AN EFFICIENT PARALLEL ALGORITHM FOR EXTREME EIGENVALUES OF SPARSE NONSYMMETRIC MATRICES

S. K. Kim and A. T. Chronopoulos

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF MINNESOTA
MINNEAPOLIS, MINNESOTA 55455

Summary

Main memory accesses for shared-memory systems or global communications (synchronizations) in message passing systems decrease the computation speed. In this paper, the standard Arnoldi algorithm for approximating a small number of eigenvalues, with largest (or smallest) real parts for nonsymmetric large sparse matrices, is restructured so that only one synchronization point is required; that is, one global communication in a message passing distributed-memory machine or one global memory sweep in a shared-memory machine per each iteration is required. We also introduce an *s*-step Arnoldi method for finding a few eigenvalues of nonsymmetric large sparse matrices. This method generates reduction matrices that are similar to those generated by the standard method. One iteration of the *s*-step Arnoldi algorithm corresponds to *s* iterations of the standard Arnoldi algorithm. The *s*-step method has improved data locality, minimized global communication, and superior parallel properties. These algorithms are implemented on a 64-node NCUBE/7 Hypercube and a CRAY-2, and performance results are presented.

The International Journal of Supercomputer Applications, Volume 6, No. 1, Spring 1992, pp. 98–111.

Introduction

It is now feasible to achieve high performance in numerical computations with parallel processing technology. To use parallel computers in a specific application, algorithms have to be developed and mapped onto a parallel computer architecture (Aykanat et al., 1988; Chronopoulos and Gear, 1989a, 1989b; Kim and Chronopoulos, 1991). Parallel computers consist of many processors communicating through an interconnection network. Depending on the structure of the memory system, two extremes of parallel computers are shared-memory multiprocessors, in which all memory modules are equally accessible to all processors, and distributed-memory parallel processors, in which each memory module is physically associated with each processor. The significant difference between the two is that a distributed-memory parallel computer has no shared memory. Consequently, to use data in a remote memory, it is necessary to explicitly get the data from a remote memory. This and all other interprocessor communication is achieved by passing messages among the processors.

Memory contention on shared-memory machines constitutes a severe performance bottleneck. The same is true for communication costs on a message passing system. It would be desirable to have methods for specific problems that have low communication costs compared to the computation costs. This is interpreted as a small number of global memory accesses for the shared-memory systems and a small number of global communications for the message passing systems. Also, we consider the design issues involved in partitioning and mapping data parallel algorithms (Li, King, and Prins, 1987). Data parallel algorithms are suitable for problems with a large volume of data. Parallelism is achieved by partitioning the dataset rather than partitioning the control of the program. The algorithm must be designed so that both computation and data can be distributed to the processors in such a way that computations can be run in parallel, balancing the loads of the processors.

Many important scientific and engineering problems require the computation of a small number of eigenvalues of nonsymmetric large sparse matrices.

The most commonly used algorithm for solving such an eigenproblem is the Arnoldi algorithm. It has three basic types of operations: matrix-vector products, inner products, and vector updates. The sequential and parallel complexity (Chronopoulos and Gear, 1989b) of these computations is shown in Table 1 (n_d = the number of nonzero diagonals of A). For the purposes of Table 1, the parallel system is assumed to have $O(N)$ processors. From Table 1 we can draw the conclusion that for massively parallel systems the inner products dominate the three operation types in the Arnoldi method because of the global communication. The same might be true for hypercube systems whose nodes communication delay is much longer than the floating point operation execution. Thus, grouping together for execution the inner products of each iteration in the Arnoldi method may lead to a speedup on this type of computer.

On shared-memory systems with a memory hierarchy such as the CRAY-2 the data locality of the computations is very important in achieving high execution speed. A good measure of the data locality of a computation is the size of the

Ratio =
(Memory References)/(Floating Point Operation).

The data locality of a computation is good if this ratio is as low as possible. This ratio being low implies that data can be kept for "a long time" in fast registers or local memories and many operations can be performed on them. Thus, restructuring the three types of operations in the Arnoldi method may lead to a speedup on shared memory systems with a memory hierarchy.

In past work *s*-step methods have been introduced for the conjugate gradient (CG) method (Chronopoulos and Gear, 1989a, 1989b) and the Lanczos method (Kim and Chronopoulos, 1991). In this paper we introduce the *s*-step Arnoldi method. In the *s*-step method, *s* consecutive steps of the standard method are performed simultaneously. This means, for example, that the inner products (needed for *s* steps of the standard method) can be performed simultaneously and the vector updates are replaced by linear combinations. In the *s*-step Arnoldi method, the vector updates and storage are

"Data parallel algorithms are suitable for problems with a large volume of data. Parallelism is achieved by partitioning the dataset rather than the control of the program. The algorithm must be designed so that both computation and data can be distributed to the processors in such a way that computations can be run in parallel, balancing the loads of the processors."

Table 1
Serial and Parallel Complexity of Arnoldi Parts

Operation	Sequential	Parallel
Vector update	$O(N)$	$O(1)$
Inner products	$O(N)$	$O(\log_2 N)$
Matrix-vector products	$O(n_d N)$	$O(\log_2 n_d)$

decreased slightly relative to the standard method because the s-step method generates s vectors so that these vectors are orthogonal to all preceding s-step Arnoldi vector sets, not all preceding vectors. Also, their parallel properties and data locality are improved and the s-step Arnoldi method has only one global communication for one s-step iteration. A disadvantage of the s-step Arnoldi method is that one more matrix-vector multiplication is required for one s-step iteration compared to s iterations of the standard method.

In the following sections we discuss the standard Arnoldi algorithm, describe the NCUBE and the CRAY-2 in detail, and discuss how to implement efficiently the Arnoldi algorithm. We then restructure the Arnoldi algorithm to gain better performance, develop the s-step method, which is the new version of the Arnoldi method, and give numerical examples and experimental results on the NCUBE/7 Hypercube and CRAY-2.

