SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1, 1996	Final 15 Aug 91 - 14 Dec 95

4. TITLE AND SUBTITLE

A Scalable Parallel Library for Numerical Linear Algebra

5. FUNDING NUMBERS

DAAL03-91-C-0047

6. AUTHOR(S)

Jack J. Dongarra

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)

University of Tennessee
Department of Computer Science
107 Ayres Hall
Knoxville, TN 37996-1301

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

ARO 29113.58-MA

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report details the main technical results of the research project "A scalable parallel library for numerical linear algebra," and constitutes the final report for this project. The research led to the development of a substantial body of software, both in the area of parallel linear algebra, and in infrastructure for building parallel libraries. The main parallel software libraries produced were ScaLAPACK for dense linear algebra, ARPACK for large-scale eigenproblems, and CAPSS for solving sparse linear systems. The project also played a central role in initiating and supporting the creation of the MPI message passing interface. In addition, the project has promoted the use of software templates as building blocks for iterative methods, and project personnel have co-authored a book on this subject.

14. SUBJECT TERMS	15. NUMBER IF PAGES
numerical linear algebra, parallel software libraries, dense linear systems, large-scale eigenproblems, sparse linear systems;	41
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank)*

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as; prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NORFORN, REL, ITAR).

DOD	- See DoDD 4230.25, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only).*

Block 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

MASTER COPY: PLEASE KEEP THIS "MEMORANDUM OF TRANSMITTAL" BLANK FOR REPRODUCTION PURPOSES. WHEN REPORTS ARE GENERATED UNDER THE ARO SPONSORSHIP, FORWARD A COMPLETED COPY OF THIS FORM WITH EACH REPORT SHIPMENT TO THE ARO. THIS WILL ASSURE PROPER IDENTIFICATION. NOT TO BE USED FOR INTERIM PROGRESS REPORTS; SEE PAGE 1 FOR INTERIM PROGRESS REPORT INSTRUCTIONS.

MEMORANDUM OF TRANSMITTAL

U.S. Army Research Office
ATTN: AMXRO-ICA-L (Hall)
P.O. Box 12211
Research Triangle Park, NC 27709-2211

____ Reprint (Orig + 2 copies)
____ Technical Report (Orig + 2 copies)
____ Manuscript (1 copy)
_X_ Final Progress Report (Orig + 2 copies)
____ Related Material (1 copy)

CONTRACT/GRANT NUMBER: DAAL03-91-C-0047

TITLE: A Scalable Parallel Library for Numerical Linear Algebra

is forwarded for your information.

SUBMITTED FOR PUBLICATION TO (applicable only if report is manuscript):

Sincerely,

DO NOT REMOVE LABEL BELOW
THIS IS FOR IDENTIFICATION PURPOSES

Jack J. Dongarra 34387MA
Department of Computer Science
University of Tennessee,
Knoxville
Knoxville, TN 37916

A Scalable Parallel Library for Numerical Linear Algebra

Final Progress Report
Jack J. Dongarra
January 1996

U.S. Army Research Office

Research Agreement DAAL03-91-C-0047

University of Tennessee at Knoxville
Oak Ridge National Laboratory
University of California at Berkeley
Rice University
University of Illinois at Urbana-Champaign
University of California at Los Angeles

Contents

List of Figures

1 Introduction

The research project "A scalable parallel library for numerical linear algebra" consisted of a number of closely related topics involving researchers at a number of institutions. ScaLAPACK was developed at the University of Tennessee at Knoxville, Oak Ridge National Laboratory, and the University of California at Berkeley. ScaLAPACK is a prototype library of software for performing dense linear algebra computations on message-passing computers. ARPACK was developed at Rice University, and is a software package for solving large, sparse, nonsymmetric eigenproblems using a variant of the Arnoldi method. CAPSS, developed at the University of Illinois at Urbana-Champaign, is a fully parallel package for solving sparse linear systems of the form $Ax = b$ on message passing computers using matrix factorization. Researchers at the University of California at Los Angeles have developed a PDE solver package that provided support for the solution of elliptic PDEs using either finite elements or finite differences in two or three dimensions. ScaLAPACK, ARPACK, and CAPSS have been placed in the public domain and are accessible via the National HPCC Software Exchange.

The research led to a number of important new software tools and standards. Recognizing that a message passing standard was necessary to ensure the easy portability of the prototype libraries, the project initiated and promoted the development of the MPI message passing interface [30]. Standards specific to parallel linear algebra were also developed. The PBLAS (Parallel Basic Linear Algebra Subprograms) [17] are message passing versions of most of the sequential BLAS routines, and have a similar interface. The BLACS (Basic Linear Algebra Communication Subprograms) are a set of routines for communicating rectangular and trapezoidal sub-matrices between processes. The BLACS and PBLAS are important building blocks of the ScaLAPACK library, and are also used in ARPACK.

A number of new scalable algorithms have been developed. A new version of the dense symmetric eigenvalue routine PDSYEVX is faster in most common cases and eliminates the need to reorthogonalize in the clustered eigenvalue case. A second eigensolver was also developed that is slower than PDSYEVX, but is more reliable. An algorithm based on the Arnoldi technique has been developed that evaluates the

eigenvalues for a selected subset of the spectrum of a large matrix.

The remaining sections of this report discuss each of the main topics of the research in more detail.

2 ScaLAPACK

Two key factors in ensuring that the ScaLAPACK algorithms have good scalability and performance characteristics are maintaining long vector lengths, and maximizing data reuse in the upper levels of memory. Long vector lengths result in more effective use of the vector or RISC processors found in many parallel computers. Thus, in implementing ScaLAPACK we have avoided performing operations on small matrices and vectors. By reusing data in the upper levels of memory (registers and cache) the longer latencies associated with accesses to lower levels of memory (main memory, off-processor memory) are avoided. In ScaLAPACK, high levels of data reuse are ensured by the use of block partitioned algorithms that exploit locality of reference. This reduces the frequency of communication between processes, thereby avoiding message startup latency. The sequential computations performed by each process are mostly expressed in terms of Level 2 and Level 3 Basic Linear Algebra Subprograms (BLAS) [21, 24]. These computations are done using commercially available assembly coded routines that have good data reuse characteristics, and make efficient use of the target chip architecture.

In many of the ScaLAPACK routines, such as the factorization routines discussed in [26], columns and/or rows of the matrix are eliminated as the computation progresses. This leads to a tradeoff between data reuse and load balance. This tradeoff is discussed in [27], and may be controlled at the user level by varying the parameters of the data distribution, as discussed in the next subsection.

2.1 Data Distribution

In many linear algebra algorithms the distribution of work may become uneven as the algorithm progresses, as in LU factorization in which rows and columns become eliminated from the computation. ScaLAPACK, therefore, makes use of the block cyclic data distribution in which ma-

trix blocks separated by a fixed stride in the row and column directions are assigned to the same process. The block cyclic data distribution is parameterized by the four numbers P, Q, r, and c, where $P \times Q$ is the process template and $r \times c$ is the block size. Although the interface of all ScaLAPACK routines can support arbitrary values of these parameters, the coding of certain routines imposes further requirements on these parameters. These temporary code restrictions simplified the code development and insured efficiency. Thus, for example, in the LU factorization, we require that the blocks be square.

Suppose we have M objects indexed by the integers $0, 1, \ldots, M-1$. In the block cyclic data distribution the mapping of the global index, m, can be expressed as $m \mapsto (p, b, i)$, where p is the logical process number, b is the block number in process p, and i is the index within block b to which m is mapped. Thus, if the number of data objects in a block is r, the block cyclic data distribution may be written,

$$m \mapsto \left(\left\lfloor \frac{m \bmod T}{r} \right\rfloor , \left\lfloor \frac{m}{T} \right\rfloor , m \bmod r \right) \tag{1}$$

where $T = rP$, and P is the number of processes. The distribution of a block-partitioned matrix can be regarded as the tensor product of two such mappings, one that distributes the rows of the matrix over P processes, and another that distributes the columns over Q processes. It should be noted that Eq. 1 reverts to the cyclic distribution when $r = 1$, with local index $i = 0$ for all blocks. A block distribution is recovered when $r = \lceil M/P \rceil$, in which case there is a single block in each process with block number $b = 0$. Thus, we have

$$m \mapsto (m \bmod P, \lfloor m/P \rfloor, 0) \tag{2}$$

for a cyclic data distribution, and

$$m \mapsto (\lfloor m/L \rfloor , 0, m \bmod L) , \tag{3}$$

for a block distribution, where $L = \lceil M/P \rceil$.

