

Applied Numerical Libraries for Parallel Software

M. Antonioletti, G. Darling

Edinburgh Parallel Computing Centre,
JCMB, King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ.

J.V. Ashby

Computational Science and Engineering Department,
Rutherford Appleton Laboratory, Chilton, Didcot, Oxon, OX11 0QX

R.J. Allan

Computational Science and Engineering Department,
Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD.

Revision : 0.8

Abstract

This report reviews the current state of applied numerical libraries. The judicious adoption of a third party library, whether commercial or free, can radically increase the performance and stability of your code base. It can also decrease code development times.

© **UKHEC 2002**. This Technical Report is the result of the UKHEC Collaboration. Copies are available from www.ukhec.ac.uk. Neither the UKHEC Collaboration nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

Contents

1	Introduction	1
2	Parallel Libraries	1
3	Adaptive and Self Optimising Libraries	2
4	Architectural influences	3
5	Review and classification of libraries	4
5.1	Linear Algebra and General Libraries	4
5.2	Eigenvalue type problems	13
5.3	Adaptive or Self Optimising Libraries	14
6	Conclusions	15

1 Introduction

Software libraries have been around in computing for a long time, indeed most compilers rely on their use for common tasks such as input/output, system calls and intrinsic routines. More specialised libraries are also familiar to most programmers, for example graphics libraries such as GINO, GKS, Phigs, OpenGL, etc., and the subject of this report, numerical libraries.

For UK academics the primary exemplar of a numerical library has for long been that provided by NAG (see section 5), a set of routines available in Fortran or C to perform a wide range of numerical operations including interpolation, integration, matrix solution and eigenproblems. The value of using such a library is well known. It speeds up development time as a programmer is free to concentrate on the novel aspects of a code rather than re-inventing or re-coding established methods. The library software, if it is from a reputable source, will be robust and efficient, employing well-proven techniques and algorithms. A well-designed library will be stable between releases, offering improvements in performance with minimal disruption to an application code, and a popular library will be available on a wide range of operating systems and hardware platforms, aiding portability (Though there are many VAX programmers who relied too heavily on DEC libraries and were surprised when they weren't on their new unix systems - will the same be true of Windows developers?).

Nowadays any serious numerically intensive computation is likely to be performed on a parallel or distributed computer. Libraries exist to exploit parallelism at the fine level, for example message passing libraries such as PVM and MPI, but this report seeks to answer the questions of what is available at the higher level – a level corresponding to the NAG library – and what factors should influence a programmer's choice of library?

2 Parallel Libraries

In choosing a parallel library to incorporate in an application, a number of decisions have to be made. The first is the target architecture or architectures on which the application will be, or is likely to be, run. Section 4 discusses this more fully, but it is clear that a library designed for a shared memory implementation will not necessarily be optimal for a distributed memory environment. One solution would be for libraries to be available at a sufficiently high level (solve linear system, Fourier Transform data, etc.) in a variety of implementations optimised for various architectures. Another is the provision of low-level building blocks similarly optimised; this is the philosophy behind BLAS and its parallel implementation, PBLAS (see section 5).

The advantage of high-level libraries is that they take a large amount of the grind out of application code development by using well-understood and robust algorithms, efficiently coded by experts. Their disadvantage is that they may be slow to become available for a novel architecture, and that they are perforce general, so may never perform as well as an algorithm made-to-measure for a particular application which can take advantage of known properties. Conversely, low-level libraries allow the efficient building of such tailored applications but reduce portability to other architectures since they may depend

on a specific mapping of problem to architecture (e.g. domain decomposition). More importantly they require an application programmer to invest in detailed knowledge of parallel computing which might well be better spent in developing the application domain.

Parallel libraries exist for a range of numerical techniques such as Digital Signal Processing, Linear Algebra, Multigrid solution of pdes, etc. The bulk of this report will concentrate on linear algebra and eigenproblem solution since these are the workhorses of most scientific application codes running on High-end computers ranging from electronic structure calculations to computational fluid dynamics, from protein folding to stellar structure.