Results and Discussion

1. THE ARNOLDI METHOD

The method of Arnoldi can be used successfully for computing a small number of eigenvalues with largest (or smallest) real part for large nonsymmetric matrices (Saad, 1980, 1984, 1985). The Arnoldi algorithm is based on the Arnoldi recursion for reducing to upper Hessenberg matrices of a real nonsymmetric matrix A. The basic Arnoldi procedure can be viewed as the Gram-Schmidt orthogonalization of the Krylov subspace basis $\{q_1, Aq_1, \ldots, A^{j-1}q_1\}$. Furthermore, for each j,

$$H_j = Q_j{}^T A Q_j \tag{1}$$

is the orthogonal projection of A onto the subspace spanned by the Arnoldi vectors $Q_j = \{q_1, \ldots, q_j\}$ such that $Q_j{}^T Q_j = I_j$ where I_j is the identity matrix of order j. The eigenvalues of the upper Hessenberg matrices H_j are called Ritz values of A in Q_j. For many matrices and for relatively small j, several of the extreme eigenvalues of A—that is, several of the algebraically-largest or algebraically-smallest of the eigenvalues of A—are well approximated by eigenvalues of the matrices H_j. The Ritz vector $Q_j y (= z)$ obtained from an eigenvector y of a given H_j is an approximation to a corresponding eigenvector of A. The standard Arnoldi algorithm is as follows:

Algorithm 1: The Arnoldi Algorithm
Choose q_1 with $\|q_1\|_2 = 1$
For $j = 1$ **until** Convergence **Do**

1. Compute and store Aq_j
2. Compute $h_{i,j} = (Aq_j, q_i)$, $i = 1, \ldots, j$
3. $r_j = Aq_j - \Sigma_{i=1}^{j} h_{i,j} q_i$
4. Compute (r_j, r_j)
5. $h_{j+1,j} = \sqrt{(r_j, r_j)}$
6. $q_{j+1} = r_j / h_{j+1,j}$

EndFor

This algorithm generates a set of vectors Q_j and each new Arnoldi vector generated is orthogonal to all preceding Arnoldi vectors. Therefore computational time and storage increase proportionally to the number of steps. The storage of orthogonal basis Q_j requires $N \times j$ memory locations. When N is large, the number of steps, j, is limited by the available main memory. This difficulty may be overcome by using the incomplete orthogonalization process and iterative Arnoldi algorithm (Saad, 1980). Step (3) of this algorithm is usually replaced by the more stable modified Gram-Schmidt (MGS) scheme. But the MGS scheme in step (3) requires many more synchronization points on parallel computers. In this paper we have not used the MGS scheme for the Arnoldi algorithm. If the matrix A is symmetric, then H_j reduces to a symmetric tridiagonal matrix and this algorithm reduces to the symmetric Lanczos method (Golub and Van Loan, 1989). The upper Hessenberg matrices H_j explicitly satisfy the equation

$$AQ_j = Q_j H_j + r_j e_j{}^T, \tag{2}$$

where $e_j{}^T = (0,0,\ldots,0,1)$ is a j-dimensional vector. The simplest posteriori bound on the accuracy of a Ritz value λ is obtained from the residual norm of the associated Ritz vector z, $\|Az - z\lambda\|$. The residual norm of the Ritz pair λ, z can be computed by using the formula

$$\|(A - \lambda_i I)z_i\| = |h_{j+1,j} e_j{}^T y_i|, \quad \text{for } i = 1, \ldots, j, \tag{3}$$

where y_i is the ith eigenvector of H_j. This is used as a stopping criterion. The following step is added after

step (5) in algorithm 1 to test the accuracy of computed approximate eigenvalues:

5′. Compute $h_{j+1,j}$, y_i. If $|h_{j+1,j}e_j^T y_i| < \in$ then stop.

The quantity $h_{j+1,j}$ is calculated by the algorithm. Since j is usually very small compared to N the calculation of y_i (via a dense matrix eigenvector method applied on the matrix H_j) is relatively inexpensive. Also, y_i can be calculated in parallel with $h_{j+1,j}$. Thus, step (5′) does not add an extra synchronization point. In practice, the Arnoldi method is used iteratively together with various acceleration techniques (Saad, 1984, 1985).

2. PARALLELISM ON THE NCUBE HYPERCUBE SYSTEM

2.1 A HYPERCUBE COMPUTER

A hypercube model is a particular example of a distributed-memory message passing parallel computer. In a hypercube of dimension d, there are 2^d processors. Assume that these are labeled $0,1,\ldots,2^d - 1$. Two processors i and j are directly connected if the binary representation of i and j differ in exactly one bit. Each edge of Figure 1 represents a direct connection of a dimension 4 hypercube (lines and dotted lines are communication links). Thus, in a hypercube of dimension d, each processor is connected to d others, and 2^d processors may be interconnected such that the maximum distance between any two is d.

Table 2 represents a summary of an experimental study of interprocessor communication time and time to perform arithmetic operations on the NCUBE (Ranka, Won, and Sahni, 1988). We see that an 8 byte message transfer between two directly connected processors takes 42 times as long as an 8 byte real addition and 32 times as long as an 8 byte real multiplication. Furthermore, longer messages are transferred at a higher rate (i.e., bytes/sec) than shorter ones going the same distance. In a linear time model of nearest neighbor communication a message of length M bytes requires approximately $446.7 + 2.4M$ μsec (Ranka, Won, and Sahni, 1988) where the constant term is a startup time that consists of the software overhead at each end and the time to set up the circuit and the second term is

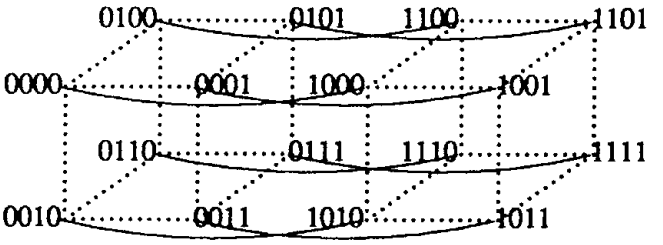


Fig. 1 Dimension 4 Hypercube

Table 2
Computation and Communication Time on NCUBE/7 Hypercube

Operation	Time	Comm./Comp.
8 byte transfer	470 μsec	
8 byte real addition	11.2 μsec	42 times
8 byte real multiplication	14.7 μsec	32 times

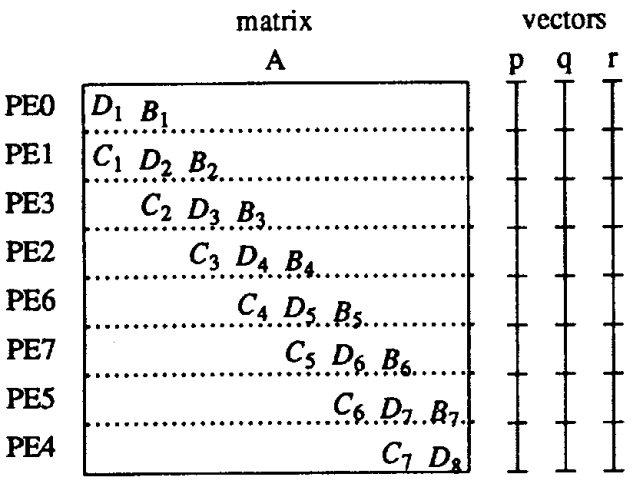


Fig. 2 Distribution of matrix and vectors to each processor

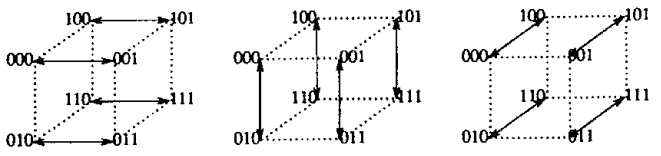


Fig. 3 Inner product process on Hypercube

the actual transmission time. The startup time for short messages is the dominating factor in the communication cost on the NCUBE.

