A Parallelizable Eigensolver for Real Diagonalizable Matrices with Real Eigenvalues*

Steven Huss-Lederman† Anna Tsao† Thomas Turnbull†

Abstract. In this paper, preliminary research results on a new algorithm for finding all the eigenvalues and eigenvectors of a real diagonalizable matrix with real eigenvalues are presented. The basic mathematical theory behind this approach is reviewed and is followed by a discussion of the numerical considerations of the actual implementation. The numerical algorithm has been tested on thousands of matrices on both a Cray-2 and an IBM RS/6000 Model 580 workstation. The results of these tests are presented. Finally, issues concerning the parallel implementation of the algorithm are discussed. The algorithm's heavy reliance on matrix-matrix multiplication, coupled with the divide and conquer nature of this algorithm, should yield a highly parallelizable algorithm.

1. Introduction. Computation of all the eigenvalues and eigenvectors of a dense matrix is essential for solving problems in many fields. The ever-increasing computational power available from modern supercomputers offers the potential for solving much larger problems than could have been contemplated previously. The characteristics and diversity of multiprocessor architectures have made the task of finding suitable parallel algorithms for dense problems a challenging one. Indeed, it appears likely that algorithms such as the QR algorithm, which has been so effective on serial machines, must be supplanted by algorithms that map more readily onto parallel architectures. For the symmetric eigenvalue problem, promising algorithms that have been investigated include bisection/multisection, followed by inverse iteration [21, 22, 20], Cuppen's divide and conquer algorithm [9, 14, 28], Jacobi methods [29, 7, 10, 30], and homotopy methods [25]. Parallelizable algorithms for

*This paper is PRISM Working Note #20, available via anonymous ftp to `ftp.super.org` in the directory pub/prism.

†Center for Computing Sciences, 17100 Science Drive, Bowie, MD 20715

1991 *Mathematics Subject Classification.* 65F15.
Key words and phrases. eigenvalues, divide and conquer algorithm, invariant subspaces, parallel algorithm.

dense nonsymmetric matrices that have been investigated include the QR algorithm [3, 32], Jacobi-like methods [31], homotopy methods [24], and the matrix sign function approach to computing invariant subspaces [6, 11, 12, 19, 26, 4].

The purpose of this paper is to present preliminary research results on a new algorithm for finding all the eigenvalues and eigenvectors of a real diagonalizable matrix with real eigenvalues. Although this class of matrices is not completely general, it includes the important class of real symmetric matrices. Our algorithm is based on theoretical ideas of Auslander and Tsao [2]. They propose an algorithm for approximating invariant subspaces of a matrix through the computation of matrix polynomials with special properties. This, in turn, would allow block triangularization of the matrix into two independent subproblems of smaller size via a suitably chosen orthogonal similarity transformation. The computation of polynomials results in an algorithm rich in matrix-matrix multiplication, and computation of the orthogonal transformation matrix is equivalent to solving a system of linear equations. The preponderance of "fast" parallel primitives, such as matrix-matrix multiplication and solving systems of equations, coupled with the divide and conquer nature of the block triangularization, yields a highly parallelizable algorithm, in principle. A similar divide and conquer algorithm using rational functions can be found in [6].

We first introduce some standard notation that will be used throughout the paper. Matrices and vectors will be represented by upper- and lower-case letters, respectively. We denote by $\mathbb{R}^m$, $\mathbb{R}^{m\times n}$, and $\mathbb{R}[x]$ the vector space of m-dimensional real vectors, the algebra of $m \times n$ real matrices, and the algebra of real polynomials, respectively.

The problem we consider is the following: Given a diagonalizable matrix $A \in \mathbb{R}^{n\times n}$ with real eigenvalues, find all the eigenvalues and eigenvectors of A. The algorithm we describe computes an orthogonal matrix Z such that $T = Z^t A Z$ is upper triangular, i.e.,

$$T = \begin{bmatrix} T_{11} & \cdots & T_{1n} \\ & \ddots & \vdots \\ 0 & & T_{nn} \end{bmatrix}. \tag{1.1}$$

The T_{ii}, $i = 1, \ldots, n$ are the eigenvalues of A, and the vectors Zx_i, $i = 1, \ldots, n$ are the eigenvectors of A, where x_i is the solution to the system of equations given by

$$Tx_i = T_{ii}x_i. \tag{1.2}$$

The matrix T in (1.1) is the *Schur decomposition* of A.

We first review some basic facts from invariant subspace theory. Let $\mathcal{X}$ be an invariant subspace of A having dimension r. Any orthogonal matrix, $Q = [\,X \quad Y\,]$, such that $\mathcal{X} = \mathcal{R}(X)$, has the property that

$$Q^t AQ = \begin{bmatrix} A_1 & H \\ 0 & A_2 \end{bmatrix},$$

where A_1 and A_2 are $r \times r$ and $(n-r)\times(n-r)$ matrices, respectively. Here, $\mathcal{R}(X)$ denotes the range space of X. The original problem has thus been decomposed into two independent subproblems, A_1 and A_2, which can be solved totally independently.

We now describe the method proposed by Auslander and Tsao for computing invariant subspaces of A. Assume that A has eigenvalues $\lambda_1, \ldots, \lambda_n$. Consider a matrix polynomial $\mathfrak{a}(A)$, where $\mathfrak{a} \in \mathbb{R}[x]$. It is well known [18] that $\mathfrak{a}(A)$ has eigenvalues $\mathfrak{a}(\lambda_1), \ldots, \mathfrak{a}(\lambda_n)$. Suppose that $\mathcal{R}(\mathfrak{a}(A))$ is a nonempty proper subspace of $\mathbb{R}^n$ of dimension r, i.e., $\mathfrak{a}$ maps exactly $n-r$ eigenvalues of A to 0, counting multiplicities. Then, $\mathcal{R}(\mathfrak{a}(A))$ is an invariant subspace of A, and we say that $\mathfrak{a}$ (or $\mathfrak{a}(A)$) is a *rank-r invariant subspace annihilator* of A. Let $Q = [\,X \quad Y\,]$ be an orthogonal matrix such that $\mathcal{R}(X) = \mathcal{R}(\mathfrak{a}(A))$. Then it is clear that Q has the desired properties.

The Schur decomposition of A can be effected by a recursive application of the following algorithm.

Invariant subspace decomposition algorithm (ISDA).

I. (**Invariant Subspace Annihilation**) *Compute a polynomial in A, $\mathfrak{a}(A)$, which maps $n-r$ ($0 < r < n$) of the eigenvalues of A to 0.*

II. (**Invariant Subspace Computation**) *Compute an orthogonal matrix*

$$Q = [\,X \quad Y\,]$$

such that $\mathcal{R}(X) = \mathcal{R}(\mathfrak{a}(A))$.

III. (**Decoupling**) *Compute* X^tAX *and* Y^tAY.

IV. (**Invariant Subspace Accumulation**) *To compute the eigenvectors, use* Q *to update both the upper triangle of* A *and the eigenvector matrix.*

This idea can be applied recursively until all subproblems are upper triangular matrices, leading to a divide and conquer algorithm having a tree-like structure where the number of subproblems doubles at each level in the tree. Ideally, one would like r to be as close to $n/2$ as possible. If the invariant subspaces are also desired, subsequent change-of-basis matrices arising from solving A_1 and A_2 are accumulated and used to perform appropriately chosen left and right multiplications of the upper triangle of Q^tAQ, respectively.

We remark that if A is symmetric, then Q^tAQ is block diagonal, eliminating both the need to update the upper triangle in succeeding stages and the backsolve given by (1.2). Note that orthogonality in the computed eigenvectors is guaranteed by ISDA in this case.