This report is not a detailed comparison of available libraries, and in particular it makes no recommendations that one is better or worse than another. As outlined above, in choosing a parallel library many factors come into play and the balance of these will be different for different applications and even for different researchers developing similar applications. In choosing which libraries to discuss in more detail we have used two criteria: we believe that numerically intensive parallel applications are mainly written in Fortran, in C or in C++ and consequently have chosen libraries which are written in or interface to those languages, and we have looked at the more established and widely available libraries as likely to offer more stability and robustness.

3 Adaptive and Self Optimising Libraries

Using highly optimised numerical libraries could potentially boost the performance of your application by up to several orders of magnitude compared to using natively compiled code. However optimisations tend to be highly machine specific, what works for one system could result in a slowdown for another. Hand optimisation of a library is a very laborious process requiring a highly skilled individual with knowledge of the target architecture, compiler and operating system for which the library is to be optimised as well as requiring knowledge of the underlying numerical algorithms concerned. With the proliferation and rapid development of hardware and software this is becoming an untenable methodology. Hence within the last five years a new model for library optimisation has come on to the scene where libraries automatically optimise themselves through empirical means for the target architecture. Highly optimised libraries can thus be produced and are ready to use for the target architecture within a matter of hours.

There are two possible ways that one can go about empirically adapting software in order to maximise performance:

1. *parameterised adaption*: where a set of parameters can be used to optimise cache usage, by determining the blocking factors in a blocked algorithm at compile time, running the code and then trying a new one if performance is not improved.
2. *source code adaption*: actually involve some level of source code modification. Various software alternatives can be bundled into the code and each tried in turn. Some packages even provide a code generator that will produce code on the fly. Also, it allows developers to provide *codelets* that can be bundled in to what can be tried out as the optimisations work at a local level and thus do not necessarily require an overall knowledge of the library. The combination of software modifications that

gives the best performance are then the ones that are bundled into the library. Alternative code can be generated to take into account instruction cache size, pipelines, *etc* and thus produce code ideally suited for the target architecture.

When there are many possible software transformations that can be performed to produce the optimal code for the given algorithm the parameter search must be done efficiently in order not to have a factorial growth in the search space that has to be tried. In practice trees are used to enumerate the possible spaces that can be explored and branches are pruned off as quickly as possible in order to reduce the search space quickly. This automated approach can reveal unexpected parameter combination that yield better performance than would have been obtained by careful hand crafted optimisations. Indeed if one looks at the performance improvements gained by libraries that use this approach like ATLAS (which reproduces the BLAS functionality) or FFTW (FFTs) (see section 5), one sees that these libraries often produce higher performance than the counterparts provided by the hardware vendors themselves and are competitive with highly optimised versions also provided by vendors.

The timers used for the initial internal benchmarking of codes fragments for the source code adaption must be reliable and produce reproducible results regardless of the load on the machine that is used to optimise the library. The parameter search should also try to, in so far as is possible, replicate the conditions under which each branch of the search space is explored. For instance if a routine is going to be used that expects a cold cache then this condition should be replicated for each of the tests otherwise one will get misleading results. Clearly for a large library this kind of self optimisation can only be carried out for small performance critical sections of the code. In order to port code to use this model one of the library developers must identify what these sections are and take the appropriate action to port the library to use this way of working. Clearly this is not a small task but in the long run it should allow highly optimised versions of the library to be ported to many platforms relatively quickly and straightforwardly. Some of these types of libraries will be reviewed in section 5.

