Parallel Performance of a Symmetric Eigensolver based on the Invariant Subspace Decomposition Approach

Christian Bischof[†] Steven Huss-Lederman[‡] Xiaobai Sun[†]
Anna Tsao[‡] Thomas Turnbull[‡]

[†]Math. & Comp. Sci. Div., Argonne National Laboratory, Argonne, IL 60439
[‡]Supercomputing Research Center, Bowie, MD 20715

Abstract

In this paper, we discuss work in progress on a complete eigensolver based on the Invariant Subspace Decomposition Algorithm for dense symmetric matrices (SYISDA). We describe a recently developed acceleration technique that substantially reduces the overall work required by this algorithm and review the algorithmic highlights of a distributed-memory implementation of this approach. These include a fast matrix-matrix multiplication algorithm, a new approach to parallel band reduction and tridiagonalization, and a harness for coordinating the divide-and-conquer parallelism in the problem. We present performance results for the dominant kernel, dense matrix multiplication, as well as for the overall SYISDA implementation on the Intel Touchstone Delta and the Intel Paragon.

1. Introduction

Computation of eigenvalues and eigenvectors is an essential kernel in many applications, and several promising parallel algorithms have been investigated [26, 3, 28, 22, 25, 6]. The work presented in this paper is part of the PRISM (Parallel Research on Invariant Subspace Methods) Project, which involves researchers from Argonne National Laboratory, the Supercomputing Research Center, the University of California at Berkeley, and the University of Kentucky. The goal of the PRISM project is the development of algorithms and software for solving large-scale eigenvalue problems based on the invariant subspace decomposition approach (ISDA) suggested by Auslander and Tsao [1].

The symmetric invariant subspace decomposition algorithm (SYISDA) for an $n \times n$ symmetric matrix A proceeds as follows.

Scaling: Compute upper and lower bounds on the spectrum $\lambda(A)$ of A and compute α and β such that for $B = \alpha A + \beta I$ we have $\lambda(B) \subseteq [0,1]$, with the mean eigenvalue of A being mapped to $\frac{1}{2}$.

Eigenvalue Smoothing: Let $p_i(x)$, $i = 1, 2, \ldots$ be polynomials such that $\lim_{i\to\infty} p_i([0,1]) = \{0,1\}$, that is, in the limit all values are mapped to either 0 or 1. Iterate

$$C_0 = B, C_{i+1} = p_i(C_i), i = 0, 1, \ldots,$$

until $||C_{i+1} - C_i||$ is numerically negligible (in iteration k, say), at which point all the eigenvalues of C_k are near either 0 or 1.

Invariant Subspace Computation: Find an orthogonal matrix $[U, V]$ such that the columns of U and V form orthonormal bases for the range space of C_k and its complementary orthogonal subspace, respectively. That is, $U^T U = I$, $V^T V = I$, $U^T V = 0$, and the range of $C_k U$ is U.

Decoupling: Update the original A with $[U, V]$, i.e., form

$$[U,V]^T A\, [U,V] = \begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix}.$$

Since the invariant subspaces of any matrix polynomial of a symmetric matrix A are also invariant subspaces of A, the columns of U and V span complementary invariant subspaces of A, and hence their application to A decouples the spectrum of A. The subproblems A_1 and A_2 can now be solved independently and the algorithm applied further recursively, with the number of subproblems doubling at every stage. If eigenvectors are desired as well, we also update the current eigenvector matrix. Orthogonality of the eigenvectors is guaranteed due to the exclusive use of orthogonal transformations.

We have considerable freedom in implementing SYISDA, particularly with respect to choosing the

*This paper is PRISM Working Note #15, available via anonymous ftp to `ftp.super.org` in the directory pub/prism.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX polynomials p_i and the method for computing the invariant subspaces. As in [27, 25, 6], we use predominantly the first incomplete Beta function $\mathfrak{B}_1(x) = 3x^2 - 2x^3$ in our implementation. In addition, in Section 2, we describe an acceleration technique we have recently developed that substantially reduces the number of iterations required in the **Eigenvalue Smoothing** step. We note that other functional iterations, such as approximation methods for the matrix sign function [14, 15, 21, 2], can be used in the **Eigenvalue Smoothing** step as well.