In a hypercube with a high communication latency, the algorithm designer must structure the algorithm so that large amounts of computation are performed between communication steps; an algorithm requiring frequent and random exchange of messages will not perform well.

The two main issues in programming this machine are load balancing and reduction of communication overhead. A program is load balanced if all the processors are busy all the time. If one processor has most of the work, then the others will end up being idle most of the time and the program will run inefficiently. To arrive at an efficient algorithm for a hypercube, one must consider that both computations and data can be distributed to the processors in such a way that computational tasks can be run in parallel, balancing the computational loads of the processors as much as possible.

2.2 MAPPING THE ARNOLDI ALGORITHMS ON THE NCUBE

The Arnoldi algorithm has three basic types of operations: matrix-vector product, inner products, and vector updates. We only consider A to have the structured nonzero pattern of a sparse banded matrix. This is the class of matrices that are used in our numerical tests. Parallelization for matrices with irregular sparsity patterns is more problematic, and global communication may be required. The matrix-vector multiplication can be performed concurrently by distributing the rows of A, and the corresponding elements of vectors among processors of the NCUBE (Fig. 2). We divide the vector length N by the number of processors P. Each processor gets a subvector. When P does not divide N exactly, one processor gets a shorter vector. The simplest representations of an $N \times N$ matrix are row or column oriented (McBryan and van de Velde, 1987). In the contiguous-row representation, each row is stored entirely in one processor. During a distributed matrix-vector product computation, processors need the most recently updated values of the vector elements which are mapped to neighbor processors. In Figure 2, we

demonstrate this for a block tridiagonal matrix A, where D_i is a tridiagonal matrix and B_i, C_i are diagonal matrices. Hence, only nearest neighbor communication (i.e., local interprocessor communication) is required. The distributed vector updates can be performed concurrently without interprocessor communication, only after each processor computes the updated global scalar values $h_{t,j}$ in step (2), step (5) of the Arnoldi algorithm 1.

For uniprocessors or multiprocessors with a small number of processors (e.g., four or eight processors) the matrix-vector products dominate the computation, whereas on parallel computer systems having the hypercube interconnection network, the inner products dominate because they require global communication (synchronization of all processors) of the system. An inner product is computed by assigning an equal part of the vector (if possible) to each node. This allows each processor to work on local segments independently of other processors for most operations. Each node computes in parallel the sum of squares of its part of the vector. Then we used the exchange-add algorithm (Aykanat et al., 1988). Processors $P_{(z_{d-1},\ldots,z_{i+1},0,z_{i-1},\ldots,z_0)}$, $(i = 0,\ldots,d-1)$ concurrently exchange their most recent partial sum with their neighbor $P_{(z_{d-1},\ldots,z_{i+1},1,z_{i-1},\ldots,z_0)}$ and then concurrently form their new partial sum. At the end of $2d$ concurrent nearest neighbor communication steps, each processor has its own copy of the inner product. The exchange-add algorithm is illustrated in Figure 3 ($d = 3$).

3. PARALLELISM ON THE CRAY-2

The CRAY-2 is an example of a shared-memory four-processor computer with memory hierarchy. All processors have equal access to a very large central memory of 256 megawords, and each processor has its own local memory. Each CRAY-2 processor has eight vector registers (each 64 words long) and has data access through a single path between its vector registers and main memory. Each processor has 16 Kwords of local memory with no direct path to central memory but with a separate data path between local memory and its vector registers, and the six parallel vector pipelines: common memory to vector register, vector register to local memory, floating additive/subtractive, floating multiplicative/divisible, integer additive/subtractive, and logical pipelines. It is possible to design assembly language kernels that exhibit a performance commensurate with the 4.2 nsec cycle time of the CRAY-2 if the computations allow it. This means that a rate of 459 MFLOPS is possible on one processor if all arithmetic pipes can be kept busy (Dongarra and Sorensen, 1986). The combination of fast cycle time, a local memory, and a large central memory make the CRAY-2 an exciting choice of computer for use in large-scale scientific computation.

The maximum performance of the CRAY-2 for specific applications comes from data movement minimization, good vectorization, and division into multiprocessing tasks. Because of single paths between vector register and central or local memory on the CRAY-2 system, memory transfers constitute a severe bottleneck for achieving maximum performance. Therefore, minimization of data movement results in faster execution times. Algorithms must provide good data locality: that is, the organization of the algorithm should be such that the data can be kept as long as possible in fast registers or local memories and have many operations performed on them. The key to optimal performance on Cray supercomputers is to ensure that arithmetic pipes are busy almost all of the time. All access to memory must be performed concurrently with the arithmetic.

Macrotasking (often called multitasking) on a data parallel algorithm is most often applied to parallel work found in the independent iterations of DO-loops. If the loop has N iterations, we map the N iterations onto P processors or tasks so that each task has the same amount of work to do. To achieve load balancing, we must consider static and dynamic partitioning. We use static partitioning when the times for each of the loop iterations are approximately equal. Example of contiguous static partitioning with $P = 4$:

Processor	Assigned iterations
P0	I = 1,N/4
P1	I = (N/4) + 1,2*N/4
P2	I = 2*(N/4) + 1,3*N/4
P3	I = 3*(N/4) + 1,N

The distribution of matrix and vectors is similar to that in Figure 2 with four processors. The extremely

large memory of the CRAY-2 means that many jobs can usually be resident in the main storage at the same time, leading to very efficient multiprogramming. We must minimize calls to the multitasking library because multitasking introduces an overhead that increases CPU time.

4. THE MODIFIED ARNOLDI METHOD

In the standard Arnoldi algorithm iteration, the inner products cannot be performed in parallel. Algorithms based on restructuring the standard Arnoldi algorithm to decrease the global communication cost and to get better performance in distributed-memory message passing systems are introduced here.

For shared-memory systems with few processors, processor synchronization is fast but accessing data from the main memory may be slow. Thus, the data localities of the three basic operation parts of the Arnoldi algorithm determine the actual time complexity of the algorithm. The data locality of the modified Arnoldi algorithm is better than that of the standard algorithm. The Lanczos algorithm for finding a few eigenvalues of symmetric large sparse matrices has the same shortcomings for parallel processing as the Arnoldi algorithm. Examples of parallel Lanczos algorithms are discussed in Kim and Chronopoulos (1991).

