Overview of Iterative Linear System Solver Packages

Victor Eijkhout

July, 1998

Abstract

Description and comparison of several packages for the iterative solution of linear systems of equations.

1 Introduction

There are several freely available packages for the iterative solution of linear systems of equations, typically derived from partial differential equation problems. In this report I will give a brief description of a number of packages, and give an inventory of their features and defining characteristics.

The most important features of the packages are which iterative methods and preconditioners supply; the most relevant defining characteristics are the interface they present to the user's data structures, and their implementation language.

2 Discussion

Iterative methods are subject to several design decisions that affect ease of use of the software and the resulting performance. In this section I will give a global discussion of the issues involved, and how certain points are addressed in the packages under review.

2.1 Preconditioners

A good preconditioner is necessary for the convergence of iterative methods as the problem to be solved becomes more difficult. Good preconditioners are hard to design, and this especially holds true in the case of parallel processing. Here is a short inventory of the various kinds of preconditioners found in the packages reviewed.

2.1.1 About incomplete factorisation preconditioners

Incomplete factorisations are among the most successful preconditioners developed for single-processor computers. Unfortunately, since they are implicit in nature, they cannot immediately be used on parallel architectures. Most packages therefore concentrate on more parallel methods such as Block Jacobi or Additive Schwarz. There are a few implementations of multicolour incomplete factorisations.

BlockSolve95 is the only package catching breakdown of the ILU or IC factorisation. The manual outlines code that successively shifts the diagonal until the factorisation succeeds.

2.1.2 Preconditioners for sequential processing

On sequential architectures, the BPKIT package provides sophisticated block factorisations, and LASpack contains multigrid solvers. Most other packages supply the user with variants of ILU (Incomplete LU) or IC (Incomplete Cholesky).

2.1.3 Preconditioners for parallel iterative methods

These are the approaches towards parallel preconditioning taken in the packages under review here.

Direct approximations of the inverse SPAI (section 3.13) is the only package that provides a direct approximation method to the inverse of the coefficient matrix. Since such an approximation is applied directly, using a matrix-vector product, it is trivially parallel. The SPAI preconditioner is in addition also generated fully in parallel.

Block Jacobi Each processor solves its own subsystem, and there is no communication between the processors. Since this strategy neglects the global/implicit properties of the linear system, only a limited improvement in the number of iterations can result. On the other hand, this type of method is very parallel.

All parallel preconditioner packages provide some form of Block Jacobi method.

Additive Schwarz As in the Block Jacobi method, each processor solves a local subsystem. However, the local system is now augmented to include bordering variables, which belong to other processors. A certain amount of communication is now necessary, and the convergence improvement can by much higher.

This method is available in Aztec (3.1), Petsc (3.10), ParPre (3.8), PSparselib (3.12).

Multicolour factorisations It is possible to perform global incomplete factorisation if the domain is not only split over the processors, but is also augmented with a multicolour structure. Under the reasonable assumption that each processor has variables of every colour, both the factorisation and the solution with a system thus ordered are parallel. The number of synchronisation points is equal to the number of colours.

This is the method supplied in BlockSolve95 (3.2); it is also available in ParPre (3.8).

Block factorisation It is possible to implement block SSOR or ILU methods, with the subblocks corresponding to the local systems (with overlap, this gives the Multiplicative Schwarz method). Such factorisations are necessarily more sequential than Block Jacobi or Additive Schwarz methods, but they are also more accurate. With an appropriately chosen processor ordering (e.g., multicolour instead of sequential) the solution time is only a small multiple times that of Block Jacobi and Additive Schwarz methods.

Such block factorisations are available in Parpre (3.8); PSparselib (3.12) has the Multiplicative Schwarz method, under the name 'multicolour SOR'.

Multilevel methods Petsc (3.10) and ParPre (3.8) are the only packages supplying variants of (algebraic) multigrid methods in parallel.

Schur complement methods ParPre (3.8) and PSparselib (3.12) contain Schur complement domain decomposition methods.

2.2 Data structure issues

Every iterative method contains a matrix-vector product and a preconditioner solve. Since these are inextricably tied to the data structures used for the matrix and the preconditioner (and possibly even to the data structure used for vectors), perhaps the most important design decision for an iterative methods package is the choice of the data structure.

2.2.1 The interface to user data

The more complicated the data structure, the more urgent the question becomes of how the user is to supply the structures to the package. This question is particularly important in a parallel context.

The packages reviewed here have taken the following list of approaches to this problem.

- Fully internal, not directly accessible, data structures (basically object oriented programming). This approach is taken in Petsc (3.10), ParPre (3.8), and BlockSolve (3.2).

  The data structures here are only accessible through function calls. This means that the package has to supply routines for constructing the structures, for inspecting them, and for applying them.

- Prescribed, user supplied, data structures. This approach is taken in Aztec (3.1); it is available in PCG (3.9).

  Here the user has to construct the data structures, but the package supplies the product and solve routines.

- User supplied, arbitrary, data structures, with user supplied product and solve routines. This approach is available in PCG (3.9) and Petsc (3.10); the object-oriented package IML++ (3.5) can also be considered to use this approach.