The two key primitives of the algorithm are matrix-matrix multiplication, which accounts for the majority of the computation, and computation of the range and null space of a matrix having eigenvalues clustered around 0 and 1. The sequential complexity of SYISDA, when applied to dense matrices, is considerably greater than that of other algorithms. Nonetheless, the algorithm is promising from both a scalability and a numerical point of view [27, 6]. First, since most of the computation is in matrix multiplication, high efficiencies and near optimal speedups are achieved on large problems. Second, since the algorithm performs only orthogonal transformations, orthogonality in the computed eigenvectors is guaranteed.

In the next section, we briefly describe the parallel algorithm. In Section 3, we give recent performance results for our implementation of SYISDA on the Intel Touchstone Delta and the Intel Paragon. Section 4 concludes with some brief comments about the algorithm presented.

2. The parallel algorithm

The parallel implementation incorporates several algorithmic improvements developed since our initial investigations on serial machines. The first improvement is our method of computing the subspaces in the **Invariant Subspace Computation** step. To find the orthogonal transformation that decouples A, we have to find the range and null space of a matrix that has only two distinct eigenvalues, 0 and 1. As it turns out, the **Invariant Subspace Computation** step can be achieved essentially via a tridiagonalization of C_k, using a methodology we shall refer to as rank-revealing tridiagonalization (RRTRID).

In order to reduce the given matrix to tridiagonal form, we employ a variant of the successive band reduction (SBR) framework suggested by Bischof and Sun in [9], which eliminates subdiagonals of C_k in a piecemeal fashion as illustrated in Figure 1. In comparison, conventional Householder tridiagonalization approaches [19] or block variants thereof [16] eliminate all subdiagonals at one time. This traditional approach also underlies the parallel implementations described in [20] and [17]. The SBR variant used in our implementation is discussed in detail in [7]; performance results are given in [6] and in another paper at this conference [10]. An exciting new development discussed in [10] is the ability to fully block all stages of the succession of band reductions.

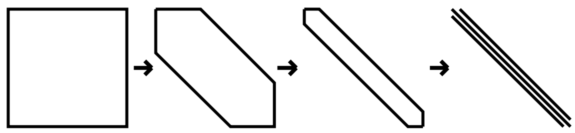

FIGURE 1. Reduction to tridiagonal form by a sequence of band reductions

The key observation in the context of SYISDA is that, under some very general conditions, a banded matrix having only two distinct eigenvalues and bandwidth $n/2^j$ must be block diagonal, with each block being of size at most $n/2^{j-1}$. In particular, a tridiagonal matrix with such a spectrum is block diagonal with blocks of size at most 2×2. Hence, after the matrix has been reduced to tridiagonal form, one only needs to solve some (completely independent) 2×2 eigenvalue problems to obtain the desired invariant subspaces. We are able to skip large numbers of the orthogonal transformations, since the block diagonality of the matrices generated results in many transformations that would act on columns that are already negligible and hence need not be performed. This property has been demonstrated in extensive testing [6, 11], particularly of the sequential algorithm. Algorithmic details of RRTRID, as well as some of the subtle numerical issues are discussed in [8].

It is important to realize that, unlike other approaches for computing so-called rank-revealing factorizations [4, 5, 12, 30], tridiagonalization does not involve any data-dependent pivoting strategies. In particular, in the parallel setting, the predictability of data flow greatly contributes to simplicity of implementation as well as to the ability to overlap communication and computation.

We have developed a sequential code for performing both SBR and RRTRID that incorporates some of the blocking schemes discussed in our previous papers, with promising results [6, 11]. However, the parallel implementation of RRTRID currently only performs unblocked operations. Nonetheless, the skipping alluded to previously still leads to good performance.