In algorithm 1, step (2), or step (5), must be completed before the rest of the computations in the same step start. This forces double access of vectors q,r,Aq from the main memory at each iteration.

Algorithm 2: The Modified Arnoldi Algorithm
Choose r_0 with $r_0 \neq 0$
For $j = 0$ **until** Convergence **Do**

1. Compute and store Ar_j
2. Compute (r_j,r_j), (Ar_j,r_j), (Ar_j,q_t) $t = 1,\ldots,j$
3. $h_{j+1,j} = \sqrt{(r_j,r_j)}$
   $h_{j+1,j+1} = (Ar_j,r_j)/(r_j,r_j)$
   $h_{t,j+1} = (Ar_j,q_t)/h_{j+1,j}$, $t = 1,\ldots,j$
4. $q_{j+1} = r_j/h_{j+1,j}$
5. $r_{j+1} = Ar_j/h_{j+1,j} - \Sigma_{t=1}^{j+1} h_{tj}q_t$

EndFor

Algorithm 2 is a variant of algorithm 1, and the orthonormal vectors q_j are generated in the same way as the standard Arnoldi method. Computationally the difference between algorithms 2 and 1 is the computation of r_j and the elements of upper Hessenberg matrix H_j. We need one more vector operation to compute r_j in algorithm 2. Algorithm 2 is better for parallel processing because the $j + 1$ (at jth iteration) inner products can be executed simultaneously. Also, one memory sweep through the data is required to complete each iteration, allowing better management of slower memories.

In algorithm 2 we must add a step similar to step (5′) of algorithm 1 to check the stopping criterion. The stopping criterion requires the computation of $h_{j+1,j}$ and eigenvectors of the reduced matrix H_j. Each processor in parallel computers has H_j and computes the stopping criterion after step (5) of algorithm 1 or step (3) of algorithm 2. Therefore another synchronization point is not required to check convergence in algorithms 1 and 2.

In the next section we propose an s-step Arnoldi algorithm, which executes simultaneously in a certain sense s consecutive steps of algorithm 1.

5. THE s-STEP ARNOLDI ALGORITHM

Let us denote by k the iteration number in the s-step Arnoldi method. Given the vectors $v_k^1, v_k^2, \ldots, v_k^s$ we will use $\overline{V}_k$ (each of dimension N) to denote the matrix of $\{v_k^1, v_k^2, \ldots, v_k^s\}$. Initially we start with a vector v_1^1 and form $v_1^2 = Av_1^1, \ldots, v_1^s = A^{s-1}v_1^1$. One way to obtain an s-step Arnoldi algorithm is to use these s linearly independent vectors. We next form $\overline{V}_2$ from $v_2^1 = Av_1^s$, $v_2^2 = Av_2^1, \ldots, v_2^s = A^{s-1}v_2^1$. Then we orthogonalize $\overline{V}_2$ against $\overline{V}_1$. Inductively we form $\overline{V}_k$ for $k > 1$. The subspaces $\overline{V}_1, \overline{V}_2, \ldots \overline{V}_k$ are mutually orthogonal, but the vectors $\{v_k^1, v_k^2, \ldots, v_k^s\}$ are not orthogonal; that is, $\overline{V}_k{}^T \overline{V}_k$ is not a diagonal matrix.

Each subspace $\overline{V}_k$ can be decomposed into $\overline{Q}_k * \overline{R}_k$, where $\overline{Q}_k$ is an orthonormal basis of $\overline{V}_k$ and $\overline{R}_k$ is an $s \times s$ upper triangular matrix.

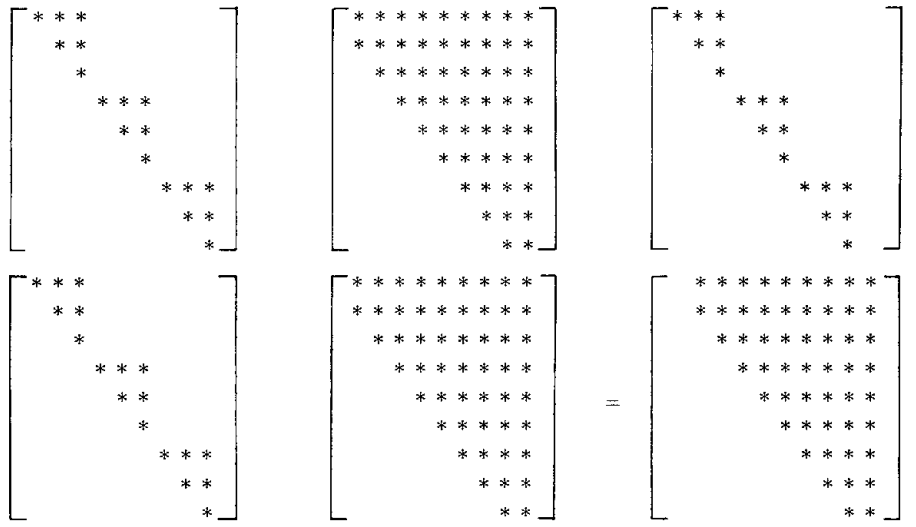
REMARK 1. Let $\overline{V}_{i_1}$ be orthogonal to $\overline{V}_{i_2}$ for $i_1 \neq i_2$. Then $V_k = \{\overline{V}_1, \overline{V}_2, \ldots, \overline{V}_k\}$ can be decomposed into $Q_k * R_k$, where $Q_k = \{\overline{Q}_1, \overline{Q}_2, \ldots, \overline{Q}_k\}$ and $R_k = \text{diag}(\overline{R}_1, \overline{R}_2, \ldots, \overline{R}_k)$.

PROPOSITION 1. *Let $\tilde{H}_j$ be a upper Hessenberg matrix and $\overline{H}_k = R_k^{-1}\tilde{H}_j R_k$ for $j = sk$. Then $\overline{H}_k$ is similar to the matrix $\tilde{H}_j$ and $\overline{H}_k$ is a block upper Hessenberg matrix whose each block $H_{i,j}$, for $1 \leq i, j \leq k$, is $s \times s$ matrices and the subdiagonal block $H_{i+1,i}$ has only nonzero element at location (1,s). We further assume that the degree of the minimal polynomial of v_1^1 is greater than $s * k$.*

Proof. By remark 1, R_k is nonsingular if all $\overline{R}_i$, for $i = 1, \ldots, k$ are nonsingular, which is the case if $v_i^1, \ldots, v_i^s$ are linearly independent. So $\overline{H}_k$ is similar to the matrix $\tilde{H}_j$. Since $R_k = \text{diag}(\overline{R}_1, \overline{R}_2, \ldots, \overline{R}_k)$ and $\overline{R}_i$, for $i = 1, \ldots, k$ is an $s \times s$ upper triangular matrix, R_k^{-1} has the same structure as R_k. Thus, the product $R_k^{-1}\tilde{H}_j R_k$ for $j = sk$ is a block upper Hessenberg matrix with lower diagonal blocks in a special form; that is, the main subdiagonal blocks have only one nonzero entry in the upper right corner. We will demonstrate this for the special case $s = 3, k = 3$. The general case is shown similarly, but the description is more complicated.

