LBNL-45446

A Survey of Packages for Large Linear Systems

Kesheng Wu and Brent Milne

Lawrence Berkeley National Laboratory/NERSC, Berkeley, CA 94720.

February 11, 2000

1 Executive Summary

This paper evaluates portable software packages for the iterative solution of very large sparse linear systems on parallel architectures. While we cannot hope to tell individual users which package will best suit their needs, we do hope that our systematic evaluation provides essential unbiased information about the packages and the evaluation process may serve as an example on how to evaluate these packages. The information contained here include feature comparisons, usability evaluations and performance characterizations. This review is primarily focused on self-contained packages that can be easily integrated into an existing program and are capable of computing solutions to very large sparse linear systems of equations. More specifically, it concentrates on portable parallel linear system solution packages that provide iterative solution schemes and related preconditioning schemes because iterative methods are more frequently used than competing schemes such as direct methods. The eight packages evaluated are: Aztec, BlockSolve, ISIS++, LINSOL, P_SPARSLIB, PARASOL, PETSc, and PINEAPL.

Among the eight portable parallel iterative linear system solvers reviewed, we recommend PETSc and Aztec for most application programmers because they have well designed user interface, extensive documentation and very responsive user support. Both PETSc and Aztec are written in the C language and are callable from Fortran. For those users interested in using Fortran 90, PARASOL is a good alternative. ISIS++ is a good alternative for those prefer the C++ language. Both PARASOL and ISIS++ are relatively new and are continuously evolving. Thus their user interface may change. In general, those packages written in Fortran 77 are more cumbersome to use because the user may need to directly deal with a number of arrays of varying sizes. Languages like C++ and Fortran 90 offer more convenient data encapsulation mechanisms which make it easier to implement a clean and intuitive user interface.

In addition to reviewing these portable parallel iterative solver packages, we also provide a more cursory assessment of a range of related packages, from specialized parallel preconditioners to direct methods for sparse linear systems.

2 Introduction

Solving a linear system of equations is one of the common tasks performed in many scientific and engineering computations. There are a large number of software packages available on many different platforms for this task. These software packages provide varying functionalities and information regarding them is scattered in many locations. This makes it difficult for a prospective user to make an educated choice. There are a number of software archives and listings that are devoted to scientific software or numerical linear algebra software, e.g., NETLIB[1], ACM TOMS archive[2], GAMS[3]. Typically, software packages are listed without quality judgment or usability information. One important exception to this generalization is the National High-performance Software Exchange (NHSE)[4] which provides review of many scientific software packages. One extensive review of available software for solving linear systems was published in 1998 by Victor Eijkhout [15]. Since then, a number of new software packages have become available; reviewed packages have also been updated. In particular, some new software using C++ and advanced user interface design practices have the potential to revolutionize the way linear system solvers are used. Eijkhout's review [15] covers 15 different software but does not provide in-depth assessment. In this survey, we will follow a more rigorous user-interface evaluation procedure [37, 36] and provide more comprehensive review based on the up-to-date information about the linear system solver packages. We will concentrate on a subset of the available packages and perform a more in-depth analysis and evaluation. The evaluation includes critical information on usability, interface, features, and performance characteristics of prominent iterative linear system solvers for distributed parallel machines. This should reduce the need for users to conduct their own evaluations and make it easier to select the right packages for their application programs. We also hope that it will serve to correct some prominent misconceptions related to performance and usability.

In Section 3, we describe the scope of the comparisons and the methodology used for comparisons. The basic information about the eight reviewed packages are given in Section 4. Section 5 contains the comparison of functionalities. Section 6 is devoted to assess the usability which is subjective in nature. To evaluate the

[1] NETLIB URL is `http://www.netlib.org`.
[2] ACM TOMS archive is located at `http://www.toms.acm.org`.
[3] GAMS archive on the web `http://gams.nist.gov/`.
[4] NSHE URL is `http://www.nhse.org`.

performance of these software packages, we conduct a small number of tests using a selected set of large test problems. The performance evaluation is not extensive, but the information provided should be instructive. This limited performance evaluation is recorded in Section 7. Section 8 contains a short summary of the comparisons and our overall impression on the eight reviewed packages. Some information on related software packages are given in the appendix.

3 Scope

There are a number of software packages that are primarily devoted to handling large matrices and finding solutions to linear systems of equations. This review concentrates on those portable stand-alone packages that are designed for solving very large linear systems where the coefficient matrices are too large to be stored as simple two-dimensional arrays. In most cases, either only the nonzero entries of these matrices are stored in some sparse matrix format or the matrices are not stored at all. Even without storing the extra zero entries or without storing the matrix at all, the application program still may not fit into the main memory of an average desktop computer. These programs usually run on distributed-memory parallel machines. The typical method used to solve these linear systems is a Krylov subspace based method [7, 44]. Using one of these methods, the coefficient matrix is primarily needed to perform matrix-vector multiplications, and the user can store the matrix in any format that is convenient for the application. Furthermore, these methods require only a small amount of computer memory in addition to the storage required for the linear system. Direct methods [14, 18] based on Gaussian Elimination typically require much more computer memory, therefore may not be a viable option in this case. For this reason, our evaluation concentrates on iterative methods. In a typical application, a preconditioner is often needed to reduce the time to compute the solution [7, 44]. In the following comparisons, we will only consider those packages that contain all basic components of a self-contained iterative solvers, are reasonably easy to portable to different type of distributed computing environment. More specifically, we will evaluate sparse iterative solver packages that

- contain their own
  - solvers,

Table 1: The web addresses for the eight parallel iterative packages.

Aztec	http://www.cs.sandia.gov/CRF/aztec1.html
BlockSolve	http://www-unix.mcs.anl.gov/sumaa3d/BlockSolve
ISIS++	http://z.ca.sandia.gov/isis
LINSOL	http://www.rz.uni-karlsruhe.de/Uni/RZ/Forschung/Numerik/linsol
P_SPARSLIB	http://www.cs.umn.edu/Research/arpa/p_sparslib
PARASOL	http://www.genias.de/projects/parasol
PETSc	http://www.mcs.anl.gov/petsc
PINEAPL	http://www.nag.com/projects/PINEAPL

  - preconditioners,
  - matrix-vector multiplication;
- run on distributed memory parallel machines;
- are portable;
- are easy to obtain either the source code or the library;
- contains no prohibitive licensing;
- are active supported;
- can be used independently – not a component tightly integrated into an application program;
- and can potentially be interchanged with one another.

Based on these criteria, we have searched all the software archives that are known to us and have also requested for information from experts in the field of numerical linear algebra through NA digest[5]. The software package found are listed in Table 1.

While we will evaluate all of these packages, we will not be able to evaluate them uniformly because of constraints such as software maturity, availability, and stability. Even though the developers promised to distribute LINSOL to the research user for free, however at the time of this writing, we are unable to obtain it from the developers. BlockSolve fails to run a simple test program we developed due to an apparent bug in the package and we are unable to get satisfactory responses from the developers (funding for its

[5] Information about NA net and NA digest can be found at `http://www.netlib.org/na-digest-html`.

development has stopped). PINEAPL was not able to run correctly on one of our main testing platform, a Cray T3E, because of floating-point arithmetic problems. We thus focus much of the discussion on Aztec, Isis++, PARASOL, P_SPARSLIB, and PETSc.

The evaluation of the software will be divided into three categories, functionality, usability and performance. The information for functionality is based on our survey of the documentation and the implementation. The comparison among the packages will include the following aspects,

- breadth of iterative methods provided,
- breadth of preconditioning options,
- input data formats and support functions,
- sophistication of the methods implemented,
- quality of implementation, e.g., error-checking/handling and extensibility.

For usability, our evaluation is based on commonly used criteria. The information provided here is primarily based our own experiences of using these packages and on the user comments collected. The specific items checked are

- clarity of documentation,
- user interface design and consistency,
- helpfulness of the feedback from the software,
- ease of use,
- ease of learning to use,
- interoperability with other packages,
- licensing and developer support.

We will provide some performance information about Aztec and PETSc. We have chosen to only study the performance of these two packages because they are the most commonly used packages from our user survey.

Since there are a large number of different features implemented by these packages, attempting to perform an exhaustive study will take too long and make this report less timely and less useful. The performance data are collected from "real world" test problems, see Table 7 for brief descriptions. These test matrices have large condition numbers and are difficult for most iterative methods. Only a simple set of tests are performed on these test problems. These tests apply the GMRES method without preconditioning for a fixed number of iterations. We report on the time used to set up the iterative methods and the time spent during the iterations. The iteration time primarily reflect the speed of the parallel sparse matrix-vector multiplication routines of the packages. Since the matrix-vector multiplication is a very well-defined operation, the difference in the speed can be taken as indirect evidence of how well the package is implemented. Everything else being equal, the package with faster matrix-vector multiplication package can be thought as a better package. The setup time is also an indicator of the quality of the software packages. The setup stage should be faster and should require minimal amount of workspace.