In Section 2, we first discuss the serial algorithm and, in particular, describe our algorithm for computing the desired matrix polynomials. Numerical and timing results in single precision on a single processor of a Cray-2 and on an IBM RS/6000 Model 580 workstation are given in Section 3. Our experimental results indicate that the resulting eigensolver is extremely effective numerically on matrices with real eigenvalues. In Section 4, we indicate why the algorithm has a high potential for parallelism.

2. The numerical algorithm.

A reasonable candidate for an approximate invariant subspace annihilator is a polynomial $\hat{\mathfrak{a}}$ such that $\hat{\mathfrak{a}}(A)$ is strongly numerically rank deficient. Loosely speaking, this means that $\hat{\mathfrak{a}}(A)$ must have a large gap in its eigenvalues. We begin then by describing our algorithm for computing such matrices. Ideally, one would like the matrix $\hat{\mathfrak{a}}(A)$ to map approximately half the eigenvalues of A near 0. Our algorithm constructs $\hat{\mathfrak{a}}$ by first performing a **Scaling** step followed by an **Eigenvalue Smoothing** step. We borrow the term "smoothing" from digital filter theory [17]. The **Scaling** and **Eigenvalue Smoothing** steps proceed as follows:

Scaling: Compute bounds on the spectrum $\lambda(A)$ of A and use these bounds to compute α and β such that for $\ell(x) = \alpha x + \beta$, $\lambda(\ell(A)) \subseteq [0,1]$, with the mean eigenvalue of A being mapped to $1/2$.

Eigenvalue Smoothing: Let $p_i(x)$, $i = 1, 2, \ldots$ be polynomials such that in the limit values in $[0, 1/2)$ are mapped near 0 and values in $(1/2, 1]$ are mapped near 1. Iterate

$$B_0 = \ell(A), B_i = p_i(B_{i-1}), i = 1, 2, \ldots,$$

until $||B_i - B_{i-1}||$ is numerically negligible (in iteration K, say), at which point all the eigenvalues of the iterated matrix are near either 0 or 1.

In other words, $\hat{\mathfrak{a}}$ is the composition $p_K \circ \cdots p_1 \circ \ell$.

2.1. Scaling scheme. The requirement that the polynomial ℓ map $\lambda(A)$ into $[0,1]$ is just a convenience. Note, however, that in order for $\hat{\mathfrak{a}}$ to map half the spectrum of A near 0, ℓ must map roughly half the eigenvalues of A into $[0, 1/2)$. Furthermore, when computing in finite precision, it is desirable to cluster the nonzero eigenvalues in order to maximize the dynamic range available for estimating the size of the gap. There is no computationally inexpensive means to compute the median of $\lambda(A)$, but certainly the mean $\mu = \mathrm{tr}(A)/m$, of $\lambda(A)$ suffices in many instances, where $\mathrm{tr}(A)$ denotes the trace of A. Let ω and Ω be a lower and upper bound on $\lambda(A)$, respectively. In our implementation, we use the bounds provided by Gershgorin disks [16] as ω and Ω. Then we let ℓ be the linear map that maps $\lambda(A)$ into as large a subinterval of $[0,1]$ as possible so that $\ell(\mu) = 1/2$. That is,

$$\ell(x) = \begin{cases} \frac{1}{2}\left(\frac{x-\mu}{\Omega-\mu} + 1\right), & \text{if } \mu \leq \frac{\omega+\Omega}{2}, \\ \frac{1}{2}\left(\frac{x-\mu}{\mu-\omega} + 1\right), & \text{if } \mu > \frac{\omega+\Omega}{2}. \end{cases}$$

The behavior of ℓ is illustrated in Figure 2.1.

2.2. Eigenvalue Smoothing.

2.2.1. Iteration scheme. We now consider construction of the polynomials p_i $(i = 1, 2, 3, \ldots)$.

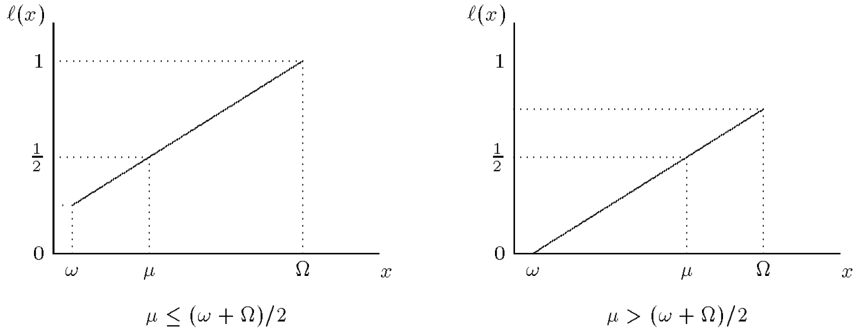


FIGURE 2.1. Function ℓ

The suitably normalized incomplete Beta functions [17] (Section 7.2) given by

$$\mathfrak{B}_j(x) = \frac{\displaystyle\int_0^x t^j(1-t)^j\,dt}{\displaystyle\int_0^1 t^j(1-t)^j\,dt} = \sum_{k=0}^{j} \binom{2j+1}{j-k}\binom{j+k}{k}(-1)^k x^{j+k+1}, \quad j \in \mathbb{N} \tag{2.1}$$

form an infinite family of candidates for the p_i. Note that for each j, $\mathfrak{B}_j$ is a polynomial of degree $2j+1$ that increases on $[0,1]$ and has fixed points at 0, 1/2, and 1. Let χ be the function defined on $[0,1]$ by

$$\chi(x) = \begin{cases} 0, & \text{if } 0 \le x < \frac{1}{2}; \\ \frac{1}{2}, & \text{if } x = \frac{1}{2}; \\ 1, & \text{if } \frac{1}{2} < x \le 1. \end{cases}$$

An obvious approach is to let $p_i = \mathfrak{B}_i$ $(i = 1, 2, 3, \ldots)$ since for $x \in [0,1]$,

$$\lim_{j\to\infty} \mathfrak{B}_j(x) = \chi(x).$$

It is clear that in this approach, K would need to be prohibitively high, making this approach infeasible. A better approach is to simply choose one polynomial in the family given by (2.1) and apply it recursively, i.e., since for fixed $k \in \mathbb{N}$ and $x \in [0,1]$,

$$\lim_{i\to\infty} \mathfrak{B}_k^{(i)}(x) = \chi(x). \tag{2.2}$$

Here

$$\mathfrak{B}_k^{(i)}(x) = \underbrace{\mathfrak{B}_k(\mathfrak{B}_k(\cdots(\mathfrak{B}_k(x))))}_{i \text{ times}}.$$

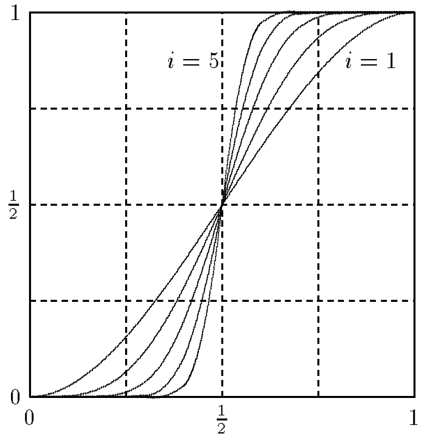


FIGURE 2.2. Behavior of $\mathfrak{B}_1^{(i)}$

In our implementation, we choose $k = 1$. Note that $\mathfrak{B}_1(x) = 3x^2 - 2x^3$. In Figure 2.2, we see how quickly this iteration converges.