4 Architectural influences

The majority of HEC applications are MIMD (Multiple Instruction, Multiple Data), where several processors perform related, though not necessarily the same, computation on subsets of the overall program data. The main architectural distinction to be drawn is the organisation of the data subsets. In Distributed Memory (DM) systems, the subsets are parcelled up and associated with the processors which are working on them. Information which needs to be communicated between processors is sent using messaging protocols such as MPI or PVM. In contrast, Shared memory (SM) systems keep all the data together in an area which any processor can access. In these systems, the main difficulty is in avoiding contentions when several processors may wish to update the same piece of data. The next generation of High End Computers are likely to be hybrid systems where each element of a DM system is itself a SM parallel machine. The simplest example of such an architecture is a Beowulf cluster where each node has a dual processor motherboard. These will present their own opportunities and difficulties for novel algorithms.

Clearly the way in which an algorithm can be parallelised will depend on which architecture is present, although it is possible to emulate a distributed memory algorithm on a shared memory system. Thus when choosing a high level parallel numerical library it is important to consider which form of parallelisation it will support.

5 Review and classification of libraries

Table 1 and table 2, based on <http://www.netlib.org/utk/people/JackDongarra/la-sw.html>, give an overview of the different linear algebra packages available. Table 1 details the capabilities of libraries for serial processors and table 2 covers libraries which support parallelism in some way. Clearly even serial libraries can be used within parallel codes, for example in parameter farming or within a single process in a multi-process algorithm.

In the tables, the columns have the following interpretation: *Complex* refers to support for complex arithmetic – all libraries support real arithmetic; the languages (*Language*) supported by the packages (F for Fortran which may be Fortran77 or Fortran90); *Dense* stands for dense, triangular, tridiagonal matrices; *Sp Direct*: a direct approach is used to factor and solve the sparse system where the matrices are *SPD* symmetric and positive definite and/or *Gen*, General; *Sp Iter* an iterative method is used to solve the sparse system where the matrix is *SPD* or *GEN* as before; *Sp Eigen* an iterative method is used to find some of the eigenvalues where the sparse matrix is *Sym* symmetric (Hermitian in the complex case) and/or *Gen* general. In Table 2 the additional columns labelled *Mode* specify the architecture support where *SM* refers to SMP/multithreaded versions and *DM* refers to message passing with *M* indicating that there is an *MPI* implementation and/or *P* indicating *PVM*.

Table 3 gives the URL where more information about each of these packages can be obtained.

In the rest of this section we give a few salient details about those libraries from the tables which support parallelism. Libraries which exist only in serial versions are not included. Also mentioned are a few libraries which are not included in the tables.

5.1 Linear Algebra and General Libraries

BLAS

<http://www.netlib.org/blas/>

The BLAS (Basic Linear Algebra Subprograms) are high quality "building block" routines for performing basic vector and matrix operations. Level 1 BLAS do vector-vector operations, Level 2 BLAS do matrix-vector operations, and Level 3 BLAS do matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they're commonly used in the development of high quality linear algebra software, LINPACK and LAPACK for example.

Software Package	Complex	Language	Dense	Sp Direct		Sp Iter		Sp Eigen	
				SPD	Gen	SPD	Gen	Sym	Gen
MTL		C++	✓						
NIST S-BLAS	✓	F/C		✓	✓	✓	✓		
SparseLib++	✓	C/C++		✓	✓	✓	✓		
MA28		F		✓	✓				
MFACT		C		✓					
SPARSE	✓	C		✓	✓				
SPARSEQR		C/C++		✓	✓				
UMFPACK	✓	F		✓	✓				
Y12M		F		✓	✓				
ITPACK		F				✓	✓		
LASPack		C				✓	✓		
LSQR		F					✓		
QMRPACK	✓	F				✓	✓	✓	✓
SPLIB		F				✓	✓		
SYMMLQ		F		✓	✓				
Templates		F/C				✓	✓		
LASO		F						✓	

Table 1: Serial libraries (or libraries for which no information on parallel implementations could be found).

PBLAS

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

The Parallel Basic Linear Algebra Subprograms (PBLAS) are distributed memory versions of the Level 1, 2 and 3 BLAS. The software is available as part of the ScaLAPACK distribution tar file.