4 Software packages

Given these stated criteria, we searched the internet and literature for software packages that meet them. In addition to our own search, we have also broadcasted a request for information through the NA digest. Through these searches we are able locate the following ten software packages:

- **Aztec** This package is written in C. The developers are a group of scientists from Sandia National Laboratories, Ray S. Tuminaro, John N. Shadid, Scott A. Hutchinson, Lydie Prevost, and Charles H. Tong [26]. The software is available from the developer's web site at `http://www.cs.sandia.gov/-CRF/aztec1.html`.

- **BlockSolve** This package is also written in C. The developers are Mark T. Jones and Paul E. Plassmann [27]. This software is not being actively updated but the iterative methods and preconditioners implemented are reasonable current. The latest version is BlockSolve95 which is available from `http://www-unix.mcs.anl.gov/sumaa3d/BlockSolve`.

- **ISIS++** This package is developed by Benjamin Allan, Robert Clay, Kyran Mish and Alan Williams from Sandia National Laboratories [2]. It is a relative new development project using C++ and one of its stated goal is to provide linear system solution service as a simple interchangeable component. The software is available at `http://z.ca.sandia.gov/isis`.

- **LINSOL** This parallel interactive solver package is by H. Häfner, W. Schönauer and R. Weiss of Universität Karlsruhe [21, 22]. The code is available free to scientific researchers. It contains many variants of iterative solvers that are not commonly available in other packages. In addition, it also contains a set of polyalgorithms that might reduce the difficult of selecting an appropriate iterative method.

- **P_SPARSLIB** This package is mostly written in Fortran 77. Some C examples are also provided with the latest distribution. The main contributors include Yousef Saad, Andrei V. Malevsky, G. C. Lo, Sergey Kuznetsov, Masha Sosonkina, Irene Moulitsa. Other contributors to the software package are indicated in the source code of the software. The latest version of the software is available at `http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html`.

- **PARASOL** PARASOL is developed by a collaborative effort of a number of European research organizations and their industrial partners. It includes not only iterative methods for linear system solutions but also direct methods. The implementation is in Fortran 90 with MPI for interprocessor communication. Some documents about the software are available, for example, an overview document [5] and the user interface specification [19], but not all aspects of the software are documented at this moment.

- **PETSc** This software is written in C but follows the basic features of Object-Oriented design. The developers are Satish Balay, William Gropp, Lois Curfman McInnes, and Barry Smith, from Argonne National Laboratory [8]. It is one of the first public available parallel iterative solver packages adopting the MPI standard and it has built up a large user group. The latest version of the software is available at `http://www.mcs.anl.gov/petsc`.

- **PINEAPL** This parallel linear algebra package is developed by NAG (Numerical Algorithms Group) [35]. The implementation language is Fortran and inter-processor data communications are performed through BLACS. This package contains an extensive set of iterative methods and preconditioners for large sparse linear systems.

5 Functionality

The majority of the successful iterative methods in use today is some form of the Krylov subspace method. A Krylov subspace method can function with two basic components, an accelerator, also called an iterative method, and a matrix-vector multiplication routine. But in most cases, a robust preconditioner is necessary to successfully compute the solution of most real-world problems. Based on our survey of the documentation and the source code, our comparison of the functionality focuses on three main categories, the iterative methods, the preconditioners, and the date structure for representing the linear systems. Comments on quality of implementation, error-checking, robustness and extensibility are given near the end of this section.

5.1 Iterative methods

This subsection lists the different iterative methods implemented by the eight packages, see Table 2. The details of the algorithms listed here are described in most standard textbooks [7, 20, 44]. Table 2 has three parts. The first part lists which ones of the seven most commonly used iterative methods are implemented by the eight packages, the second part lists whether any poly-algorithms are implemented, and the third part lists other iterative methods implemented. In the first part, the seven methods are displayed as row headings and the eight packages are displayed as column headings, when the intersection of a row and a column is marked with 'Y', the corresponding package implements the iterative method. If the intersection is not marked, the package does not implement the named iterative method. The second part of the table indicate where the package also supply *intelligent* polyalgorithms that are able to automatically switch to a different method when the current one stagnates. The third part of the table is the last row of the table that names the iterative methods outside of the seven methods in the top part of the table.

Table 2: Iterative methods provided by different packages.

	Aztec	BlockSolve	ISIS++	LINSOL	P_SPARSLIB	PARASOL	PETSc	PINEAPL
CG	Y	Y	Y	Y	Y		Y	Y
SYMMLQ		Y						Y
BiCGSTAB	Y		Y	Y	Y		Y	Y
GMRES	Y	Y	Y	Y	Y		Y	Y
TFQMR	Y				Y		Y	
BiCG					Y			
QMR			Y	Y	Y			
poly-algorithm					Y			Y
other	CGS		FGMRES, DefGMRES, CGS, CGNR, CGNE[2]	ORTHOMIN, Fridman, ORTHORES, GMERR, BICO, CGS, CGNE, CGNE-ATPRES [46]	DBCG [41], DQGMRES [39], FGMRES [43], FOM [44]	direct methods [6], FETI, DD	CGS, CR, LSQR	CGS

Among the seven iterative methods, the first two are for symmetric linear systems, the rest are for nonsymmetric linear systems. In most cases, when CG is mentioned, the preconditioned version (PCG) is implemented [7, 20]. The only exception is P_SPARSLIB [38] which implements the standard CG algorithm. The CG algorithm is designed to solve symmetric positive definite linear systems. When a preconditioner is applied from one side, the preconditioned linear system is very likely to become nonsymmetric. To keep the preconditioned linear symmetric positive definite, the PCG algorithm is designed to implicitly split the preconditioner. When using CG from P_SPARSLIB, the user must split the preconditioner explicitly and apply preconditioning from both left and right. When using other packages, the user specifies one preconditioner. In order for PCG to function properly, the preconditioner has to be positive definite. When the symmetric linear system or the preconditioned linear system is indefinite, SYMMLQ should be used. A method for nonsymmetric system might also be effective in this case.

The first three of the five nonsymmetric methods only require the matrix A to be used in the operation Ax, but the last two require both the operations of Ax and $A^T x$. Often providing the function to perform $A^T x$ requires significant amount of work, therefore, the last two methods are not as popular as the first three.

At every iteration of the GMRES algorithm, it expands the size of the Krylov subspace by one. All the packages that implement GMRES, implement a restarted version, i.e., when the Krylov subspace reaches a specified size, the algorithm is restarted with the latest solution as the starting solution. On many problems, the behaviors of restarted GMRES is quite different from that of TFQMR or BiCGSTAB [47]. Since it is difficult to predict which one of them will work the best, most packages offer at least GMRES and one of TFQMR or BiCGSTAB.

When solving a difficult linear system, an iterative method may converge very slowly. This phenomenon is known as stagnation. For example, GMRES is often very effective in reducing the residual norm during the first few restarts, however, it may experience stagnation later. It has been observed that when one method stagnates a different method often can reduce the residual further [22, 47]. This suggests that one might be able to design some intelligent way of switching among available iterative methods in order solve a difficult linear system. This kind of polyalgorithm is also helpful for novice users since the user is not required to select a method from the many choices presented.

Among the special methods provided by different packages, the most popular one is CGS. Since it may have erratic convergence behavior, BiCGSTAB or TFQMR should be used in its place. Other iterative methods with notable features are FGMRES and DQGMRES which are variants of GMRES. They allow the user to change preconditioner after every iteration. This feature may be useful, for example, it allows one to use another Krylov subspace method as preconditioner. Varying the preconditioner may also help to alleviate the stagnation problem in some cases. DQGMRES is a truncated variant of the GMRES algorithm, it has some different features from the restarted GMRES and may be used as an alternative to the commonly used restarted version. DefGMRES implemented in ISIS++ is another variant of GMRES. It is a restarted GMRES that explicitly deflates out the eigenvector components corresponding to eigenvalues close to zero. When GMRES is used with deflation, it often achieves much better performance than without deflation by avoiding stagnation [12]. The authors of LINSOL are experts in developing new iterative algorithms and their package contains a number of special algorithms of their invention. These methods can be interesting alternatives for those who are interested in trying out new methods.

PARASOL does not implement any of the common iterative methods yet. The two main iterative

Table 3: Preconditioners available from parallel iterative solver packages.

	Aztec	BlockSolve	ISIS++	LINSOL	P_SPARSLIB	PARASOL	PETSc	PINEAPL
diag	Y	Y	Y		Y		Y	
SOR		Y			Y			Y
ILU/IC		Y	Y	Y				
B-Jacobi	Y	Y	Y				Y	
ODD	Y				Y		Y	
AI			Y					
other	Neumann and least-squares polynomials	coloring and clique finding available	Neumann and least squares polynomials up to order 10 [2]		SOR implemented with coloring schemes, Schur complement	use domain decomposition scheme as solver	Richardson, Chebyshev, IC and ILU on 1 PE, can also use BLOCKSOLVE	coloring options available

schemes currently available are domain decomposition method[6] and FETI. These methods are intended to solve linear systems from discretizing PDE problems where information about the physical domain is available. The multifrontal sparse direct method [4, 3, 6] from PARASOL is relatively stable and has been extensively used be other components of PARASOL.