2.2 Building Blocks

The ScaLAPACK routines are built out of a small number of modules. The most fundamental of these are the Basic Linear Algebra Communication Subprograms (BLACS) [22, 23, 25], that perform common

matrix-oriented communication tasks, and the sequential Basic Linear Algebra Subprograms (BLAS) [21, 24, 38], in particular the Level 2 and 3 BLAS. ScaLAPACK can be ported with no code modification to any machine on which the BLACS and the BLAS are available. The Parallel BLAS (PBLAS) are message passing versions of most of the sequential BLAS routines. The BLACS, the sequential BLAS, and the PBLAS are the modules from which the higher level ScaLAPACK routines are built. Thus, the entire ScaLAPACK package contains modules at a number of different levels. For many users the top level ScaLAPACK routines will be sufficient to build applications. However, more expert users may make use of the lower level routines to build customized routines not provided in ScaLAPACK.

The BLACS package attempts to provide the same ease of use and portability for MIMD message-passing linear algebra communication that the BLAS provide for linear algebra computation. Therefore, future software for dense linear algebra on MIMD platforms could consist of calls to the PBLAS for computation and calls to the BLACS for communication. Since both packages will have been optimized for each particular platform, good performance should be achieved with relatively little effort.

In the ScaLAPACK routines all interprocess communication takes place within the PBLAS and the BLACS, so the source code of the top software layer of ScaLAPACK looks very similar to that of LAPACK. The BLACS have been implemented for the Intel family of computers, the TMC CM-5, the IBM SP-X, the Cray T3D, PVM, and MPI.

2.3 Contents of ScaLAPACK

The initial public release of ScaLAPACK occurred in February, 1995, and included routines for the solution of a general system of linear equations via LU and Cholesky factorizations, orthogonal factorizations (QR, RQ, LQ, and QL), reduction to condensed form (upper Hessenberg, tridiagonal form, and bidiagonal), and the symmetric eigenproblem.

Since this initial public release, many improvements have been underway to increase the flexibility and functionality of the library. Flexibility is being increased by expanding the spectrum of PBLAS operations. For instance the ability to operate on non-aligned data not only

greatly simplifies the writing of ScaLAPACK codes but also makes it possible to express divide-and-conquer algorithms in terms of calls to ScaLAPACK and PBLAS routines. The functionality of ScaLAPACK is being expanded by the introduction of new routines such as condition estimation and iterative refinement for LU and Cholesky, full rank and rank deficient linear least squares, generalized RQ and QR factorizations, generalized linear least squares, singular value decomposition, symmetric positive definite banded routines, and tridiagonal solvers.

2.4 Symmetric Eigenvalue Problem

We designed, produced and released a parallel algorithm for the symmetric eigenvalue problem, PDSYEVX. This was released as part of the 1995 ScaLAPACK release. This code can return some or all of the eigenvalues of a dense symmetric matrix, and the corresponding eigenvectors. The constituent subroutines can be applied to the symmetric tridiagonal problem. We modeled this routine on the LAPACK routine DSYEVX, and encountered a significant tradeoff between speed and the ability to guarantee orthogonality of the computed eigenvalues in certain pathological cases. We chose speed over orthogonality, mostly because of related work on a new algorithm that promises to eliminate this problem (and will also replace the LAPACK implementation). This work is discussed in the section 8.2. We also produced a detailed performance model of the current code, and variations on it, which we have used in performance tuning and algorithm design [20].

2.5 Documentation

We are in the process of writing the ScaLAPACK Users' Guide. This book is patterned after the LAPACK Users' Guide and will contain descriptions of all routines contained in the package, as well as example programs of their usage. It has been frequently requested by the user community.

We maintain homepages for ScaLAPACK and the BLACS at the respective URLs:

http://www.netlib.org/scalapack/index.html

and

http://www.netlib.org/blacs/index.html

and provide software support via the mailing aliases `scalapack@cs.utk.edu` and `blacs@cs.utk.edu`.

3 ARPACK

We have developed mathematical software for large scale eigenvalue problems based upon a new variant of the Arnoldi process. This new variant of the Arnoldi process employs an implicit restarting scheme that may be viewed as a truncation of the standard implicitly shifted QR-iteration for dense problems. Numerical difficulties and storage problems normally associated with Arnoldi and Lanczos processes are avoided. The algorithm is capable of computing a few eigenvalues with user specified features such as largest real part or largest magnitude using a predetermined storage requirement proportional to matrix order times the desired number of eigenvalues.

The ARPACK software, which is based upon an implementation of this algorithm, has been designed to be efficient on a variety of high performance computers. Parallelism within the scheme is obtained primarily through the matrix-vector operations that comprise the majority of the work in the algorithm. The software is capable of solving large scale symmetric, nonsymmetric, and generalized eigenproblems from significant application areas.

3.1 Large Sparse Eigenvalue Software

The most general problem addressed by this software is the generalized eigenproblem

$$Ax = \lambda Mx, \tag{4}$$

where both A and M are real $n \times n$ matrices and M is symmetric. We assume that the pair (A, M) is a regular definite pencil if A is also symmetric or that M is positive semi-definite if A is nonsymmetric.

Arnoldi's method is a Krylov subspace projection method. It obtains approximations to eigenvalues and corresponding eigenvectors of

a large matrix A by constructing the orthogonal projection of this matrix onto the Krylov subspace $Span\{v, Av, ..., A^{k-1}v\}$. The Arnoldi process begins with the specification of a starting vector v and in k steps produces the decomposition of an $n \times n$ matrix A into the form

$$AV = VH + fe_k^T, \tag{5}$$

where v is the first column of the matrix $V \in \mathbf{R}^{n\times k}$, $V^TV = I_k$; $H \in \mathbf{R}^{k\times k}$ is upper Hessenberg, $f \in \mathbf{R}^n$ with $0 = V^Tf$ and $e_k \in \mathbf{R}^k$ the kth coordinate basis vector. The vector f is called the residual. This factorization may be advanced one step at the cost of a (sparse) matrix-vector product involving A and two dense matrix vector products involving V^T and V. The dense products may be accomplished using level 2 BLAS. The new column of V will be $v_{k+1} = f/\beta$ where $\beta = \|f\|$, and β will become the next subdiagonal element of H.

The columns of V form an orthonormal basis for the Krylov subspace and H is the orthogonal projection of A onto this space. Eigenvalues and corresponding eigenvectors of H provide approximate eigenvalues and eigenvectors for A. If $Hy = y\theta$ and we put $x = Vy$, then x, θ is an approximate eigenpair for A with

$$\|Ax - x\theta\| = \|f\||e_k^Ty|,$$

and this provides a means for estimating the quality of the approximation.

The information obtained through this process is completely determined by the choice of the starting vector. Eigen-information of interest may not appear until k gets very large. In this case it becomes intractable to maintain numerical orthogonality of the basis vectors V and it also will require extensive storage. Failure to maintain orthogonality leads to a number of numerical difficulties. Our method provides a means to extract interesting information from very large Krylov subspaces while avoiding the storage and numerical difficulties associated with the standard approach. It does this by continually compressing the interesting information into a fixed size k dimensional subspace. This is accomplished through the implicitly shifted QR mechanism. An Arnoldi factorization of length $k + p$ is compressed to a factorization of length k by applying p implicit shifts resulting in

$$AV_{k+p}^+ = V_{k+p}^+H_{k+p}^+ + f_{k+p}e_{k+p}^T\hat{Q}, \tag{6}$$

where $V_{k+p}^+ = V_{k+p}\hat{Q}$, $H_{k+p}^+ = \hat{Q}^T H_{k+p}\hat{Q}$, and $\hat{Q} = Q_1 Q_2 \cdots Q_p$, with Q_j the orthogonal matrix associated with the shift μ_j. It may be shown that the first $k-1$ entries of the vector $e_{k+p}^T\hat{Q}$ are zero. Equating the first k columns on both sides yields an updated $k-$step Arnoldi factorization

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^T, \tag{7}$$

with an updated residual of the form $f_k^+ = V_{k+p}^+ e_{k+1}\hat{\beta}_k + f_{k+p}\sigma$. Using this as a starting point it is possible to use p additional steps of the Arnoldi process to return to the original form. Each of these applications implicitly applies a polynomial in A of degree p to the starting vector. The roots of this polynomial are the shifts used in the QR process and these may be selected to filter unwanted information from the starting vector and hence from the Arnoldi factorization. Full details may be found in [48].

