The PRISM Project: Infrastructure and Algorithms for Parallel Eigensolvers*

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

[†]Argonne National Laboratory, Argonne, IL 60439.
[‡]Supercomputing Research Center, Bowie, MD 20715.

Abstract

The goal of the PRISM project is the development of infrastructure and algorithms for the parallel solution of eigenvalue problems. We are currently investigating a complete eigensolver based on the Invariant Subspace Decomposition Algorithm for dense symmetric matrices (SYISDA). After briefly reviewing SYISDA, we discuss 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 also present performance results of these kernels as well as the overall SYISDA implementation on the Intel Touchstone Delta prototype.

1. Introduction

Computation of eigenvalues and eigenvectors is an essential kernel in many applications, and several promising parallel algorithms have been investigated [29, 24, 3, 27, 21]. 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 originally suggested by Auslander and Tsao [1].

The algorithm described here is the Symmetric Invariant Subspace Decomposition Algorithm (SYISDA) for an $n \times n$ symmetric matrix A, which 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).

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 $[U, V]$; that is, 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,

*This paper is PRISM Working Note #12, available via anonymous ftp to `ftp.super.org` in the directory pub/prism.

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 System operated by Caltech on behalf of the Concurrent Supercomputing Consortium was provided by NSF.

This paper will appear in the Proceedings of the Scalable Parallel Libraries Conference, 6–8 October, 1993.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

with the number of subproblems doubling at every step. If eigenvectors are desired as well, we also update the current eigenvector matrix. Orthogonality of the eigenvectors is guaranteed because of the exclusive use of orthogonal transformations.

For ease of reference, we henceforth call the matrices produced in the **Eigenvalue Smoothing** step, having only the two distinct eigenvalues 0 and 1, PRISM matrices. Note that we have considerable freedom in implementing SYISDA, in particular with respect to choosing the polynomials p_i as well as the method for computing the invariant subspaces. We also mention that any other method that produces invariant subspaces, such as approximation methods for the matrix sign function [12, 13, 20, 2], could be used in the **Eigenvalue Smoothing** step as well. As in [26], we use predominantly the first incomplete beta function $3x^2 - 2x^3$ in our implementation. The experiments in [26] also confirm the numerical robustness of SYISDA.

While the SYISDA algorithm can be used to compute a full eigendecomposition, it is worthwhile to point out certain mathematical features that distinguish it from other approaches:

Ordering of Eigenvalues: Assuming that p maps all eigenvalues in $[0, a)$ to 0 and all eigenvalues in $[a, 1]$ to 1, $\lambda(A_1)$ contains all the eigenvalues of A that are smaller than $\frac{a-\beta}{\alpha}$, and $\lambda(A_2)$ contains the rest. Hence, if one is interested only in eigenvalues in a certain part of the spectrum, one need not further resolve diagonal blocks corresponding to uninteresting parts of the spectrum.

Subspaces before Eigenvalues: SYISDA is primarily an algorithm that computes and refines invariant subspaces. If such a subspace has become one-dimensional, an eigenvalue/eigenvector pair has been found. However, if one is only interested in finding an orthogonal basis for the subspace spanned by a certain set of eigenvectors, there is no need to expend the effort to compute all eigenvalues, and one can terminate the divide-and-conquer procedure when the subspace corresponding to the desired eigenvalue range has been identified.

No Problems with Repeated Eigenvalues: Clusters of eigenvalues are quickly gathered in the same subproblem and are, in fact, likely to increase the speed of convergence of the **Eigenvalue Smoothing** step. The orthogonality of eigenvectors is not affected at all by repeated eigenvalues.

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 zero and one. 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. First, since most of the computation is in matrix multiplication, high efficiencies and near-optimal speedups can be expected on large problems. Second, since the algorithm performs only orthogonal transformations, orthogonality in the computed eigenvectors is guaranteed.