5.2 Preconditioning

The preconditioning provided by the eight packages are listed in Table 3. The lay-out of this table is close to the previous one. The top half of the table shows which package implements which of the six common parallel preconditioners. The last row of the table shows the other preconditioners the packages also implemented and notes about the preconditioning schemes.

The six parallel preconditioner listed in Table 3 are: diagonal scaling (diag), SOR/SSOR [20], incomplete LU (in symmetric case incomplete Cholesky) factorization [33], block-Jacobi iteration [44], overlapping domain-decomposition (ODD) [44] and approximate inverse (AI) [44]. The diagonal scaling can be performed by itself as a preconditioner, but more frequently, it is combined with another one. The SOR scheme listed in this table refers to the version that is used on sequential machines. Some forms of the multiplicative domain decomposition based preconditioner are regarded as block SOR. These preconditioners are listed

[6] The source code distribution contains document about domain decomposition algorithms in directory `solver/ddm/doc`.

under the title of domain decomposition. Since the SOR scheme offers limited parallelism, it is often used with a multi-coloring scheme to enhance the parallelism. Incomplete factorization schemes are successfully used on many different types of linear systems in the sequential environment. However on distributed machines, they have only limited parallelism and are not as widely used as block-Jacobi preconditioner or the domain-decomposition based preconditioners.

The block-Jacobi preconditioner is one of the most commonly used parallel preconditioners. It is one of the simplest preconditioners available, easy to implement and very inexpensive to use. It has good parallel efficiency and it often reduces the number of iterations needed to compute a solution. Implementations of this scheme differ from each other in how they decide the block size and how they solve the local problems on each processor. In most cases, the block size is chosen so that each processor owns a block of the matrix and the the diagonal blocks are inverted either exactly or approximately. Those packages that supports sparse matrix with dense block entries usually also support a version of Jacobi method that operates on the dense block entries as a whole. For example, Aztec, BlockSolve and ISIS++ support this form of the block-Jacobi preconditioner. Similar choices also exist for the overlapping domain decomposition preconditioners. Additionally, the overlapping domain decomposition has more options such as how to decide the overlap and how to integrate the approximate solutions of the overlapping nodes. For the five packages that are marked to have either block-Jacobi preconditioner or the overlapping domain decomposition preconditioner, here is more information on their implementation.

- **Aztec** implements a set of overlapping domain decomposition schemes based on additive Schwarz iterations. When no overlap is used, this scheme reduces to a block-Jacobi preconditioner. The methods used to solve the local problem include: LU, ILUT, ILU(k), RILU, ICC when the matrix is symmetric and BILU(k) when the input matrix has block entries. The user can control overlapping by specifying the width of the overlapping region and Aztec takes care of the details of forming the preconditioner.

  Aztec also implements a special version of this scheme (AZ_sym_GS) which is a non-overlap additive Schwarz with Gauss-Seidel iteration as local solver. The block Jacobi preconditioner in this package is a version only for the block-entried sparse matrices.

- **BlockSolve** The block-Jacobi preconditioner works on dense diagonal blocks. The input matrix need not be in a block-entried form, BlockSolve will analyze the graph of the sparse matrix to discover cliques in the graph and reorder the matrix so that nodes of a clique are ordered next to each other. This causes dense submatrices to be formed on the diagonal. This form of the block-Jacobi preconditioner is often more effective than the simple Jacobi preconditioner.

- **ISIS++** implements the block Jacobi preconditioner both with overlap and without overlap. The block size can be controlled by the user. The ISIS++ reference guide does not mention how the local problems are solved.

- **P_SPARSLIB** has a number of incomplete LU factorizations for solving the local linear systems, e.g., ILU(0), MILU, ILUT, ILUTP, and Gauss-Seidel iterations. In fact, if FGMRES or DQGMRES is used, the local problems can be also solved by another iterative method such as GMRES. The user is responsible for generating the overlapping decomposition of the matrix. Though a number of helper routines are available from P_SPARSLIB, performing your own overlapping decomposition still takes some amount of programming.

- **PETSc** offers block Jacobi, block Gauss-Seidel and overlapping additive Schwarz preconditioners with a full set of options [8]. By default, ILU(0) is used to solve the local linear systems, the width of the overlapping region is 1, and the additive Schwarz uses full restriction and ignores off-processor values during interpolation. All these options can be modified by the user either on command-line or inside the program.

Approximate inverse is a relatively new type of preconditioning strategy for parallel machines [9, 13, 17, 30]. Explicit forms of the approximate inverse preconditioners can be easily applied as parallel sparse matrix-vector multiplication which can usually be performed faster than triangular solutions required by all incomplete LU preconditioners. Approximate inverse preconditioners can be easily constructed for matrices with zero diagonal elements or singular diagonal submatrices. It is likely that more parallel iterative solver packages will offer this option in the future. At the moment, only ISIS++ offers this type of preconditioners.

Among the other preconditioners implemented by various packages, polynomial preconditioners are the most common ones. Two types of popular polynomials implemented are Neumann series and least-squares polynomials. Typically, relative low order polynomials are used, say 5 – 10. Aztec does not have a preset limit on the order of polynomials but ISIS++ limits the maximum order to be 10.

Another type of preconditioners worth noting is the Schur's complement based scheme implemented in P_SPARSLIB. At the moment, the developers have not put out any document on this feature, but there are examples in the latest P_SPARSLIB distribution (version 3.0).

5.3 Specifying the linear system

How to specify the linear system to the iterative solver packages is the third topic to be discussed in this section. This part of the discussion mostly concentrates on the data structures used to specify the linear system. Closely associated with the external data structure is the internal data structure used by the packages to perform its own operations, e.g., the parallel matrix-vector multiplication and construction of preconditioners. We will not discuss the issues related to performing parallel sparse matrix-vector multiplication or construction of preconditioners. Consult references [11, 40, 45] if you are interested in reading further about how to construct an efficient parallel matrix-vector multiplication. The references given during the discussion of the preconditioners should be a good starting point for learning about preconditioning.

A linear system of equations consists of three entities: the matrix, the right-hand side and the solution vector. Typically, the iterative method uses a starting vector as the initial guess of the solution vector. Altogether, there are three vectors and a matrix. Usually, the vectors are stored as simple one dimensional array. The matrices in this discussion are assumed to be sparse, i.e., the majority of their entries is zero. There are a number of schemes commonly used to store the sparse matrices in sequential environment [10, 42], there is no accepted standard for storing sparse matrices on distributed machines. Table 4 summarizes the data structure used by the eight packages, more information about the data structure and overall schema are provided below.

- **Aztec** uses two data structures for storing the sparse matrices, DMSR and DVBR. They are distributed versions of MSR and VBR. The major difference between the two formats is that DMSR is intended to be used with point entry sparse matrices and DVBR is to be used with block entry sparse matrices.

Table 4: Summary of data structures and overall schema.

	Aztec	BlockSolve	ISIS++	LINSOL	P_SPARSLIB	PARASOL	PETSc	PINEAPL
comm[a]	MPI	MPI	MPI	MPI	MPI	MPI	MPI	BLACS
vector	array	array	class	vector	vector	vector	struct	array
matrix	DMSR, DVBR	distributed CSR, or access internal data structure	put matrix into one of the defined classes		distributed CSR		distributed COO	distributed COO
precond exposed[b]	no	no	yes		yes	no	no	yes[c]
schema	construct matrix data structure, set options, solve	construct matrix, set options, solve	construct vectors, matrix, preconditioner and linear system objects with desired options, solve		construct matrix, process for matrix-vector multiplication, factor local submatrices, enter reverse communication loop to solve	distribute matrix, set options, solve	construct matrix, set options, solve	distribute matrix, construct preconditioner, solve, or use *expert* routine to avoid constructing preconditioner

[a] Underlying communication layer.

[b] Is preconditioner exposed to user? Exposed preconditioner may require to constructed it explicitly.

[c] Even though PARASOL is able to construct preconditioners, it requires the user to allocate enough computer memory for it.

The users are required to put their matrices into one of these formats and while Aztec is using the matrices the users can not modify or deallocate the data structures. The solution process is controlled by modifying an options array. After converting the user data into DMSR or DVBR, all further modification to the matrix are performed by calling Aztec functions.

The sparse matrices are expected to be distributed by the user and each row of the matrix must belong to an unique processor. The vectors are stored as simple one-dimensional arrays and their distribution must conform to the distribution of the matrix, in other words, if row i of the matrix belongs to processor p, the ith element of every vector must be on the same processor. This conformity in data distribution is typically assumed in most packages.