A second algorithmic change was necessitated by the use of the rank-revealing tridiagonalization scheme of Bischof and Sun. The rank-revealing tridiagonalization scheme requires that the rank-deficient matrix, C_k, have only two distinct eigenvalues. As in [27], we began by using first incomplete Beta function, $\mathfrak{B}_1(x) = 3x^2 - 2x^3$, exclusively in the **Eigenvalue Smoothing** step. However, because $\mathfrak{B}_1$ has a fixed point at 1/2, C_k occasionally has eigenvalues at 1/2. Fortunately, we have discovered an inexpensive means of detecting and addressing this situation. Suppose that C_k has eigenvalues 0, 1/2, and 1 of multiplicities m_1, m_2, and m_3, respectively. It is easily shown that $m_2 = 4(\text{trace}(C_k) - \text{trace}(C_k^2))$. Both trace($C_k$) and $\text{trace}(C_k^2) = \|C_k\|_F^2$ are inexpensive computations. Here, $\|\cdot\|_F$ denotes the Froebenius norm. If $m_2 > 0$, we then apply either $3x - 2x^2$ or $2x^2 - x$ to C_k. Both these quadratic functions map 0 and 1 to themselves; the pertinent difference between them is that one maps 1/2 to 1 and the other maps 1/2 to 0. The quadratic used is chosen to make the divide as even as possible. In practice, after the application of this quadratic, the clusters of eigenvalues at 0 and 1 require "tightening up." We accomplish this by a few further applications of the p_i.

The final algorithmic feature found in our current parallel implementation is an acceleration technique that significantly reduces the number of iterations of $\mathfrak{B}_1$ required in the early divides. Since most of the work in SYISDA occurs in the early divides, 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 [19]. Since these bounds are generally quite poor, B from the **Scaling** step 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 of the incomplete Beta function iteration is very slow for values near 1/2, we sought strategies to improve the rate of convergence for matrices having eigenvalues near 1/2.

$\mathfrak{B}_1$ 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 [29]. They essentially observed that if we take the matrix B and "stretch" it so that its eigenvalues now lie over some interval, say $[-\alpha, 1+\alpha]$, where $0 < \alpha \leq \rho - 1$, then the eigenvalues of B near 1/2 are moved away from 1/2 and $\mathfrak{B}_1$ will still map the eigenvalues of B into [0, 1]. By "stretching", we mean to apply a linear function that maps 0 and 1 to $1-\alpha$ and α, 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, we must be careful not to "stretch" too aggressively. We have found that applying this strategy six or seven times with $\alpha = .3$ leads to about a 1/3 reduction in the number of iterations required in the early stages of the algorithm. Figure 2 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.

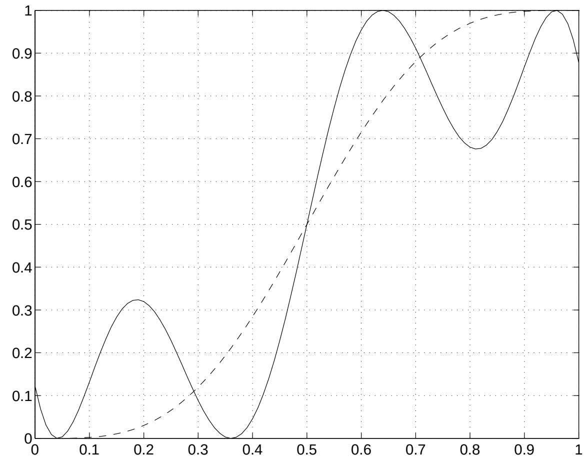

FIGURE 2. Behavior of Acceleration Technique

In the latter stages of SYISDA, 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 SYISDA 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 original incomplete Beta function iteration.