- User defined data structures, not passed to the iterative method at all; product and solve operations are requested through a reverse communication interface.

  This approach is taken in PIM (3.11); it is also available in PCG (3.9).

2.2.2 Parallel data layout

There are several ways of partitioning a sparse matrix over multiple processors. The scheme supported by all packages is partitioning by block rows.

- Each processor receives a contiguous series of rows of the matrix. This is the approach taken in Petsc (3.10); it is available in BlockSolve95 (3.2).

  Under this approach a processor can determine the ownership of any variable, by keeping a short list of first and last rows of each processor.

- Each processor receives an arbitrary set or rows. This is the approach taken in Aztec (3.1); it is available in BlockSolve95 (3.2).

  Under this scheme, each processor needs to maintain the mapping of its local rows to the global numbering. It is now no longer trivial to determine ownership of a variable.

When a sparse matrix is distributed, the local matrix on each processor needs to have its column indices mapped from the global to the local numbering. Various packages offer support for this renumbering.

2.3 High performance computing

Sparse matrix problems are notoriously low performers. Most of this is related to the fact that there is little or no data reuse, thereby preventing the use of BLAS kernels. See for example the tests on vector architectures in [4].

Other performance problems stem from parallel aspects of the solution methods.

Here are then a few of the approaches taken to alleviate performance problems.

2.3.1 Quotient graph computation

A matrix defines a graph by introducing an edge (i, j) for every nonzero element a_ij. A dense matrix in this manner defines a graph in which each node is connected to every other node. This is also called a 'clique'. Sparse matrices, on the other hand, induce a graph where, no matter the number of nodes, each node is connected to only a small number of other nodes.

However, sparse matrices from problems with multiple physical variables will have dense subblocks, for instance corresponding to the variables on any given node. It is possible to increase the efficiency of linear algebra operations by imposing on the matrix the block structure induced by these small dense subblocks. For instance, the scalar multiplication/division $a\_ika\_kk^{-1}a\_ki$ that appears in Gaussian elimination now becomes a matrix inversion and a matrix-matrix multiplication, in other words, BLAS3 kernels.

Identifying cliques in the matrix graph, and deriving a quotient graph by 'factoring them out', is the approach taken in the BlockSolve package; section 3.2.

The PCG package (section 3.9) takes the opposite approach in its Regular Grid Stencil data format. To get dense subblocks, the degrees of freedom have to be numbered first in regular storage, but in PCG they are numbered last. This approach is more geared towards vector architectures.

2.3.2 Inner products

Most linear algebra operations in the iterative solution of sparse systems are easily parallelised. The one exception concerns the inner products that appear in all iterative methods (with the exception of the Chebyshev iteration). Since the collection and redistribution of data in an inner product will inevitably have some processors waiting, the number of inner products should be kept low.

The conjugate gradients and bi-conjugate gradients algorithms have two interdependent inner products, and various people have suggested ways to reduce this to two independent ones, which can then combine their communication stages. See [3] and the references cited therein. This approach has been taken in the PIM package; section 3.11.

The second source of inner products is the orthogonalisation stage in the GMRES algorithm. Using Gram-Schmidt orthogonalisation, the inner products are independent and can be combined; however, this makes the resulting algorithm unstable. Modified Gram-Schmidt gives a more stable algorithm, but the inner products are interdependent, and have to be processed in sequence. A middle ground is to apply (unmodified) Gram-Schmidt twice.

2.4 Language

The languages used to implement the packages here are C, C++, and Fortran. To some extent the implementation language determines from what language the library can be called: C++ constructs can not be used from C, and if a C routine returns an internally allocated array, this routine cannot directly be used from Fortran. The Petsc library addresses this last point in its Fortran interface.

3 The packages

3.1 Aztec

Aztec is a parallel library of iterative solution methods and preconditioners. Its main aim is to provide tools that facilitate handling the distributed data structures. For this, there is a collection of data transformation tools, as well as query functions of the data structures.

3.1.1 Basic information

Available from Web site[1]; registration required

Author(s) Scott A. Hutchinson, John N. Shadid, Ray S. Tuminaro

Latest version 1.1, October 1995

Status Unknown

3.1.2 Contents

Iterative methods CG, GMRES, CGS, TFQMR, BiCGstab, Direct solve (on one processor only).

Preconditioners Block Jacobi with ILU on the subblocks; also Additive Schwarz, with an overlap limited to one.

Data structures Distributed variants of Compressed Row and Variable Block Row; see below.

Manual User's Guide, 43 pages

Example codes Yes

3.1.3 Parallelism and data layout

Aztec can handle arbitrary assignments of matrix rows to processors. Since this quite general arrangement makes it harder to construct the local numbering of the matrices (see section 2.2.2), the user can supply the matrix with global numbering and a preprocessing routine performs the localisation. However, the user does have to supply all rows on the appropriate processor.

3.1.4 Other

Aztec requires routines from `Blas`, `Lapack`, `Linpack`, `Y12m`.