The resulting software ARPACK based upon this mechanism provides several features which are not present in existing (single vector) codes to our knowledge:

- Reverse communication interface
- Ability to return k eigenvalues which satisfy a user specified criterion such as largest real part, largest absolute value, largest algebraic value (symmetric case), etc.
- A fixed pre-determined storage requirement suffices throughout the computation. Usually this is $n * O(2k) + O(k^2)$ where k is the number of eigenvalues to be computed and n is the order of the matrix. No auxiliary storage or interaction with such devices is required during the course of the computation.
- Eigenvectors may be computed on request. The Arnoldi basis of dimension k is always computed. The Arnoldi basis consists of vectors which are numerically orthogonal to working accuracy.
- Accuracy: The numerical accuracy of the computed eigenvalues and vectors is user specified and may be set to the level of working precision. At working precision, the accuracy of the computed

eigenvalues and vectors is consistent with the accuracy expected of a dense method such as the implicitly shifted QR iteration.

- Multiple eigenvalues offer no theoretical or computational difficulty other than additional matrix vector products required to expose the multiple instances. This cost is commensurate with the cost of a block version of appropriate blocksize.

3.2 Applications of ARPACK

ARPACK has been used in a variety of challenging applications, and has proven to be useful both in symmetric and nonsymmetric problems. It is of particular interest when there is no opportunity to factor the matrix and employ a "shift and invert" form of spectral transformation,

$$\hat{A} \leftarrow (A - \sigma I)^{-1} . \tag{8}$$

Existing codes often rely upon this transformation to enhance convergence. Extreme eigenvalues $\{\mu\}$ of the matrix $\hat{A}$ are found very rapidly with the Arnoldi/Lanczos process and the corresponding eigenvalues $\{\lambda\}$ of the original matrix A are recovered from the relation $\lambda = 1/\mu + \sigma$. Implementation of this transformation generally requires a matrix factorization. In many important applications this is not possible due to storage requirements and computational costs. The implicit restarting technique used in ARPACK is often successful without this spectral transformation.

One of the most important classes of application arises in computational fluid dynamics. Here the matrices are obtained through discretization of the Navier-Stokes equations. A typical application involves linear stability analysis of steady state solutions. Here one linearizes the nonlinear equation about a steady state and studies the stability of this state through the examination of the spectrum. Usually this amounts to determining if the eigenvalues of the discrete operator lie in the left halfplane. Typically these are parametrically dependent problems and the analysis consists of determining phenomena such as simple bifurcation, Hopf bifurcation (an imaginary complex pair of eigenvalues cross the imaginary axis), turbulence, and vortex shedding as this parameter is varied. Our method is well suited to this setting as

it is able to track a specified set of eigenvalues while they vary as functions of the parameter. Our software has been used to find the leading eigenvalues in a Couette-Taylor wavy vortex instability problem involving matrices of order 4000. One interesting facet of this application is that the matrices are not available explicitly and are logically dense. The particular discretization provides efficient matrix-vector products through Fourier transform. Details may be found in [29].

Alvarez-Cohen and McCarty have studied a groundwater remediation problem through a large nonsymmetric eigenanalysis [2]. They use a pore-scale model to understand macroscopic groundwater transport phenomena. Convection, diffusion, and biochemical reactions occur at the pore level. The equations model flow through a single pore, whose lining reacts with the flowing solute. Boundary conditions are periodic. The eigenvalues of this boundary value problem provide useful information about the flow through an aggregate of pore cells. Solution of the eigenproblem is discussed in [28]. Preliminary computational studies indicate that ARPACK can provide a means to extract a number of interesting eigenvalues and eigenvectors more efficiently than the inverse power method that is currently employed.

Our software has been used to study the stability of the core of a civil nuclear power plant, as modeled by the two-group neutron diffusion equation. Vaudescal [51] reports improved performance using ARPACK over results obtained in [36] using explicitly restarted Arnoldi.

Very large symmetric generalized eigenproblems arise in structural analysis. One example that we have worked with at Cray Research through the courtesy of Ford motor company involves an automobile engine model constructed from 3D solid elements. Here the interest is in a set of modes to allow solution of a forced frequency response problem $(K - \lambda M)x = f(t)$, where $f(t)$ is a cyclic forcing function which is used to simulate expanding gas loads in the engine cylinder as well as bearing loads from the piston connecting rods. This model has over 250,000 degrees of freedom. The smallest eigenvalues are of interest and the ARPACK code appears to be very competitive with the best commercially available codes on problems of this size. For details see [49].

Nonlinear eigenvalue problems also arise in structural analysis. We

are collaborating with researchers at Stanford University in this area. In [47] we present an implicitly restarted Lanczos-based eigensolution technique for evaluating the natural frequencies and modes from frequency dependent eigenproblems in structural dynamics. The new solution technique is used in conjunction with a mixed finite element modeling procedure which utilizes both the polynomial and frequency dependent displacement fields in formulating the system matrices. The method is well suited to the solution of large scale problems. The solution methodology presented in [47] is based upon the ability to evaluate a specific set of parameterized nonlinear eigenvalue curves at given values of the parameter using the symmetric generalized eigensolvers available in ARPACK. Numerical examples illustrate that the implicitly restarted Lanczos method with secant interpolation accurately evaluates the exact natural frequencies and modes of the nonlinear eigenproblem and verifies that the new eigensolution technique coupled with the mixed finite element modeling procedure is more accurate than the conventional finite element models. In addition, the eigenvalue technique presented here is shown to be far more computationally efficient on large scale problems than the determinant search techniques traditionally employed for solving exact vibration problems. These techniques are being extended to solve damped problems (which are nonsymmetric) and interior eigenvalue problems.

Computational chemistry provides a rich source of problems. ARPACK is being used in two applications currently and holds promise for a variety of challenging problems in this area. We are collaborating with researchers at Ohio State on large scale three-dimensional reactive scattering problems. The governing equation is the Schroedinger equation and the computational technique for studying the physical phenomena relies upon repeated eigenanalysis of a Hamiltonian operator consisting of a Laplacian operator discretized in spherical co-ordinates plus a surface potential. The discrete operator has a tensor product structure from the discrete Laplacian plus a diagonal matrix from the potential. The resulting matrix has a block structure consisting of $m \times m$ blocks of order n . The diagonal blocks are dense and the off diagonal blocks are scalar multiples of the order n identity matrix. It is virtually impossible to factor this matrix directly because the factors are dense in any ordering. We are using a distributed memory parallel

version of ARPACK together with some preconditioning ideas to solve these problems on distributed memory machines. Encouraging computational results have been obtained on Cray Y-MP machines and also on the Intel Delta. See [32], [49] for further details.

Nonsymmetric problems also arise in quantum chemistry. Researchers at University of Washington have used the code to investigate the effects of the electric field on InAs/GaSb and GaAs/Al_xGa_{1-x} as quantum wells. ARPACK was used to find highly accurate solutions to these nonsymmetric problems which couldn't be solved by other means. See [39] for details.

Another source of problems arise in magnetohydrodynamics (MHD) involving the study of the interaction of a plasma and a magnetic field. The MHD equations describe the macroscopic behavior of the plasma in the magnetic field. These equations form a system of coupled nonlinear PDE. Linear stability analysis of the linearized MHD equations leads to a complex eigenvalue problem. Researchers at the Institute for Plasma Physics and Utrecht University in the Netherlands have modified the codes in ARPACK to work in complex arithmetic and are using the resulting code to obtain very accurate approximations to the eigenvalues lying on the Alfven curve. The code is not only finding extremely accurate solutions, it is doing so far more efficiently than the existing method of choice. Currently problems of order 3216 are being solved. The complex version of ARPACK produced 45 good approximations of eigenvalues in 27 seconds of Cray Y-MP CPU time while the method currently in use needed 32 seconds to find 25 poorly converged approximations. See [37] for details.

There are many other applications. In addition to the examples just mentioned, ARPACK has been used to solve large scale problems in the optimal design of a membrane and in the design of dielectric waveguides. It may also be used to to compute the singular value decomposition (SVD) of large matrices. There are many important applications of the SVD including analysis and enhancement of digital images. Several applications of this technology arise in Computational Biology as well as many other fields. As we gain experience with the ARPACK software, we find an increasing number of new interesting and challenging applications. The dramatic increase in modern computing power combined with the new algorithms available in the ARPACK software can

provide solutions to eigenproblems that were previously intractable.

