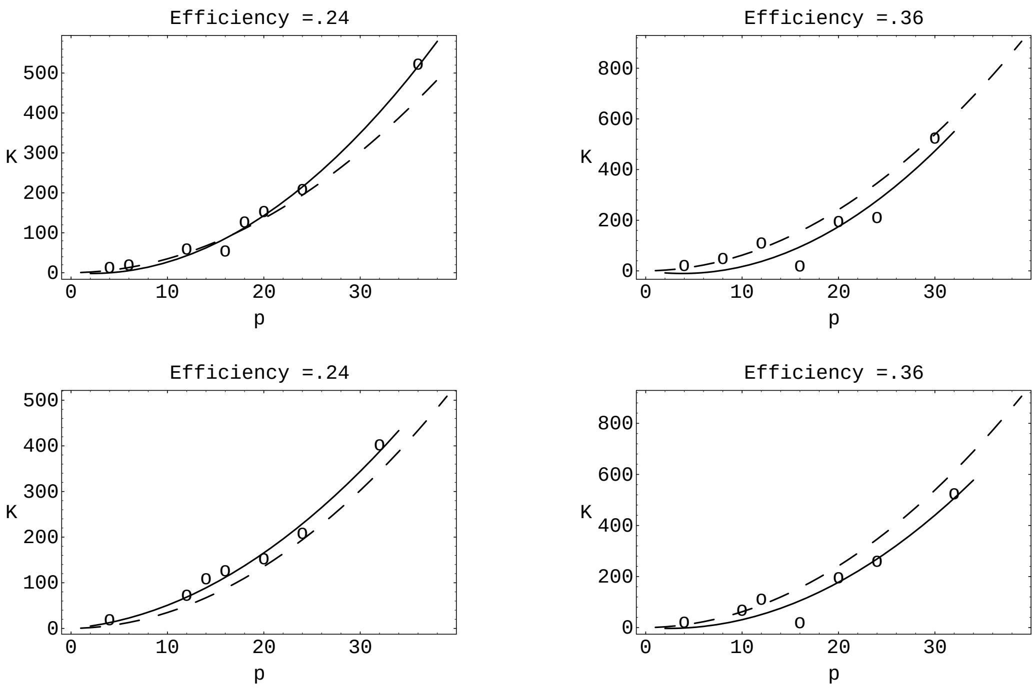


FIG. 13. *Isoefficiency curves for HG_T ($k$ = 15, top) and MHG_T ($k$ = 15, bottom). Dashed line—theoretical; solid line—fit to data.*

$f_{E_T}$ on the Cray T3E differs significantly from the isoefficiency function $f_{E_S}$ on the IBM SP2. In particular, for $E = 0.42$,

$$f_{E_T} = 0.76p^2 + 0.48p \qquad \text{and} \qquad f_{E_S} = 0.58p^2 + 0.33p.$$

The coefficient of the leading term of $f_{E_T}$ is larger than that of $f_{E_S}$ since the computing rate $1/t_c$ on the Cray T3E is faster than on the IBM SP2, while the improvement in $t_s$ and $t_w$ is not sufficient to hide the communication latency. Hence, the idling time on the Cray T3E is larger than on the IBM SP2. The isoefficiency function of the Paragon architecture presents the case when moderate values for *all* the machine-dependent constants involved result in good isoefficiency scalability characteristics. However, with scaling of the problem size, less powerful architectures, such as the Intel Paragon, tend to exhaust their resources faster than more powerful ones, such as the IBM SP2 and the Cray T3E.

**6. Efficiency of adaptive GMRES($k$).** The erratic behavior of the efficiency values for adaptive GMRES($k$) (Figure 14 top and bottom left plots) on the Paragon suggests that not only the convergence for adaptive GMRES($k$) differs from that for the restarted GMRES(see Section 2), but also their parallel algorithm-architecture performances differ.

In the case of problem size scaling considered here, changing matrix dimensions ($K$ times) can lead to a variation in test problem difficulty as well as in its size. This often happens in practical applications, such as postbuckling analysis of structures, with increase in problem size. By definition, the adaptive GMRES($k$) algorithm proceeds qualitatively differently on different problems. Therefore, for the adaptive