[1]http://www.cs.sandia.gov/CRF/aztec1.html

3.1.5 Installation

As part of the installation, Aztec requests auxiliary routines from `netlib`, and places them in a subdirectory to be compiled later. This precludes native `Blas` or `Lapack` libraries to be used. There is no hint on how to substitute these.

AzTec uses a distributed version of the MSR (Modified Sparse Row) storage format, which is close to the Compressed Row Storage format. I find this an unfortunate choice:

- I do not see any advantage over the CRS format.
- While conversion to and from CRS is straightforward, the arrays need to be longer by one position. This means that interfacing AzTec to a code using CRS entails deallocating and reallocating the arrays.
- The extra storage location in the real array is not used; the location in the integer array duplicates the scalar parameter giving the number of rows.

3.2 BlockSolve95

The main focus of the BlockSolve package is the implementation of SSOR and ILU preconditioners based on a parallel multi-colouring algorithm by the authors. The algorithm first computes a quotient graph of the matrix graph by eliminating cliques and identical nodes. Operations on the nodes of this quotient graph then become of BLAS2/3 type, leading to a high performance.

BlockSolve can be used with the Petsc package; section 3.10.

3.2.1 Basic information

Available from Web site[2]

Author(s) Mark T. Jones and Paul E. Plassmann

Latest version 3.0, June 1997

Status Being maintained and further developed

3.2.2 Contents

Iterative methods CG, SYMMLQ, GMRES are provided, though these are not the main focus of package; BlockSolve can be interfaced to Petsc for more iterative methods.

Preconditioners diagonal scaling, block Jacobi with blocks corresponding to the cliques factored out of the quotient graph, incomplete LU and Cholesky.

Data structures Internal, as a C structure..

> **Access operations on the data structure** None; the definition of the structure is given in the manual.
>
> **Operations using the data structure** Iterative solution of the system and of a shifted system, application of the matrix and the preconditioner[3].

Manual Users Manual; 34 pages. This is a complete reference; the user is suggested to use the example codes as templates for applications.

Example codes Yes

[2] http://www.mcs.anl.gov/blocksolve95/

[3] The separate application of the matrix and the preconditioner are not documented in the manual

3.2.3 Parallelism and data layout

The user has two ways to pass the distributed matrix to BlockSolve.

1. Since the definition of the data structure is given in the manual, the user can explicitly construct it.
2. The user can supply a compressed row storage matrix, with column indices in the local numbering (section 2.2.2), to the routine `BSeasy_A`, which yields the matrix structure.

In the second case, the matrix rows need to be consecutively numbered. In the first case the assignment of rows over the processors can be arbitrary; the user has to construct the mapping functions between local and global numberings. There are example codes illustrating this.

3.3 BPKIT2

BPKIT is a package of block preconditioners, that is, factorisation preconditioners that treat the matrix first on a subblock level, then recursively factor the blocks on the element level. One of the selling points of BPKIT is the object-oriented approach to this two-level factorsation.

3.3.1 Basic information

Available from Web site[4]

Author(s) E. Chow and M. A. Heroux

Latest version 2.0, September 1996

Status Maintained

3.3.2 Contents

Iterative methods Flexible GMRES, though this is not the focus of the package

Preconditioners Block SSOR and ILU, possibly with block fill-in, and with various methods for explicit and implicit approximation of inverses of pivot blocks

Data structures Internal, as a C++ class.

> **Access operations on the data structure** Retrieve rows and scalar information such as the number of nonzeros of the matrix.
>
> **Operations using the data structure** Multiply and multiply transpose by the matrix; solve and solve transpose of the preconditioner, both the whole preconditioner, and the left and right split parts.

Manual Reference manual, 53 pages

Example codes Yes, in Fortran, C, and C++

3.3.3 Installation

The makefile in the `app` subdirectory requires editing for the location of MPI and for compiler options.

The manual is not well written. Many parameters and auxiliary routines are under-documented.

[4]http://www.cs.umn.edu/%7Echow/bpkit.html/

3.4 GPS: General Purpose Solver

3.4.1 Basic information

Available from Web site[5]; requires registration by postal mail.

Author(s) Olaf O. Storaasli, Majdi Baddourah, Duc Nguyen

Latest version 03/08/95 (submission to NASA Langley Software Server)

Status Approval for downloading the software did not come in in time for this report.

[5]http://www.larc.nasa.gov/LSS/ABSTRACT/LSS-1995-0002.html

3.5 IML++

The IML++ package consists of C++ implementation of the iterative methods from the templates project [2]. It relies on the user supplying Matrix, Vector, and Preconditioner classes that implement the required operations.

An example implementation of such classes called Sparselib++ is available from the same authors. It contains uni-processor matrix classes based on compressed row and column and coordinate storage, a vector class, and Jacobi and ILU/IC preconditioners for the compressed row and column matrices. Additionally it contains tools for converting a Harwell-Boeing matrix or matrix file to these formats.

3.5.1 Basic information

Available from Web site[6]

Author(s) Jack Dongarra, Andrew Lumsdaine, Roldan Pozo and Karin A. Remington

Latest version 1.2, April 1996

Status Being maintained; IML++ will eventually be superseded by the Template Numerical Toolkit, a package not currently available.