4 Application Templates

A new generation of even more massively parallel computers will soon emerge. Concurrent with the development of these more powerful parallel systems is a shift in the computing practices of many scientists and researchers. Increasingly, the tendency is to use a variety of distributed computing resources, with each individual task assigned to the most appropriate architecture, rather than to use a single, monolithic machine. The pace of these two developments, the emergence of highly parallel machines and the move to a more distributed computing environment, has been so rapid that software developers have been unable to keep up. Part of the problem has been that supporting system software has inhibited this development. Consequently, exploiting the power of these technological advances has become more and more difficult. Much of the existing reusable scientific software, such as that found in commercial libraries and in public domain packages, is no longer adequate for the new architectures. If the full power of these new machines is to be realized, then scalable libraries, comparable in scope and quality to those that currently exist, must be developed.

One of our goals as software designers is to communicate to the high-performance computing community algorithms and methods for the solution of system of linear equations. In the past we have provided black-box software in the form of a mathematical software library, such as LAPACK, LINPACK, NAG, and IMSL. These software libraries provide for:

- Easy interface with hidden details
- Reliability; the code should fail as rarely as possible
- Speed.

On the other hand, the high-performance computing community, which wants to solve complex, large-scale problems as quickly as possible, seems to want

- Speed

- Access to internal details to tune data structures to the application
- Algorithms that are fast for the particular application even if not reliable as general methods.

These differing priorities make for different approaches to algorithms and software. The first set of priorities leads us to produce "black boxes" for general problem classes. The second set of priorities seems to lead us to produce "template codes" or "toolboxes" where the users can assemble, modify and tune building blocks starting from well-documented subparts. This leads to software which is not going to be reliable on all problems, and requires extensive user tuning to make it work. This is just what the block-box users do not want.

In scientific high-performance computing we see three different computational platforms emerging, each with a distinct set of users. The first group of computers contains the traditional supercomputer. Computers in this group exploit vector and modest parallel computing. They are general purpose computers that can accommodate a large cross section of applications while providing a high percentage of their peak computing rate. They are the computers typified by the Cray Y-MP and C90, IBM ES/9000, and NEC SX-3; the so-called general purpose vector supercomputers.

The second group of computers are the highly parallel computers. These machines often contain hundreds or even thousands of processors, usually RISC in design. The machines are usually loosely coupled having a switching network and relatively long communication times compared with computation times. These computers are suitable for fine-grain and coarse-grain parallelism. As a system, the cost is usually less than the traditional supercomputer and the programming environment is very poor and primitive. There is no portability since user's programs depend heavily on a particular architecture and on a particular software environment.

The third group of computers are the clusters of workstations. Each workstation usually contains a single very fast RISC processor. Each workstation is connected through a Local Area Network, or LAN, and as such the communication time is very slow, making this setup not very suitable for fine-grain parallelism. They usually have a rich software environment and operating system on a workstation node, usually

UNIX. This solution is usually viewed as a very cost-effective solution compared to the vector supercomputers and highly-parallel computers.

Users are in general not a monolithic entity, but in fact represent a wide diversity of needs. Some are the sophisticated computational scientists who eagerly move to the newest architecture in search of ever-higher performance. Others want only to solve their problems with the least change to their computational approach.

We hope to satisfy the high-performance computing community's needs by the use of reusable software templates. With the templates we describe the basic features of the algorithms. These templates offer the opportunity for whatever degree of customization the user may desire, and also serve a valuable pedagogical role in teaching parallel programming and instilling a better understanding of the algorithms employed and results obtained. While providing the reusable software templates we hope to retain the delicate numerical details in many algorithms.

We believe it is important for users to have trust in the algorithms, and hope this approach conveys the spirit of the algorithm and provides a clear path for implementation where the appropriate data structures can be integrated into the implementation. We believe that this approach of templates allows for easy modification to suit various needs. More specifically, each template should have:

- Working software for matrices as general as the method allows.
- A mathematical description of the flow of the iteration.
- Algorithms described in a Fortran-77 program with calls to BLAS [21, 24, 38], and LAPACK routines [3].
- Discussion of convergence and stopping criteria.
- Suggestions for extending a method to more specific matrix types (for example, banded systems).
- Suggestions for tuning (for example, which preconditioners are applicable and which are not).
- Performance: when to use a method and why.
- Reliability: for what class of problems the method is appropriate.

- Accuracy: suggestions for measuring the accuracy of the solution, or the stability of the method.

An area where this will work well is with sparse matrix computations. Many important practical problems give rise to large sparse systems of linear equations. One reason for the great interest in sparse linear equations solvers and iterative methods is the importance of being able to obtain numerical solutions to partial differential equations. Such systems appear in studies of electrical networks, economic-system models, and physical processes such as diffusion, radiation, and elasticity. Iterative methods work by continually refining an initial approximate solution so that it becomes closer and closer to the correct solution. With an iterative method a sequence of approximate solutions $\{x^{(k)}\}$ is constructed which essentially involve the matrix A only in the context of matrix-vector multiplication. Thus the sparsity can be taken advantage of so that each iteration requires $O(n)$ operations.

Many basic methods exist for iteratively solving linear systems and finding eigenvalues. The trick is finding the most effective method for the problem at hand. The method that works well for one problem type may not work as well for another. Or it may not work at all. We have written a book on templates for large sparse linear systems [7] to help address the needs of users of high performance computers.

5 CAPSS

Although many large scale scientific applications require a sparse direct solver, there has been a lack of both algorithms and software to translate the high execution rates of massively parallel multiprocessors into faster solution times for such applications. The primary goal of this portion of the research is the development of algorithms and software for the scalable, fully parallel solution of large sparse systems of linear equations using direct methods.

The main step in direct methods is matrix factorization. For sparse matrices this step is preceded by a symbolic phase in which the matrix is permuted so that sparsity will be preserved in the subsequent factorization. For a fully parallel solver, both numeric and symbolic steps must be parallelized. Several parallel numeric factorization algorithms

have been developed, but by comparison the effective parallelization of the ordering step has lagged. Our work is based on a central idea, that of developing new scalable parallel algorithms for nested dissection ordering and using the resulting partition tree in formulating various parallel matrix factorization schemes, including Cholesky (LL^T), Gaussian elimination with partial pivoting (LU), and orthogonal factorization (QR).

We have developed a scalable parallel nested dissection scheme, called Cartesian nested dissection [35]. This algorithm is suitable for sparse matrices associated with an embedding in Euclidean space, such as those typically arising from finite-element and finite-difference methods for partial differential equations. We have also developed a more general purpose parallel nested dissection algorithm based on parallel graph contraction [45]. Using a multifrontal approach based on the resulting partition tree, we have developed algorithms for the numeric steps of the parallel solution process for symmetric positive definite systems [33] as well as nonsymmetric systems [43]. Some of these algorithms have been packaged and distributed as CAPSS, a fully parallel solver for message-passing multiprocessors [34].

We have used the basic framework for a fully parallel sparse direct solver embodied in CAPSS as a basis for further development and enhancement of the various steps involved. For example, we are exploring the use of numeric kernels for dense subproblems, based on ScaLapack, to enhance both performance and scalability. Another phase of the computation that has been a substantial bottleneck in the past is the triangular solution, due to its unfavorable ratio of communication to computation. We have developed a new "selective inversion" algorithm [44] for the triangular solution phase that exhibits vastly improved performance and scalability over previous methods for sparse triangular solutions on message-passing, distributed-memory architectures. This breakthrough is especially critical for applications that require the solution of several linear systems involving the same matrix but different right-hand-side vectors. We have also studied the use of local refinement techniques for improving the quality of nested dissection orderings.

Other research concerns the use of sparse direct methods in applications. To gain some experience with the use of scalable software

for solving real-world problems, we have applied CAPSS to such problems as the stress analysis of foam-like materials, extrusion problems, and crack-tip propagation. Interfacing with such applications codes has emphasized to us the necessity of developing better interfaces that are easier to use and/or more transparent to the user. One approach we are exploring is the use of an interactive user interface that hides the complex details of data distribution. We have also used sparse matrix algorithms and results to evaluate the complexity of a finite-element formulation in electromagnetic scattering [40], and to reorder term-document matrices during information retrieval [8].