The paper is organized as follows. The next section briefly discusses the implementation of matrix-matrix multiplication. Section 3 discusses the rank-revealing tridiagonalization algorithm employed for the **Invariant Subspace Computation** step. It is based on the successive band reduction (SBR) framework developed by Bischof and Sun [9] and, while completely general, derives significant benefit from the special eigenvalue structure of the matrices at hand [8]. Section 4 discusses the overall divide-and-conquer strategy employed to orchestrate the various subproblems and presents preliminary performance results of the first SYISDA implementation on the Intel Touchstone Delta. We conclude with our findings, in particular with respect to the effect of data layout on the design of scalable libraries for the support of matrix computations, and describe how the kernels we have developed form the infrastructure for a "library" of parallel eigensolvers.

2. Matrix Multiplication

As noted previously, the computational cost for SYISDA is dominated by dense matrix multiplication. Hence, its performance depends heavily on having a scalable matrix multiplication code for our initial target machine, the Intel Touchstone Delta. We have developed a distributed matrix multiplication code that calculates the products $C = \alpha AB + \beta C$ and $C = \alpha A^t B + \beta C$ in double precision. Our objectives were to

(1) provide a highly efficient algorithm suitable for use by SYISDA,
(2) strive for high performance on large square matrices,
(3) provide robust performance for mesh configurations with poor aspect ratios, and
(4) use an algorithm whose kernels match well with the expected capabilities of future machines.

As usual, a critical issue in achieving high efficiency is data locality, that is, maximum reuse of data in floating point computations. To this end, we utilize the highly optimized assembly-coded double precision general matrix multiplication, DGEMM, on single nodes. It is capable of sustaining 36.5 Mflops (empirically determined using Release 1.4 of Intel NX/M OS and related software) of the 40 Mflops possible. On the Delta, optimal single-node performance for DGEMM often decreases significantly as the matrix shapes become less square or when matrix granularities become finer. Thus, our first general rule is to strive for matrices having all their dimensions as large as possible in local computations.

Let us first consider the case of square meshes. In the parallel SBR strategy, it is extremely desirable from both a performance and ease of programming point of view to have row and column blocks physically spread across processor columns in a torus wrap fashion (see [6], for example). Furthermore, the divide-and-conquer strategy we use assumes that the blocks of the matrix are spread out fairly evenly across the mesh in such a way that the generated subproblems remain spread out all over the mesh. Therefore, two-dimensional torus wrap is ideal. For the purposes of matrix multiplication alone, optimal performance occurs when each node has the same amount of data for each matrix; the use of torus wrapping is neither an advantage nor a disadvantage. The situation changes, however, when we try to generalize two-dimensional torus wrap to nonsquare meshes. We found that the choice of data layout has a significant effect on both the ease of programming and granularity of local computations. We chose to use a generalization of two-dimensional torus wrap on nonsquare meshes, known as virtual two-dimensional torus wrap, that resulted in simpler, more easily tuned algorithms. Some of our preliminary findings are discussed in [6] and [22].

The Broadcast-Multiply-Roll (BMR) algorithm [16, 11, 28] 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. We have implemented a variant of BMR in C using communication primitives highly suited to the Delta [22]. In particular, since the Delta does not effectively overlap communication and computation, the algorithm we chose is highly synchronous.

Our BMR variant is able to deal with arbitrary rectangular meshes and matrix dimensions. Our code can deal with a variety of different virtual two-dimensional torus blocking schemes by means of a user-passed function providing block size information. The implementation guarantees that all matrix operands end up in place. Details of our implementation are discussed in

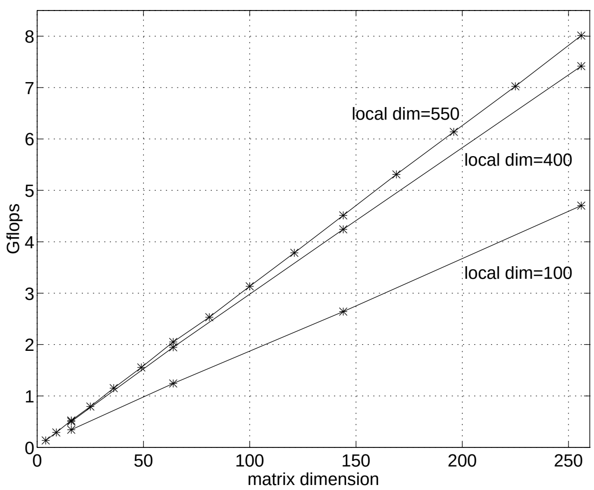


FIGURE 1. Total performance of matrix multiplication on the Delta

[22].