3.5.2 Contents

Iterative methods BiCG, BiCGstab, CG, CGS, Chebyshev, GMRES, IR, QMR

Preconditioners n/a

Data structures n/a

Manual Reference guide, 39 pages; also SparseLib++ Reference Guide, 20 pages.

Example codes no

[6]http://math.nist.gov/iml++/

3.6 Itpack 2C / ItpackV 2D

Itpack is a package of iterative methods. It runs sequentially, but ItpackV is an optimised version for vector machines such as the Cray Y-MP.

Itpack features adaptive determination of matrix bounds, to be used in accurate stopping tests, of for the Chebyshev semi-iteration.

3.6.1 Basic information

Available from Ftp site[7], also on Netlib

Author(s) David R. Kincaid, Thomas C. Oppe, David M. Young

Latest version Itpack 2C: manual dated July 1997, Itpack 2D: manual dated May 1989

Status Being maintained

3.6.2 Contents

Iterative methods Jacobi Conjugate Gradient, Jacobi Semi-iteration (i.e., Chebyshev iteration, SOR, SSOR, SSOR CG, Reduced system CG, Reduced system SI

Preconditioners n/a; see above

Data structures Itpack 2C: Compressed Row (there are auxiliary routines to facilitate building the data structure); ItpackV 2D: ellpack storage.

Manual 22/14 pages

Example codes Yes

3.6.3 Installation

Itpack There is no makefile or anything like it, but there is really no need for it either, since a complete installation consists of one file of library routines and one file of tests.

The `PROGRAM` statement in the test files had to be edited.

The test code was insufficiently documented for an easy 'same problem but larger' test.

Nspcg The `nspcg` routines come in files with undescriptive names such as `nspcg1.f`.

The `nspcg5.f` file needed replacement of the timer routine.

[7]ftp://ftp.ma.utexas.edu/pub/CNA/ITPACK

3.7 Laspack

LASpack is an object-oriented package of iterative methods, iterative methods, multigrid solvers, and auxiliary routines for the iterative solution of linear systems. It does not run in parallel.

There are data structures for vectors, general and square matrices, and preconditioners; a large number of accessing and manipulating these objects is available.

3.7.1 Basic information

Available from Web site[8]; also from Netlib

Author(s) Tomáš Skalický

Latest version 1.12.3, January 1996

Status Developed

3.7.2 Contents

Iterative methods Jacobi, SOR, SSOR, Chebyshev iteration, CG, CGN, GMRES, BiCG, QMR, CGS, BiCGstab, restarted GMRES.

Multigrid methods Conventional and Full multigrid, BPX preconditioning

Preconditioners Jacobi, SSOR, ILU(0)

Data structures Internal, created by passing elements by function call.

> **Access operations on the data structure** Many
>
> **Operations using the data structure** Matrix addition and scaling; Matrix-vector multiply, transposition, Matrix inverse vector multiply.

Manual Reference manual, 8+40 pages.

Example codes Yes.

[8]http://www.math.tu-dresden.de/ skalicky/laspack/index.html

3.8 ParPre

This is an add-on package to Petsc; section 3.10. It is solely a collection of parallel preconditioners, to be used with iterative methods either from Petsc or coded by the user. The data structures are those used by Petsc.

ParPre can be used independently of Petsc, but does require Petsc to be installed.

3.8.1 Basic information

Available from Web site[9]

Author(s) Victor Eijkhout and Tony Chan

Latest version 2.0.17

Status Maintained and developed

3.8.2 Contents

Iterative methods None

Preconditioners Additive and multiplicative Schwarz, Generalised Block SSOR, Schur complement domain decomposition, Algebraic multilevel methods (including multicolour ILU and algebraic multigrid).

Data structures Internal.

Access operations on the data structure Inherited from Petsc.

Operations using the data structure Solve, no solve transpose.

Manual Reference manual with programming examples; 32 pages.

Example codes Yes

3.8.3 Parallelism and data layout

All methods in ParPre are based on decomposition of the physical domain into subdomains, with an assignment of one subdomain per processor (or rather: process). Most methods involve a local solve on the subdomain, for which any of the non-parallel Petsc preconditioners can be used.

For the methods that do not have all subdomains operate in parallel (e.g., multiplicative Schwarz as opposed to additive Schwarz), the user can specify the strategy that determines the sequential ordering of the domains. The choices are: natural, red-black, and multicolour.

[9] http://www.math.ucla.edu/ eijkhout/parpre.html

3.9 PCG

The PCG package has a large number of iterative methods and a few simple preconditioners. The code is publically available in a uni-processor version, and one optimised for Cray YMP. An MPI version is under development. The iterative methods are addressable on three levels, each with a different way of handling the data structures.