6 MPI

MPI is a standard message passing interface. The design of MPI was a collective effort involving researchers in the United States and Europe from many organizations and institutions. MPI includes point-to-point and collective communication routines, as well as support for process groups, and application topologies.

In MPI there is currently no mechanism for creating processes, and an MPI program is parallel *ab initio*, i.e., there is a fixed number of processes from the start to the end of an application program. All processes are members of at least one process group. Initially all processes are members of the same group, and a number of routines are provided that allow the user to create (and destroy) new subgroups. Within a group each process is assigned a unique rank in the range 0 to $n-1$, where n is the number of processes in the group. This rank is used to identify a process, and, in particular, is used to specify the source and destination processes in a point-to-point communication operation, and the root process in certain collective communication operations. As in PVM, message selectivity in point-to-point communication as by source process and message tag, each of which may be wildcarded to indicate that any valid value is acceptable.

The key innovative ideas in MPI are the communicator abstraction and general, or derived, datatypes. These will be discussed before describing the communication routines in more detail. Communicators provide support for the design of safe, modular software libraries. Here "safe" means that messages intended for a particular receive routine

in an application will not be incorrectly intercepted by another receive routine. Thus, communicators are a powerful mechanism for avoiding unintentional non-determinism in message passing. This is a particular problem when using third-party software libraries that perform message passing. The point here is that the application developer has no way of knowing if the tag, group, and rank completely disambiguate the message traffic of different libraries and the rest of the application. Communicator arguments are passed to all MPI message passing routines, and a message can be communicated only if the communicator arguments passed to the send and receive routines match. Thus, in effect communicators provide an additional criterion for message selection, and hence permit the construction of independent tag spaces.

If communicators are not used to disambiguate message traffic there are two ways in which a call to a library routine can lead to unintended behavior. In the first case the processes enter a library routine synchronously when a send has been initiated for which the matching receive is not posted until after the library call. In this case the message may be incorrectly received in the library routine. The second possibility arises when different processes enter a library routine asynchronously, as shown in the example in Figure 1, resulting in nondeterministic behavior. If the program behaves correctly processes 0 and 1 each receive a message from process 2, using a wildcarded selection criterion to indicate that they are prepared to receive a message from any process. The three processes then pass data around in a ring within the library routine. If separate communicators are not used for the communication inside and outside of the library routine this program may intermittently fail. Suppose we delay the sending of the second message sent by process 2, for example, by inserting some computation, as shown in Figure 2. In this case the wildcarded receive in process 0 is satisfied by a message sent from process 1, rather than from process 2, and deadlock results. By supplying a different communicator to the library routine we can ensure that the program is executed correctly, regardless of when the processes enter the library routine.

Communicators are opaque objects, which means they can only be manipulated using MPI routines. The key point about communicators is that when a communicator is created by an MPI routine it is guaranteed to be unique. Thus it is possible to create a communicator and

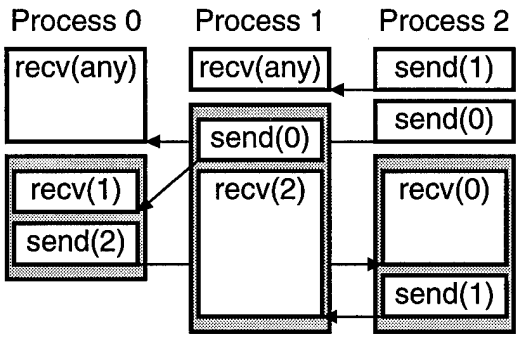

Figure 1: Use of communicators. Time increases down the page. Numbers in parentheses indicate the process to which data are being sent or received. The gray shaded area represents the library routine call. In this case the program behaves as intended. Note that the second message sent by process 2 is received by process 0, and that the message sent by process 0 is received by process 2.

pass it to a software library for use in all that library's message passing. Provided that communicator is not used for any message passing outside of the library, the library's messages and those of the rest of the application cannot be confused.

Communicators have a number of attributes. The group attribute identifies the process group relative to which process ranks are interpreted, and/or which identifies the process group involved in a collective communication operation. Communicators also have a topology attribute which gives the topology of the process group. Topologies are discussed below. In addition, users may associate attributes with communicators through a mechanism known as caching.

All point-to-point message passing routines in MPI take as an argument the datatype of the data communicated. In the simplest case this will be a primitive datatype, such as an integer or floating point number. However, MPI provides a number of routines for creating more general datatypes,and thereby supports the communication of array sections and structures involving combinations of primitive datatypes.

In many applications the processes are arranged with a particular topology, such as a two- or three-dimensional grid. MPI provides support for general application topologies that are specified by a graph in

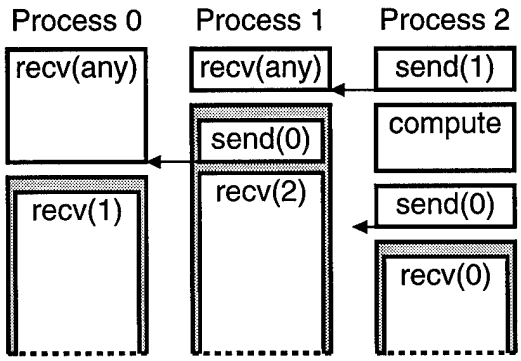

Figure 2: Unintended behavior of program. In this case the message from process 2 to process 0 is never received, and deadlock results.

which processes that communicate a significant amount are connected by an arc. If the application topology is an n-dimensional Cartesian grid then this generality is not needed, so as a convenience MPI provides explicit support for such topologies. For a Cartesian grid periodic or nonperiodic boundary conditions may apply in any specified grid dimension. In MPI, a group either has a Cartesian or graph topology, or no topology. In addition to providing routines for translating between process rank and location in the topology, MPI also:

1. allows knowledge of the application topology to be exploited in order to efficiently assign processes to physical processors,

2. provides a routine for partitioning a Cartesian grid into hyperplane groups by removing a specified set of dimensions,

3. provides support for shifting data along a specified dimension of a Cartesian grid.

By dividing a Cartesian grid into hyperplane groups it is possible to perform collective communication operations within these groups. In particular, if all but one dimension is removed a set of one-dimensional subgroups is formed, and it is possible, for example, to perform a multicast in the corresponding direction.

A set of routines that supports point-to-point communication between pairs of processes forms the core of MPI routines for sending and receiving blocking and nonblocking messages are provided. A blocking

send does not return until it is safe for the application to alter the message buffer on the sending process without corrupting or changing the message sent. A nonblocking send may return while the message buffer on the sending process is still volatile, and it should not be changed until it is guaranteed that this will not corrupt the message. This may be done by either calling a routine that blocks until the message buffer may be safely reused, or by calling a routine that performs a nonblocking check on the message status. A blocking receive suspends execution on the receiving process until the incoming message has been placed in the specified application buffer. A nonblocking receive may return before the message has been received into the specified application buffer, and a subsequent call must be made to ensure that this has occurred before the application uses the data in the message.

In MPI a message may be sent in one of four communication modes, which approximately correspond to the most common protocols used for point-to-point communication. In *ready* mode a message may be sent only if a corresponding receive has been initiated. In *standard* mode a message may be sent regardless of whether a corresponding receive has been initiated. MPI includes a *synchronous* mode which is the same as the standard mode, except that the send operation will not complete until a corresponding receive has been initiated on the destination process. Finally, there is a *buffered* mode. To use buffered mode, the user must first supply a buffer and associate it with a communicator. When a subsequent send is performed using that communicator MPI may use the associated buffer to buffer the message. A buffered send may be performed regardless of whether a corresponding receive has been initiated. In PVM message buffering is provided by the system, but MPI does not mandate that an implementation provide message buffering. Buffered mode provides a way of making MPI buffer messages, and is useful when converting a program from PVM to MPI.

In addition, MPI provides routines that send to one process while receiving from another. Different versions are provided for when the send and receive buffers are distinct, and for when they are the same. The send/receive operation is blocking, so does not return until the send buffer is ready for reuse, and the incoming message has been received.

MPI includes a rich set of collective communication routines that perform coordinated communication among a group of processes. The process group is that associated with the communicator that is passed into the routine. MPI's collective communication routines can be divided into two groups: data movement routines and global computation routines. There are five types of data movement routine: broadcast, scatter, gather, all-gather, and all-to-all. These are illustrated in Fig. 3.