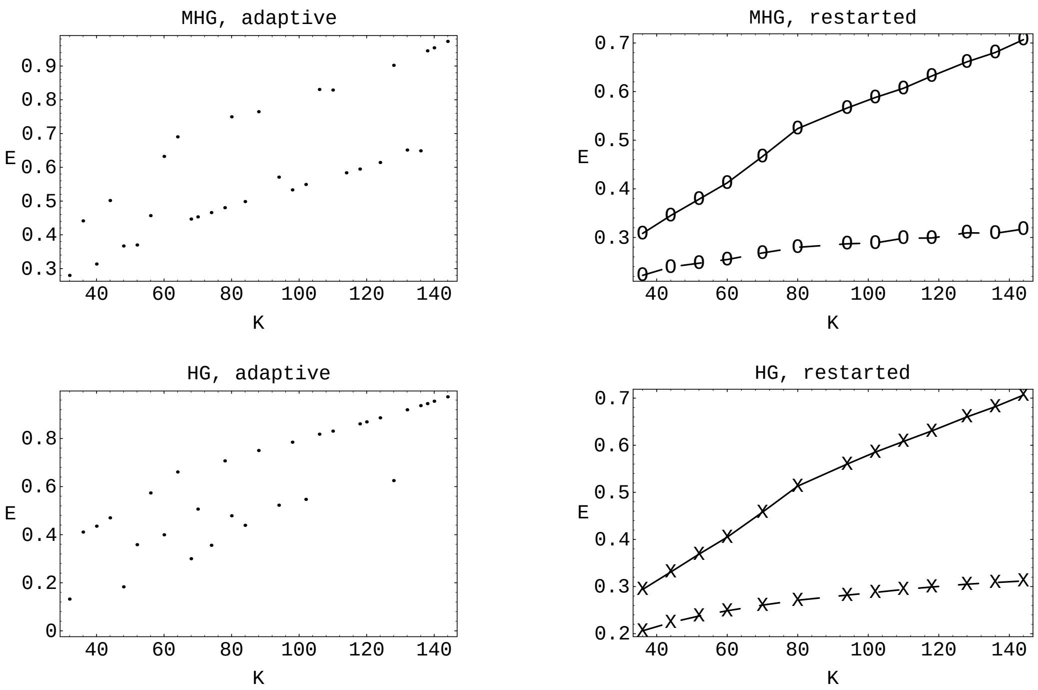


FIG. 14. *Efficiency on 21 processors of Intel Paragon, GMRES(kmax) (solid lines), GMRES(k) (dashed lines).*

GMRES($k$) algorithm, the operation count of the full convergence history (or of the convergence history until the maximum subspace dimension is reached) represents the useful work to be considered in the isoefficiency analysis. Also, from the equation $W(Kn, k) = eV(W(Kn, k), p, \bar{h})$, the parallel overhead includes terms depending on $k$ as a parameter. In the previous section, several numerical experiments deal with computing the isoefficiency function for different fixed values of $k$. These experiments indicate that, as predicted theoretically, the isoefficiency function increases with increase in $k$. However, if it is not known in advance when increases in $k$ occur during the solution process, for example, in adaptive GMRES($k$), the isoefficiency characteristics of adaptive GMRES($k$) are unpredictable. Variation in the Krylov subspace dimension makes the scalability analysis of the adaptive GMRES($k$) algorithm more complicated than the analysis of the conjugate gradient method [8], where an operation count and parallel overhead per iteration is sufficient to derive an isoefficiency function. By the same token, the definition of both useful work and parallel overhead differ from their expressions in the analysis of the GMRES($k$) method, which can be performed for a particular subspace dimension $k$ and a particular operation count of a single GMRES iteration.

One approach to estimating the efficiency of the adaptive GMRES($k$) algorithm on scaled problems is to introduce a measure of problem difficulty into the problem size definition. Since it is hard to predict convergence of the adaptive GMRES($k$) method on an arbitrary problem, making problems more uniform by preconditioning is another way of dealing with the issue of useful work scaling for the purpose of an isoefficiency analysis.

**7. Conclusions.** The developed parallel implementation of adaptive GMRES($k$) takes advantage of an efficient distributed matrix-vector multiplication and the Householder reflection orthogonalization that avoids all-to-all communications and has a high degree of parallelism. The modification proposed here extends the parallel Householder reflection orthogonalization from accepting a block-striped partitioning of rows to a general vertex-based partitioning. For a large number of processors, using the extended parallel implementation of Householder reflections preserves the advantage gained from applying graph-theoretical heuristics to partition a problem into subdomains.

In this paper, the isoefficiency analysis is carried out for parallel versions of GMRES($k$) with Householder reflection orthogonalization implemented on the IBM SP2, Intel Paragon, and Cray T3E. The theoretical part of this analysis not only establishes an asymptotic relation between the increase in problem size and number of processors, but also provides an analytic expression for the isoefficiency function. On target parallel architectures, experimental results support the claim and match closely predicted isoefficiency functions. Both theoretical and experimental results show that the isoefficiency function is a quadratic polynomial with a small leading term coefficient, which implies that the given parallel GMRES($k$) implementations are reasonably scalable on the given parallel architectures, namely, on an IBM SP2, Intel Paragon, and Cray T3E. In general, a parallel algorithm is considered scalable if the isoefficiency function (in any form) exists; that is, given a parallel architecture and a fixed efficiency value, the amount of useful work in a given algorithm can be determined such that a given parallel system achieves the desired efficiency. For adaptive GMRES($k$), the results indicate a strong dependency of the efficiency of a parallel version of the method on the variability in problem difficulty when problems are scaled.