"In the standard Arnoldi algorithm iteration, the inner products cannot be performed in parallel. Algorithms based on restructuring the standard Arnoldi algorithm to decrease the global communication cost and to get better performance in distributed-memory message passing systems are introduced here."

$$R_3^{-1}\tilde{H}_9 R_3 =$$



$$=$$

We use $\overline{H}_k$ as a block upper Hessenberg matrix, even though the structure is the same as a upper Hessenberg matrix. ■

We will use the following s-dimensional column notations for the blocks $H_{l,k}$ of the block upper Hessenberg matrix $\overline{H}_k$:

$$H_{l,k} = [\mathbf{h}_{\mathbf{l,k}}^{\mathbf{i}}], \quad \text{for } i = 1, \ldots s,$$

where $\mathbf{h_{l,k}}^{\mathbf{i}} = [h_{l,k}{}^{1,i}, \ldots, h_{l,k}{}^{s,i}]^T$ and $h_{l,k}{}^{i,j}$ is the element in position (i,j) of each block $H_{l,k}$.

THEOREM 1. *Let A be an $N \times N$ nonsymmetric matrix and $V_m = \{\overline{V}_1, \overline{V}_2, \ldots, \overline{V}_m\}$ for $N = sm$. Let*

$$V_m{}^{-1} A V_m = \overline{H}_m \quad \text{or} \tag{4}$$

$$A V_m = V_m \overline{H}_m, \tag{5}$$

then $\overline{H}_m = R_m{}^{-1} H_N R_m$ where H_N is the Arnoldi upper Hessenberg matrix and R_m is the block upper triangular matrix defined in remark 1.

Proof. From remark 1 we have

$$(\tilde{Q}_m R_m)^{-1} A (\tilde{Q}_m R_m) = R_m{}^{-1} (\tilde{Q}_m{}^{-1} A \tilde{Q}_m) R_m = \overline{H}_m.$$

From remark 1 $\tilde{Q}_m$ is an orthogonal matrix and $\tilde{Q}_m{}^T A \tilde{Q}_m$ is a upper Hessenberg matrix. By the implicit Q theorem (Golub and Van Loan, 1989), $\tilde{Q}_m{}^T A \tilde{Q}_m$ is (up to a sign) the same as the Arnoldi upper Hessenberg matrix H_N and $\tilde{Q}_m$ is the same sequence of vectors as the standard Arnoldi vectors Q_N if the initial vector v_1 is the same. Also $\overline{H}_m$ is a block upper Hessenberg matrix form in proposition 1.

COROLLARY 1. *The block upper Hessenberg matrix $\overline{H}_k$, for $k = 1, \ldots, m-1$ has the same eigenvalues as the (algorithm 1) upper Hessenberg matrix H_j, for $j = sk$.*

Proof. By theorem 1 and proposition 1, the matrices $\overline{H}_k$ and H_j for $j = sk$ are similar. ■

If $k < m$ then by equating column blocks in Eq. (5) we obtain the following equation:

$$A V_k = V_k \overline{H}_k + u_k e_{sk}{}^T \quad \text{for } k = 1, \ldots, m-1 \tag{6}$$

where the vector u_k is called the residual vector after k iterations of s-step method and the vector e_{sk} is $[0, \ldots, 0, 1]$ of dimension $s * k$. From Eq. (6) we derive the following block equations:

$$A \overline{V}_k = \sum_{l=1}^{k} \overline{V}_l H_{l,k} + u_k e_{sk}{}^T \tag{7}$$

Equations (5), (6), and (7) motivate the derivation of an s-step Arnoldi algorithm. Initially we can form $\overline{V}_1 = [v_1{}^1, A v_1{}^1, \ldots, A^{s-1} v_1{}^1]$, then we select the new Krylov vector from Eq. (7).

$$u_1 = A v_1{}^s - \overline{V}_1 \mathbf{h_{1,1}}^{\mathbf{s}}$$

$\mathbf{h_{1,1}}$ is selected to orthogonalize the residual vector u_1 against $\overline{V}_1$. We choose $v_2{}^1 = u_1$ (i.e., normalization is not applied). Then we form the vectors $\{v_2{}^2, \ldots, v_2{}^s\}$ in $\overline{V}_2$ from the vectors $\{A v_2{}^1, \ldots, A^{s-1} v_2{}^1\}$ by orthogonalizing them against $\overline{V}_1$. Thus, $v_2{}^j$, for $2 \leq j \leq s$, are determined by the linear combinations:

$$v_2{}^j = A^{j-1} v_2{}^1 - \sum_{i=1}^{s} v_1{}^i t_{1,1}{}^{i,j} \quad \text{for } j = 2, \ldots, s,$$

where $\{t_{1,1}{}^{i,j}\}$, for $1 \leq i \leq s$ and $2 \leq j \leq s$, are parameters to be determined. Let $\mathbf{t_{1,k}}^{j} = [t_{l,k}{}^{1,j}, \ldots, t_{l,k}{}^{s,j}]^T$, for $1 \leq l \leq k$, denote the parameters defining $v_{k+1}{}^j$. We now give the defining equations of the s-step Arnoldi method in the form of an algorithm.

Algorithm 3: The s-step Arnoldi Algorithm

$\overline{V}_1 = [v_1{}^1, A v_1{}^1, \ldots, A^{s-1} v_1{}^1]$

For $k = 1$ **until** Convergence **Do**

Select $[\mathbf{h_{l,k}}^{\mathbf{i}}]$, for $1 \leq l \leq k$ and $1 \leq i \leq s$, to orthogonalize $\overline{V}_k$ against $\overline{V}_{k-1}, \ldots, \overline{V}_1$ in Eq. (7). This gives also:

$$v_{k+1}{}^1 = A v_k{}^s - \sum_{l=1}^{k} \overline{V}_l \mathbf{h_{l,k}}^{\mathbf{s}} \tag{8}$$