In Figure 1, we give the total performance for our distributed matrix multiplication on square submeshes of the Delta for matrices having matrix size 100×100, 400×400, and 550×550 on each processor. In particular, our code has achieved a parallel efficiency of 86%, with overall peak performance in excess of 8 Gflops on 256 nodes for an 8800×8800 double-precision matrix and has demonstrated robust performance for non-optimal mesh aspect ratios.

While our orientation was machine-specific and aimed at optimization rather than portability, our BMR variant will port well to other distributed architectures such as the IBM SP1.

3. Successive Band Reduction

To find the orthogonal transformation that decouples A, we have to find the range and null space of a PRISM matrix. As it turns out, the **Invariant Subspace Computation** step can be achieved essentially by a tridiagonalization of C_k. The key observation is that, under some very general conditions, a band 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. These issues, as well as some of the subtle numerical issues arising in this context, are discussed in [8].

It is important to realize that, unlike other approaches for computing so-called rank-revealing factorizations [4, 5, 10, 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.

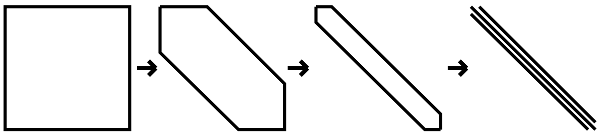


FIGURE 2. Reduction to tridiagonal form by a sequence of band reductions.

To reduce the given matrix to tridiagonal form, we employ a variant of the successive band reduction framework suggested by Bischof and Sun [9], which eliminates subdiagonals of C_k in a piecemeal fashion as illustrated in Figure 2. In comparison, conventional Householder tridiagonalization approaches [17] or block variants thereof [14] eliminate all subdiagonals at one time. This traditional approach also underlies the parallel implementations described in [19] and [15]. The SBR variant used in our implementation is discussed in detail in [7].

In our particular version of SBR, we first reduce C_k to a banded matrix of bandwidth nb, using block orthogonal transformations with blocksize nb and then, in a second step, reduce the band matrix of bandwidth nb to tridiagonal form. The first step maximally exploits block transformations, and the hope is that the little work that is left to be done in the second step (from a flop count point of view) does not add much to the overall complexity. In the context of SYISDA, we also expect to be able to skip large numbers of the orthogonal transformations, since the block diagonality of the matrices we will generate should result in many transformations that would act on columns that are already negligible and hence need not be performed.

One issue that is critical is the need to repackage the banded matrix remaining after the first initial reduction step into a more compact form that allows us to access adjacent entries of the band efficiently. For example, in a serial implementation, entries $(i + nb, i)$ and $(i + 1 + nb, i + 1)$ are at least $n - 1$ storage locations apart if an $n \times n$ matrix has been stored as the usual two-dimensional array. Hence, the final band reduction scheme would exhibit no data locality and would, as a result, suffer severe performance penalties on cache-based architectures. Even worse performance penalties would result in the parallel setting, as very little data are left for every process to work on, but a lot of communication is needed to access successive matrix elements and perform bulge chasing. To implement the second band reduction step, we redistribute the remaining band into packed storage and then employ a variant of an algorithm suggested by Lang [25] to reduce the band matrix to tridiagonal form. We made several modifications to Lang's algorithm to improve the memory locality of the algorithm, and details will be reported in a forthcoming paper.