There are two global computation routines in MPI: reduce and scan. The MPI reduction operation is similar in functionality to that provided by PVM. Different versions of the reduction routine are provided depending on whether the results are made available to all processes in the group, just one process, or are scattered cyclicly across the group. The scan routines perform a parallel prefix with respect to a user-specified operation on data distributed across a specified group. If D_i is the data item on the process with rank i, then on completion the output buffer of this process contains the result of combining the values from the processes with rank $0, 1, \ldots, i$, i.e.,

$$\mathcal{D}_i = D_0 \oplus D_1 \oplus D_2 \oplus \cdots \oplus D_i \tag{9}$$

Two versions of the MPI specification exist. One is dated May 5, 1994 (version 1.0), and the other June 12, 1995 (version 1.1). The latter document incorporates corrections and clarifications to the former, but the two do not differ in any substantial way. At the time of writing version 1.1 is only available electronically [30]. The book on using MPI by Gropp, Lusk and Skjellum, who played an active role in MPI's design, gives a good introduction to application programming with MPI [31]. An annotated reference manual based on version 1.1 of MPI is available [41]. A large amount of information about MPI is available via the web, including portable implementations of MPI, information about efforts to extend MPI, and publications related to MPI [1].

7 PDE Solver Package

The PDE Solvers Package, described in [46] is an extension of the PETSc system developed originally by W. Gropp and B. Smith, which already has a wide user base. The PDE Solvers package is mostly developed at UCLA by B. Smith, with contributions from Chan, Ciarlet

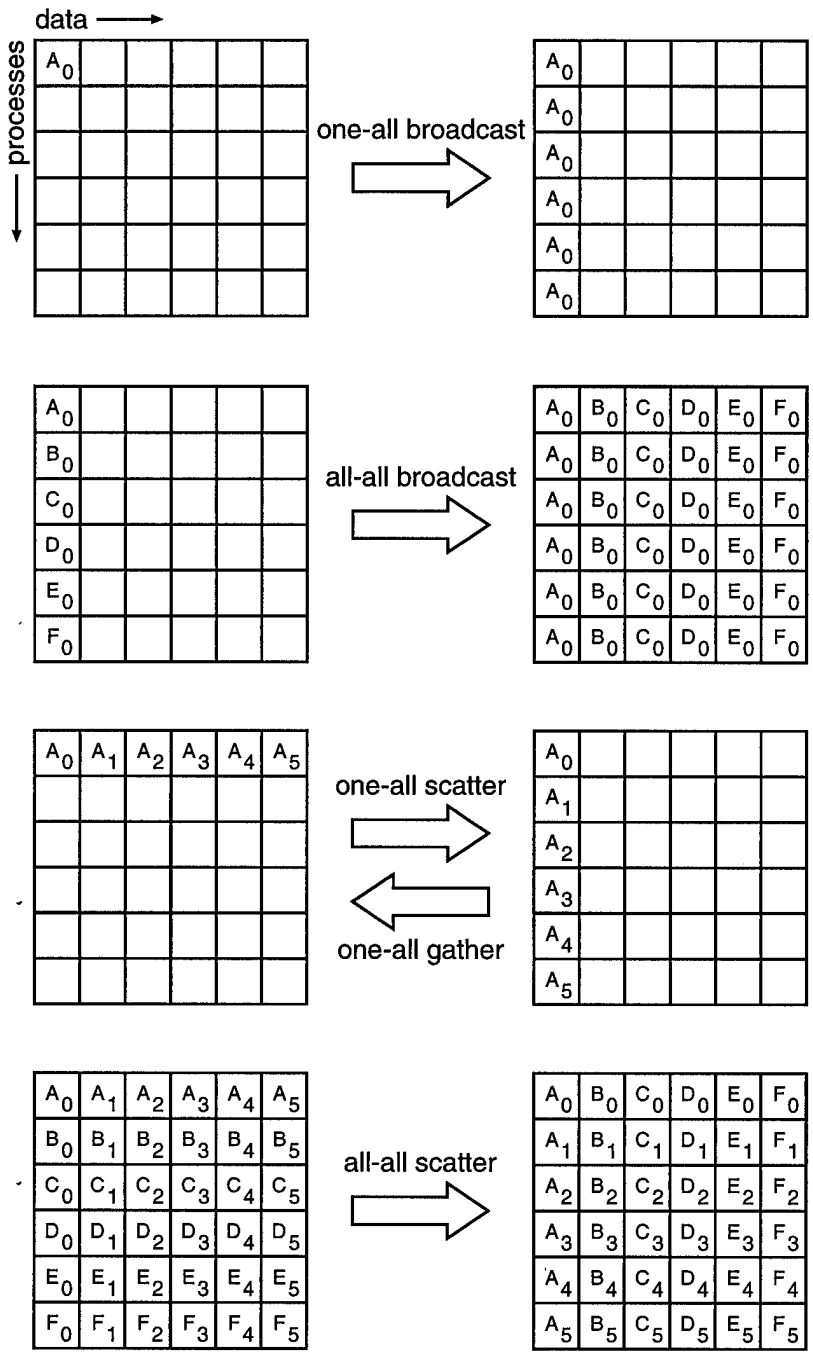

Figure 3: One-all and all-all versions of the broadcast, scatter, and gather routines for a group of six processes. In each case, each row of boxes represents contiguous data locations in one process. Thus, in the one-all broadcast, initially just the first process contains the data A_0, but after the broadcast all processes contain it.

and Lamour. This package provides support for the solution of elliptic PDEs using either finite elements or finite differences in 2 or 3 dimensions on either structured or unstructured grids. Several classical direct and iterative methods, as well as several multigrid and domain decomposition variants, may be used to solve the resulting linear systems. Although the package is fully functional and self-contained, we view it mainly as a prototype to demonstrate and learn how such packages may be organized. Currently, the package has very little explicit support for parallel computing in the package, except for parallel matrix-vector operation using the Chameleon message passing interface. We feel it is important to develop the complete package as a whole rather than just isolate the sparse iterative part because the structure of the discretization matrices can be better exploited in the solution process. We emphasize that the sparse iterative solvers in the PDE Solvers package can be used as "library" routines independent of the PDE discretization routines.

8 New Algorithms

In this section we describe new algorithms developed in the project. In addition, some new algorithms based on the Arnoldi method have already been discussed in 3.1.

8.1 Sparse Direct Methods

Numeric factorization has received most of the attention in previous research on parallel direct methods for sparse linear systems, so our work has focused on the symbolic phase that precedes factorization and the triangular solution phase that follows factorization. Prior to numeric factorization of a sparse matrix, it is essential to reorder the matrix to reduce fill (i.e., the creation of new nonzeros), and thereby reduce the work and storage required. We have developed two new parallel algorithms for ordering sparse matrices for this purpose. One, called Cartesian nested dissection, recursively subdivides the graph of the matrix based on coordinate values of the nodes embedded in Euclidean space. The other ordering algorithm is based on parallel graph contraction, and it is suitable for problems for which geometric coordinates are

not available. In addition to these algorithms for the symbolic part of the computation, we have also developed a new parallel algorithm for the triangular solution phase that follows the factorization. This algorithm is based on selective inversion of small dense submatrices of the sparse triangular factor, and it has vastly improved the performance and scalability of this phase of the computation, which had previously been a substantial bottleneck using conventional subsitution methods for triangular solution.

8.2 A New Algorithm for the Symmetric Eigenproblem

This is joint work between Inderjit Dhillon, partially supported by this grant, Professor Beresford Parlett, supported by ONR, and Vince Fernando, supported by NAG. The basis of the algorithm currently in PDSYEVX is bisection (to compute eigenvalues) followed by inverse iteration (to compute eigenvectors). Bisection costs $O(n^2)$ operations to find all n eigenvalues of an n-by-n symmetric tridiagonal matrix, and can be parallelized to run in $O(n^2/p)$ time on p processors. If the eigenvalues are not close together, inverse iteration can also be parallelized to run in $O(n^2/p)$ time with little or no communication, also perfect speedup. However, if the eigenvalues are tightly clustered, the time can increase to $O(n^3)$, with a great deal of communication. Based on a much deeper mathematical understanding of the problem, we have developed new algorithms which appear to eliminate the need for ever doing more than $O(n^2/p)$ work in an embarrassingly parallel fashion. This is a breakthrough which promises to change the way we solve this important problem on serial machines as well. A preliminary technical report describes this work [42].

8.3 Parallel Sign Function

The sign function is an algorithm for the dense nonsymmetric eigenvalue problem. It is relatively straightforward to parallelize, compared to the conventional sequential algorithm, but does several times as many floating point operations. It is the only scalable algorithm for this problem currently available. We have explored its numerical properties, performance, algorithmic variations, and suitability for mixed parallelism in a sequence of papers [4, 5, 6, 52].