Select $[\mathbf{t_{1,k}}^{\mathbf{j}}]$, $2 \leq j \leq s$ to orthogonalize $\{A v_{k+1}{}^1, \ldots, A^{s-1} v_{k+1}{}^1\}$ against $\overline{V}_k, \ldots, \overline{V}_1$ which gives

$$v_{k+1}{}^j = A^{j-1} v_{k+1}{}^1 - \sum_{l=1}^{k} \overline{V}_l \mathbf{t_{l,k}}^{\mathbf{j}}$$

$$\text{for } j = 2, \ldots, s \tag{9}$$

EndFor

Next, we demonstrate how to determine the parameters $\mathbf{h_{l,k}}^{\mathbf{i}}$, $\mathbf{t_{1,k}}^{\mathbf{j}}$ in algorithm 3. Equation (7) multiplied by $\overline{V}_l{}^T$, for $1 \leq l \leq k$ from the left yields

$$\overline{V}_l{}^T A \overline{V}_k = \overline{V}_l{}^T \overline{V}_l H_{l,k} \tag{10}$$

Equation (9) multiplied by $\overline{V}_l{}^T$ from the left yields

$$0 = \overline{V}_l{}^T A^{j-1} v_{k+1}{}^1 - \overline{V}_l{}^T \overline{V}_l \mathbf{t_{1,k}}^{\mathbf{j}} \quad \text{for } j = 2, \ldots s \tag{11}$$

Equations (10) and (11) determine $[\mathbf{h_{l,k}}^{\mathbf{i}}]$, $1 \leq i \leq s$ and $[\mathbf{t_{1,k}}^{\mathbf{j}}]$, $2 \leq j \leq s$ as solutions of linear systems of size

Table 3

Vector Operations for s Iterations from the $s(k-1)$th Iteration of the Standard Method and the kth Iteration of the s-Step Method

Operation	Standard Arnoldi Algorithm	s-Step Arnoldi Algorithm
Inner products	$(k-1)s^2 + \frac{s}{2}(s+1) + s$	$(k-1)s^2 + \frac{s}{2}(s+1) + s$
Vector updates	$(2k-1)s^2$	$2(k-1)s^2$
Matrix*vector	s	$s+1$
Vector-storage	$ks+1$	$ks+1$

s. We will introduce some notations in order to describe these linear systems.

REMARK 2. Let $W_k = \overline{V}_k{}^T \overline{V}_k = \{(v_k{}^i, v_k{}^j)\}$, $1 \leq i,j \leq s$, then W_k is symmetric and it is nonsingular if and only if $v_k{}^1, \ldots, v_k{}^s$ are linearly independent.

REMARK 3. From Eqs. (10) and (11) and remark 2 it follows that the following linear systems must be solved to determine $[\mathbf{h}_{1,k}{}^{\mathbf{i}}]$, $1 \leq i \leq s$ and $[\mathbf{t}_{1,k}{}^{\mathbf{j}}]$, $2 \leq j \leq s$:

$$W_l \mathbf{h}_{1,k}{}^{\mathbf{i}} = \mathbf{c}_{1,k}{}^{\mathbf{i}}, \quad \textit{where } \mathbf{c}_{1,k}{}^{\mathbf{i}} = [(v_l{}^1, Av_k{}^i), \ldots, (v_l{}^s, Av_k{}^i)]^T, \tag{12}$$

$$W_l \mathbf{t}_{1,k}{}^{\mathbf{j}} = \mathbf{b}_{1,k}{}^{\mathbf{j}} \quad \textit{where } \mathbf{b}_{1,k}{}^{\mathbf{j}} = [(v_l{}^1, A^{j-1}v_{k+1}{}^1), \ldots, (v_l{}^s, A^{j-1}v_{k+1}{}^1)]^T \tag{13}$$

The following corollary reduces the matrix W_k to the computed scalars in previous iterations and the following inner products

$$(A^i v_k{}^1, v_l{}^j) \quad \text{for } i = 1, \ldots, s \; j = 1, \ldots, s \; l = 1, \ldots k-1 \text{ and} \tag{14}$$

$$(A^i v_k{}^1, A^j v_k{}^1) \quad \text{for } i = 0, \ldots, s-1 \; j = i, \ldots, s \tag{15}$$

COROLLARY 2. *The computation of matrix $W_k = (v_k{}^i, v_k{}^j)$, $1 \leq i,j \leq s$ can be reduced to the inner products in Eqs. (14) and (15) and the computed scalars in previous iterations.*

Proof. We use the block orthogonality of V_k and Eqs. (8) and (9). The matrix of inner products W_k can be formed from the inner products in Eqs. (14) and (15) and the s-dimensional vectors $\mathbf{b}_{1,k-1}{}^{\mathbf{j}}$ as follows:

$$(v_k^i, v_k^j) = (A^{i-1}v_k^1, A^{j-1}v_k^1) - \sum_{l=1}^{k-1}\sum_{r=1}^{s} t_{l,k-1}^{r,i-1}(v_l^r, A^{j-1}v_k^1), \text{ for } 2 \leq i,j \leq s$$

Also, the vectors $\mathbf{c_{1,k}}^{\mathbf{i}}$, $\mathbf{b_{1,k}}^{\mathbf{j}}$ can be reduced to computing the inner products in Eqs. (14) and (15) and previous scalar work using a proof similar to corollary 2. Thus, the inner products (on vectors of size N) which are required in forming $\overline{H}_k$, $\overline{V}_{k+1}$ in algorithm 3 equals $(k-1)s^2 + [2 + \ldots + (s+1)]$. ■

We now reformulate the s-step Arnoldi algorithm taking into account the theory developed above.

Algorithm 4: The s-step Arnoldi Algorithm

Select v_1^1
Compute $\overline{V}_1 = [v_1^1, Av_1^1, \ldots, A^{s-1}v_1^1]$
Compute inner products in Eq. (15)
For $k = 1$ **until** Convergence **Do**
1. Call Scalar1
2. Compute $v_{k+1}^1 = Av_k^s - \Sigma_{l=1}^k \overline{V}_l \mathbf{h_{1,k}}^{\mathbf{s}}$
3. Compute $Av_{k+1}^1, A^2v_{k+1}^1, \ldots, A^sv_{k+1}^1$
4. Compute the inner products in Eqs. (14) and (15)
5. Call Scalar2
6. Compute $v_{k+1}^j = A^{j-1}v_{k+1}^1 - \Sigma_{l=1}^k \overline{V}_l[\mathbf{t_{1,k}}^{\mathbf{j}}]$ for $j = 2, \ldots, s$

EndFor

Scalar1: Decomposes W_k and solves $W_l\bar{\mathbf{h}}_{\mathbf{1,k}}^{\mathbf{i}} = \mathbf{c_{1,k}}^{\mathbf{i}}$ for $i = 1, \ldots, s$ as explained in Eq. (12).

Scalar2: Solves $W_l\mathbf{t_{1,k}}^{\mathbf{j}} = \mathbf{b_{1,k}}^{\mathbf{j}}$, for $j = 2, \ldots, s$ as explained in Eq. (13).