3.9.1 Basic information

Available from Web site[10]

Author(s) W.D. Joubert, G.F. Carey, N.A. Berner, A. Kalhan, H. Khli, A. Lorber, R.T. McLay, Y. Shen

Latest version 1.0, September 1996

Status Being further developed

3.9.2 Contents

Iterative methods Richardson, CG, CG with Neuman polynomial preconditioning, BiCG, BiCG with restart, Lanczos / Orthores, CGS, BiCGstab, BiCGstab2, BiCGstab(ℓ), QMR, TFQMR, truncated OrthoMin and OrthoRes, Incomplete Orthogonalisation, GMRES: restarted, restarted with Householder reflections, and nested restarted GMRESR; CGNE and CGNR, LSQR and LSQE.

Preconditioners Richardson and Jacobi

Data structures Regular Grid Stencil storage supported; also data structure free mode by passing product and solve routines, or through reverse communication.

Manual Reference manual, 64 pages; also Getting Started manual, Examples manual, and guide to the XPCG Xwindows front end.

Example codes Yes.

3.9.3 Interface to user data structures

PCG has three ways of handling the matrix and preconditioner data structures.

First of all, PCG has one supported data structure: the Regular Grid Stencil storage. This corresponds to the type of matrices arising from having the same finite difference or finite element stencil on each grid point of a Cartesian product grid. Such matrices are trivially stored in multi-dimensional arrays. After the user sets up the matrix array and the right hand side vector, PCG solves the system (this is called Top Level Usage of PCG).

[10] http://www.cfdlab.ae.utexas.edu/pcg/index.html

Secondly, the user can pass matrix and preconditioner arrays, plus two integer arrays for each, to iterative method routines on the Iterative Method Level of PCG. Additionally, now the user has to supply routines to perform the matrix vector product and the preconditioner solve, plus their transposes for methods such as BiCG that need them.

Thirdly, PCG supports Reverse Communication Usage: no matrix or preconditioner structures whatsoever are passed to the iterative routines. Instead, when a product or solve operations needs to be performed, the iterative routine will save its internal state, and return control to the user program with a request to perform that operation.

```
1     continue
      call CG( .... IREQ .... IVA, IVQL ...  FWK ...)

      if ( IREQ .eq. JAV ) then
c perform matrix-vector product to the vector FWK(IVQR)
c leaving the result in FWK(IVA)
      else if ( IREQ .eq. JQLV )
c apply left preconditioner to the vector FWK(IVA)
c leaving the result in FWK(IVQR)
      else .....
      end if

      goto 1
```

Control is also returned to the user for the convergence test, but inner products are still performed inside the iterative routine.

The Regular Grid Stencil storage scheme incorporates the possibility of having multiple physical variables per grid point; see section 2.3.1.

3.10 Petsc

Petsc is a package for the solution of PDE problems. It contains everything from linear algebra tools, through linear solvers, nonlinear solvers, and time-stepping methods. Since it is written in an object-oriented style, all data structures are hidden from the user. A large number of construction and inspection routines give access to the numerical data and parameters of the objects.

Petsc can use the preconditioners of the BlockSolve package; section 3.2.

3.10.1 Basic information

Available from Web site[11]

Author(s) Satish Balay, William Gropp, Lois Curfman McInnes,Barry Smith

Latest version 2.0.17

Status Begin maintained and further developed

3.10.2 Contents

Iterative methods Richardson, Chebyshev, CG, GMRES, TCQMR, BCGS, CGS, TFQMR, CR, LSQR

Preconditioners Identity, Jacobi, Block Jacobi, Block Gauss-Seidel (only sequential), SOR and SSOR, IC and ILU (sequential only?), Additive Schwarz, full factorisation (sequential only), user supplied.

Data structures Internal, elements passed through function calls.

> **Access operations on the data structure** Function calls yielding the internal arrays of matrices and vectors, matrix rows and the matrix diagonal; other statistics.
>
> **Operations using the data structure** Vector-vector and matrix-vector operations; creation and application of preconditioners, linear and nonlinear system solvers.

Manual Users manual, 196 pages; the manual and the man pages are also available in html format.

Example codes Yes

3.10.3 Parallelism and data layout

Petsc supports both dense and sparse data structures sequential and in parallel; there is support for multiple physical variables per unknown.

The data structures are completely hidden from the user, only accessible through function calls. Creation of the vector and matrix data structures is completely general: any processor can specify elements for any other processor.

[11]http://www.mcs.anl.gov/petsc/petsc.html

3.11 Parallel Iterative Methods (PIM)

The sole focus of this package is on iterative methods. PIM let's the user supply external routines for the matrix-vector product, preconditioner application, and norms and inner products. This makes the package largely independent of data structures and communication protocols, in fact of whether the program is running in parallel or not. It also puts a considerable burden on the user.

3.11.1 Basic information

Available from Web site[12]

Author(s) Rudnei Dias da Cunha, Tim Hopkins

Latest version 2.2, May 1997

Status Maintained

3.11.2 Contents

Iterative methods CG, CG on the normal equation (CGNE and CGNR), BiCG, CGS, BiCGstab (normal and restarted), restarted GMRES, restarted GCR, QMR with reduced synchronisation overhead, TFQMR, Chebyshev iteration.