- **BlockSolve** internally uses a compressed format for storing sparse matrices. This internal data structure is a general form compressed row storage format. The user may construct this data structure himself or use a supporting routine to convert from a distributed CSR data format. The conversion routine requires the matrix to be distributed in block row format, in other words, each processor can only have a set of consecutive rows of the global matrix. This is a small restriction since the user can use one of many reordering packages to reorder the sparse matrix so that the required simple partitioning scheme does not harm the parallel performance of the iterative methods [29]. When the conversion function is used, the user's data structure is implicitly used. Even though no further operations of BlockSolve explicitly involves the user data structure, the user's distributed CSR data structure can not be modified or deallocated.

  Once the user data is converted to the required internal format, all the manipulation required to construct preconditioners and perform matrix-vector multiplications are done by BlockSolve functions, so that the user does not need to directly manipulate the internal data. The user controls the operations of the BlockSolve through calls to a set of functions that changes the BlockSolve context. The vectors are expected to be distributed conformally as the matrix.

- **ISIS++** requires the users to put their sparse matrices into one of several distributed sparse matrix classes defined in the package and put vectors into a distributed vector class. A number of functions are provided to input the nonzero entries into the matrices after the classes are initialized. The distribution of the matrix and the vectors are controlled by a map class. After the matrices are constructed the user may construct preconditioners from them. With the matrix, the right-hand side and the initial guess vector, the user can declare a linear equation class which will handle the final solution process. The linear equation class needs an iterative method class and a preconditioner class in order to compute the final solution.

- **LINSOL** allows one to input the matrix in a number of forms, for example, one row at a time, one column at a time, or even an arbitrary submatrix. Presumably, LINSOL takes care of assembling all the pieces together and compute the solution without further requiring the user to manipulate the matrix. The documentation [46] is somewhat vague on the specifics.

- **P_SPARSLIB** employs the reverse communication scheme to interact with the matrix and the preconditioner. This means that the user has complete control over how the matrices and the preconditioners are stored in memory. The distributed matrix-vector multiplication routine of P_SPARSLIB uses a distributed version of CSR format to store matrices. To use this matrix-vector multiplication routine, the user need to distribute their matrix in row-wise fashion and call the preprocessing routines to reorder the local portion of matrix and generate necessary information for the matrix-vector routine. To prepare a preconditioner, the use need to call an incomplete LU factorization routine on the diagonal blocks of the matrix on each processor. The iterative methods return control to the user at every iteration and ask the user to perform matrix-vector multiplications or preconditioning operations.

  The vectors are expected to be distributed conformally as the matrix.

- **PARASOL** has specified a fairly complete set of mechanisms for the user to input the matrix and the related information [19]. Users need to first enable PARASOL and compute a mapping for the linear system. Another important distinction between this package and the others is that the sparse matrices are expected to be in compressed column format.

- **PETSc** allows one to input a sparse matrix in row index, column index and nonzero value triplet format (also known as coordinate format) or simply specify that the user will perform the matrix-vector multiplication (matrix-free). To input the matrix to PETSc, the user need to first initialize a PETSc sparse matrix data structure before specifying the nonzero entries. Once all the nonzero entries are specified, the user can construct preconditioners and compute the solution of a linear system without having to directly manipulate the matrix data structure again. PETSc can distribute the input sparse matrix without user intervention and because of this the user may input all nonzero entries from one single processor. The distributed vectors are generated in a similar manner. In both cases, the user is not explicitly aware of the distribution schemes used internally.

- **PINEAPL** requires the user to store the matrix in a distributed coordinate format and the matrix distribution is specified in block cyclic fashion following BLACS and ScaLAPACK. The corresponding vectors must be distributed conformally. The user may either choose a set of three functions to perform setup, solve and postprocessing, or a single *expert* routine to compute the solution.

  Matrix-free versions of the iterative methods are also provided for those who want more control over the data storage format and preconditioning schemes. The matrix-free versions use reverse communication similar to that used in P_SPARSLIB.

5.4 Quality of implementation

A number of issues related to software quality are discussed here. There are many quality issues, we limit the discussion to the issues related to error handling and extensibility.

The majority of the eight software provide limited feed-back about error conditions by either returning an integer error code or passing back an integer error code as a function argument. When a package is implemented in Fortran or C, this is the primary mechanism for providing feed-back about errors. Even though ISIS++ uses C++, it also returns an error code as its primary error feedback mechanism. BlockSolve also provides a global variable for some error conditions and provides a set of macro functions for checking this variable. PETSc has a much more extensive error checking than the other packages. A program using PETSc can be instructed to start a debugger when encountering an error.

Documentation about error code is incomplete or nonexistent in most cases. PINEAPL is an exception in this case, it does have documentation on every error code of every function. ISIS++ provides explanations about its error code for major functions such as the `solve` function of its `Solver` class. PETSc requires the user to use one of the two macro functions CHKERRA and CHKERRQ to handle error codes returned. If the user fail to use these functions, the program may hang in some cases such as one of the process fail to allocate the required workspace. Aztec and P_SPARSLIB has comments in the source code that explains most of the error code returned. The source code of ISIS++ and PARASOL have very few comments other than the copyright notices.

PETSc was designed to be extensible. There is a mechanism for users to register their own functions such as an iterative solver or a preconditioner. The users who are interested in this feature should make sure to use the latest version of the program and the documentation since the mechanism has changed considerably since earlier releases. Those packages that employ reverse communication, such as P_SPARSLIB and PINEAPL, are also extensible since the user can use any preconditioning or matrix-vector multiplication routine.

6 Usability

We evaluate the usability of the packages by evaluating the documentation, user interface design, learning time, interoperability and support. Each of these categories can be subdivided. Based on our own experiences and the evaluations we have received from the users of the National Energy Research Scientific Center (NERSC `<http://www.nersc.gov>`), we give scores to each package. Table 5 shows the scores we have assigned. The heading of each category is meant to be self-explanatory and footnotes are provided within the table to explain the terms that might not be obvious. We have chosen to use only three values for scores. As we accumulate more experience in using the different linear system solvers, we might refine the scoring system to fine-tune the scores. In most cases, a score of 1 means that the package is weak in the category, 2 means acceptable, and 3 means good. We will next give some general comments about all the packages, specific comments about each individual packages will follow.

- **Documentation** of most packages are reasonably well prepared. In particular, PETSc and PINEAPL have extensive documentation on all aspects of their implementations. LINSOL and P_SPARSLIB lack documentation about the software interface and their existing documents are out of date. The first

Table 5: Usability scores for the eight portable parallel iterative solver packages (1 needs improvement, 2 acceptable, 3 good, no score is given if we can not evaluate).

	Aztec	BlockSolve	ISIS++	LINSOL	P_SPARSLIB	PARASOL	PETSc	PINEAPL
documentation								
clarity	2	3	2	1	2	3	3	3
depth	3	3	2	1	1	2	3	3
breadth	2	3	3	2	1	1	3	3
structure	2	3	3	2	1	2	3	2
learning curve								
examples	2	2	2		1	1	2	2
simple task[a]	2	3	3		1	2	3	2
advanced features[b]	3	3	3		1	2	3	2
user interface								
consistency	2	3	3		2	3	3	3
arg list[c]	1	3	3		1	3	3	1
complexity[d]	2	2	3		1		3	1
feed back[e]	2	2	2		1		3	3
other								
availability	2	3	3		3	3	3	
installation	1	2	2		2		3	
support	3	1	3		3	3	3	1
recommendation								
application	3	1	2		1	2	3	1
research	2	2	2		3	3	3	1

[a] How easy is it to solve a simple linear system?
[b] How easy is it to use the most advanced features offered?
[c] How short and intuitive are the argument list of the user-interface?
[d] How easy is it to prepare the required arguments before using the iterative methods?
[e] How useful is the feedback from the functions of the packages?

thing a user document should explain is the overall schema of the software interface design, but most documents often miss this important part.

- **Examples** are usually provided by the packages. Here we mostly look for the presence of simple examples that can help users to get start quickly and easily. An ideal example of this type should be less than a page when printed on paper, have just enough information to solve an example linear system of equations and have enough comment in the code for the user to follow along. A score of 1 in this category means there is no simple example provided or the examples are not well explained for a user to follow, 2 means some examples are provided but there is a lack of an "ideal" example.

- **Ease of use** is measured by the effort required to solve a simple problem (simple task) and the effort required to utilize the advanced features (advanced features). In these two categories, a score of 3 indicates that we anticipate an experienced programmer can integrate the package into his or her program in less than a day, 2 means that the integration may need less than one week's work, and 1 means that the integration needs more than a week's work. Generally, the packages that utilize data encapsulation provided by C and C++ programming language require less time to use and those that require the user to allocate every array composed of the sparse matrix data structure and the preconditioner data structure need more time to learn and to use.

  The time needed to use the advanced features are measured as the additional time needed after one is able to solve simple problems. This measures the effort required for a novice to become an expert user. Those packages that do not require the user to explicitly construct preconditioners are relatively easy to master.