ScaLAPACK

<http://www.netlib.org/scalapack/>

The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory MIMD parallel computers. It is currently written in a Single-Program-Multiple-Data style using explicit message passing for interprocessor communication. It assumes matrices are laid out in a two-dimensional block cyclic decomposition.

ScaLAPACK is designed for heterogeneous computing and is portable on any computer that supports MPI or PVM.

Like LAPACK, the ScaLAPACK routines are based on block-partitioned algorithms in order to minimise the frequency of data movement between different levels of the memory hierarchy. (For such machines, the memory hierarchy includes the off-processor memory of other processors, in addition to the hierarchy of registers, cache, and local memory on each processor.) The fundamental building blocks of the ScaLAPACK library are distributed

Software Package	Complex	Language	Mode		Dense	Sp Direct		Sp Iter		Sp Eigen	
			SM	DM		SPD	Gen	SPD	Gen	Sym	Gen
ATLAS	✓	F/C	✓		✓						
BLAS	✓	F/C	✓		✓						
PLAPACK	✓	F/C		M	✓						
PRISM		F	✓	M	✓						
ScaLAPACK	✓	F/C		M/P	✓						
MP_SOLVE	✓	F		M			✓				
MUMPS		F	✓	M		✓	✓				
PSPASES		F/C		M		✓					
SPOOLES	✓	C	✓	M		✓	✓	✓	✓		
SuperLU	✓	F/C	✓			✓	✓				
AZTEC		C	✓	M				✓	✓		
BILUM		F	✓					✓	✓		
BlockSolve95		F/C/C++		M				✓	✓		
IML++		F/C/C++	✓					✓	✓		
ISIS++		C++		M				✓	✓		
ITL		C++	✓					✓	✓		
PARPRE		C		M				P	P		
PCG		F/C/C++		P				✓			
PETSc	✓	F/C	✓	M				✓	✓		
P-SparsLIB		F		M					✓		
SPAI		C	✓	M				✓	✓		
P_ARKPACK	✓	F/C/C++	✓	M/P						✓	✓
PLANSO		F	✓	M						✓	
TRLAN		F	✓	M						✓	

Table 2: Parallel libraries.

Software Package	More information
MTL	http://www.lsc.nd.edu/research/mtl
NIST S-BLAS	http://math.nist.gov/spblas/
SparseLib++	http://math.nist.gov/sparselib++/
MA28	http://www.netlib.org/harwell/
MFACT	http://www.cs.utk.edu/~padma/mfact.html
SPARSE	http://www.netlib.org/sparse/
SPARSEQR	http://www.arc.unm.edu/~trobey/
UMFPACK	http://www.cise.ufl.edu/research/sparse/umfpack/
Y12M	http://www.netlib.org/y12m/
ITPACK	http://www.netlib.org/itpack/
LASPack	http://www.tu-dresden.de/mwism/skalicky/laspack/laspack.html
LSQR	http://www.stanford.edu/~saunders/lsqr/
QMRPACK	http://www.netlib.org/linalg/qmrpack.tgz
SPLIB	http://ftp.cs.indiana.edu/pub/bramley/splib.tar.gz
SYMMLQ	http://www.stanford.edu/~saunders/symmmlq/
Templates	http://www.netlib.org/templates/
LASO	http://www.netlib.org/laso/
ATLAS	http://www.netlib.org/atlas/
BLAS	http://www.netlib.org/blas/
PLAPACK	http://www.cs.utexas.edu/users/plapack/
PRISM	http://www-unix.mcs.anl.gov/prism/
ScaLAPACK	http://www.netlib.org/scalapack/
MP_SOLVE	http://emlab2.nmsu.edu/mp_solve/
MUMPS	http://www.enseeiht.fr/apo/MUMPS/
PSPASES	http://www.cs.umn.edu/~mjoshi/pspases/
SPOOLES	http://www.netlib.org/linalg/spooles/spooles.2.2.html
SuperLU	http://www.nerisc.gov/~xiaoye/SuperLU/
AZTEC	http://www.cs.sandia.gov/CRF/aztec1.html
BILUM	http://www.cs.uky.edu/~jzhang/bilum.html
BlockSolve95	http://www.mcs.anl.gov/sumaa3d/BlockSolve/
BPKIT	http://www-users.cs.umn.edu/~chow/bpkit.html/
IML++	http://math.nist.gov/iml++
ISIS++	http://z.ca.sandia.gov/isis/
ITL	http://www.osl.iu.edu/research/itl/
PARPRE	http://www.cs.utk.edu/~eijkhout/parpre.html
PCG	http://www.cfdlab.ae.utexas.edu/pcg/
PETSc	http://www-fp.mcs.anl.gov/petsc
P-SparsLIB	http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html
SPAI	http://www.sam.math.ethz.ch/~grote/spai
P_ARPACK	http://www.caam.rice.edu/~kristyn/parpack_home.html
PLANSO	http://www.nerisc.gov/research/SIMON/planso.html
TRLAN	http://www.nerisc.gov/research/SIMON/trlan.html