Preconditioners None

Data structures None

Manual User's guide; 81 pages

Example codes Yes; both sequential and parallel, and for dense and sparse data formats. The manual contains a good discussion of the example programs.

3.11.3 Interface to user data structures

PIM iterative method routines need parameters corresponding to external routines for the matrix-vector (possible matrix-transpose-vector) product, and the preconditioner application.

The calling interface for these external routines is fairly simple, e.g.,

```
subroutine matvec{u,v,ipar}
double precision u(*),v(*)
integer ipar(*)
```

[12]http://www.mat.ufrgs.br/pim-e.html

where the `ipar` array is the information array that is passed to the iterative method.

Unfortunately this puts a number of restriction on the user's choices. For instance, it implies that the matrix has to be in common blocks, and that the vectors need to be simple arrays; they can not be pointers to more elaborate structures.

3.12 PSparselib

This seems to be very much a package under development. There are various discrepancies between the manual and the code, and the manual is far from being a reference.

3.12.1 Basic information

Available from Web site[13]

Author(s) Yousef Saad and Gen-Ching Lo

Latest version 2.15, May 1997 (manual is dated June 1996)

Status Being developed; future version may be for internal use only.

3.12.2 Contents

Iterative methods Flexible GMRES, CG, BiCG, BiCGstab, GMRES, DQGMRES, TFQMR

Preconditioners Additive and multiplicative Schwarz, Schur complement domain decomposition

Data structures Undocumented, the user is referred to tech reports and articles.

Manual Users manual, 14 pages; this does not document calling sequences or data structures.

Example codes Yes

3.12.3 Parallelism and data layout

PSparselib uses reverse communication to abstract away from particulars of the communication layer and the data structure: the `fgmres` routine returns control to the user for each matrix-vector product and preconditioning operation. However, inner products are still performed by hard MPI instructions in the `fgmres` routine.

3.12.4 Installation

The makefile required editing for some macros, as described in the `README` file.

[13] http://www.cs.umn.edu/Research/arpa/p˙sparslib/psp-abs.html

3.13 Sparse Approximate Inverse (SPAI) Preconditioner

SPAI is a research code, implementing in parallel an approximate inverse preconditioner, based on computing a minimum norm approximation to the inverse of the coefficient matrix. Both the computation and application of the preconditioner are fully parallel.

3.13.1 Basic information

Available from Web site[14]

Author(s) Steve Barnard

Latest version

Status Maintained and developed

3.13.2 Approximate inverse

Most popular preconditioners are implicit, that is, to apply them one has to solve a system. One might say that they compute an approximation to the coefficient matrix that is easier to solve with than the matrix itself.

The approximate inverse class of preconditioners is different in that they compute explicitly an approximation to the inverse. Hence the application is an explicit matrix-vector product operation, and therefore trivially parallel.

The method in the SPAI code is based on ideas from [5]: the minimisation problem

$$\min \|AM - I\|$$

or

$$\min \|MA - I\|$$

is solved, with the sparsity pattern of M predetermined or adaptively determined. This minimisation problem turns out to reduce to independent subproblems for the rows or columns of M, and is therefore executable in parallel.

An other advantage of this method is that it is not subject to breakdown the way factorisation based methods are.

[14]http://lovelace.nas.nasa.gov/NAS/SPAI/download.html

3.14 SPlib

SPlib is a package of uni-processor iterative methods and preconditioners, primarily designed for ease of use.

3.14.1 Basic information

Available from Ftp site[15]

Author(s) Randall Bramley and Xiaoge Wang

Latest version Unknown

Status Being maintained

3.14.2 Contents

Iterative methods CG-stab, BiCG, CGNR and CGNE, CGS, BiCGstab, GMRES, TFQMR, templates version of CGS, templates version of GMRES, Jacobi, Gauss-Seidel, SOR, Orthomin

Preconditioners Identity, ILU(s), MILU(s, r), ILUT(s, t), SSOR(ω), TRID(s), ILU0, ECIMGS; where s is the number of levels of fill, r is the relaxation parameter [1], t is the drop tolerance.

Data structures Compressed Sparse Row

Manual 26 pages

Example codes Driver program that read a Harwell-Boeing matrix and solves a problem with it.

[15] ftp://ftp.cs.indiana.edu/pub/bramley/splib.tar.gz

3.15 Templates

The templates codes are meant as example implementations of the methods in the Templates book [2]. As such they are somewhat unsophisticated, more illustrations of the principles than optimised realisations.

3.15.1 Basic information

Available from Netlib[16]

Author(s) Richard Barrett et. al.

Latest version

Status Maintained

3.15.2 Contents

Iterative methods BiCG, BiCGstab, CG, CGS, Chebyshev, GMRES, Jacobi, QMR, SOR

Preconditioners Identity, Jacobi

Data structures User supplied: each iterative method is given in two versions.

1. The matrix-vector product routine is an external, passed to the iterative routine, and the matrix is assumed to be in common.
2. The iterative routine uses reverse communication.

Manual The Templates book [2] is available commercially, or for download from Netlib.

Example codes Yes

[16]http://www.netlib.org/templates/

4 Comparison chart of features

The following chart gives some basic information about the packages. Please consult the previous section for a more detailed discussion of the individual packages.