- **User interface** should be clean, intuitive, and always provide useful feedback. Consistency in naming the functions and conformance to the overall schema of the package design is the first subcategory evaluated here. Most of the eight packages that we have seen provides reasonably consistent interface. The next two subcategories measure the length of the argument list and the effort required to prepare those arguments. On the argument list length, we give a score of 1 for those with functions that has more than 20 arguments in a key function, 2 for those with 10 to 20 arguments and 3 for those with less than 10 arguments. Typically data encapsulation is used to reduce the argument list size and having a better designed interface for the solvers can also reduce the size of the argument list. Time needed to understand the resource requirement of each argument is a major part of the time needed to use the software package. In many cases, it is appropriate to hide this type of complexities. Aztec and P_SPARSLIB do not hide this complexity; PETSc is regarded as the *standard* for this type of software; and PARASOL is a good effort to hide this type of complexity in Fortran programs.

- **Availability** of the eight packages are generally good. Table 1 lists where are the packages can be found on the web. Most of them can be downloaded from the web, only Aztec and PINEAPL need extra work. Aztec requires the user to fill out a web form and the user will receive a password by email to enter the actual download page. PINEAPL needs a formal commercial license.

- **Installation** of the packages are usually pretty easy. The easiest installation requires two command on a unix machine, `configure` and `make install`. PETSc and BlockSolve is rated 3 on this category because they come as close two this idea situation as can be expected. Most of the other packages require one to manipulate `Makefile`s in one way or another or require one to install a number of other packages and obtaining extra files from an archive.

- **Interoperability** should be good for these packages since solving a linear system of equations is a very well defined task involving small number of objects. However because the differences in data structure and the user interface, it is not trivial to change from one package to another. Table 6 lists the score based on the estimated time required to switch from using a package heading the rows to a package heading the columns. Again, the score are based on 3 for a one-day effort, 2 for a one-week effort and 1 for a multi-week effort.

- **User support** from the developers are generally good since most of the packages have a relatively small user community. Most of the developers can respond to an inquiry in a day or so. In addition, there is an effort by the US Department of Energy to offer unified support through the Advanced Computational Testing and Simulation toolkit, for further information see `http://acts.nersc.gov` which can further enhance the user support for scientific computing softwares.

Table 6: Ease of change from using one package to another

$\Longrightarrow$	Aztec	BlockSolve	ISIS++	LINSOL	P_SPARSLIB	PARASOL	PETSc	PINEAPL
Aztec		2	3		1	2	2	1
BlockSolve	1		2		1	2	2	1
ISIS++	3	2			1	2	3	1
LINSOL								
P_SPARSLIB	2	2	2			2	2	1
PARASOL	2	3	3		1		3	1
PETSc	3	3	2		1	2		1
PINEAPL	2	2	2		1	2	2	

In addition to the above general comments about the score and the scoring system, we also offer a few comments about some of the packages that we have acquired personal experience.

Aztec Version 2.0 of the source code and the corresponding documentation are available as of July of 1998. The documentation could use some more editing to improve readability but it contains the essential information needed for a user to get started. Table 5 contains the usability score for this package, the following are more specific comments that can be regarded as footnotes to the scores.

- The solution vector needs to contain extra space to provide workspace for matrix-vector multiplication. This feature is not mentioned in the documentation and it not obvious in the example program either.
- After calling the required transformation routine to process the input matrix, the user need to explicitly reorder the right-hand side and the initial guess. The documentation neglected to mention this.
- The description of the DVBR is not very clear. The title of the section suggest some hierarchical relation between DVBR and DMSR which is misleading.
- No provision for repeatedly solving linear systems with coefficient matrix of the same nonzero pattern.

BlockSolve This package was last updated in 1997. The focus appears to be providing two parallel incomplete factorization preconditioners which are missing in many other packages. The preferred way of using this package would be to use it through PETSc.

ISIS++ This is a package that is actively being developed. The developers are also involved in an effort to componentize linear system solution, see *Equation Solver Interface* forum at `http://z.ca.sandia.gov/esi`. As part of this effort, this package is designed to access a number of other existing packages. However, the developers appear to have put in too much emphasis on the object-orientation offered by C++ and have not paid enough attention to inter-language related issues. As of this writing, the developers have released version 1.2.0 of the package.

P_SPARSLIB The language used to implement this package is Fortran 77 which can not perform dynamic memory allocation. Because of this, the user is required to allocate enough workspace for each array used in the sparse matrix data structures. Since Fortran 77 does not have user defined data type, there are many individual arrays which are exposed in the user interface. This leads to too many arguments in the calling sequence. Understanding the memory requirement can be a complicate task and users often need to spend significant amount of time on this. P_SPARSLIB and PINEAPL appear to both have this kind of problem. P_SPARSLIB has a set of strong preconditioners based on incomplete LU factorization and a selection of update-to-date iterative methods. It is mostly designed to be a toolkit for research scientists in the numerical analysis field. There has been some minor user interface modification between release 2 and release 3 of the software.

PARASOL Since this package is implemented in Fortran 90, it has the benefit of the data encapsulation and it is able to allocate computer memory dynamically. This makes it more convenient to offer more easy-to-use software interface than earlier Fortran based packages. This package has a complete set of user interface design documents and it has a number of industrial partners which provide direct user input to the development process. Its strongest linear system solution feature is a multifrontal direct solver.

PETSc This package is well designed and widely used. It is among the first MPI based program packages available to the public. It has a good set of iterative methods but has a weak set of preconditioners. However, since it is able to access the preconditioners provided by BlockSolve, the user can easily use preconditioners provided by both PETSc and BlockSolve. Recent integration effort is likely to make it easier to access more packages such as Aztec without modification to the user code. One perceived shortcoming of this package is that it appears to have higher learning curve though our own experiences demonstrate otherwise. Part of this perception comes from the fact that the package is bigger than others. PETSc requires the user to use its own initialization and clean-up procedure instead of the corresponding functions from MPI. This gives the user the impression that PETSc is taking over the program. It is also possible that this may be a source of real concern if the user program uses another package that requires one to use its initialization and clean-up procedures as well. PETSc compiles into a number of archive files which are to be linked into the user program. However the selection of these archive files is not well explained. The users have to use the makefile stubs provided by PETSc in order to link with the correct files. Simplifying this complexity could also go a long way in removing the perceived high learning curve.

7 Performance

This section provides a very limited amount of performance information about two of the eight portable parallel iterative solver packages: Aztec and PETSc. We only collected performance information on these two package because they are the two most commonly used ones among the users that we have come into contact during the operations of the National Energy Research Scientific Center. Trying to collect comprehensive performance information will take significantly more time which will delay the completion of the report and rendering it less timely for the reader. The performance of the linear system solvers are significantly affected by particularities of the iterative methods, preconditioners, linear systems and the computer system used, we have chosen to make our tests as simple as possible. The test matrices are listed in Table 7; the right-hand sides of the test linear systems are vectors with all elements equal to one; the initial guess is always zero. Our tests run the restarted version of GMRES(30) without preconditioning for 500 iterations[7].

The test program performs the following operations,

- read in the matrix from disk,

[7] We did plan to run the iterative methods with preconditioning, however, our numerous attempts failed to find a combination of iterative method and preconditioner that is able to reduce the residual norm by six order of magnitude in 500 iterations using Aztec and PETSc.

Table 7: Information about the five test problems. (MB is short for Mega Bytes.)

name	order(N)	NNZ	file size	description
A1	199911	14955879	180 MB	a finite element model of deformation of a complex composite structure [1]
Aug7	1060864	9313876	116 MB	the support operator discretization of a radiation transport model [25, 34]
Aug9	1060864	8313856	104 MB	the Morel-Hall discretization of a radiation transport model [25, 34]
CPC.K	637244	30817002	253 MB	the stiffness matrix from structural analysis
Omega3P	1066302	17117586	210 MB	a finite element discretization of the Maxwell equation in complex domain [32]

- feed the matrix to Aztec or PETSc, i.e., convert the matrix to the format required by the package to be used,
- generate a zero vector as initial guess and a vector of all ones as the right-hand side of the linear system,
- set parameters to indicate that GMRES(30) should be used without preconditioning for 500 iterations,
- run the iterative method.

All steps except the last one are considered setup steps for the iterative solution. When measuring time, the first four steps are counted as setup time, the last step is labeled as GMRES time. The matrices are stored in a portable binary format. When read into computer memory, they are distributed in a simple block-row format.

The setup time and the iteration time are reported in Table 8 and Figures 1 and 2. Overall, Aztec and PETSc use about the same amount of time when solving the same test problem. From Table 8 and Figures 1 and 2, one preeminent feature is that the setup time fluctuates wildly. To a large extend, this fluctuation is due to the file IO contention. From Table 7, we see that the size of the matrix files are fairly large, 100 – 250 MB. When the test program uses a small number of processors, it is likely there are a number of other jobs running at the same time which might also require file IO operations. This may cause significant delay in reading of the matrices. When the packages prepare the input matrix for performing sparse matrix-vector multiplications, significantly amount of data may be exchanged among the processors.