Table 3: *URLs* for the packages from Table 1 and Table 2 where more information can be found out about them.

memory versions (PBLAS) of the Level 1, 2 and 3 BLAS, and a set of Basic Linear Algebra Communication Subprograms (BLACS) for communication tasks that arise frequently in parallel linear algebra computations. In the ScaLAPACK routines, all interprocessor communication occurs within the PBLAS and the BLACS. One of the design goals of ScaLAPACK was to have the ScaLAPACK routines resemble their LAPACK equivalents as much as possible.

NAG

http://www.nag.co.uk/numeric/numerical_libraries.asp

NAG's serial libraries have long been standard for UK academics and industrialists. They also offer an SMP (Symmetric Multi-Processor) library and a parallel library specifically designed for use on distributed memory systems and on clusters of workstations or PCs. Message passing is handled through MPI, mostly through the use of BLACS. There are routines for Dense Linear Algebra, eigenvalue and singular value problems, data distribution, sparse linear algebra, quadratures, optimisation and various support routines.

The NAG parallel library is available for Linux, SGI 6, Sun Solaris, HP-Compaq Alpha, IBM SP3, Hitachi SR8000 and Fujitsu VPP5000. It uses an SPMD programming model and supplies examples to provide tutorial assistance in programming using this model. The routine interfaces are designed to be as close as possible to the serial equivalents to aid program migration.

Note that this is a commercial library, although available at advantageous rates to academics.

HSL

<http://hsl.rl.ac.uk/>

The Harwell Subroutine Library contains three parallel routines for the solution of finite-element systems, HSL_MP42, HSL_MP43 and HSL_MP62. These are MPI based versions of the serial routines MA42, MA43 and MA62. The library, which has a long history, is marketed by a combination of CLRC, AEA Technology and Hyprotech. It is free to UK academics for academic purposes.

PESSL

<http://usgibm.lbl.gov/pessl/>

Parallel Engineering and Scientific Subroutine Library (PESSL)

Parallel ESSL is a scalable mathematical subroutine library that supports parallel processing applications on IBM RS/6000 SP Systems and clusters of IBM RS/6000 workstations. Parallel ESSL supports the Single Program Multiple Data (SPMD) programming model using either the Message Passing Interface (MPI) signal handling library or the MPI threaded library. Parallel ESSL provides subroutines in six major areas of mathematical

computations.

Parallel ESSL provides subroutines in the following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms
- Random Number Generation

The subroutines run under the AIX operating system and can be called from application programs written in Fortran, C, C++, and High Performance Fortran (HPF). On the SP, Parallel System Support Programs (PSSP) is also required.