The orchestration of multiple subproblems cur-

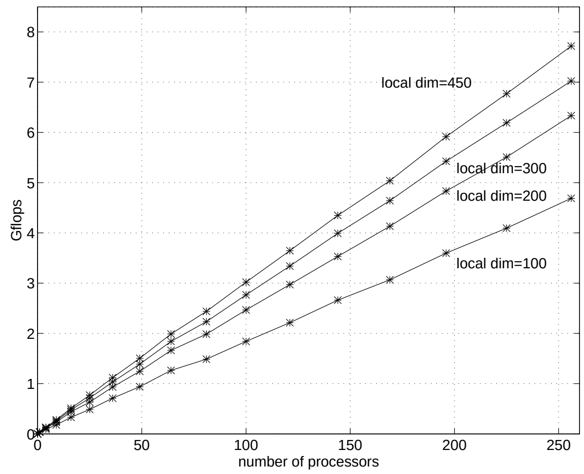

FIGURE 3. Total performance of BiMMeR on Delta

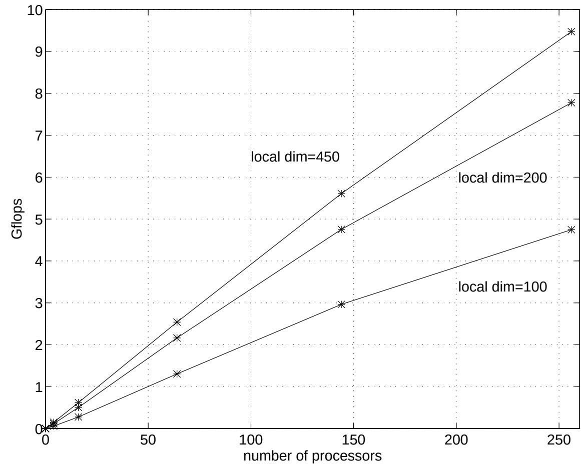

FIGURE 4. Total performance of BiMMeR on Paragon

rently consists of two separate stages [25, 6]. The first stage encompasses the early divides, where large subproblems are solved sequentially and the scalability of the dense matrix multiplication and rank-revealing tridiagonalization lead to high efficiencies. As the subproblem size decreases, there is a point at which communications overhead makes it impractical to solve that subproblem over the entire mesh. The amount of work required to find the eigensolution of the remaining subproblem is very small, but the cost of the update of the eigenvector matrix Z, whose leading dimension is still the size of the original matrix A, is still substantial. The second stage, or End Game, handles the small subproblems remaining, exploiting parallelism at two levels: first, by solving individual subproblems in parallel on single nodes and second, by performing distributed updates of the accumulated eigenvector matrix.

Through the use of two-dimensional torus wrap, no data redistribution is required between divides. As the subproblem size decreases, the proportion of the total time required for communication during individual matrix multiplications increases and the granularity of local computation decreases. However, this approach guarantees near-perfect load balancing in the costly early stages. A discussion of the rationale used to arrive at this scheme is presented in [25]. When the subproblems become small (dimension of a few hundred), they are currently solved using DSYEV (QR algorithm) from LAPACK.

Since the computational cost for SYISDA is dominated by dense matrix-matrix multiplication, its performance depends heavily on the matrix multiplication code. We have developed a double precision distributed matrix multiplication code, BiMMeR, that incorporates features needed by SYISDA. As usual, a critical issue in achieving high efficiency is data locality, i.e., maximum reuse of data in floating point computations. To this end, we designed BiMMeR to exploit the availability of highly optimized assembly-coded implementations of matrix multiplication on single nodes and to ensure maximal granularity in local computations. The Broadcast-Multiply-Roll (BMR) algorithm [18] has been demonstrated to scale extremely well on loosely coupled square processor meshes and uses two readily portable communication kernels: one-dimensional broadcast and roll. Since many distributed-memory machines do not currently overlap communication and computation effectively, BiMMeR uses a variant of BMR that is highly synchronous [24, 23].

In Figures 3 and 4, we give the total performance for BiMMeR on square partitions of the Intel Delta for matrices having size 100×100, 200×200, 300×300, and 450×450 on each processor and some preliminary results on the Intel Paragon for matrices having size 100×100, 200×200, and 450×450 on each processor. Our code has also achieved a parallel efficiency of 86%, with overall peak performance in excess of 8 Gflops on 256 nodes of the Delta for an 8800×8800 double precision matrix [24]. Note that BiMMeR has achieved a parallel efficiency of about 80%, with peak performance thus far of about 9.5 Gflops on 256 nodes of the Paragon for a 7200×7200 double precision matrix. We expect performance on the Paragon to improve as the machine matures.

3. Parallel performance of eigensolver

We are currently studying both the numerical performance and scaling properties of our algorithm. Since our code currently uses only the unblocked versions of SBR, we expect our performance to continue to improve. Preliminary results indicate that SYISDA provides excellent scaling and accuracy [6]. Our code can currently accommodate matrices of local dimension up to about 450. The terminology *local dimension* refers to the size of the matrix block on each processor.