Table 2.1 gives empirical support of our belief that either $k = 1$ or $k = 2$ is the best choice in terms of amount of computation that would be required. Let $\mathbf{u}$ be the machine roundoff unit; then the number $1/2 - \mathbf{u}$ is the largest number in $[0, 1/2)$ that can be distinguished from $1/2$. The second column of Table 2.1 gives the smallest integer, N, such that

$$\mathfrak{B}_k^{(N)}(\frac{1}{2} - \mathbf{u}) < \mathbf{u},$$

where $\mathbf{u}$ is the Cray-2 machine roundoff unit 2^{-48}. The third column gives the approximate degree of $\mathfrak{B}_k^{(N)}(A)$, and the last column gives the number of matrix multiplications that would be required to compute $\mathfrak{B}_k^{(N)}(A)$ if A has an eigenvalue equal to $1/2 - \mathbf{u}$. Although the table indicates that $\mathfrak{B}_2$ may be preferable to $\mathfrak{B}_1$, $\mathfrak{B}_1$ was chosen over $\mathfrak{B}_2$ because it has a local minimum and maximum at 0 and 1, respectively. This property ensures that eigenvalues mapped outside $[0, 1]$ because of machine roundoff will tend to be mapped back into $[0, 1]$ by subsequent applications of $\mathfrak{B}_1$.

It is clear that the more accurately ω and Ω bound $\lambda(A)$, the fewer iterations will be required. For each of the two subproblems generated by $\hat{\mathfrak{a}}(A)$, the mean value of $\lambda(A)$, $\mu(A)$, provides either an upper or a lower bound on the spectrum. The scheme just described is supplemented by the values of $\mu(A)$ to provide better bounds for subsequent subproblems.

k	N	approximate degree of $\mathfrak{B}_k^{(N)}$	# matrix multiplications
1	86	1.1×10^{41}	172
2	55	2.8×10^{38}	165
3	45	1.1×10^{38}	180
4	39	1.6×10^{37}	195
5	36	3.1×10^{37}	216
6	33	5.8×10^{36}	231

TABLE 2.1. Computation needed to map $1/2 - \mathbf{u}$ to a value less than $\mathbf{u}$ ($\mathbf{u} = 2^{-48}$)

2.2.2. Accelerated iteration scheme. We actually employ a modified version of this basic iteration that significantly reduces the number of iterations of $\mathfrak{B}_1$ required in the early stages of the divide and conquer. As we discuss in Section 4, most of the work in ISDA occurs in the early divides and hence efforts to improve performance must be aimed at these divides. In fact, in the early divides, the number of applications of $\mathfrak{B}_1$ required tends to be larger than in later stages. One reason for this is that when no *a priori* spectral information is available, **Scaling** is done using bounds obtained from Gershgorin disks. Since these bounds are generally quite poor, $\ell(A)$ tends to have eigenvalues closer to 1/2 than would be the case if better bounds on the spectrum were available, as is the case in later divides. Since the convergence rate for values near 1/2 is very slow using only $\mathfrak{B}_1$, we sought strategies to improve the rate of convergence for matrices having eigenvalues near 1/2.

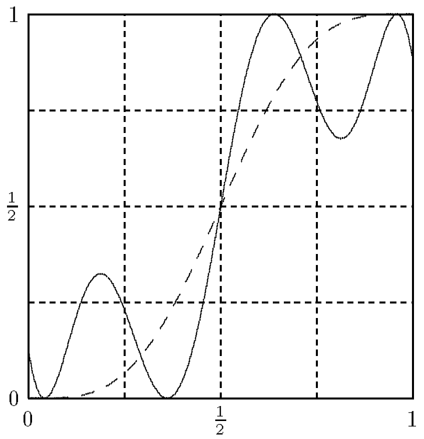


FIGURE 2.3. Behavior of Acceleration Technique

$\mathfrak{B}_1$ takes on the value 1/2 three times: at 1/2, ρ, and $1-\rho$, where $\rho = (1+\sqrt{3})/2 \approx 1.366$. We propose the following scheme, which is a slight modification of a technique suggested by Pan and

Schreiber [27]. They essentially observed that if we take the matrix $\ell(A)$ from the **Scaling** step and "stretch" it so that its eigenvalues now lie over some interval, say $[-s, 1+s]$, where $0 < s \leq \rho - 1$, then the eigenvalues of $\ell(A)$ near 1/2 are moved further away from 1/2 and $\mathfrak{B}_1$ will still map the eigenvalues of $\ell(A)$ into $[0, 1]$. By "stretching", we mean to apply a linear function that maps 0 and 1 to $1-s$ and s, respectively, leaving 1/2 fixed. Repeating this strategy several times, namely, a "stretch" followed by one application of $\mathfrak{B}_1$, at the beginning of the **Eigenvalue Smoothing** step leads to a substantial reduction in the number of iterations required in the early stages of the algorithm. Since values near $(1 \pm \sqrt{3})/2$ are mapped near 1/2, there is a tradeoff to be made in our choice of s. We have found that applying this strategy six times with $s = .3$ leads to about a 1/3 reduction in the number of iterations required in the early stages of the algorithm. Figure 2.3 compares the effect of two iterations of this acceleration strategy (solid curve) versus two regular iterations of $\mathfrak{B}_1$ (dashed curve). Note the poorer behavior of this iteration near 0 and 1; this is offset by the substantially improved convergence for values near 1/2. In any case, values away from 1/2 converge quadratically to either 0 or 1 in the later iterations, so this boundary behavior does not in fact prove to be detrimental.

In the latter stages of ISDA, because good bounds can be ascertained from previous divides, divides tend to occur quickly without acceleration and use of the acceleration strategy often leads to increased numbers of iterations. Therefore, we do not apply this technique to small problems. In any case, since the majority of the computation performed by ISDA occurs in the early divides, the savings realized results in a significant performance improvement. We have observed improvements in run time of roughly 25%. The number of iterations required is now typically between 15 and 20 for the first divide, as opposed to between 25 and 30 for the basic iteration without the acceleration technique.

2.3. Convergence criterion. Since the matrix A is diagonalizable, the sequence of matrices, $\{B_i\}_{i=1}^{\infty}$, in the **Eigenvalue Smoothing** step converges when performing exact arithmetic. In practice, we check for convergence by examining the behavior of

$$\Delta_i(A) \equiv \frac{\|B_i - B_{i-1}\|_1}{\|B_i\|_1}, \quad i = 2, 3, \ldots.$$

In most cases, we use the following test for convergence:

$$\Delta_i(A) \leq C_s \mathbf{u}, \tag{2.3}$$

where C_s is a positive constant. This stopping criterion is a necessary but not sufficient condition for convergence of the sequence $\{B_i\}_{i=1}^{\infty}$. It has proven to be very reliable in practice and eliminates the need to check for rank deficiency after each iteration. Application of $\mathfrak{B}_1$ in the later iterations leads to quadratic convergence when the eigenvalues are far enough from 1/2. The thresholds given in Table 2.2 were used to obtain the results presented in Section 3 and were empirically determined to perform satisfactorily in the ranges of dimension shown in the figures in Section 3.

Architecture	precision	$\mathbf{u}$	$C_s\mathbf{u}$
Cray-2	single	2^{-48}	2^{-33}
RS/6000	single	2^{-24}	2^{-16}
RS/6000	double	2^{-52}	2^{-36}

TABLE 2.2. Convergence thresholds

The values of the mean eigenvalue, μ, are also of great practical value in detecting clusters of nearly identical eigenvalues. Since early cluster detection can greatly reduce the amount of work done, we use a simple heuristic scheme that chooses whichever of A_1 or A_2 has all of its eigenvalues on the same side of 0 as the mean eigenvalue of A. Furthermore, $|\mu|$ is always a lower bound on the spectral radius of the original matrix. A running estimate, Λ, of the largest mean eigenvalue in magnitude from already completed divides is kept. When the bounds used in the **Scaling** step of ISDA indicate that all the eigenvalues of the current subproblem are either $O(\mathbf{u}\Lambda)$ or within $O(\mathbf{u}\Lambda)$ of each other (recall $\mathbf{u}$ is the machine epsilon), then the subproblem is declared to have clustered eigenvalues and to be "done."