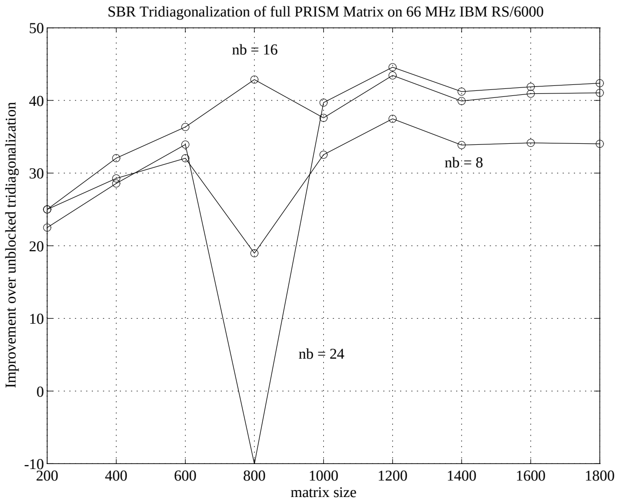
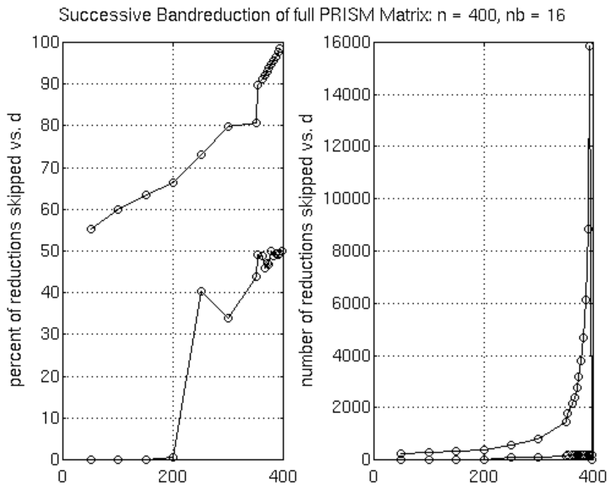


FIGURE 3. Improvement over standard tridiagonalization approach through SBR for PRISM matrices on IBM RS/6000.

Our SBR approach significantly outperforms the usual tridiagonalization approach on SYISDA matrices. For example, as is shown in Figure 3, running on a 66-MHz IBM RS/6000 with 128 MBytes of memory and using assembler-coded BLAS, our SBR approach runs a good 30% faster than the standard tridiagonalization procedure. Note that these times reflect both the time for the reduction from full to tridiagonal form and the accumulation of the orthgonal transformation matrices. If we ignore the usual timing variations one is bound to expect on a nondedicated system, the blocksize chosen does not seem to have a significant effect on the overall performance. The anamolous behavior exhibited for matrices of size 800 is currently under investigation.

As hinted earlier, we profit from the underlying structure of the matrices arising in SYISDA. Figure



top: 2nd band reduction   bottom: 1st band reduction

FIGURE 4. Work savings as a result of exploitation of eigenvalue structure via SBR

4 shows how many reductions we can skip in the first and second band reduction steps, where we first eliminate d subdiagonals ($d \geq nb = 16$) of a 400×400 dense PRISM matrix and then reduce the resulting matrix to tridiagonal form. While a standard tridiagonalization approach, which eliminates all subdiagonal entries in one shot, would never incur any numerically zero columns and hence never skip a transformation, SBR can take advantage of the special block structure exhibited by banded matrices with only two eigenvalues. The left plot shows the percentage of transformations we could skip for a particular value of d; the right plot shows the total number of transformations skipped. We see that it is beneficial to reduce the matrix to a relatively narrow band in the first step. One benefit is that we get closer to the desired tridiagonal form, and another is that we skip just about half the transformations in the first band reduction step, and over 90% in the second band reduction step. On the other hand, reducing a matrix to too narrow a band is counterproductive, as the blocksize becomes too small and the number of transformations needed for bulge chasing grows considerably. Based on these experiments, we consider an initial reduction to between 8 and 24 bands a reasonable choice.

The parallel implementation of this SBR scheme is in progress, with the first step having been completed. In fact, we implemented a general band reduction tool that allows us to reduce a matrix of bandwidth $nb*k$ to a matrix of size $nb*l$, with $1 \leq l \leq k-1$. Furthermore, nb is the block size used for the blocked torus wrap mapping and also the block size used for orthogonal transformations. Blocked reductions of full matrices to narrow bands and the unblocked tridiagonalization of a full matrix are special instances of this code. To develop a portable code and to allow a maintainable implementation, we chose to base our implementation on the Chameleon parallel programming tools [18].