For communication, Parallel ESSL includes the Basic Linear Algebra Communications Subprograms (BLACS), which use the Parallel Environment (PE) Message Passing Interface (MPI). Communications using the User Space (US) require either the High Performance Switch or SP Switch. Communications using the Internet Protocol (IP) may use Ethernet, Token Ring, Fiber Distributed Data Interface (FDDI), High Performance Switch or SP Switch. For computations, Parallel ESSL uses the ESSL for AIX subroutines.

PETSc

<http://www-fp.mcs.anl.gov/petsc/>

PETSc (Portable Extensible Toolkit for Scientific Computation) is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication.

PIM

<http://www.mat.ufrgs.br/~rudnei/pim/pim-i.html>

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.

PINEAPL

<http://www.nag.co.uk/projects/PINEAPL/>

The PINEAPL (Parallel Industrial NumERical Applications and Portable Libraries) project is a 31.4 man-year, European Commission funded project in the area of High Performance Computing and Networking (HPCN). PINEAPL is intended to benefit a large range of industries that make use of numerically intensive application codes for design and simulation as part of their business. Much of the project work will be publicised to encourage a wide audience to contribute useful input and commentary and help to shape the final project results.

AZTEC

<http://www.cs.sandia.gov/CRF/aztec1.html>

Aztec is a parallel iterative library for solving linear systems, which is both easy-to-use and efficient. Simplicity is attained using the notion of a global distributed matrix. The global distributed matrix allows a user to specify pieces (different rows for different processors) of his application matrix exactly as he would in the serial setting (i.e. using a global numbering scheme). Issues such as local numbering, ghost variables, and messages are ignored by the user and are instead computed by an automated transformation function. Efficiency is achieved using standard distributed memory techniques; locally numbered submatrices, ghost variables, and message information computed by the transformation function are maintained by each processor so that local calculations and communication of data dependencies is fast. Additionally, Aztec takes advantage of advanced partitioning techniques (Chaco) and utilizes efficient dense matrix algorithms when solving block sparse matrices.

Methods: CG, CGS, BiCGSTAB, GMRES, TFQMR

Preconditioners: Point and block Jacobi, Gauss-Seidel, least-squares polynomials, and overlapping domain decomposition using sparse LU, ILU, BILU within domains.

PLAPACK

<http://www.cs.utexas.edu/users/plapack/>

PLAPACK codes dense linear algebra algorithms at a high level of abstraction. Cholesky, LU and QR decompositions are currently included in a library with both C and Fortran bindings, although the Fortran binding is still under construction. A book on PLAPACK, which also discusses the components and design of a high-performance parallel numerical library, is available from MIT Press. PLAPACK is designed for distributed memory machines and attempts to reduce the need for user programming to a bare minimum.

MP_SOLVE

http://www.emlab2.nmsu.edu/mp_solve

The MP_SOLVE package uses LU factorisation to solve large sparse systems with multiple right-hand sides on distributed memory architectures. MPI is the default system used for communications, although BIP for Beowulf class machines can also be used, and the

sparse system must be partitioned before solution using the Chaco partitioning software from Sandia Labs. Currently the package supports HP Convex SPP-2000 and Intel-based beowulf systems with either myrinet networking + BIP low-level communications software, or ethernet networking.

MUMPS

<http://www.enseeiht.fr/apo/MUMPS>

MUMPS (MULTifrontal Massively Parallel sparse direct Solver) is public domain software partly funded by the European Community. It allows specification of the matrix in a variety of forms, from fully assembled to elemental. MUMPS is Fortran 90 and MPI based, and has a simple but powerful interface. Iterative refinement and backward error analysis are used to provide high quality solutions, and BLAS, LAPACK and ScaLAPACK are used at the lower levels to aid portability.

PSPASES

www.cs.umn.edu/~mjoshi/pspases

PSPASES stands for Parallel SParse Symmetric dirEct Solver and contains parallel implementations of each of the four primary phases of direct solution of sparse systems: the computation of fill-reducing ordering, symbolic factorisation, numerical factorisation, and the solution of triangular systems of equations. The library is callable from both C and Fortran90 and uses MPI, LAPACK and BLAS.