Parallel Does the package run in parallel? All the parallel packages are based on MPI, other protocols are noted.

Iterative Does the package contain iterative methods? A few packages have preconditioners as their main focus, but suppply one or a few iterative methods for the user who doesn't have any yet.

Prec Does the package contain preconditioners?

Data How does the package interface to user data? See note 3 below.

Lang What is the implementation language of the package?

Inst Is the library instrumented, reporting on flops and timing?

Package	Parallel	Iterative	Prec	Data[3]	Language	Inst
Aztec	yes	yes	yes	internal[3a]	C	Yes
BlockSolve95	yes	yes[1]	yes	internal[3a]	C	Yes
BPKIT	no	yes[1]	yes	internal[3b]	C++[8]	No
IML	n/a[2]	yes	yes[9]	supplied	C++	
Itpack	no	yes	yes[7]	prescribed	Fortran	No
Laspack	no	yes	yes	internal	C	
ParPre	yes	no	yes	internal[4]	C	
PCG	coming	yes	yes	prescribed/supplied/free	Fortran	
Petsc	yes	yes	yes	internal/supplied	C[8]	Yes
PIM	n/a[2]	yes	no	free	Fortran	
PSparselib	yes	yes	yes	free	Fortran	No
SPAI	yes	yes1	yes		C	
SPlib	no	yes	yes	prescribed	Fortran	
templates	no	yes	no[5]	supplied[6]/free	Fortran/C/Matlab	

Notes

1 Not the main focus of this package.

2 The library abstracts away from data structure implementation aspects; parallelism is possible, but is the user's responsibility.

3 For the explanation of terms 'internal', 'prescribed', 'supplied', and 'free', see section 2.2.1.

3a converted from compressed row format.

3b converted from Harwell-Boeing format.

4 Identical to Petsc format.

5 Nothing beyond Jacobi.

6 The external product and solve routines are presumed to find the matrix in a common block.

7 Can not be chosen independently of the iterative method: the user picks a combination.

8 Fortran interface provided.

9 Preconditioners provided in an example C++ matrix class library, SparseLib++.

5 Performance tests

We have run performance tests on a number of packages. Ideally, these tests combine all of the following possibilities:

- Single processor and parallel where possible.
- Using all data structures supplied or accepted by the package.
- Comparing various iterative methods, in particular where they have different parallel behaviour, such as the Chebyshev method versus the Conjugate Gradient method.
- Assessing the efficacy of different preconditioners, measuring separately and combined:
  - Cost of setup,
  - Reduction in numbers of iterations,
  - Number of flops per iteration,
  - Flop rate of the solve performed in each iteration.
- Solving different systems of the same size and structure is not of much use, as this only changes the number of iterations performed; one could note how many iterations are necessary to offset the initial setup cost.

5.1 Machines used

The following machines at the University of Tennessee, Knoxville, were used:

nala Sun Ultra-Sparc 2200 with Solaris 5.5.1. 200MHz, 16K L1, 1Mb L2. Compilers: `f77 -O5` and `gcc -O2`.

cetus lab Sun Ultra-Sparc 1, 143 Mhz, connected by 10Mbps Ethernet.

N	Mfl MSR	Mfl VBR (nb=4)
2500	23	
2744		23
10,000	20	
9261		22
22,500	19	22

Table 1: AzTec performance on Nala (section 5.1.

N	np=1	np=2	np=4	np=8
2500	26	20	18	16
10,000	20	26	37	45
22,500	19	27	49	68
90,000	17	26	50	89
250,000	16	25	49	95

Table 2: AzTec aggregate Mflop rating for Jacobi preconditioned CG on the Cetus lab (section 5.1.

5.2 Results

5.2.1 AzTec

Problem tested: five-point Laplacian solved with Jacobi CG. We used the sample main program provided, and altered only parameter settings

- CG instead of CGS,
- Block Jacobi preconditioner,
- 5-point instead of 7-point matrix.

We also tested the 7-point Laplacian with 4 variables per grid point, using the VBR format. Since this uses level 2 BLAS routines, it should in principle be able to get higher performance, but in practice we do not see this happening. In the single processor tests in table 1 we see that for small problems there is a slight performance increase due to cache reuse, but not on the order that we would see for dense operations. The use of Blas2 in the VBR format seems to have no effect.

AzTec's built in timing and flop count does not support the ILU(0) preconditioner, so we added that. The flop count is approximate, but does not overestimate by more than a few percent. We omit the $N = 400$ tests because they were too short for the timer to be reliable.

From table 2 we see for small problem sizes the communication overhead dominates; for larger problems the performance seems to level off at 13 Mfl per processors, about 10 percent of peak performance. Performance of an ILU(0)-preconditioned method (table 3) is slightly lower. The explanation for this is not immediately clear. Note that, since we used a regular grid problem, it is not due to indirect addressing overhead.