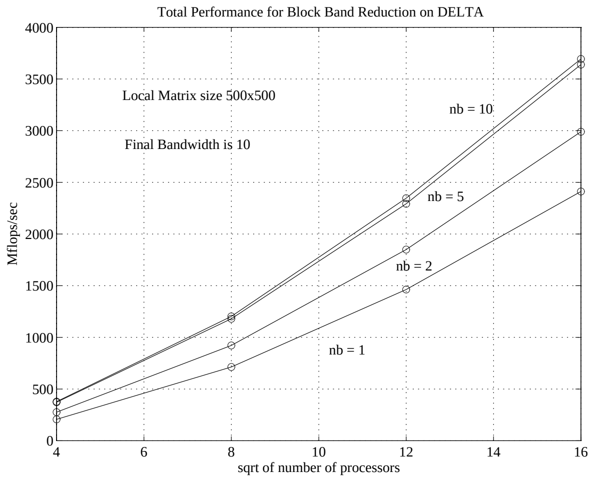


FIGURE 5. Total performance of blocked band reduction code on the Delta

The performance of this code is promising, and its performance on the reduction of a full random matrix to bandwidth 10 and the accumulation of orthogonal transformations on the Intel Touchstone Delta are shown in Figure 5. In these experiments, the matrix size on each processor was kept constant at 500×500, and the execution rates are based on the standard symmetric flop count of $(8n^3)/3$. These experiments were performed in double precision. We recall that for random matrices, no transformations can be skipped.

TABLE 1. Preliminary performance results on 16-node IBM SP1

n	nb	b	Mflops/proc	Total Mflops
1000	1	20	10.1	162
1020	15	15	18.7	299
2000	1	20	10.0	160
2000	10	10	26.2	419
3000	1	10	10.9	174
3000	15	15	33.7	539
4000	20	20	38.2	611

Some preliminary performance results on an IBM SP1, using the EUI-H transport layer, are shown in

Table 1. Here n is the overall matrix size, nb is the block size used, b is the number of bands the full matrix is reduced to, and the last two columns are per-processor performance and overall performance for a double-precision run, again based on the standard flop count. These results should be taken as a lower bound on the ultimate performance, as the performance of the IBM software and of Chameleon are improving rapidly.

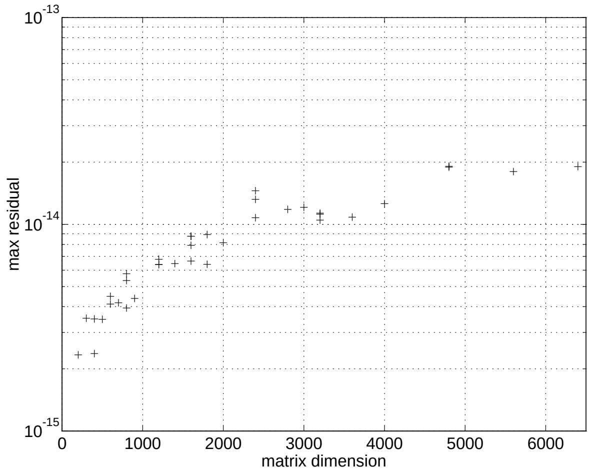
We see that, even now, the performance of the blocked code is superior to that of the unblocked code on both Delta and SP1. In particular, on the SP1 the blocked code outperforms the unblocked one by a wide margin. For example, in the time that we can reduce a 2000×2000 matrix to tridiagonal form using the unblocked approach, we can reduce a 3000×3000 matrix to bandwidth 10, even though the latter reduction involves roughly 3.5 times as many floating-point operations.

We are implementing a parallel version of Lang's algorithm, where the banded matrix (but not the orthogonal transformation matrix) is replicated in each row of the processor mesh. Given the performance gap between blocked and unblocked band reduction and the savings we are likely to incur as a result of orthogonal transformations being skipped, the final parallel SBR code should be even better in comparison with the unblocked code than the serial one.