8.4 Iterative Methods

The main goal of this portion of the project has been to develop parallel and scalable algorithms and software for iterative methods for solving sparse linear systems of equations, particularly those that arise in the discretizations of partial differential equations. The methods include standard Krylov subspace methods such as Conjugate Gradients, GMRES, BiCGSTAB etc.; generally applicable preconditioners such as point and block relaxation methods, incomplete factorization methods; and PDE-specific preconditioners such as multigrid and domain decomposition methods. Our studies involved not only a careful study of existing methods but also the development of new methods, especially considering that iterative methods are generally not as fully developed as direct methods. Parallel implementation considerations also motivated us to study data and task partitioning algorithms and sparse distributed data structures for sparse matrices and vectors.

In the area of iterative methods work has focused on the study of scalable iterative methods:

1. *Domain decomposition and multigrid algorithms*: see [9, 10, 16, 11, 12, 13, 15]. Our main contribution is to extend the standard algorithms to handle unstructured meshes., which are becoming increasingly popular, and to study their parallel implementation issues. In [13] we prove a somewhat surprising result that the part of the boundary with Neumann boundary conditions (e.g. outflow) must be covered by the coarse meshes in order to retain the usual optimal convergence rate of MG and DD.

2. *Krylov subspace methods and their parallel implementations*: For a state-of-the-art survey see [50].

3. *Incomplete factorization preconditioners*: For a state-of-the-art survey see [14].

4. *Partitioning algorithms*: In the parallel implementation of sparse iterative methods, the data (i.e. matrices and vectors or grid functions) must be partitioned to achieve load balance and minimize communication [19, 18]. We have developed a very fast (compared to spectral partitioning methods) greedy partitioning algorithm which empirically does not seem to lose very much in

terms of the quality of the partition and the size of the cut set. This method, and others, have been incorporated into the PDE Solvers package.

9 List Of Publications And Technical Reports

1. *Software Libraries for Linear Algebra Computations on High Performance Computers*, J. Dongarra and D. Walker, SIAM Review, Vol. 37, No. 2, June 1995.

2. *Performance Complexity of LU Factorization with Efficient Pipelining and Overlap on a Multiprocessor*, J. Dongarra, F. Desprez, and B. Tourancheau, Parallel Processing Letters, Vol. 5, No. 2, 1995.

3. *PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines*, J. Choi, J. Dongarra, and D. Walker,
   To appear Concurrency: Practice and Experience.

4. *The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines*, J. Choi, J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, Technical report, University of Tennessee, CS-94-246, LAPACK Working Note 80, September 1994.
   To appear in Scientific Programming, 1995.

5. *The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form*, J. Choi, J. Dongarra, and D. Walker, Numerical Algorithms, Vol. 10, No. 3 and 4, pp. 379–400, 1995.

6. *A Highly Parallel Algorithm for the Reduction of a Nonsymmetric Matrix to Block Upper-Hessenberg Form*, M. Berry, J. Dongarra, and Y. Kim, Parallel Computing, Vol 21, No.8, pp. 1189–1212, August, 1995.

7. *Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers*, J. Choi, J. Dongarra, and D. Walker, Parallel Computing, pp. 1387–1405, Vol. 21, 1995.

8. *The Spectral Decomposition of Nonsymmetric Matrices on Distributed Memory Computers*, Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, and K. Stanley, Technical Report, University of Tennessee, CS-95-273, LAPACK Working Note 91, January 1995.
To appear.

9. *Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems*, Z. Bai, J. Demmel, and M. Gu.
To appear in *Numerische Mathematik*.

10. *Sparse matrix reordering schemes for browsing hypertext*, M. Berry, B. Hendrickson, and P. Raghavan, *Lec. Appl. Math., Amer. Math. Soc.*, 1995.
To appear.

11. *A Cartesian nested dissection algorithm*, M. T. Heath and P. Raghavan, *SIAM J. Matrix Anal. Appl.*, 16(1):235–253, 1995.

12. *Distributed sparse Gaussian elimination and orthogonal factorization*, P. Raghavan, *SIAM J. Sci. Comput.*, 16(6):1462–1477, 1995.

13. *Efficient parallel triangular solution with selective inversion*, P. Raghavan, Technical Report CS-95-314, University of Tennessee, Dec 1995.
Submitted to SIAM J. Sci. Comput.

14. *Parallel ordering using edge contraction*, P. Raghavan, Technical Report CS-95-293, University of Tennessee, May 1995.
Submitted to Parallel Computing.

15. *Constructing Numerical Software Libraries for High-Performance Computing Environments*, J. Choi, J. Dongarra, R. Pozo, and D. Walker, Workshop on Parallel Scientific Computing, Lyngby, Denmark, Lecture Notes in Computer Science Number 879, Springer-Verlag, pp 147–168, 1995.

16. Workshop on Environments and Tools for Parallel Scientific Computing, SIAM Publications, Editors J. Dongarra and B. Tourancheau, The Design of a Parallel, Dense Linear Algebra Software Library:

Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form, J. Choi, J. Dongarra, and D. Walker.

17. ScaLAPACK: A Dense, Linear Algebra Library for Message-Passing Computers, J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and C. Whaley, Manchester Linear Algebra and Applications Conference, July 10-12, 1995.

18. *The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers*, J. Demmel and K. Stanley, In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing.* SIAM Press, 1995.

19. *Modeling the benefits of mixed data and task parallelism*, K. Yelick and S. Chakrabarti, In *Proceedings of the 1995 Symposium on Parallel Algorithms and Architectures*, 1995.

10 Scientific Personnel Supported By This Project and Degrees Awarded

- Robert Block, Graduate Student, University of Illinois at Urbana-Champaign, MS 1995.
- Tony Chan, Professor, University of California at Los Angeles.
- Chee-Whye Chin, Undergraduate Student, University of California at Berkeley.
- Jaeyoung Choi, Research Assistant Professor, University of Tennessee at Knoxville.
- Patrick Ciarlet, Post-doc visitor, University of California at Los Angeles.
- Andrew Cleary, Research Assistant Professor, University of Tennessee at Knoxville.
- James Demmel, Professor, University of California at Berkeley.

- Jack Dongarra, Distinguished Professor at University of Tennessee, Knoxville, and Distinguished Scientist at Oak Ridge National Laboratory.
- Inderjit Dhillon, Graduate Student, University of California at Berkeley.
- Victor Eijkhout, Research Assistant, University of California at Los Angeles.
- Michael Heath, Professor, University of Illinois at Urbana-Champaign.
- Greg Henry, Research Assistant Professor, University of Tennessee at Knoxville.
- Dan Yu Hu, Post-doc, Rice University.
- Ajay Kalhan, Graduate Student, University of Tennessee at Knoxville.
- Chik Tung Dominic Lam, Undergraduate Student, University of California at Berkeley.
- Francois Lamour, Post-doc visitor, University of California at Los Angeles.
-
- Richard Lehoucq, Graduate Student, Rice University, PhD 1995.
- Xiaoye Li, Graduate Student, University of California at Berkeley.
- Robert Manchek, Graduate Student, University of Tennessee, MS 1994.
- Susan Ostrouchov, Research Associate, University of Tennessee at Knoxville.

- Antoine Petitet, Graduate Student, University of Tennessee at Knoxville.
- Chris Puscasiu, Undergraduate Student, University of California at Berkeley.
- Padma Raghavan, Assistant Professor, University of Tennessee at Knoxville.
- Huan Ren, Graduate Student, University of California at Berkeley.
- R. Seccomb, Graduate student, University of Tennessee at Knoxville.
- Danny Sorensen, Professor, Rice University.
- Ken Stanley, Graduate student, University of California at Berkeley.
- Ted Szeto, Assistant Professor, University of California at Los Angeles.
- JinqChong Teo, Undergraduate Student, University of California at Berkeley.
- David Walker, Oak Ridge National Laboratory
- Clint Whaley, Graduate Student, University of Tennessee at Knoxville, MS 1994.
- Scotti Whitmire, Graduate Student, University of Tennessee at Knoxville.
- Jun Zou, Assistant Professor, University of California at Los Angeles.

11 Report Of Inventions (by title only):

None.