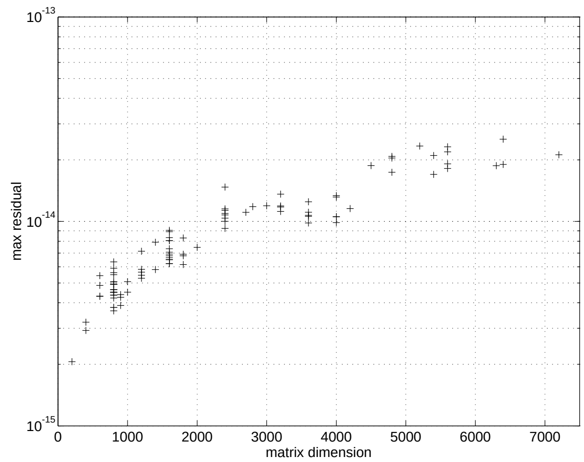

FIGURE 5. Eigenvalue Accuracy on Delta

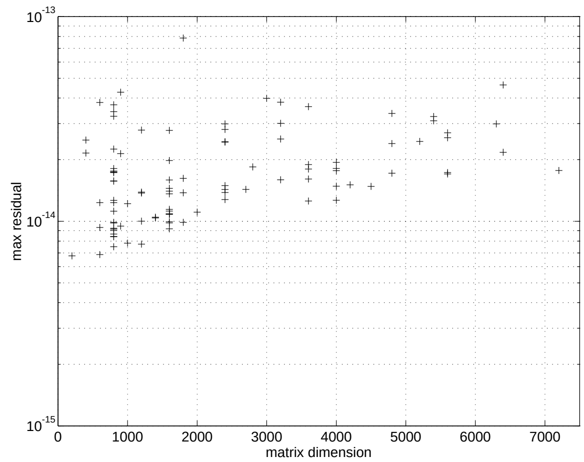

FIGURE 6. Eigenvector Orthogonality on Delta

Easily generated symmetric test cases with uniformly distributed eigenvalues of dimensions 100–7200

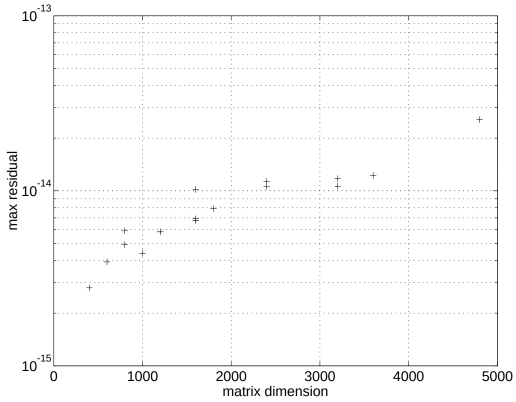

FIGURE 7. Eigenvalue Accuracy on Paragon

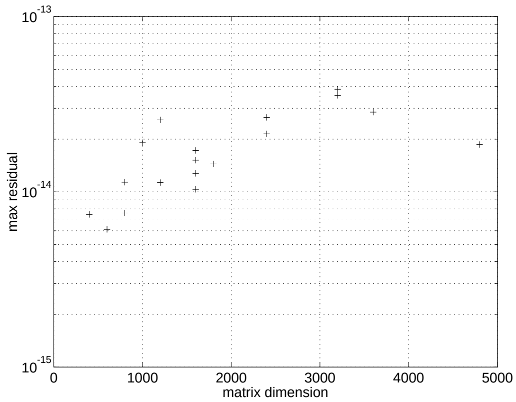

FIGURE 8. Eigenvector Orthogonality on Paragon

have been tested so far. For a given matrix dimension, one of two matrices was generated and solved on different size partitions. We plan to expand our test suite substantially in the future.

Preliminary results on the Paragon were obtained under Release 1.1.3 of the operating system and by running our code compiled with the "-g" option and default parameters. Due to a variety of complications, the Paragon data is incomplete. Nonetheless, our results are quite promising and we expect improvements in performance under future releases of the operating system.