Thus, for instance, matrices with exponentially distributed eigenvalues did not prove to be as expensive computationally as might be expected. A matrix with exponentially distributed eigenvalues could require $O(n^4)$ computation if such monitoring of μ is not done. This is avoided in practice because poorly conditioned matrices have clustered eigenvalues that are quickly detected by this scheme. Note that the problem of invariant subspace sensitivity is also avoided. We just remark that eigenvalues that are extremely tightly clustered around $1/2$ after the application of the function ℓ tend to all move in the same direction away from $1/2$ under the action of $\mathfrak{B}_1$.

The case of clustered eigenvalues merits additional discussion. The number of iterations is limited to a maximum of 50 in our implementation. If the stopping criterion fails to be satisfied after 50 iterations, we check for rank deficiency anyway. If the matrix fails to be rank deficient, we conclude that the subproblem must have only one eigenvalue. The stopping criterion is augmented by an additional check for divergence,

$$\Delta_i(A) > \Delta_{i-1}(A), \tag{2.4}$$

when $\Delta_i(A) \leq \sqrt{\mathbf{u}}$. This check was necessary in a few cases where the matrix had clustered eigenvalues and our stopping criterion was too restrictive. We do not fully understand this phenomenon at this time. "Divergent" behavior was also observed when the matrix had imaginary eigenvalues, since our algorithm is not always well-behaved in this case.

In general, if K is the smallest positive integer for which (2.3) is satisfied, we verify that the resulting matrix, B_K, does, indeed, have a large gap in its singular values. This was done by computing its *QR factorization with column pivoting* [13], given by

$$B_K \Pi = QR, \tag{2.5}$$

where Π is a permutation matrix, Q is an orthogonal matrix, and $R = [\, R_{ij} \,]$ is an upper triangular matrix whose diagonal elements are arranged in order of decreasing absolute value. In practice, if $|R_{r+1,r+1}/R_{rr}|$ is small, then there is a large gap between the r-th and $(r+1)$-st singular values of B_K, and the first r columns of $B_K\Pi$ will form a good approximate basis for $\mathcal{R}(B_K)$. We declare the matrix B_K to have rank r if

$$|R_{r+1,r+1}| \leq \sqrt{\mathbf{u}} |R_{rr}|. \tag{2.6}$$

We then let $\hat{\mathfrak{a}}(A) = B_K$ and perform the orthogonal change of basis given by Q. As noted in [15], if $\hat{\mathfrak{a}}(A)$ has a large gap in the singular values, then QR factorization with column pivoting should generally perform well at detecting rank deficiency and as a means of computing $\mathcal{R}(\hat{\mathfrak{a}}(A))$. We used the routine xGEQPF in LAPACK [1] for this computation. Rather surprisingly, our experiments showed that requiring a gap larger than $\sqrt{\mathbf{u}}$ produced a less effective algorithm.

2.4. Decoupling problem. The computations in the **Decoupling** and **Invariant Subspace Accumulation** steps are straightforward. However, the algorithms used for the symmetric and nonsymmetric cases do differ in that symmetry is enforced after all operations when the matrix is symmetric. First, we perform the operations in the **Decoupling** step using a sequence of rank-1 updates, thereby enforcing symmetry. Additionally, in the symmetric case, the application of p_i requires computing $M^3 = M^2 M$, where M is a symmetric matrix. Symmetry is maintained by computing M^3 as follows. We first perform the dense matrix multiplication M^2M and then average symmetric entries with respect to the diagonal. This corresponds mathematically to computing $(M^3 + (M^3)^t)/2$. These methods of symmetrizing M^3 were chosen for convenience rather than efficiency.

Since all the change of basis matrices are orthogonal, if the norm of the lower triangular block, $\|Y^tAX\|_2$, is small for each subproblem A, then we are guaranteed that our solution is the exact eigensystem of a small perturbation of A. We monitored the size of Y^tAX at each stage of the algorithm and have never encountered a test case where this value is large, even for nonsymmetric matrices.

We note that $\mathfrak{B}_1(x) = (\mathfrak{n}(2x-1)+1)/2$, where $\mathfrak{n}$ is the Newton-Schulz iteration given by $\mathfrak{n}(x) = (3x - x^3)/2$. A discussion of the behavior of the Newton-Schulz iteration can be found in [23]. In particular, the discussion in [23] illustrates the difficulties of extending our methodology to the complex case.

Another method of performing the **Invariant Subspace Annihilation** is to scale A so that the mean eigenvalue is mapped to 0 and to let $p_i = \mathcal{S}$, $i = 1, 2, \ldots$, where $\mathcal{S}(x) = (x + 1/x)/2$ is the matrix sign function. In the limit all eigenvalues that are not purely imaginary are mapped to either -1 or 1. One can then scale the result to produce a matrix having eigenvalues 0 and 1. We considered this approach, but did not adopt it for three reasons. First, the number of iterations required for the matrix sign approach and the accelerated incomplete Beta function approach are comparable, but we expect dense matrix multiplication to be more scalable on modern multiprocessor architectures. Second, the computation of matrix inverses is more problematic numerically than matrix multiplication. Lastly, $\mathcal{S}$ has a singularity at the origin, so the algorithm could fail to converge. This difficulty can be overcome by applying simple shifting techniques, but at the expense of more computation. We therefore feel that the Beta function approach promises more robust, scalable performance than the matrix sign approach for the matrices we are considering. However, for the general nonsymmetric eigenvalue problem where the matrices may have complex eigenvalues, the matrix sign approach is quite promising [6, 11, 12, 19, 26, 4].

3. Test cases. Testing of the algorithm described was performed on both nonsymmetric and symmetric matrices. Even though the code performs dense computations and does not take advantage of sparsity, we tested our algorithm on both dense and upper Hessenberg matrices, since the reduction to upper Hessenberg form is a standard one. Analogously, in the symmetric case, we tested ISDA on both dense and symmetric tridiagonal matrices. Since, in our testing, accuracy in the residuals was comparable for the dense and sparse forms, we present only results for dense matrices.

A large suite of test matrices were generated using the LAPACK test generation routines xLATME (nonsymmetric) and xLATMS (symmetric) [1] . xLATME allows one to generate matrices of the form

$$A = (U^t \Sigma V)^{-1} D (U^t \Sigma V),$$

where U, V are random orthogonal matrices and D, Σ are diagonal matrices. In addition, xLATME provides options for varying the distribution of the diagonal entries of Σ and D, $\mathrm{cond}(\Sigma)$, $\mathrm{cond}(D)$,

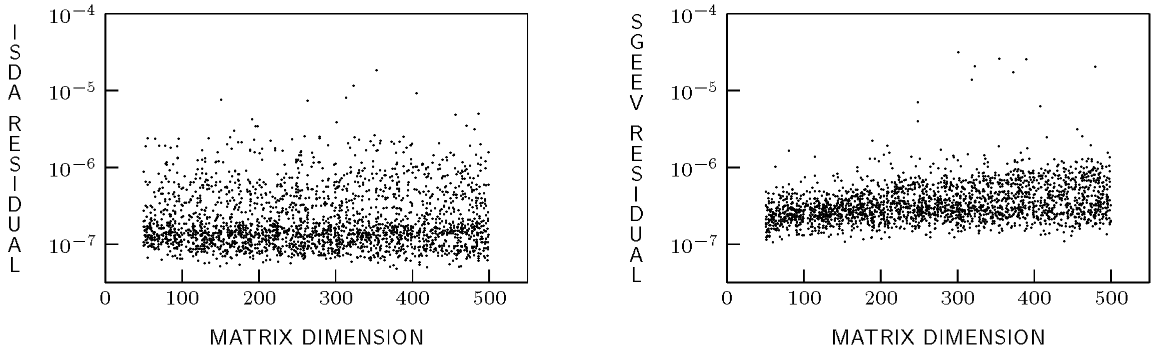


FIGURE 3.1. Residuals for dense nonsymmetric matrices (RS/6000, single precision)