Table 8: Performance information collected from solving the five test problems on a 512-processor Cray T3E 900. (PE is short for processing element, i.e., processor.)

Aztec								
	elapsed time (seconds)						speed-up	
	8-PE case			128-PE case			8-PE → 128-PE	
	setup	GMRES	total	setup	GMRES	total	GMRES	total
A1	48.1	56.4	104.5	44.7	7.44	52.1	7.6	2.0
Aug7	749.2	210.2	959.4	32.9	20.1	53.0	10.5	18.1
Aug9	229.9	180.0	409.9	28.8	19.1	47.9	9.4	8.6
CPC.K	74.3	159.4	233.7	70.2	23.4	93.6	6.9	2.5
Omega3P	68.8	189.1	257.9	53.9	19.8	73.7	9.6	3.5
PETSc								
	elapsed time (seconds)						speed-up	
	8-PE case			128-PE case			8-PE → 128-PE	
	setup	GMRES	total	setup	GMRES	total	GMRES	total
A1	48.8	54.5	103.3	43.2	6.79	49.0	8.0	2.1
Aug7	33.1	227.9	261.0	30.7	25.1	55.8	9.1	4.7
Aug9	28.8	246.3	275.1	27.3	22.7	50.0	10.9	5.5
CPC.K	77.9	146.5	224.4	74.9	19.8	94.7	7.4	2.4
Omega3P	65.5	211.5	277.0	53.4	18.8	72.2	11.3	3.8

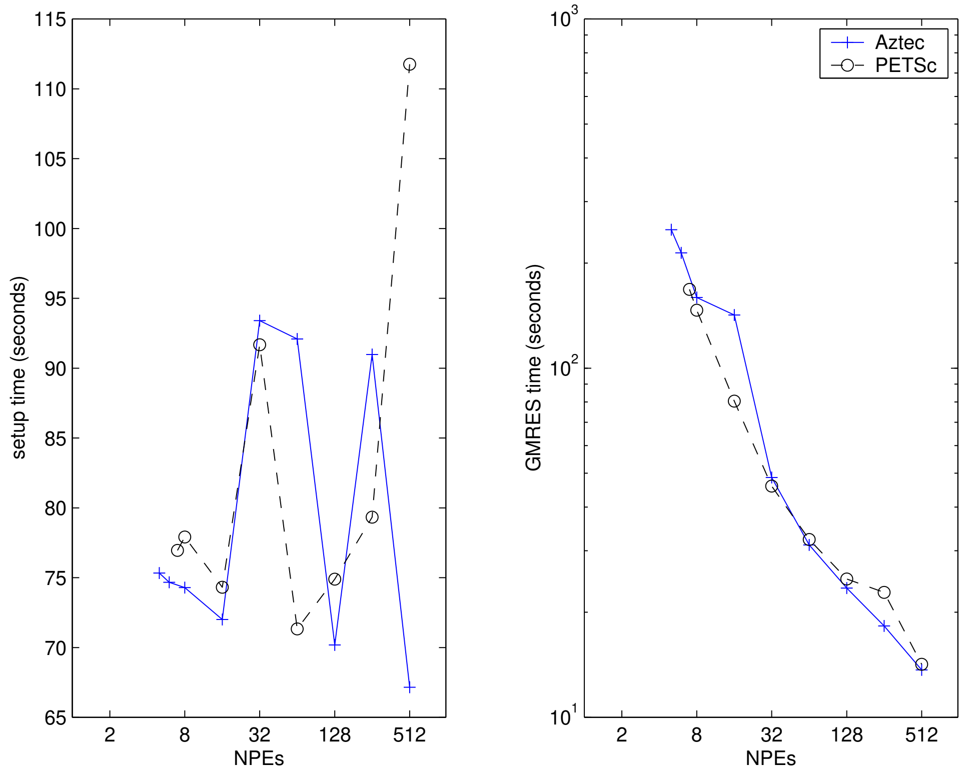

Figure 1: Time (seconds on T3E) to run the test problem CPC.K.

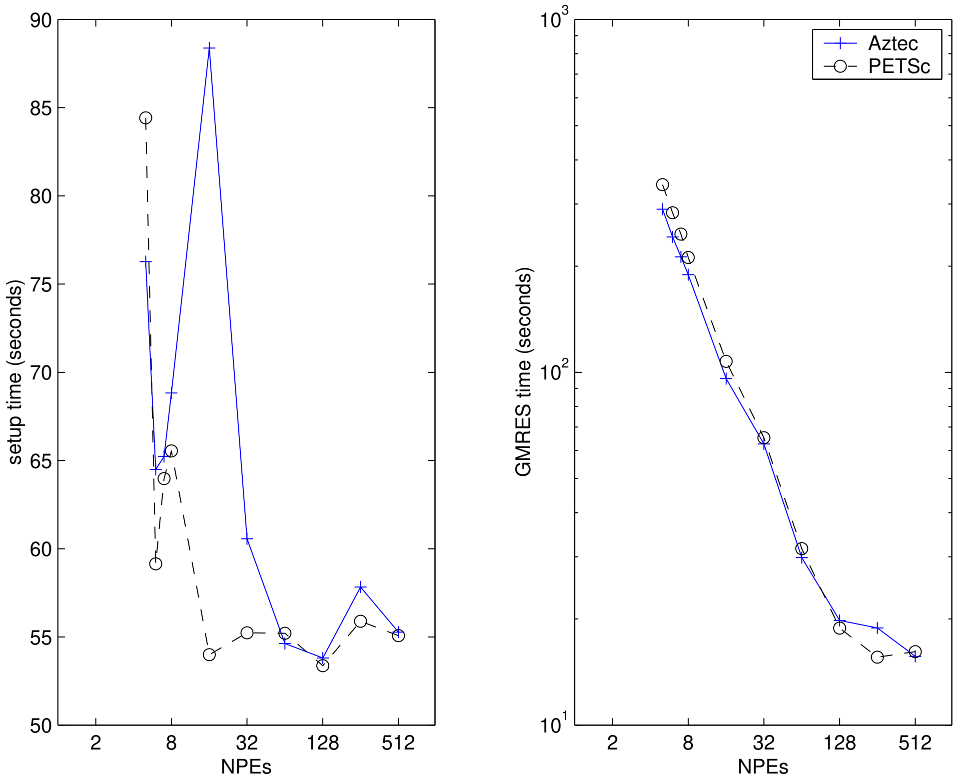

Figure 2: Time (seconds on T3E) to run the test problem Omega3P.

Since this process involves complex data communication, it may cause network congestion which can also introduce large variations when different number of processors are used. The data in Table 8 seem to suggest that only the Aztec test program suffers from this setup variation, in fact both test programs suffer the same problem. The variations are usually larger when less than 32 processors are used.

The time used by GMRES reduces monotonically until 256 or 512 processors are used. When the test matrices are distributed onto 256 or 512 processors, each processor has a relatively small amount of arithmetic work to do and the communication overhead is likely to dominate the total time spend in the parallel sparse matrix-vector multiplications. In short, even though the test problems are reasonably large, they are still to small to effectively utilize 256 or 512 processors. All five test problems can be easily fit onto 8 processors of the Cray T3E 900 that has 256 MB memory on each processor[8]. For these reasons, Table 8 shows the time on 8 processors and 128 processors.

Increasing the number of processor from 8 to 128, the time used to run GMRES for 500 iterations reduces

[8] See NERSC web page `http://hpcf.nersc.gov/computers/T3E` for more information about the T3E.

by a factor ranging from 8 to 11. However, the total time, including both setup time and GMRES time, only decrease by a factor of 2 to 5 because the time required to read the matrices stay relatively constant when the number of processors increases. The GMRES method from PETSc shows slightly better speed-up than that from Aztec. This might indicate the parallel sparse matrix-vector multiplication in PETSc is slightly more efficient than the one in Aztec.

While running the test programs, we observe that the Aztec test program encounters *Operand range error* when working on Aug7 and Aug9 test problems on 512 processors. This is a typical symptom of a programming error.

8 Summary

We have reviewed eight portable parallel iterative method packages that are currently available. Among the eight packages, Aztec and PETSc are the most commonly used ones. Both packages are fairly well documented, have reasonably clean user interface and provide good user support. Between the two packages, PETSc has slightly better designed user interface. In our tests, it also behaves better when large number of processors are used.