Figures 5–8 show the residuals for eigenvalue accuracy and orthogonality on the Delta and Paragon. Accuracy in the residuals for a given matrix A is measured by computing the maximum normalized ∞-norm residual

$$\max_i \frac{||AZ_i - \lambda_i Z_i||_\infty}{||A||_\infty}$$

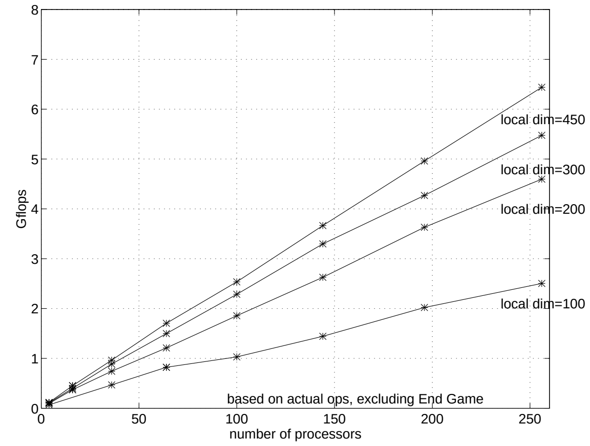

FIGURE 9. Total performance of SYISDA on Delta

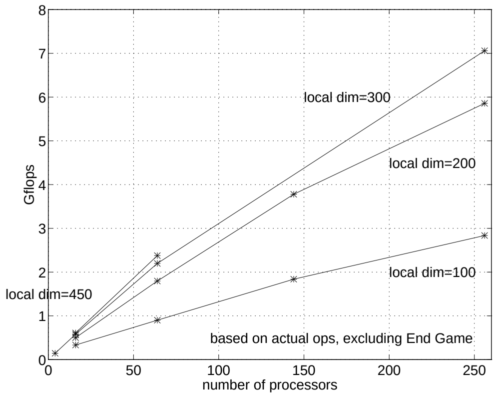

FIGURE 10. Total performance of SYISDA on Paragon

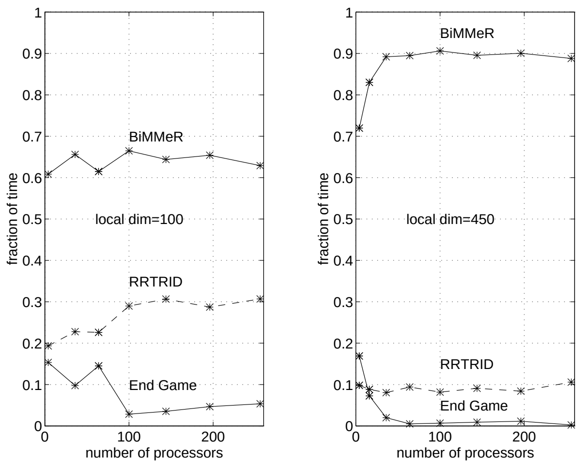

FIGURE 11. Time Breakdown on Delta

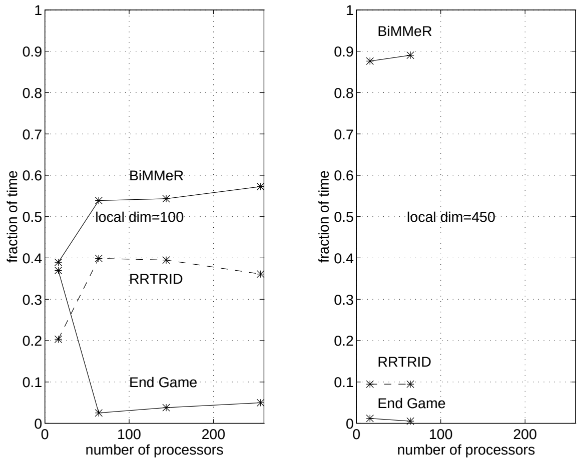

FIGURE 12. Time Breakdown on Paragon

where Z_i is the computed eigenvector corresponding to the eigenvalue λ_i. We also measured 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.

	time (min.)			
n	$p=4$	$p=8$	$p=12$	$p=16$
800	2.5	1.1	.9	.8
1600	17.4	6.3	4.1	3.1
2400	–	17.9	10.5	7.5
3200	–	38.5	21.6	14.8
4000	–	–	38.8	25.6
4800	–	–	63.0	41.1
5600	–	–	–	61.7
6400	–	–	–	88.3
7200	–	–	–	122.1

TABLE 1. Time for complete eigensolution of $n \times n$ matrix on $p \times p$ partition of Delta

	time (min.)			
n	$p=4$	$p=8$	$p=12$	$p=16$
800	1.7	1.0	–	–
1600	12.3	4.3	3.1	2.7
2400	–	12.2	7.3	–
3200	–	–	15.4	11.6
4800	–	–	–	31.9

TABLE 2. Time for complete eigensolution of $n \times n$ matrix on $p \times p$ partition of Paragon

Figures 9 and 10 show the total performance of SYISDA on square partitions of the Delta and Paragon for test matrices having various local dimensions on each processor. It is clear that SYISDA scales extremely well. We see from Figure 3 that on the Delta, BiMMeR runs at about 7.8 Gflops on a 16×16 partition for 7200×7200 matrices; SYISDA achieves 6.4 Gflops, about 82% of what we believe to be the maximum possible performance for this algorithm.