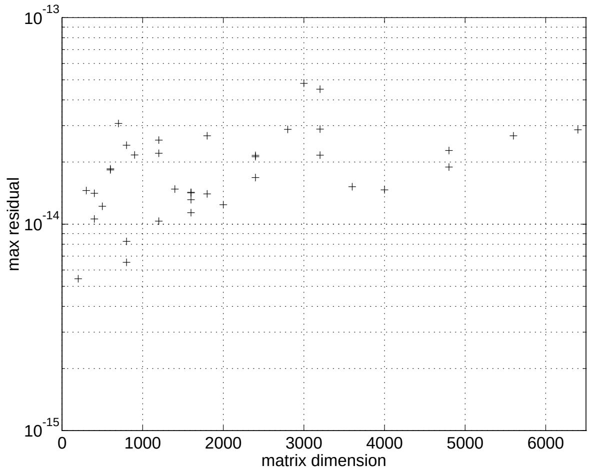
4. Load Balancing Strategies and Performance Results

The orchestration of multiple subproblems currently consists of two separate stages. 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 pointed out in Section 1, clusters tend to be gathered into the same subproblem. 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. 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 SYISDA indicate that all the eigenvalues of the current subproblem are either $O(\epsilon_M \Lambda)$ or within $O(\epsilon_M \Lambda)$ of each other (ϵ_M is the machine epsilon), then the subproblem is declared to have clustered eigenvalues and to be "done."

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 [23].



(a) Residuals



(b) Eigenvector orthogonality

FIGURE 6. Numerical accuracy of SYISDA

As the subproblem size decreases, there is a point, probably machine-dependent, 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. The strategy used in the end game is also described in [23].

Numerical testing in double precision of our parallel algorithm, using the unblocked version of the SBR code, is in progress. Easily generated symmetric test cases with uniformly distributed eigenvalues of dimensions 100–6400 have been tested so far. Accuracy in the residuals for a given matrix A is measured by computing the maximum normalized ∞-norm residual

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

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. Figure 6 shows that in terms of both accuracy in the residuals and orthogonality of the eigenvectors, SYISDA is performing well. We have done some limited testing on Wilkinson matrices and have thus far encountered no problems with accuracy.

TABLE 2. Time for complete eigensolution of $n \times n$ matrix on $p \times p$ submesh of the Delta

	time (min.)			
n	$p = 4$	$p = 8$	$p = 12$	$p = 16$
1600	23.16	7.98	4.97	3.70
3200	–	50.96	27.97	18.60
4800	–	–	83.11	52.60
6400	–	–	–	115.51

Table 2 gives the solution time in minutes for test matrices of sizes $1600, 3200, 4800$, and 6400 on $p \times p$ meshes, where $p = 4, 8, 12$, and 16.

Figure 7 shows the total performance of SYISDA for our test matrices on square submeshes of the Delta for matrices having matrix dimensions of 100, 200, 300, and 400 on each processor. Clearly, SYISDA scales extremely well. We see from Figure 1 that our distributed matrix multiplication runs at about 7.4 Gflops on a 16×16 mesh for 6400×6400 matrices; SYISDA achieves 6.4 Gflops, about 86% of what we

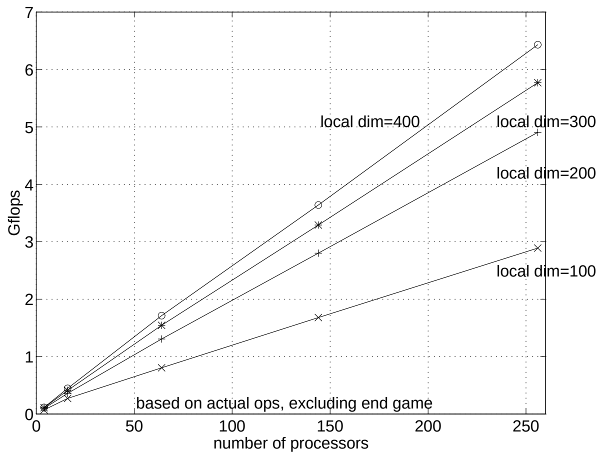


FIGURE 7. Total performance of SYISDA on square submeshes of the Delta

believe to be the maximum possible performance for this algorithm.

Our assembled SYISDA code currently runs only on square submeshes of the Delta using the unblocked SBR code. We expect its performance to improve when the blocked SBR has been completed. We are porting the code to other architectures and plan to generalize the code to nonsquare processor meshes in the future.