From Eq. (6) it follows that in the s-step Arnoldi method, the Ritz values of A in V_k are the eigenvalues λ_k of $\overline{H}_k$, and the Ritz vectors are vectors $V_k x_k (= z_k)$, where x_k are eigenvectors of $\overline{H}_k$ associated with λ_k. The residual norms of the Ritz value λ and Ritz vector z can be computed by using the formula $\|(A - \lambda I)z\| = \|u_k\| |\bar{s}_k|$ where $\bar{s}_k$ is the last element of eigenvectors of $\overline{H}_k$. This can be used as a stopping criterion.

We next compare the computational work and storage of the s-step Arnoldi method to the standard Arnoldi method (Table 3). We present only the vector operations on vectors of dimension N and neglect the operations on vectors of dimension s. If the matrix A is symmetric then $\overline{H}_k$ reduces to a block tridiagonal matrix $\overline{T}_k$ and this algorithm reduces to the s-step Lanczos method (Kim and Chronopoulos, 1991). Approximations to the eigenvalues of A are obtained by computing eigenvalues of the block upper Hessenberg matrices H_k.

6. NUMERICAL EXPERIMENTS

Large, sparse problems arise frequently in the numerical integration of partial differential equations (PDEs). Thus, we borrow our model problem from this area. Elliptic PDEs are often steady state equations for time-dependent PDEs describing physical models. The largest (or smallest) in real part eigenvalues of the steady state operators are computed in order to determine the dynamical system stability of the approximated physical model.

The test problems were derived from the five-point discretization of the following partial differential operator. Problem:

$$-(bu_x)_x - (cu_y)_y + (du)_x + (eu)_y + fu$$

on the unit square, where

$$b(x,y) = e^{-xy},\ c(x,y) = e^{xy},\ d(x,y) = \beta(x+y)$$

$$e(x,y) = \gamma(x+y) \text{ and } f(x,y) = 1./(1+x+y)$$

subject to the Dirichlet boundary conditions $u = 0$ on the boundary. The parameters β and γ are useful for changing the degree of symmetry of the resulting coefficient matrix of the linear systems. Note that the matrix A resulting from the discretization remains positive real independent of these parameters. We denote by n the number of interior nodes on each side of the square and by $h = 1/(n+1)$ the mesh size. In this paper we set $\gamma = 50$ and $\beta = 1$ and obtain a nonsymmetric matrix of size $N = n^2$.

We use the following programs: (1) Saad's program (*Code*), which implements the standard Arnoldi method with reorthogonalization and with iterative deflation; an Arnoldi procedure is said to be iterative if within the Arnoldi procedure a sequence of upper Hessenberg matrices is generated, each of which corresponds to a different starting vector; (2) the standard Arnoldi method implementing algorithm 1; (3) the modified Arnoldi method implementing algorithm 2; (4) the s-step Arnoldi method implementing algorithm 4.

First, we conduct accuracy tests. We choose $\in = 10^{-6}$ for the stopping criterion in Saad's program. The matrix A has dimension $N = 4{,}096$ in our accuracy tests. In the standard, modified, and s-step Arnoldi method we computed the largest eigenvalues after upper Hessenberg matrices H_j of size j are generated. We used EISPACK on the matrix H_j. In Table 4, as the size of upper Hessenberg matrices of standard method increase, the largest eigenvalue of the upper Hessenberg matrix is very close to actual eigenvalue by Saad's program with 10^{-6} accuracy. Tables 4 and 5 show that upper Hessenberg matrices generated by the modified and s-step methods have the same largest eigenvalues in real parts as the standard method. These tests were run on the single-processor CRAY-2 with single precision. For large $s > 5$ loss of accuracy for eigenvalues has been

Table 4
Largest Eigenvalues of Matrix A on CRAY-2

	Code		Standard Arnoldi Method	
Size of H_j	Largest Eigenvalue	Matrix-Vector Products	Largest Eigenvalue	Matrix-Vector Products
10	0.10204000E+02	81	0.9575713E+01	10
20	0.10204000E+02	81	0.10199149E+02	20
30	0.10203999E+02	91	0.10204783E+02	30
40	0.10203866E+02	81	0.10204008E+02	40

Table 5
Largest Eigenvalues Using the Modified and s-Step Arnoldi Methods on CRAY-2

H_j	Modified	2-Step	3-Step
10 × 10	0.9575713E+01	0.9575713E+01	–
20 × 20	0.10199149E+02	0.10199149E+02	–
30 × 30	0.10204783E+02	0.10204783E+02	0.10204783E+02
40 × 40	0.10204008E+02	0.10204008E+02	–
H_j	4-Step	5-Step	6-Step
10 × 10	–	0.9575713E+01	–
20 × 20	0.10199149E+02	0.10199149E+02	
30 × 30	–	0.10204783E+02	0.10204782E+02
40 × 40	0.10204008E+02	0.10204008E+02	