A large part of PETSc's popularity is due to the fact that it is one of the earliest available MPI based linear system solution package. Over the years it has grow to be a mature software package with a large user basis. Its main weakness is its size. PETSc was designed to solve many problems of scientific computing, e.g., linear system of equation, nonlinear equations, and ODE problems, and it even has its own compiling procedure consisting of a set of makefile stubs. The user who committed to use PETSc has to embrace the whole design of the package, such as, using PETSc version of initialization and finalization functions instead of the corresponding functions of MPI, using PETSc error checking functions, and using the makefile stubs to build the programs correctly. This is a significant commitment and a one that can not be easily reversed. PETSc has a well designed error handling capability. It is surprising to encounter a case where it hangs when failed to allocate internal variables.

Because of the significant commitment associated with using PETSc, many application programmers just started to use linear system packages are turning to Aztec and other packages. Many users choose Aztec because it provides a good set of functionality for solving linear systems and it is modest in size. There is also a perception that since Aztec is dedicated to solving linear systems, its core operations such as the sparse matrix-vector multiplication might be better implemented than those in PETSc. Our limited tests suggest that this perception might not be true.

Among the other six packages, PARASOL has a strong emphasis on direct solver and is best suited for those users who are required to use Fortran; ISIS++ provides hooks for users to access many different packages and is best suited for C++ users. For most application programmer, we believe the Fortran 77 based packages require too much effort to get started because the user has to provide numerous arrays of correct sizes. Among the three Fortran 77 based packages, LINSOL, P_SPARSLIB and PINEAPL, P_SPARSLIB is the best one. We recommend it for the specialist who want to control every aspects of the linear system solution process. P_SPARSLIB also has an impressive set of preconditioners which can be critical in solve difficult linear systems.

References

[1] Mark Adams. *Multigrid Equation Solvers for Large Scale Nonlinear Finite Element Simulations*. Ph.D. dissertation, University of california, Berkeley, 1998. Tech. report UCB//CSD-99-1033.

[2] B. A. Allan, R. L. Clay, K. M. Mish, and A. B. Williams. ISIS++ reference guide. Technical Report SAND99-8231, Sandia National Laboratories, 1999.

[3] P. R. Amestoy, I. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. Technical Report TR/PA/98/22, CERFACS, France, 1998.

[4] P. R. Amestoy, I. Duff, and J. Y. L'Excellent. MUMPS multifrontal massively parallel solver version 2.0. Technical Report TR/PA/98/02, CERFACS, France, 1998.

[5] P. R. Amestoy, I. Duff, J. Y. L'Excellent, and P. Plecháč. PARASOL: an integrated programming environment for parallel sparse matrix solvers. Technical Report TR/PA/98/13, CERFACS, France, 1998.

[6] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, J. Koster, and M. Tuma. *MUltifrontal Massively parallel solver (MUMPS Version 4.0) Specification sheets*, 1999.

[7] Owe Axelsson. *Iterative Solution Methods.* Press Syndicate of the University of Cambridge, Cambridge, UK, 1994.

[8] S. Balay, W. Gropp, L. C. McInnes, and B. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11, Mathematics and Computer Science Division, Argonne National Laboratory, 1995. Latest source code available at `http://www.mcs.anl.gov/petsc`.

[9] M. Benzi, C. D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17(5):1135–1149, 1996.

[10] S. Carney, M. Heroux, G. Li, and K. Wu. A revised proposal for a sparse BLAS toolkit: SPARKER working note # 3. Technical Report 94-034, Army High Performance Computing Research Center, Minneapolis, 1994.

[11] U. V. Catalyurek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. In A. Ferreira, J. Rolim, Y. Saad, and T. Yang, editors, *Parallel Algorithms for Irregularly Structured Problems. Proceedins of Third International Workshop IRREGULAR '96 (Santa Barbara, CA, USA, 19-21 Aug. 1996)*, pages 75–86, Berlin, Germany, 1996. Springer-Verlag.

[12] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. Technical Report UMSI 95/181, Minnesota Supercomputing Institute, University of Minnesota, 1995.

[13] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 18(6):1657–1675, 1997.

[14] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse matrices.* Oxford University Press, Oxford, OX2 6DP, 1986.

[15] Victor Eijkhout. Overview of iterative linear system solver packages. *NHSE Review*, 3(1), 1998. On the web at `http://www.nhse.org/NHSEreview/98-1.html`.

[16] Elegant Mathematics, Inc. *N_SPARSE_SP Release 1.2*, 1999. More information available at `http://www.elegant-math.com`.

[17] D. J. Evans and G.-Y. Lei. Approximate inverses of multidiagonal matrices and application to the block PCG method. *BIT*, 35:48–63, 1995.

[18] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive-Definite Systems*. Prentice-Hall, New Jersey, 1981.

[19] GMD SCAI, Germany. *PARASOL Interface Specification*, 1998. On the web at `http://www.genias.de/projects/parasol`.

[20] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD 21211, third edition, 1996.

[21] H. Häfner, W. Schönauer, and R. Weiss. Parallelization and integration of the LU and ILU algorithm in the LINSOL program package. In *The Proceedings of Pact_99 conference, St. Petersburg, Russia, 1999*, 1999.

[22] H. Häfner, W. Schönauer, and R. Weiss. The progrm package LINSOL: basic concepts and realizations. *Applied Numerical Mathematics*, 30:213–224, 1999.

[23] M. J. Hagger. Automatic domain decomposition on unstructured grids (DOUG). Technical Report maths9706, School of Mathematical Sciences, University of Bath, Bath, U.K., 1997.

[24] M. J. Hagger and L. Stals. *DOUG — Doman Decomposition on Unstructured Grids User Guide v 1.98*. School of Mathematical Sciences, University of Bath, Bath, U.K., 1998.

[25] M. L. Hall and J. E. Morel. Diffusion discretization schemes in augustus: A new hexahedral symmetric support operator method. Technical Report LA-UR-98-3146, Los Alamos National Laboratory, 1998.

[26] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro. AZTEC user's guide. Technical Report SAND95-1559, Sandia National Laboratories, Albuquerque, NM, 1995.

[27] M. T. Jones and P. E. Plassmann. Blocksolve95 users manual: scalable library software for parallel solution of sparse linear systems. Technical Report ANL-95/48, Mathematics and Computer Science Division, Argonne national laboratory, Argonne, IL, 1995.

[28] M. Joshi, G. Karypic, and V. Kumar. *PSPASES: Scalable parallel director solver library for sparse symmetric positive definite linear systems*, 1998. The latest version of the software package is available at `http://www-users.cs.umn.edu/~mjoshi/pspases/`.

[29] G. Karypis, K. Schloegel, and V. Kumar. *ParMETIS: parallel graph partitioning and sparse matrix ordering library*, 1998. Further information available at `http://www.cs.umn.edu/~karypis`.

[30] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings I. theory. *SIAM J. Matrix Anal. Appl.*, 14(1):45–58, 1993.

[31] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings II. solution of 3D FE systems on massively parallel computers. *International Journal of High Speed Computing*, 7(2):191–215, 1995.

[32] B. McCandless, Z. Li, V. Srinivas, Y. Sun, and K. Ko. Omega3P: A parallel eigensolver for modeling large, complex cavities. In *Proceedings ICAP98*, 1998.

[33] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31:148–162, 1977.

[34] J. E. Morel, Jr. J. E. Dendy, M. L. Hall, and S. W. White. A cell-centered lagrangian-mesh diffusion differencing scheme. *J. Comput. Phys.*, 103(2):286–299, December 1992.

[35] NAG. *PINEAPL final report — experience and results*, February 1999. For more information, visit `http://www.nag.com/projects/PINEAPL`.

[36] Jeffrey Rubin. *Handbook of usability testing: how to plan, design, and conduct effective tests.* Wiley, New York, 1994.

[37] Tony Rubin. *User Interface Design for computer systems.* Ellis Horwood, Chichester, UK, 1988.

[38] Y. Saad and K. Wu. Design of an iterative solution module for a parallel matrix library (P_SPARSLIB). *Applied Numerical Mathematics*, 19:343–357, 1995. Was TR 94-59, Department of Computer Science, University of Minnesota.

[39] Y. Saad and K. Wu. DQGMRES: a quasi-minimal residual algorithm based on incomplete orthogonalization. *Numerical Linear Algebra with Applications*, 3(4):329–343, 1996.

[40] Y. Saad, K. Wu, and S. Petiton. Sparse matrix computations on the CM-5. In R. Sincovec, D. Keyes, M. Leuze, L. Petzold, and D. Reed, editors, *Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 414–420, Philadelphia, 1993. SIAM.

[41] Yousef Saad. Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 5:203–228, 1984.

[42] Yousef Saad. SPARSKIT: A basic toolkit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990. Software currently available at `ftp://ftp.cs.umn.edu/dept/sparse/`.

[43] Yousef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, March 1993.

[44] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, Boston, MA, 1996.

[45] R. S. Tuminaro, J. N. Shadid, and S. A. Hutchinson. Parallel sparse matrix-vector multiply software for matrices with data locality. *Concurrency: Practice and Experience*, 10(3):229–47, March 1998.