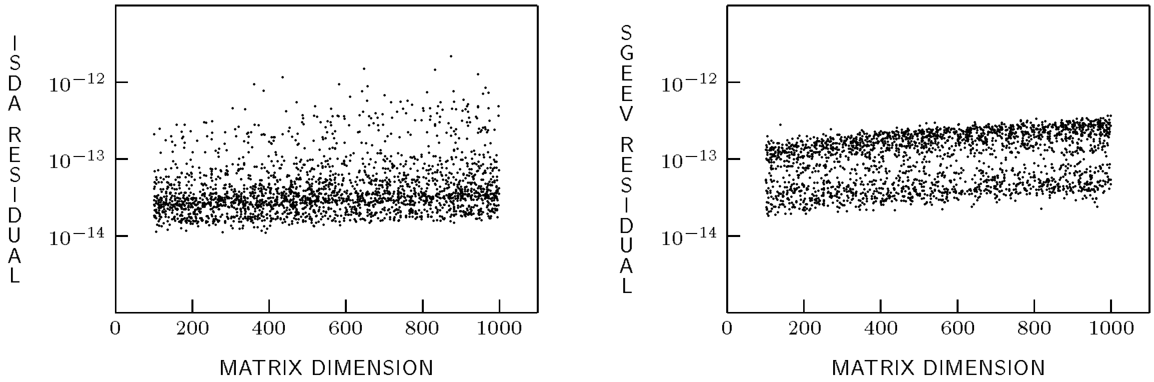


FIGURE 3.2. Residuals for dense nonsymmetric matrices (Cray-2, single precision)

$\lambda(A)$, and $\max_{i,j} |A_{ij}|$. These options allow the user to generate a wide variety of ill-conditioned eigenvalue problems. Due to the fact that our algorithm can handle only matrices with real eigenvalues, we restricted our attention to cases where we believed the eigenvalues to actually be real by fixing $\text{cond}(\Sigma)$ to be between one and ten. The performance of ISDA for both dense and upper Hessenberg matrices was compared to the LAPACK implementations (Release 1.1) of the QR algorithm for dense (xGEEV) and upper Hessenberg matrices (xHSEQR), respectively. Since the eigenvalues are somewhat insensitive to perturbation under these conditions [5], it was reasonable to rely on xGEEV or xHSEQR to filter out cases with complex eigenvalues. Our algorithm was only applied to those matrices where the eigenvalues were "close" to real according to xGEEV or xHSEQR. Analogously, xLATMS constructs symmetric matrices of the form

$$A = U^t D U,$$

where U is a random orthogonal matrix and D is a diagonal matrix. xLATMS provides options for choosing the distribution of the diagonal entries of D, $\mathrm{cond}(D)$, and $\lambda(A)$. Except for the restriction on $\mathrm{cond}(\Sigma)$ noted above, matrices for testing were generated by randomly selecting input parameters for xLATME and xLATMS that covered a substantial subset of the dynamic range of the machine's arithmetic.

3.1. Numerical results. Symmetric and nonsymmetric test cases of dimensions 100–1000 and 50–500 were generated as described above for testing of our algorithm on a Cray-2 and an IBM RS/6000 Model 580, respectively. Accuracy in the residuals for a given matrix A was quantified by computing the maximum normalized 2-norm residual

$$\max_i \frac{\|Ax_i - \lambda_i x_i\|_2}{\|A\|_F}, \qquad \|x_i\|_2 = 1,$$

where x_i is the computed eigenvector corresponding to the eigenvalue λ_i. For symmetric matrices, we also computed the departure from orthogonality residual given by

$$\max_{i,j} \left| [Z^t Z - I_n]_{ij} \right|$$

to verify that the computed eigenvectors were, indeed, orthonormal. Here Z is the matrix of eigenvectors.

Between 2000 and 3000 test cases were run on a Cray-2 in single precision (64-bit) and on an RS/6000 in single (32-bit) and double (64-bit) precision for both the dense nonsymmetric and symmetric cases. Figures 3.1-3.6 show plots of single precision residuals for dense matrices on both a Cray-2 and an RS/6000. The double precision results on the RS/6000 produced analogous results. Figures 3.1 and 3.2 show plots of the residuals for dense nonsymmetric diagonalizable matrices with real eigenvalues from both ISDA and SGEEV plotted versus matrix dimension. In Figures 3.3–3.4 we give plots of the maximum residual versus matrix dimension for dense symmetric matrices for both ISDA and SSYEV in LAPACK [1]. Figures 3.5 and 3.6 show plots of the departure from orthogonality residuals for both ISDA and SSYEV plotted versus matrix dimension. The accuracy

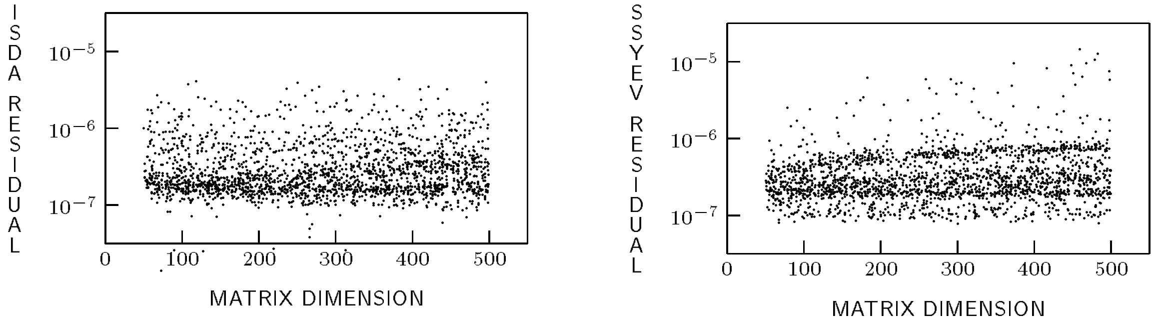


FIGURE 3.3. Residuals for dense symmetric matrices (RS/6000, single precision)

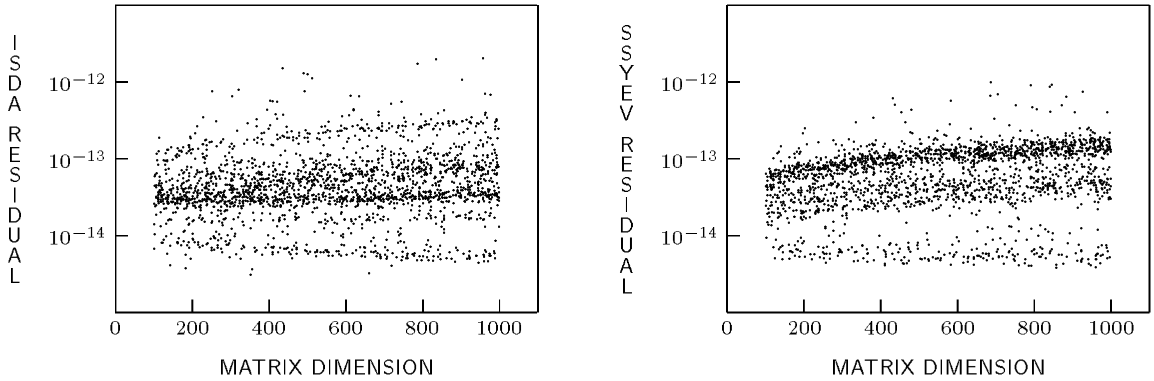


FIGURE 3.4. Residuals for dense symmetric matrices (Cray-2, single precision)

of ISDA, as measured by the maximum residual and the departure from orthogonality, is comparable to that of SSYEV on the cases tested.