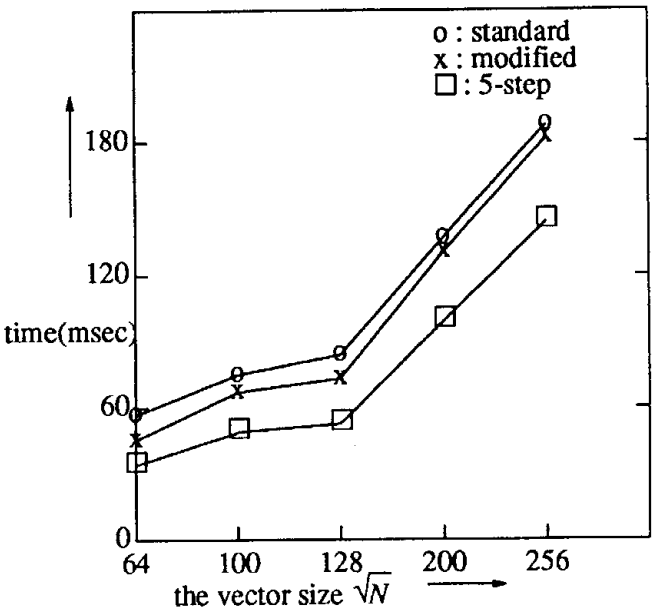


Fig. 4 CRAY-2 performance (msec) using four processors for the Arnoldi methods

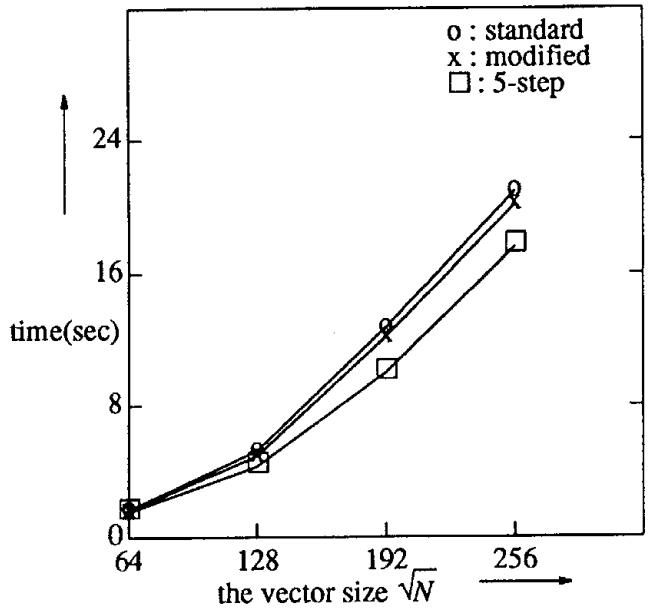


Fig. 5 NCUBE/7 performance (sec) using 64 nodes for the Arnoldi methods

observed. Loss of accuracy comes from solving $s \times s$ linear systems in algorithm 4 for large s, because these linear systems are not well conditioned.

Second, we run performance tests on the 64-node NCUBE/7 and four-processor CRAY-2 of the University of Minnesota. We used explicit multitasking on the CRAY-2. In these performance tests we compare the execution times for programs (2), (3), and (4). In these tests we varied the dimension of A from $N = 64^2$ to $N = 256^2$. We obtained 20×20 upper Hessenberg matrices. Figure 4 shows the execution times of these methods for different size test problems on the CRAY-2 with four processors. Memory reference time and calls to the Multitasking Library are decreased by making possible grain size large and by decreasing synchronization points in the modified and s-step methods, so those methods have better performance than the standard one on CRAY-2.

Figure 5 shows the execution times of the standard, modified, and s-step methods for different sizes of test problems with 64 on the NCUBE/7. The s-step method is faster than the modified and the modified is faster than the standard method. This can be explained as follows. The speed of the matrix-vector products and linear combinations is the same for all three methods. However, performing individual inner products is much slower than performing them simultaneously in groups (because of high global communication costs, see Table 3). The s-step method has an overhead in matrix-vector products, it has fewer vector updates (see Table 3), and it performs $2s$ inner products simultaneously. The modified method performs two inner products simultaneously.

7. CONCLUSION

The Arnoldi algorithm was restructured in this paper. The modified algorithm decreases the global communication bottleneck of the standard Arnoldi algorithm by restructuring computations in such a way as to increase the number of inner products that are accumulated during one iteration. The modified algorithm improves data locality by decreasing the memory contention bottleneck.

We have also introduced an s-step Arnoldi method

and proved that *s*-step methods generate block upper Hessenberg matrices that are similar to reduction matrices generated by the standard Arnoldi method. The resulting algorithm has better data locality and parallel properties than the standard one. In the *s*-step method, the inner products needed for *s* steps of the standard method can be performed simultaneously and the vector updates are replaced by linear combinations. The *s*-step Arnoldi method requires less computational work than the standard one.

ACKNOWLEDGMENT

We thank the anonymous referees and editor whose comments helped enhance significantly the quality of presentation of this article.

The research was partially supported by University of Minnesota Graduate School grant 0350-2104-07, and National Science Foundation grants CER DCR-8420935 and CCR-8722260. The Minnesota Supercomputing Institute provided time on CRAY-2.

BIOGRAPHIES

S. K. Kim obtained a B.S. in mathematics (in 1979) from Ehwa Womans University (Korea) and an M.S. in computer science from the Korea Advanced Institute of Science and Technology (in 1982). She received a Ph.D. in computer science from the University of Minnesota (in 1991). Her research interests include parallel numerical algorithms and mathematical software.

A. T. Chronopoulos obtained a B.S. in mathematics (in 1979) from the University of Athens (Greece) and an M.S. in applied mathematics from the University of Minnesota (in 1981). He received a Ph.D. in computer science from the University of Illinois at Urbana-Champaign (in 1986). Since 1987 he has been an assistant professor of computer science at the University of Minnesota in Minneapolis. His research interests include numerical analysis and parallel processing.

SUBJECT AREA EDITOR

Iain Duff

REFERENCES

Aykanat, C., Ozguner, F., Ercal, F., and Sadayappan, P. 1988. Iterative algorithms for solution of large sparse systems of linear equations on Hypercubes. *IEEE Trans. Comput.* 37 (12):1554–1568.

Chronopoulos, A. T., and Gear, C. W. 1989a. S-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.* 25:153–168.

Chronopoulos, A. T., and Gear, C. W. 1989b. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Comput.* 11:37–52.

Dave, A. K., and Duff, I. S. 1987. Sparse matrix calculations on the CRAY-2. *Parallel Comput.* 5:55–64.

Dongarra, J. J., and Sorensen, D. C. 1986. Linear algebra on high-performance computer. Conf. Parallel Computing 1985 proceed. M. Feilmeier et al. eds. Elsevier Pub. 1986.

Golub, G. H., and Van Loan, C. F. 1989. *MATRIX Computations.* Baltimore: Johns Hopkins University Press, pp. 219–225.

Gustafson, J. L., Montry, G. R., and Benner, R. E. 1988. Development of parallel methods for a 1024-processor hypercube. *SIAM J. Sci. Statist. Comput.* 9(4).

Kim, S. K., and Chronopoulos, A. T. 1991. A class of Lanczos algorithms implemented on parallel computers. *Parallel Comput.* 17.

Meurant, G. 1987. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Comput.* 5:267–280.

McBryan, O. A., and van de Velde, E. F. 1987. Matrix and vector operations on hypercube parallel processors. *Parallel Comput.* 5:117–125.

Ni, L. M., King, C. T., and Prins, P. 1987. Parallel algorithm design considerations for hypercube multiprocessors. *Proc. of the 1987 International Conf. on Parallel Processing.*

Ranka, S., Won, Y, and Sahni, S. 1988. Programming the NCUBE Hypercube. Tech. Rep. CSci No. 88-13. Minneapolis: University of Minnesota.

Ruhe, A. 1982. The two sided Arnoldi algorithm for nonsymmetric eigenvalue problems. Springer-Verlag *Lecture Notes in Math.* 973:104–120.

Saad, Y. 1980. Variation on the Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Algebra Appl.* 34:269–295.

Saad, Y. 1984. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.* 42:567–588.

Saad, Y. 1985. Partial eigensolutions of large nonsymmetric matrices. Research Report YALUE/DCS/RR-397.

Saylor, P. E. 1988. Leapfrog variants of iterative methods for linear algebraic equations. *J. Comput. Appl. Math.* 24:169–193.

Wilkinson, J. H. 1965. *The algebraic eigenvalue problem.* New York: Oxford University Press.