References

[1] The URL of a good MPI web page is `http://www.mcs.anl.gov/mpi/`. This has links to other extensive MPI pages at Mississippi State Engineering Center and Oak Ridge National Laboratory.

[2] L. M. Alvarez-Cohen and P. L. McCarty. A cometabolic biotransformation model for halogenated aliphatic compounds exhibiting product toxicity. *Environmental Science and Technology*, 25(8):1381–1387, 1991.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide.* SIAM, Philadelphia, PA, 1992.

[4] Z. Bai and J. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, part i. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing.* SIAM Press, 1993.

[5] Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, , and K. Stanley. The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers. To appear.

[6] Z. Bai, J. Demmel, and M. Gu. Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems. To appear in *Numerische Mathematik.*

[7] R. Barrett, Michael Berry, Tony F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems.* SIAM, Philadelphia, 1994.

[8] M. Berry, B. Hendrickson, and P. Raghavan. Sparse matrix reordering schemes for browsing hypertext. *Lec. Appl. Math., Amer. Math. Soc.*, 1995. To appear.

[9] T. F. Chan and T. Mathew. *Acta Numerica*, chapter Domain Decomposition Algorithms, pages 61–143. Cambridge University Press, 1995.

[10] T. F. Chan and J.-P. Shao. Optimal coarse grid size in domain decomposition. *J. Comp. Math.*, 12(4):291–297, 1994.

[11] T. F. Chan and J.-P. Shao. Parallel complexity of domain decomposition methods and optimal coarse grid size. *Parallel Computing*, 1994.

[12] T. F. Chan, B. Smith, and J. Zou. Multigrid and domain decomposition methods for unstructured meshes. In *Proceedings of the Third International Conference on Numerical Methods and Applications*, 1994.

[13] T. F. Chan, B. Smith, and J. Zou. Overlapping schwarz methods on unstructured meshes using non-matching coarse grids. Technical Report 94-8, Department of Mathematics, University of California, Los Angles, 1994.

[14] T. F. Chan and H. A. van der Vorst. Approximate and incomplete factorizations. Technical Report 94-27, Department of Mathematics, University of California, Los Angles, 1994.

[15] T. F. Chan and J. Zou. Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes. Technical Report 93-4, Department of Mathematics, University of California, Los Angles, 1993.

[16] T. F. Chan and J. Zou. Domain decomposition algorithms for nonsymmetric parabolic problems on unstructured meshes. Technical Report 94-22, Department of Mathematics, University of California, Los Angles, 1994.

[17] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. A proposal for a set of parallel basic linear algebra subprograms. Computer Science Dept. Technical Report CS-95-292, University of Tennessee, Knoxville, TN, May 1995. (LAPACK Working Note 100).

[18] P. Ciarlet and F. Lamour. An efficient low cost greedy graph partitioning heuristic. Technical Report 94-1, Department of Mathematics, University of California, Los Angles, 1994.

[19] P. Ciarlet and F. Lamour. On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint. Technical Report 94-37, Department of Mathematics, University of California, Los Angles, 1994.

[20] J. Demmel and K. Stanley. The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing.* SIAM Press, 1995.

[21] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, 1990.

[22] J. Dongarra and R. C. Whaley. A user's guide to the blacs v1.0. Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, TN, March 1995. (LAPACK Working Note 94).

[23] J. J. Dongarra. LAPACK Working Note 34: Workshop on the BLACS. Computer Science Dept. Technical Report CS-91-134, University of Tennessee, Knoxville, TN, May 1991. (LAPACK Working Note #34).

[24] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of Fortran basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.

[25] J. J. Dongarra and R. A. van de Geijn. Two-dimensional basic linear algebra communication subprograms. Computer Science Dept. Technical Report CS-91-138, University of Tennessee, Knoxville, TN, October 1991. (LAPACK Working Note 37).

[26] J. J. Dongarra, R. A. van de Geijn, and D. W. Walker. Scalability issues affecting the design of dense linear algebra library. *Journal of Parallel and Distributed Computing*, 1994. Accepted for publication.

[27] J. J. Dongarra and D. W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 1994. Accepted for publication.

[28] B. Dykaar. Macroscopic groundwater flow and transport coefficients. Ph. D. Thesis Proposal, Stanford University, 1993.

[29] W. S. Edwards, L. S. Tuckerman, R. A. Friesner, and D. C. Sorensen. Krylov methods for the incompressible navier-stokes equations. *Journal of Computational Physics*, 1993. To appear.

[30] The MPI forum. MPI: A message passing interface standard (version 1.1). Available electronically from `http://www.mcs.anl.gov/mpi/`.

[31] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI.* The MIT Press, 1994.

[32] E. F. Hayes, P. H. Pendergast, Z. Darakjian, and D. C. Sorensen. Scalable algorithms for three-dimensional reactive scattering: evaluation of a new algorithm for obtaining surface functions, 1993. Submitted to the Journal of Computational Physics.

[33] M. T. Heath and P. Raghavan. Distributed solution of sparse symmetric positive definite systems. In *Scalable Parallel Libraries Conf.*, pages 114–122, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[34] M. T. Heath and P. Raghavan. Performance of a fully parallel sparse solver. In *Scalable High Performance Computing Conf.*, pages 334–341, Los Alamitos, CA, 1994. IEEE Computer Society Press. Full version submitted to International Journal of Supercomputer Applications.

[35] M. T. Heath and P. Raghavan. A Cartesian nested dissection algorithm. *SIAM J. Matrix Anal. Appl.*, 16(1):235–253, 1995.

[36] J. Jaffre and J.-L. Vaudescal. Arnoldi's method for two-group neutron diffusion. In *Proceedings of the International Conference on Mathematical Methods and Supercomputing in Nuclear Applications*, pages 19–23, 1993.

[37] M. N. Kooper, H. A. van der Vorst, S. Poedts, and J. P. Goedbloed. Application of the implicitly updated arnoldi method with a complex shift and invert strategy in mhd. Technical report, Institute for Plasmaphysics, FOM Rijnhuizen, Nieuwegein, The Netherlands, 1993.

[38] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5:308–323, 1979.

[39] Tsung L. Li and Kelin J. Kuhn. Finite element solution to quantum wells by irreducible formulations. Technical report, University of Washington, Seattle, 1993.

[40] M. A. Nasir, W. C. Chew, P. Raghavan, and M. T. Heath. A comparison of computational complexities of hfem and abc based finite element methods. In *Proc. IEEE APS Internat. Symp.*, pages 447–450, June 1994. Full version submitted to IEEE Trans. on Antennas and Propagation.

[41] S. W. Otto, M. Snir, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra. *MPI: The Complete Reference.* The MIT Press, 1995. Should be available by the end of 1995.

[42] B. Parlett and I. Dhillon. Fernando's method for finding the most redundant equation in a tridiagonal system. Department of mathematics, cpam report 635, University of California at Berkeley, 1995.

[43] P. Raghavan. Distributed sparse Gaussian elimination and orthogonal factorization. *SIAM J. Sci. Comput.*, 16(6):1462–1477, 1995.

[44] P. Raghavan. Efficient parallel triangular solution with selective inversion. Technical Report CS-95-314, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, Dec 1995. Submitted to SIAM J. Sci. Comput.

[45] P. Raghavan. Parallel ordering using edge contraction. Technical Report CS-95-293, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, May 1995. Submitted to Parallel Computing.

[46] B. Smith. Extensible pde solvers package users manual. Technical report, Department of Mathematics, University of California, Los Angeles, 1995.

[47] H. A. Smith, D. C. Sorensen, and R. K. Singh. A lanczos-based eigensolution technique for exact vibration analysis. *International Journal for Numerical Methods in Engineering*, 36:1987–2000, 1993.

[48] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Numerical Analysis (Series B)*, 28:1752–1775, 1992.

[49] D. C. Sorensen, Z. A. Tomasic, and P. A. Vu. Algorithms and software for large scale eigenproblems on high performance computers. In Adrian Tentner, editor, *Proceedings of High Performance Computing 93*, pages 149–154. Society for Computer Simulation, 1993.

[50] H. A. van der Vorst and T. F. Chan. Linear system solvers: sparse iterative methods. Technical Report 94-28, Department of Mathematics, University of California, Los Angles, 1994.

[51] J.-L. Vaudescal, 1993. Private communication.

[52] K. Yelick and S. Chakrabarti. Modeling the benefits of mixed data and task parallelism. In *Proceedings of the 1995 Symposium on Parallel Algorithms and Architectures*, 1995.