We use the notation (b_i, d_j, b_i) to denote the symmetric tridiagonal matrix having diagonal entries d_j, $j = 1, \ldots, n$ and symmetric off-diagonal bands with entries b_i, $i = 1, \ldots, n-1$. In addition to "random" testing, we also tested the symmetric version of our algorithm on a few standard classes of special tridiagonal matrices: $(1, 2, 1)$ matrices; Wilkinson matrices $W^+_{2k+1} = (1, |k+1-i|, 1)$, $i = 1, \ldots, 2k+1$; and "glued" W^+_{21} matrices, $G^{k,\epsilon}_{21}$. For $k \in \mathbb{N}$ and $\epsilon > 0$, $G^{k,\epsilon}_{21}$ is defined to be a matrix of dimension $21k$ where

$$b_i = \begin{cases} \epsilon, & \text{if } i = 0 \mod 21, \\ 1, & \text{otherwise} \end{cases}$$

and

$$d_j = |10 - ((j-1) \mod 21)|.$$

The Wilkinson matrices W_{2k+1}^{+} have increasingly pathologically close pairs of eigenvalues as k increases. The "glued" Wilkinson matrices are pathological for values of ϵ that are large relative to $\mathbf{u}$. We tested ISDA on this class of matrices for a sampling of such values of ϵ and accuracy was comparable in all cases to that shown in Figures 3.3–3.6.

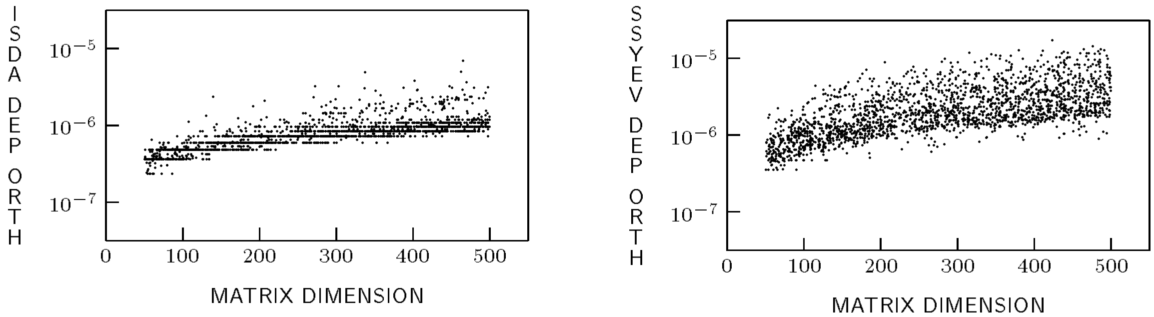


FIGURE 3.5. Departure from orthogonality for dense symmetric matrices (RS/6000, single precision)

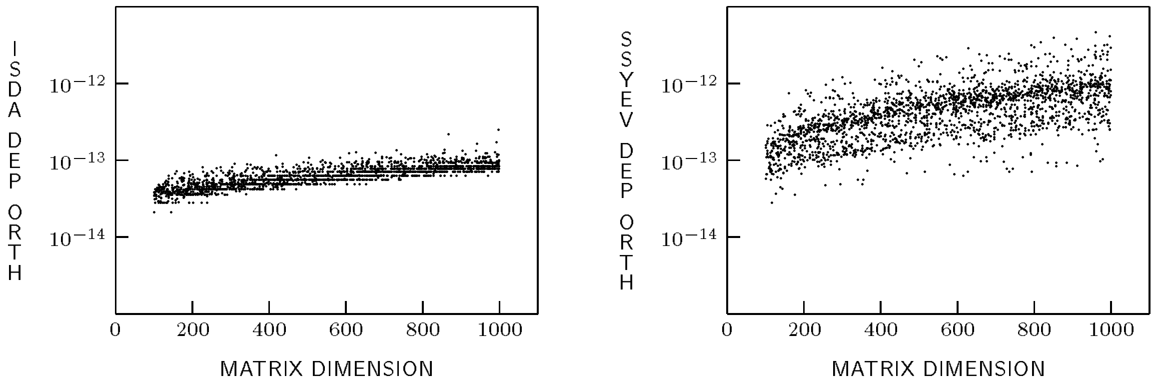
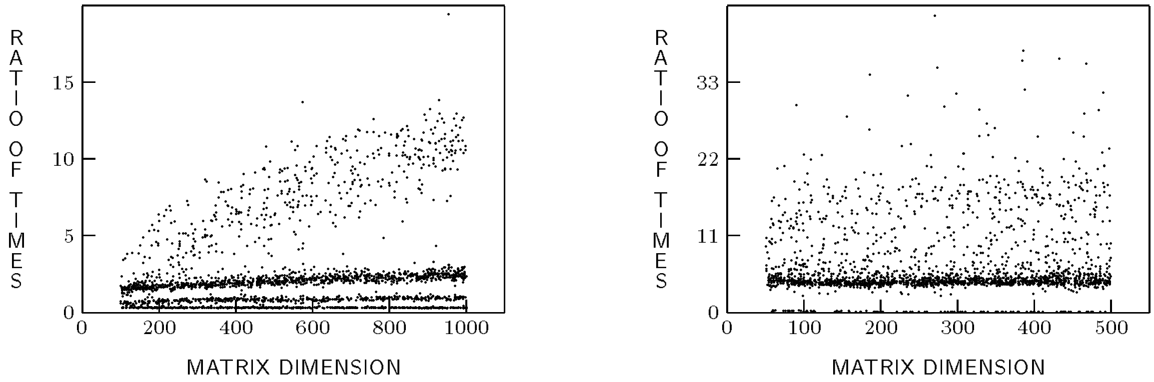


FIGURE 3.6. Departure from orthogonality for dense symmetric matrices (Cray-2, single precision)

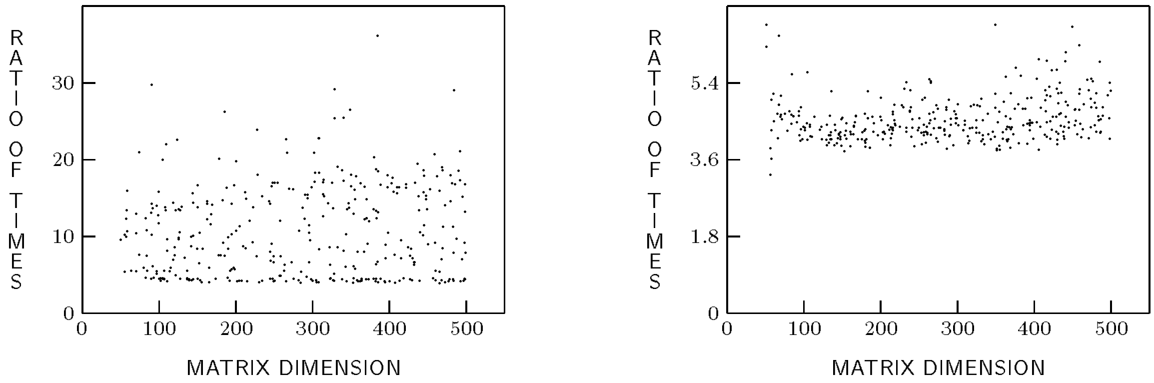
3.2. Timing results. Although this research was primarily directed towards understanding the numerical issues of this new algorithm, efficiency of the algorithm is also important. Figure 3.7 (a) shows the ratio of times for ISDA as compared to SGEEV for single precision dense nonsymmetric matrices on the Cray-2. It should be pointed out that all of the scatter above a ratio of 4 is attributable to test cases having mode ± 3, i.e., exponentially distributed eigenvalues, from the generation routine xLATME. We are examining better ways for the algorithm to detect and handle such distributions. Figure 3.7 (b) shows the ratio of times for ISDA as compared to SSYEV for single precision dense symmetric matrices on the RS/6000. Again, much, but not all, of the scatter is attributable to matrices having exponentially distributed eigenvalues. Figure 3.8 points out the effect of the eigenvalue distribution on the runtime of the algorithm–mode ± 3 matrices require considerably more time than do, say matrices produced with mode ± 4, i.e., uniformly distributed eigenvalues.