N	np=1	np=2	np=4	np=8
2500	17	19	19	17
10,000	15	21	35	47
22,500	14	21	38	65
90,000	13	20	39	73
250,000		20	38	76

Table 3: AzTec aggregate Mflop rating for ILU(0) preconditioned CG on the Cetus lab (section 5.1.

N	p=1	p=2	p=4
400	5.6	8.8	4.5
2500	5.5	2.4	2.4
10,000	5.5	3.7	4.6
90,000	5.0	5.3	8.4
250,000	4.8	5.5	9.5

Table 4: BlockSolve95 aggregate megaflop rates on the Cetus lab (section 5.1; one equation per grid point.

5.2.2 BlockSolve95

We tested the supplied `grid5` demo code, with the timing and flop counting data supplied in BlockSolve95. The method was CG preconditioned with ILU.

From table 4 we see that the performance of BlockSolve95 is less than of other packages reported here. This is probably due to the more general data format and the resultant indirect addressing overhead. Results in table 5 show that by inode/clique identification BlockSolve95 can achieve performance comparable to regular grid problems in other packages.

Larger problems than those reported led to segmentation faults, probably because of insufficient memory. Occasionally, but not always, BlockSolve aborts with an 'Out of memory' error.

5.2.3 Itpack

Problem tested: five-point Laplacian solved with Jacobi CG. We wrote our own main program to generate the Laplacian matrix in row compressed and diagonal

N	p=1	p=2	p=4	p=8
400	23(10)	10(2)	5(2)	06(2)
2500	19(9)	20(6)	17(5)	24(5)
10,000	18(8)	25(7.5)	38(9)	54(10)

Table 5: BlockSolve95 aggregate megaflop rates on the Cetus lab (section 5.1; five equations per grid point; parenthesized results are without inode/clique isolation.

N	alloc (Mb)	Mfl CRS	Mfl Dia
400	.05	19	1
2500	.3	20	8
10,000	1.2	17	14
22,500	2.8	16	15

Table 6: Megaflop rates for Itpack on a single Cetus machine (section 5.1.

N	p=1	p=2	p=4	p=8
400	17	4	2	1
2500	18	12	8	7
10,000	15	20	20	24
90,000	13	22	44	75
250,000	13	22	44	88

Table 7: Aggregate megaflop rates for unpreconditioned CG under Petsc on the Cetus lab (section 5.1).

storage format.

Certain Itpack files are provided only in single precision. We took the single precision files and compiled them with `f77 -r8 -i4`, which makes the REALs 8 bytes and INTEGERs 4. It is not clear why diagonal storage will only give good performance on larger problems.

5.2.4 Petsc

We tested the Petsc library on Sun UltraSparcs that were connected by both Ethernet and an ATM switch. The results below are for the Ethernet connection, but the ATM numbers were practically indistinguishable.

We wrote our own main program to generate the five-point Laplacian matrix. The method in table 7 is an unpreconditioned CG algorithm.

We tested the efficacy of ILU by specifying

```
PCSetType(pc,PCSOR);
PCSORSetSymmetric(pc,SOR_LOCAL_SYMMETRIC_SWEEP);
```

which corresponds to a block Jacobi method with a local SSOR solve on-processor. This method, reported in table 8, has a slightly lower performance than the unpreconditioned method, probably due to the larger fraction of indirect-addressing operations.

5.2.5 PSparsLib

We added flop counting to the example program `dd-jac`, which is an additive Schwarz method with a local solve that is ILUT-preconditioned GMRES.

Larger problem sizes ran into what looks like a memory-overwrite. Attempts to allocate more storage failed.

N	p=1	p=2	p=4	p=8
400	14	2	2	1
2500	15	9	7	6
10,000	12	13	18	20
90,000	10	13	26	45
250,000		14	27	52

Table 8: Aggregate megaflop rates for ILU CG under Petsc on the Cetus lab (section 5.1).

N	p=1	p=2
400	29	10
2500	26	5

Table 9: Aggregate megaflop rates for PSparsLib on the Cetus lab (section 5.1).

5.3 Discussion

Although our tests are nowhere near comprehensive, we can state a few general conclusions.

- A sequential code should be able to attain 10–20% of the peak speed of the machine. This value was attained by most packages, using a variety of storage formats.
- Parallel codes have significant overhead; for large enough problems this is amortized to where the per-processor performance is about half of that of the sequential code.
- Inode/clique identification can make a large difference in systems that have multiple variables per node.

References

[1] Owe Axelsson and Gunhild Lindskog. On the eigenvalue distribution of a class of preconditioning matrices. *Numer. Math.*, 48:479–498, 1986.

[2] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia PA, 1994.

[3] E.F. D'Azevedo, V.L. Eijkhout, and C.H. Romine. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Technical Report CS-93-185, Computer Science Department, University of Tennessee, Knoxville, 1993. to appear.

[4] Jack Dongarra and Henk van der Vorst. Performance of various computers using sparse linear equations software in a fortran environment. *Supercomputer*, 1992.

[5] L. Yu. Kolotilina and A. Yu. Yeremin. On a family of two-level preconditionings of the incomlete block factorization type. *Sov. J. Numer. Anal. Math. Modelling*, pages 293–320, 1986.