It has also been observed that, because of resource saturation, there is an upper bound on the problem size beyond which the sequential execution time is affected by extensive memory hierarchy operations. In this case, the time measurement of solving large problems on a single processor may impede the isoefficiency analysis, since the efficiency $E = T_1/pT_p$ becomes greater than one and the relation between the useful work and the parallel overhead $W(\bar{N}) = eV(W(\bar{N}), p, \bar{h})$ cannot be applied to study scalability of an algorithm-machine combination.

## REFERENCES


[1] D. C. S. Allison, K. M. Irani, C. J. Ribbens, and L. T. Watson, *High-dimensional homotopy curve tracking on a shared-memory multiprocessor*, J. Supercomput., 5 (1992), pp. 347–366.

[2] Å. Björck, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1–21.

[3] C. H. Bischof and P. T. P. Tang, *Robust incremental condition estimation*, Tech. Rep. CS-91-133, LAPACK Working Note 33, Computer Sci. Dept., Univ. of Tennessee, May, 1991.

[4] A. Chakraborty, D. C. S. Allison, C. J. Ribbens, and L. T. Watson, *The parallel complexity of embedding algorithms for the solution of nonlinear equations*, IEEE Trans. Parallel Distributed Systems, 4 (1993), pp. 458–465.

[5] I. Duff, R. G. Grimes, and J. G. Lewis, *Users' guide for the Harwell-Boeing sparse matrix collection (Release I)*, Tech. Report TR/PA/92/86, CERFACS, France, October, 1992.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[7] A. Y. Grama, A. Gupta, and V. Kumar, *Isoefficiency: Measuring the scalability of parallel algorithms and architectures*, IEEE Parallel and Distrib. Technol., 1 (1993), pp. 12–21.

[8] A. Gupta, and V. Kumar, *Performance and scalability of preconditioned conjugate methods on parallel computers*, IEEE Trans. Parallel and Distrib. Systems, 6 (1995), pp. 455–469.

[9] B. A. Hendrickson and R. W. Leland, *A multilevel algorithm for partitioning graphs*, Tech.Rep., SAND92-1301, Sandia National Laboratories, November 1993.

[10] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, Tech. Report, 95-036, Computer Sci Dept., Univ. of Minnesota, Minneapolis, MN 55455, October, 1996.

[11] G. Karypis and V. Kumar, *MeTiS: Unstructured graph partitioning and sparse matrix ordering system*, User's Guide—Version 2.0, Dept. of Computer Sci., Univ. of Minnesota, MN 55455, 1995.

[12] G.-C. Lo and Y. Saad, *Iterative solution of general sparse linear systems on clusters of workstations*, Tech. Report, UMSI 96/117, Supercomputer Institute, Univ. of Minnesota, Minneapolis, MN, August, 1996.

[13] D. M. Nicol and W. Mao, *On bottleneck partitioning of $k$-ary $n$-cubes*, Parallel Programming Letters, 6 (1996), pp. 389–399.

[14] D. A. Reed, L. M. Adams, and M. L. Patrick, *Stencils and problem partitionings: Their influence on the performance of multiple processor systems*, IEEE Trans. Comput., C-36 (1987), pp. 845–858.

[15] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual method for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856 – 869.

[16] R. B. Sidje, *Alternatives for parallel Krylov subspace basis computation*, Numer. Linear Algebra Appl., 4 (1997), pp. 305–331.

[17] J. P. Singh, J. L. Hennessy, and A. Gupta, *Scaling parallel programs for multiprocessors: methodology and examples*, Computer, 7 (1993), pp. 42–50.

[18] M. Sosonkina, L. T. Watson, R. K. Kapania, and H. F. Walker, *A new adaptive GMRES algorithm for achieving high accuracy*, Numer. Linear Algebra Appl., submitted.

[19] H. F. Walker, *Implementation of the GMRES method using Householder Transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 152–163.

[20] L. T. Watson, *Globally convergent homotopy methods: a tutorial*, Appl. Math. Comput., 31BK (1989), pp. 369–396.

[21] L. T. Watson, M. Sosonkina, R. C. Melville, A. P. Morgan, and H. F. Walker, *HOMPACK90: A suite of FORTRAN 90 codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software., to appear.

[22] D. B. West, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ, 1996.