5. Data Layout and Library Considerations

Our work has reinforced the need for systemic support of basic communication kernels that are inevitably required when data are physically distributed. Because communication is highly machine-dependent, portability and compatibility with other software are often obtained at the price of performance or increased software development effort. Support for data redistribution is needed at two levels. The first is to support basic operations that are needed within the same data layout. In our case, the use of data layouts with symmetric placement of rows and columns led to simpler algorithms than with other layouts, without sacrificing the performance design objectives. However, since the data layout we used (as is the case for most data layouts) is not supported in a systemic manner, many "low-level" communication routines had to be written to perform what should be compiler-level operations. For instance, routines were needed for the basic operations described below.

Scattering: The problem to be solved must be distributed on the processors in the layout used.

Redistribution: In both the SBR and the end game, blocks of data scattered over one submesh had to be redistributed or replicated over a different submesh, sometimes in a different layout.

Layout conversion: In both SBR and for the invariant subspace update, data redistribution is required whenever the matrix being updated is not aligned with the update matrix.

Specialized routines had to be developed to deal with each of these situations. The second level of support should be in accommodating different layouts. Even within the torus wrap data layout, changing the block size requires significant data movement. Without such support, more specialized routines would be required. In fact, all the types of data movement described here pervade numerical computation and should be supported in future parallel languages such as High Performance Fortran. Otherwise, every user will be required to write his own specialized embedded routines. Continued lack of compiler support for basic communication kernels associated with data layout and movement will not only greatly impede library development, but will also impede the adoption of these libraries by users because of interface problems.

In our work on SYISDA, we have developed three kernels that provide the infrastructure for a "library" for parallel eigensolvers and, to a more limited extent, for numerical linear algebra computations. The first is matrix multiplication, the primitive of choice in current algorithmic design. The second, SBR, can in fact be used in conjunction with any tridiagonal or banded eigenvalue solver. Finally, the harness we have developed for managing the multiple subproblems produced by SYISDA readily transfers to any algorithm based on the invariant subspace decomposition approach. In particular, our approach can be readily modified to solve specialized partial spectrum problems, including those where the eigenvalues in a specified interval are desired and those where only the invariant subspace corresponding to eigenvalues in a specified interval is sought.

Our preliminary experiments indicate that our parallel implementation of SYISDA provides excellent scaling and accuracy. 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. A particularly promising algorithm, SYBISDA, 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 $O(b^2n)$ 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 [31], the run times for SYBISDA are competitive with the symmetric QR algorithm. In fact, the time spent in SBR becomes the dominant time. SYBISDA 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 SYBISDA and compare the two algorithms.

Acknowledgments

We thank Bill Gropp and Barry Smith for their help and support with the Chameleon programming system. We also thank Bruno Lang for providing us with his serial code for a packed storage bandreduction and for some stimulating discussions on this subject. Lastly, we thank Thomas Turnbull and Elaine Jacobson 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, E. M. Jacobson, X. Sun, & A. Tsao, *On the impact of HPF data layout on the design of efficient and maintainable parallel linear algebra libraries*, (available from the archives of the HPF Forum).

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

ing for Scientific Computing (Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, ed.), 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] Chandrasekaran, S., & I. Ipsen, *On rank-revealing QR factorizations*, Technical Report YALEU/DCS/RR880, Yale University, Department of Computer Science, 1991.

[11] Choi, J., J. J. Dongarra, & D. W. Walker, *Level 3 BLAS for distributed memory concurrent computers*, CNRS-NSF Workshop on Environments and Tools for Parallel Scientific Computing (Saint Hilaire du Touvet, France, September 7–8, 1992), Elsevier Science Publishers, 1992.

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

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

[14] 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).

[15] 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).

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

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

[18] Gropp, William D., & Barry Smith, *Chameleon parallel programming tools users manual*, ANL-93/23, Argonne National Laboratory (March 1993).

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

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

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

[22] 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 #11).

[23] 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).

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

[25] Lang, B., *A parallel algorithm for reducing symmetric banded matrices to tridiagonal form*, SIAM J. Sci. Stat. Comp. **14** (1993), no. 6.

[26] 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).

[27] 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.

[28] Mathur, K. K., & S. L. Johnsson, *Multiplication of matrices of arbitrary shape on a data parallel computer* (1992), Thinking Machines Corporation, preprint, also released as Technical Report TR-216.

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

[31] 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).

Preprint MCS-P383-0993, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., January 1994.