(a) ISDA to SGEEV (Cray-2, dense non-symmetric matrices)

(b) ISDA to SSYEV (RS/6000, dense symmetric matrices)

FIGURE 3.7. Ratio of times



(a) Mode ± 3

(b) Mode ± 4

FIGURE 3.8. Ratio of times on RS/6000, dense symmetric matrices

4. Parallel issues. The coarse grain parallelism in the algorithm comes from two main sources: 1) computations that can be performed by having multiple processors all work on a large subproblem and 2) the divide and conquer partitioning of the matrix into multiple smaller subproblems that can be worked on independently. These two different types of parallelism could both be exploited in any multiprocessor implementation.

In order to discuss the amount and type of work that the algorithm performs, the operation counts are presented for the four main steps associated with the ISDA given in Section 1. The analysis below is for the nonsymmetric problem; the symmetric case is analogous. Also, a straightforward unblocked implementation of the ISDA is analyzed in which Q in the **Invariant Subspace Computation** is explicitly formed at each stage. We follow Golub and Van Loan [16] in presenting our operations counts. An operation is defined as one floating point computation, e.g., squaring a matrix of order n takes $2n^3$ operations. We let m represent the size of the subproblem, $\hat{A}$, to be divided, and n is the size of the initial problem, A.

We first discuss the amount of potential parallelism in the early stages of the algorithm, where multiple processors will be working on the same large subproblem. The first step in the ISDA is the **Invariant Subspace Annihilation**. The number of operations required in the **Scaling** step is $O(m^2)$ and therefore insignificant compared to the formation of $\mathfrak{a}(\hat{A})$. Since the computation of B_i requires two matrix multiplications, N applications of $\mathfrak{B}_1$ require $2m^3 \cdot 2N = 4m^3N$ operations. The **Invariant Subspace Computation** via QR factorization with column pivoting on $\mathfrak{a}(\hat{A})$, involves $(8/3)m^3$ operations since Q is formed explicitly. The **Decoupling** step or formation of two independent subproblems via the transformation $Q^t\hat{A}Q$, necessitates two matrix multiplications, or $4m^3$ operations. The **Invariant Subspace Accumulation** step, encompassing the updates of both the invariant subspace of the subproblem of interest and the upper triangle, involves matrix multiplications totalling $4nm^2 + 2m^3$ operations. Thus, the total work associated with dividing a subproblem is $4m^3N + (26/3)m^3 + 4nm^2$ operations. To simplify the analysis, assume that the subproblem being divided is the initial matrix ($n = m$) and the

$$\text{total operations to divide } A = n^3\left(4N + \frac{38}{3}\right).$$

Note that **Eigenvalue Smoothing**, **Decoupling**, and **Invariant Subspace Accumulation** are all matrix-matrix multiplication-based. It is easy to show that the

$$\text{fraction of operations in dividing } A \text{ spent in matrix multiplication} \geq \frac{2N+5}{2N+\frac{19}{3}}.$$

Empirical results indicate that, on the first divide, N is between 15 and 20 for matrices of dimension between 500 and 1000 with uniformly distributed eigenvalues. Using $N = 15$, we find that matrix multiplication is approximately 96.3% of the total operations count for the first divide of the ISDA. This result is very encouraging since it seems reasonable to presume that any scientific multiprocessor will be able to efficiently perform matrix multiplication in parallel. For larger values of N, this percentage will, of course, increase, but at the expense of greater total work. Additionally, even though the QR with column pivoting in **Invariant Subspace Computation** is not included as being matrix multiplication-based, Bischof [8] has shown it can be run in parallel with controlled local pivoting. Thus, subproblems of sufficient size should run efficiently on a multiprocessor due to the large fraction of matrix multiplications and the existence of a parallel QR algorithm.

The second form of coarse-grain parallelism is the divide and conquer aspect of the algorithm. This allows different groups of processors to work independently on different subproblems. In order to develop a simplified model for the divide and conquer behavior of the algorithm, two assumptions are made. The first is that the two subproblems spawned are each half the size of the generating subproblem. It is clear that this is a reasonable assumption for matrices with uniformly distributed eigenvalues, and this has been confirmed in our testing. We shall, therefore, assume that $n = 2^k$ for some $k \in \mathbb{N}$. Skewed distributions, such as exponential distributions, do cause unequal divides since the mean of the eigenvalues differs greatly from the median. The second assumption is that N is the same for all subproblems. Empirical results show that N varies for different subproblems but is largest for the early divides of the problem. For the results given below, the specific choice of N does not significantly vary the result.

With these two assumptions, the divide and conquer aspect of the algorithm can be viewed as a balanced tree with levels 0 to $(\log_2 n) - 1$. The i-th level in the tree has 2^i subproblems of size $n/2^i$. Thus, the total work to solve a problem is

$$\begin{aligned} \text{ISDA total work} &= \sum_{i=0}^{(\log_2 n)-1} 2^i \left[4 \left(\frac{n}{2^i} \right)^3 N + \frac{26}{3} \left(\frac{n}{2^i} \right)^3 + 4n \left(\frac{n}{2^i} \right)^2 \right] \\ &= n^3 \left(4N + \frac{26}{3} \right) \frac{4}{3} \left(1 - \frac{1}{n^2} \right) + 4n^3 \cdot 2 \left(1 - \frac{1}{n} \right) \\ &\approx \frac{1}{3} n^3 \left(16N + \frac{176}{3} \right), \quad n >> 1. \end{aligned} \tag{4.1}$$

For $N = 15-20$ in (4.1), we see that under our assumptions, ISDA requires between $100n^3$ and $126n^3$ floating point operations to solve the complete eigenvalue problem. In particular, ISDA requires roughly four to five times as many operations as the nonsymmetric QR algorithm, assuming that the nonsymmetric QR algorithm performs roughly $25n^3$ operations [16]. But even sequentially, we see why dense matrix multiplication is such a desirable primitive. For matrices with uniformly distributed eigenvalues, ISDA is an average of 1.9 times slower than the QR algorithm on the RS/6000 and is about 2.2 times slower than the QR algorithm on the Cray-2. On the other hand, for the symmetric eigenvalue problem, ISDA is an average of 4.7 times slower than the QR algorithm on the RS/6000 and about 5.2 times slower than the QR algorithm on the Cray-2. Assuming that the QR algorithm for symmetric matrices requires $9n^3$ operations, ISDA requires about 11 to 14 times more work than the symmetric QR algorithm. Furthermore, our implementation does not exploit symmetry in the **Eigenvalue Smoothing** step and therefore performs roughly two times more operations than are actually necessary in the symmetric case. We note that matrices with other eigenvalue distributions can take significantly more or less time to solve using ISDA.