Matrix multiplication and RRTRID comprise the vast majority of the computational time required by SYISDA. Figures 11 and 12 show the fraction of the SYISDA computation time spent in each of these primitives on both the Delta and Paragon for matrices of local dimensions 100 and 450. We see that RRTRID is a significant portion of the calculation for small matrices, whereas the percentage of time spent in matrix multiplication approaches 90% for large matrices. This is particularly desirable because of the extreme scalability of matrix multiplication. We also see that the End Game is very inexpensive.

Tables 1 and 2 give the solution time in minutes for test matrices of various sizes on $p \times p$ partitions of the Delta and Paragon, where $p = 4, 8, 12, 16$.

4. Conclusions

We have discussed algorithmic improvements to the SYISDA and given performance results on a variety of machines. We have seen that in addition to providing good numerical performance, high efficiencies and good scaling are achieved, due to the preponderance of matrix multiplication in the algorithm.

In the future, we shall give performance results for SYISDA using a blocked implementation of SBR on a variety of different machines, as well as numerical results for a wide variety of test matrices. We also plan to compare the performance of our code to that of other parallel codes, as available.

In addition to continuing our studies of SYISDA and the data layout issues associated with the kernels we are interested in, we are also investigating variants of SYISDA aimed at reducing the amount of time spent doing dense matrix multiplication. One potential method of achieving this is to use a Strassen-type algorithm [31, 13] to perform the dense matrix multiplications. Another particularly promising algorithm, banded SYISDA [11], uses SBR to reduce A to a narrow band and then periodically reduces matrices in the **Eigenvalue Smoothing** step to a narrow band. Multiplication of two matrices of bandwidths b only requires $O(b^2 n)$ work versus $O(n^3)$ for two dense matrices. Furthermore, the special properties of the iterates in the **Eigenvalue Smoothing** step result in surprisingly slow band growth in the iterated matrices. In our sequential implementation, using specialized routines for multiplying symmetric band matrices [32], the run times for banded SYISDA are competitive with the symmetric QR algorithm for large problems. In fact, the time spent in SBR becomes the dominant time. Banded SYISDA uses essentially the same harness as SYISDA and the two computational kernels of SBR and banded matrix multiplication. We intend to modify our current code to obtain an implementation of banded SYISDA and compare the two algorithms.

5. Acknowledgements

This work was partially supported by the Applied and Computational Mathematics Program, Advanced Research Projects Agency, under Contract DM28E04120, and by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38. Access to the Intel Touchstone Delta and Paragon systems operated by Caltech on behalf of the Concurrent Supercomputing Consortium was provided by NSF.

The authors would particularly like to thank Heidi Lorenz-Wirzba (CCSF) for all of her assistance during our testing runs on both the Delta and Paragon(s) at Caltech and Elaine Jacobson (SRC) for many helpful discussions during the course of this work.

References

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

[2] 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).

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

[4] Bischof, C., *A parallel QR factorization with controlled local pivoting*, SIAM J. Sci. Stat. Comput. **12** (1991), 36–57.

[5] Bischof, C. H. & P. C. Hansen, *A block algorithm for computing rank-revealing QR factorizations*, also MCS-P251-0791, Numerical Algorithms **2** (1992), no. 3-4, 371–392.

[6] Bischof, C. H., S. Huss-Lederman, X. Sun, & A. Tsao, *The PRISM Project: Infrastructure and Algorithms for Parallel Eigensolvers*, Proceedings, Scalable Parallel Libraries Conference (Starksville, MS, Oct. 6–8, 1993), IEEE, 1993, (also PRISM Working Note #12).