[46] R. Weiss, H. Häfner, and W. Schönauer. LINSOL description and user's guide for the parallelized version. Technical Report 61/95, Rechenzentrum der Universitat Karlsruhe, 1995. Further information available at `http://www.rz.uni-karlsruhe.de/Uni/RZ/Forschung/Numerik/linsol`.

[47] Kesheng Wu. An experimental study of Krylov subspace accelerators. Technical Report UMSI 96/20, Minnesota Supercomputer Institute, University of Minnesota, 1996.

[48] Kesheng Wu and Horst Simon. An evaluation of parallel shift-and-invert lanczos method. In *Proceedings of The 1999 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, June 28 - July 1, 1999*, pages 2913–19, 1999.

Appendix A Related software

In addition to the packages reviewed, there are a large number of packages that are primarily devoted to solve linear systems of equations, have a major component for solve linear systems, or provide more specialize services such as preconditioning. This section provides some pointers about some of these packages. The emphasis is still on packages that solves linear systems. To facilitate searching for other related packages, we also list a few archives that the authors often visit.

A.1 Other iterative solver packages

In addition to the parallel iterative methods listed in Table 1, there are many other packages that can be used to solve linear systems and there are many larger packages that include linear system solution components as well. We refer readers to Eijkhout's 1998 review of linear system solver packages [15]. Here we will only briefly mention the two parallel iterative solver packages not reviewed.

- **DOUG** This package is primarily designed to solve linear systems arising from finite element discretization of PDE problems [23, 24]. The main developers are M. J. Hagger and L. Stals from University of Bath. It contains a good selection of domain decomposition based preconditioning schemes. The software is implemented in Fortran with MPI as the communication layer. We did not evaluate this one because it requires the user to provide information about the finite element discretization which may not be available if the linear system is generated by other means. Those interested in the software can visit `http://www.maths.bath.ac.uk/~parasoft/doug` to obtain the software and the documentation.

- **N_SPARSE_SP** This package is from Elegant Mathematics, Inc. (USA) founded by Dr. Alex Yu. Yeremin and colleagues. It is written in Fortran and has been primarily used on IBM SP systems. The greatest strength of this package is its implementation of factorized sparse approximate inverse preconditioners [30, 31]. At the time of this writing, we only have access to some documentation of the package [16], but not the package itself. For this reason, we did not provide extensive evaluation of the package. Those interested in this package should visit `http://www.elegant-math.com` for more information.

A.2 Dense direct solver

In some cases, the linear systems to be solved are reasonably small or its zero entries are too few for any algorithm to take advantage of, the coefficient matrix of these linear systems are usually stored as dense matrices. The most commonly used algorithm to solve these linear systems are based on Gauss elimination. The most commonly used package for solving these type of linear system is LAPACK on sequential machines and ScaLAPACK on distributed machines. For those who are more familiar with object oriented programming, LAPACK++ is available for the sequential machines and PLAPACK is a good alternative on distributed machines. There are a number of commercial packages that are quite successful, but the authors have very little experiences using them. Further information about these two packages can be found on the web, see Table 9.

A.3 Sparse direct solver

Often the coefficient matrices of the linear systems contain a significant portion of zero entries and are too large to be treated as dense matrices. In these cases, sparse versions of the Gauss elimination (LU factorization) may be successfully used. These types of methods are known as sparse direct methods. We refer the reader to the software archives listed below for sparse direct methods for sequential machines. Here we briefly mention the parallel sparse direct method packages that are known to us. PSPASES is one of the first publically available parallel sparse direct methods package. It implements the Cholesky factorization for symmetric positive definite linear systems and it is only able to function on power of two number of processors. Because there is no need of pivoting, it is able to run efficiently on a variety of machines [28, 48]. S+, SuperLU, UMFPACK and SPOOLES are all able to solve nonsymmetric linear systems. S+ and SuperLU use versions of supernode based algorithms, UMFPACK employs a multifrontal algorithm. All these three software packages partition both rows and columns at the same time. This partitioning scheme requires more complex data structure support, but are important in achiving high parallel efficiency and good load balance during both factorization and triangular solution phases. SPOOLES uses a unique data structure termed *chevron* which allows a flexible implementation of many different functions in a single code base. For example, SPOOLES implements LDL^T factorization for symmetrix indefinite matrices, LU factorization for nonsymmetric matrices, QR factorization and it can be run on sequential environment, multithreaded environment and distributed parallem environment. This package is especially designed with the shift-and-invert Lanczos method in mind and it is suitable for all other cases where many sparse matrices of same nonzero pattern are factored. To accommodate possibility of pivoting, most of the parallel direct solvers need to use very complex data structure which affects scalability and efficiency. However, in the cases where sparse factorization can be stored in computer memory, sparse direct methods are often more robust and/or more efficient.

A.4 Parallel preconditioner

The preconditioners implemented in the eight reviewed packages are mostly based on incomplete LU factorization or overlapping Schwarz iteration. A larger variety of preconditioning schemes are being developed constantly. Here we mention just a few to give readers a sense of what is available out there. BPKIT implements a set of block preconditioners in C++ and it offers a flexible way of varying from simple Gauss-Seidel iterations; PARPRE offers a set of domain decomposition based preconditioners and a number of ILU and SOR type of preconditioners; SPAI implements a sparse approximate inverse preconditioner that computes the sparsity pattern of the approximate inverse dynamically; Prometheus builds a domain decomposition based preconditioner on top of PETSc. Other on-going projects, such as HYPRE, are developing more sophisticated preconditioning schemes that are expected to be robust and easy to use, for more information see Table 9.

A.5 Archives

There are a number of archives on the world wide web that are devoted mostly to scientific computing software packages. The following is a short list that the authors visit frequently.

- The ACTS toolkit page (`http://acts.nersc.gov`). This web site is funded by US Department of Energy (DOE). Its main goal is to maintain useful information about the software packages that are funded by DOE.

Table 9: The web addresses for other linear system related packages mentioned.

other parallel iterative packages	
DOUG	http://www.maths.bath.ac.uk/~parasoft/doug
N_SPARSE_SP	http://www.elegant-math.com
dense direct solver	
ScaLAPACK	http://www.netlib.org/scalapack
PLAPACK	http://www.cs.utexas.edu/users/plapack
sparse direct solver	
PSPASES	http://www-users.cs.umn.edu/~mjoshi/pspases
S+	http://www.cs.ucsb.edu/Research/s%2b
SuperLU	http://www.nersc.gov:/~xiaoye/SuperLU
SPOOLES	http://www.netlib.org/linalg/spooles
UMFPACK	http://www.cise.ufl.edu/~davis/umfpack.html
parallel preconditioner	
BPKIT	http://www-users.cs.umn.edu/~chow/bpkit.html
PARPRE	http://www.cs.utk.edu/~eijkhout/parpre.html
SPAI	http://www.sam.math.ethz.ch/~grote/spai/spai-page.html
Prometheus	http://www.cs.berkeley.edu/~madams
HYPRE	http://acts.nersc.gov/hypre/main.html

- Netlib (`http://www.netlib.org`). This site has one of the most extensive collections of scientific computing software packages. It should be the first place to visit for anyone who is interested in numerical analysis and related software. This site also contains useful links to many other resources.

- ACM TOMS archive (`http://www.acm.org/calgo`). This site collects all software published in ACM Transactions on Mathematical Software. It also has an extensive links to other mathematical software, `http://www.acm.org/toms`.

- The NHSE (`http://www.nhse.org/`)is a distributed collection of software, documents, data, and information of interest to the high performance and parallel computing community. This site also publishes a web-based journal named NHSE Review that provides indepth review of scientific computing software.

- Scientific Applications on Linux (`http://SAL.KachinaTech.COM/index.shtml`) is a searchable archive with more than 2500 entries of freely available scientific programs. As linux becomes a more mainstream computer engine for the academia and smaller research institutions, software for linux platform will become more important.

- The Virtual Library (`http://src.doc.ic.ac.uk/bySubject/Computing/Overview.html`) maintained by the Department of Computing, Imperial College, UK has a long list of various on-line resources ranging from popular computer journal to esoteric algorithms research.

- NUMERICA (`http://www.numeritek.com`) is a company with a stated purpose of "bringing cutting-edge research in numerical methods closer to practical applications". It serves commercial customer for fee like NAG and others, but it also distributes its software for free, see NUMERICA website for terms and conditions.

- The popular web portals such as Yahoo, Netscape and Hotbot all provide some listing of popular mathematical software.

  - Yahoo: `http://dir.yahoo.com/Science/Mathematics/Software`
  - Netscape: `http://search.netscape.com/Science/Math/Software`
  - Hotbot: `http://dir.hotbot.lycos.com/Science/Math/Software`

Appendix B Acknowledgments

The authors gratefully acknowledge the help received from William C. Saphir and Osni A. Marques for reading the drafts of this manuscript and providing valuable suggestions for improvements.

This work was supported by the Director, Office of Science, Office of Laboratory Policy and Infrastructure Management, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.