SPOOLES

www.netlib.org/linalg/spooles/spooles.2.2.html

SPOOLES is a SParse Object Oriented Linear Equations Solver. It is implemented in C but follows an Object Oriented paradigm. It is possible to use SPOOLES to compute the minimum degree, generalized nested dissection and multisection orderings of matrices with symmetric structure, to factor and solve square symmetric linear systems of equations either in serial mode, multithreaded with Solaris or POSIX threads, or with MPI, to factor and solve overdetermined full rank systems of equations using multifrontal QR in serial or using POSIX threads and to solve square linear systems using a variety of Krylov iterative methods.

SuperLU

www.nersc.gov/~xiaoye/SuperLU

SuperLU solves large sparse non-symmetric systems of equations using LU decomposition with partial pivoting. It is written in C and is callable from C or Fortran. Three varieties of SuperLU cover serial, shared memory and distributed memory architectures. These

differ slightly in the way in which data is supplied to the solvers and in the precision they support.

BlockSolve95

<http://ww.mcs.anl.gov/sumaa3d/BlockSolve/>

BlockSolve95 is designed to solve sparse (structurally symmetric) systems with a block structure which arise, for example, from finite element models with multiple degrees of freedom per node. A high level of portability has been obtained by use of MPI for message passing. LAPACK and BLAS are also used and must be present on the computer to use BlockSolve95.

IML++

<http://math.nist.gov/iml++>

The Iterative Methods Library (IML++) is a C++ library of modern iterative solvers for both symmetric and non-symmetric systems of equations. It is fully templated so that the same user code works for dense, sparse and distributed matrices.

ISIS++

<http://ziggurat.ca.sandia.gov/isis/>

ISIS++ is a portable, object-oriented framework for solving sparse systems of linear equations. The framework includes implementations of a number of Krylov subspace iterative solution methods (CG, CGS, BiCGStab, QMR, GMRES(m), DefGMRES(m), FGMRES(m), CGNE (normalised equations), CGNR (normalised residuals)) and preconditioners (diagonal, block Jacobi, polynomial (Neumann and least squares), sparse parallel approximate inverse (SPAI), composed (combines polynomial with block Jacobi or SPAI)) as well as both uni-processor and multi-processor matrix and vector classes. It is designed to facilitate integration of components from various other sparse libraries.

P-SparseLIB

http://www.cs.umn.edu/Research/arpa/p_sparslib/sps-abs.html

P-SparseLIB is a library of sparse iterative solvers designed for distributed memory architectures. There are four main sections of the library: accelerators based on Krylov subspace methods, preprocessing tools, preconditioning routines, and message-passing tools. Reverse communication is used to give flexibility, independence of user data-structures and portability. Thus whenever a Matrix-vector multiply is required the P-SparseLIB routine returns to the calling code with a request and the multiplication is handled by a user supplied routine (which may use the message passing routines of the library). Then the library routine is recalled with the result and can continue with the computation. P-SparseLIB is written mainly in Fortran77 with a little C.

5.2 Eigenvalue type problems

PRISM

<http://www-unix.mcs.anl.gov/prism/>

The goal of the PRISM (Parallel Research on Invariant Subspace Methods) project is to develop infrastructure and algorithms for the parallel solution of eigenvalue problems. It has developed a complete eigensolver based on the Invariant Subspace Decomposition Algorithm for dense symmetric matrices (SYISDA).

SYISDA (SYmmetric Invariant Subspace Decomposition Approach) is a parallel eigensolver package. The routines in this package allow you to find all the eigenvalues and eigenvectors of real symmetric diagonalisable matrices and can easily be adapted to only resolve a certain part of the spectrum. All routines use MPI for their message passing. As a result, they should run on any machine which supports MPI. The code has been tested extensively on a wide variety of parallel architectures and test cases. There is also a User's Guide which provides directions on installing the package and explains the user interface.