[7] Bischof, C., M. Marques, & X. Sun, *Parallel bandreduction and tridiagonalization*, Proceedings, Sixth SIAM Conference on Parallel Processing for Scientific

Computing (Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadelphia, 1993, (also PRISM Working Note #8).

[8] Bischof, C. & X. Sun, *A divide-and-conquer method for tridiagonalizing symmetric matrices with repeated eigenvalues*, Preprint MCS-P286-0192, Argonne National Laboratory (1992), (also PRISM Working Note #1).

[9] Bischof, C. & X. Sun, *A framework for symmetric band reduction and tridiagonalization*, Preprint MCS-P298-0392, Argonne National Laboratory (1992), (also PRISM Working Note #3).

[10] Bischof, C. H. & X. Sun, *Parallel tridiagonalization through two-step band reduction*, Proceedings: Scalable High Performance Computing Conference '94, Knoxville, Tennessee, May 1994, IEEE Computer Society Press, 1994, (also PRISM Working Note #17) (to appear).

[11] Bischof, C., X. Sun, A. Tsao, & T. Turnbull, *A study of the Invariant Subspace Decomposition Algorithm for banded symmetric matrices*, Proceedings: Fifth SIAM Conference on Applied Linear Algebra, Snowbird, UT, June, 1994, SIAM, 1994, (also PRISM Working Note #16) (to appear).

[12] Chandrasekaran, S. & I. Ipsen, *On rank-revealing QR factorizations*, Technical Report YALEU/DCS/RR-880, Yale University, Department of Computer Science, 1991.

[13] Coppersmith, D. and S. Winograd, *Matrix multiplication via arithmetic progressions*, Proceeding of the Nineteenth Annual ACM Symposium on Theory of Computing, 1987, pp. 1-6.

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

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

[16] Dongarra, Jack J., Sven J. Hammarling and Danny C. Sorensen, *Block reduction of matrices to condensed form for eigenvalue computations*, MCS–TM–99, Argonne National Laboratory (September 1987).

[17] Dongarra, J. & R. van de Geijn, *Reduction to condensed form for the eigenvalue problem on distributed-memory architectures*, Technical Report ORNL/TM-12006, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division (January 1992).

[18] Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, & D. Walker, *Solving Problems on Concurrent Processors, Vol. I*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

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

[20] Hendrikson, B. & D. Womble, *The torus-wrap mapping for dense matrix calculations on massively parallel computers*, SAND92-0792, Sandia National Laboratories (1992).

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

[22] Huo, Y. & R. Schreiber, *Efficient, massively parallel eigenvalue computation*, RIACS Technical Report 93.02, Research Institute for Advanced Computer Science (1993).

[23] Huss-Lederman, S., E. M. Jacobson, & A. Tsao, *Comparison of Scalable Parallel Matrix Multiply Libraries*, Proceedings, Scalable Parallel Libraries Conference (Starksville, MS, Oct. 6–8, 1993), IEEE, 1993, (also PRISM Working Note #13, also appears as Technical Report SRC-TR-93-108, Supercomputing Research Center, 1993).

[24] Huss-Lederman, S., E. M. Jacobson, A. Tsao, & G. Zhang, *Matrix Multiplication on the Intel Touchstone Delta*, Technical Report SRC-TR-93-101, Supercomputing Research Center (1993), (also PRISM Working Note #14).

[25] Huss-Lederman, S., A. Tsao, & G. Zhang, *A Parallel Implementation of the Invariant Subspace Decomposition Algorithm for Dense Symmetric Matrices*, Proceedings, Sixth SIAM Conference on Parallel Processing for Scientific Computing (Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadelphia, 1993, (also PRISM Working Note #9).

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

[27] Huss-Lederman, S., A. Tsao, & T. Turnbull, *A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues*, Technical Report TR-91-042, Supercomputing Research Center (1991).

[28] 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.

[29] 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.

[30] G. W. Stewart, *Updating a rank-revealing ULV decomposition*, SIAM J. Matrix Anal. Appl. **14** (1993), no. 2.

[31] Strassen, V., *Gaussian elimination is not optimal*, Numer. Math. **13** (1969), 354-6.

[32] Tsao, A. & T. Turnbull, *A comparison of algorithms for banded matrix multiplication*, Technical Report SRC-TR-093-092, Supercomputing Research Center (1993), (also PRISM Working Note #6).