One can see that the

$$\text{fraction of work at level } i = \frac{\left(4N + \frac{26}{3}\right) 2^{-2i} + 4 \cdot 2^{-i}}{\frac{1}{3}\left(16N + \frac{176}{3}\right)}.$$

Level (i)	Fraction of work for level	Cumulative fraction of work
0	0.730	0.730
1	0.193	0.923
2	0.053	0.976
3	0.016	0.992

TABLE 4.1. Work done at level i when $N = 15$

Table 4.1 shows that, under these simplifying assumptions, coupled with letting $N = 15$ for all subproblems, 73% of the total work is expended in dividing the initial matrix. Furthermore, by the time that level 2 is completed and eight subproblems exist, only 2.4% of the total work remains. This implies that, for parallel processing, the majority of work will be performed where multiple processors are working on a single subproblem. Thus, it is important that the four steps in the ISDA can be run in parallel in the early stages of the algorithm. As the level increases, the sizes of the subproblems decrease, and the total amount of work available drops. In order to keep a reasonable amount of work available to a group of processors working on a subproblem, the number of processors associated with a given subproblem needs to decrease as the subproblem size decreases. To accomplish this while at the same time keeping all the processors active, multiple subproblems can be worked on simultaneously. Eventually, the number of processors associated with a given subproblem will decrease to the point where an alternate method could be used to solve the remaining subproblems. The combination of these two sources of coarse grain parallelism in the ISDA complement each other in such a way that, as the work associated with each subproblem decreases, the number of subproblems available will increase. This should yield an algorithm with a high parallel utilization. It is clear that the assumptions used in the above analysis will not be appropriate for all matrices, and additional issues, such as load balancing, will need to be addressed.

5. Acknowledgements. The authors would like to thank J. Fischman for his numerous contributions towards improving and testing of our algorithm.

The authors would also like to thank Z. Bai and J. Demmel for sharing their insights concerning the nonsymmetric eigenvalue problem and their LAPACK software with us, E. Jessup for recommending that we perform only symmetric operations in the symmetric case, and C. Bischof for sharing his expertise on rank-revealing orthogonal factorizations with us. We are also particularly grateful to C. Bischof and Z. Bai for their suggestions on how to improve the original draft of this paper. Finally, we would like to thank G. W. Stewart for encouragement and instructive suggestions that have had a great impact on the direction of our investigations. We would also like to thank the referee who brought the paper of Pan and Schreiber to our attention.

References

[1] Anderson, E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, & D. Sorensen, *LAPACK: A portable linear algebra library for high-performance computers*, Proceedings, Supercomputing '90, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 2–11.

[2] Auslander, L. & A. Tsao, *On parallelizable eigensolvers*, Adv. Appl. Math. **13** (1992), 253–261.

[3] Bai, Z. & J. Demmel, *On a block implementation of Hessenberg multishift QR iteration*, International Journal of High Speed Computing **1** (1989), no. 1, 97–112, (also LAPACK Working Note #8).

[4] Bai, Z. & J. Demmel, *Design of parallel nonsymmetric eigenroutine toolbox, Part I*, Research report 92-09, University of Kentucky (Dec. 1992), (also PRISM Working Note #5).

[5] Bai, Z., J. Demmel, & A. McKenney, *On the conditioning the nonsymmetric eigenproblem: theory and software*, LAPACK Working Note 13, Courant Institute (1989).

[6] Beavers, A. N., Jr. & E. D. Denman, *A computational method for eigenvalues and eigenvectors of a matrix with real eigenvalues*, Num. Math. **21** (1973), 389-96.

[7] Berry, M. & A. Sameh, *Parallel algorithms for the singular value and dense symmetric eigenvalue problem*, CSRD (1988), no. 761.

[8] Bischof, C., *A parallel QR factorization with controlled local pivoting*, SIAM J. Sci. Stat. Comput. **12** (1991), no. 1, 36–57.

[9] Cuppen, J. J. M., *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math. **36** (1981), 177–95.

[10] Demmel, J. & K. Veselic, *Jacobi's method is more accurate than QR*, Department of Computer Science, New York University **468** (1989).

[11] Denman, E. D. & A. N. Beavers, Jr., *The matrix sign function and computations in systems*, Appl. Math. Comp. **2** (1976), 63-94.

[12] Denman, E. D. & J. Leyva-Ramos, *Spectral decomposition of a matrix using the generalized sign matrix*, Appl. Math. Comp. **8** (1981), 237-50.

[13] Dongarra, J., C. B. Moler, J. R. Bunch, & G. W. Stewart, *LINPACK User's Guide*, SIAM, Philadelphia, 1979.

[14] Dongarra, J. & D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput. **8** (1987), s139–54.

[15] Golub, G., V. Klema, & G. W. Stewart, *Rank degeneracy and least squares problems*, Technical Report TR-456, University of Maryland (1976).

[16] Golub, G. & C. F. Van Loan, *Matrix Computations, 2nd ed.*, Johns Hopkins University Press, Baltimore, 1989.

[17] Hamming, R. W., *Digital Filters, 2nd ed.*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[18] Hoffman, K. & R. Kunze, *Linear Algebra*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[19] Howland, J. L., *The sign matrix and the separation of matrix eigenvalues*, Lin. Alg. Appl. **49** (1983), 221-32.

[20] Huo, Y. & R. Schreiber, *Efficient, massively parallel eigenvalue computation*, The International J. of Supercomputer Applications **7** (Winter 1993), no. 4, 292-303.

[21] Ipsen, I. & E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Tech. Rep. RR-548, Yale University (1987).

[22] Ipsen, I. & E. Jessup, *Improving the accuracy of inverse iteration* (1991) (to appear).

[23] Kenney, C. & A. J. Laub, *Rational iterative methods for the matrix sign function*, SIAM J. Matrix Anal. Appl. **12** (1990), no. 2, 273-91.

[24] Li, T. Y., Z. Zeng, & L. Cong, *Solving eigenvalue problems of real nonsymmetric matrices with real homotopies*, Dept. of Mathematics, Michigan State University (1990), preprint.

[25] Li, T.-Y., H. Zhang, & X.-H. Sun, *Parallel homotopy algorithm for symmetric tridiagonal eigenvalue problems*, SIAM J. Sci. Stat. Comput. **12** (1991), no. 3, 469–87.

[26] Lin, C.-C. & E. Zmijewski, *A parallel algorithm for computing the eigenvalues of an unsymmetric matrix on a SIMD mesh of processors*, TRCS 91-15, Dept. of Computer Science, Univ. of Calif., Santa Barbara (1991).

[27] Pan, V. & R. Schreiber, *An improved Newton iteration for the generalized inverse of a matrix, with applications*, SIAM J. Sci. Stat. Comput. **12** (1991), no. 5, 1109–1129.

[28] Rutter, J., *A serial implementation of Cuppen's Divide and Conquer Algorithm for the symmetric eigenvalue problem*, UCB/CSD 94/799, University of California at Berkeley (1994), Berkeley, California.

[29] Schreiber, R., *Solving eigenvalue and singular value problems on an undersized systolic array*, SIAM J. Sci. Stat. Comput. **7** (1986), no. 2, 441–451.

[30] Shroff, G. & R. Schreiber, *On the convergence of the cyclic Jacobi method for parallel block orderings*, SIAM J. Sci. Stat. Comput. **10** (1989), no. 3, 326–346.

[31] Stewart, G. W., *A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix*, SIAM J. Sci. Stat. Comput. **6** (1985), no. 4, 853–864.

[32] Van de Geijn, R. A., *Deferred shifting schemes for parallel QR methods*, SIAM J. Matrix Anal. Appl. **14** (1993), no. 1, 180-194.