ARPACK and PARPACK

http://www.caam.rice.edu/software/ARPACK/http://www.caam.rice.edu/~kristyn/parpack_home.html

ARPACK is a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems.

The package is designed to compute a few eigenvalues and corresponding eigenvectors of a general n by n matrix A . It is most appropriate for large sparse or structured matrices A where structured means that a matrix-vector product $w \leftarrow Av$ requires order n rather than the usual order n^2 floating point operations. This software is based upon an algorithmic variant of the Arnoldi process called the Implicitly Restarted Arnoldi Method (IRAM). When the matrix A is symmetric it reduces to a variant of the Lanczos process called the Implicitly Restarted Lanczos Method (IRLM). These variants may be viewed as a synthesis of the Arnoldi/Lanczos process with the Implicitly Shifted QR technique that is suitable for large scale problems. For many standard problems, a matrix factorization is not required. Only the action of the matrix on a vector is needed. ARPACK software is capable of solving large scale symmetric, nonsymmetric, and generalized eigenproblems from significant application areas. The software is designed to compute a few (k) eigenvalues with user specified features such as those of largest real part or largest magnitude. Storage requirements are on the order of $n \times k$ locations. No auxiliary storage is required. A set of Schur basis vectors for the desired k -dimensional eigen-space is computed which is numerically orthogonal to working precision. Numerically accurate eigenvectors are available on request.

PLANSO

<http://www.nersc.gov/research/SIMON/planso.html>

PLANSO is a parallel version of LANSO, a Lanczos iteration code for symmetric eigenvalue problems. An SPMD approach is taken with distribution of the Lanczos vectors across processors to give good parallelisation of the dot products and SAXPY operations at the heart of the algorithm. The interface to PLANSO is in Fortran77.

TRLAN

<http://www.nersc.gov/research/SIMON/trlan.html>

TRLAN implements the Thick Restart Lanczos algorithm to find extremal eigenvalues. The bulk of the parallelisation is achieved through a user supplied matrix-vector multiplication subroutine. The interface to TRLAN is in Fortran90.

5.3 Adaptive or Self Optimising Libraries

ATLAS

<http://www.netlib.org/atlas/>

The ATLAS (Automatically Tuned Linear Algebra Software) project is an ongoing research effort focusing on applying empirical techniques in order to provide portable performance. At present, it provides C and Fortran77 interfaces to a portably efficient BLAS implementation, as well as a few routines from LAPACK.

FFTW

<http://www.fftw.org/>

FFTW is a C subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data, and of arbitrary input size. FFTW is freely available as defined by the *Free Software Foundation*.

UHFFT

http://www.cs.uh.edu/~johnsson/lacsi_reviewtalk_S.pdf

UHFFT is a C subroutine library which uses automatic algorithm selection and the generation of code from high-level descriptions of the performance of a target architecture. It has been produced as part of the LASCI project. Fast Fourier Transforms are particularly suited to the adaptive approach because the details of memory access can have a dramatic impact on algorithm performance. At the time of writing the availability of this library is unclear.

6 Conclusions

The use of libraries for basic tasks in computing has a long and distinguished history. Libraries for numerical problems such as linear solvers and eigensolvers are the result of many years of detailed research and experience. It would be folly for a programmer to ignore the advantages using such a library can confer on the development of software.

This report has indicated some of the issues to be considered when choosing a numerical library for use in High Performance software. The primary goals in using such a library are to gain rapid access (in terms of programmer time) to efficient and effective algorithms (defined by run-time behaviour). However, it may be necessary to compromise either of those if portability is important, particularly portability across a variety of High Performance Architectures.

We have given a list of some of the more popular and successful libraries available and indicated some of their properties. The aim of this report is to give developers of software sufficient information to enable them to approach making an informed choice. Thus we do not say “Choose this library”, but “Go and choose the most appropriate library for your machine, your programming style and your problem”.