



An Overview of Eigensolvers for HPCx

Dr A.G. Sunderland, Dr E.Y. Breitmoser

Daresbury Laboratory, Warrington, UK, WA4 4AD

EPCC, University of Edinburgh, UK

Abstract

This report gives an overview of parallel Eigensolvers, their underlying algorithms, requirements, functionality, suitability and availability on HPCx. Following reports will investigate the performance of some of these Eigensolvers on HPCx.

This is a Technical Report from the HPCx Consortium.

Report available from <http://www.hpcx.ac.uk/research/publications/HPCxTR0312.pdf>

© UoE HPCX Ltd 2003

Neither UoE HPCX Ltd 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.

1	<i>Introduction</i>	3
2	<i>Background to the Eigenproblem</i>	3
2.1	The Real Symmetric Eigenproblem	3
2.1.1	Definition	3
2.1.2	Eigensolver Approach	4
2.1.3	Other Methods	4
2.1.4	Complexity	5
2.1.5	Overheads	5
2.1.6	Parallelisation Issues	5
2.2	The Nonsymmetric Eigenproblem	6
3	<i>Parallel Eigensolver Software</i>	6
3.1	ScaLAPACK	6
3.1.1	Programming Issues	6
3.1.2	Comments	6
3.2	PESSL (Parallel Engineering and Scientific Subroutine Library)	6
3.3	PeIGS (Parallel Eigensystem Solver)	7
3.3.1	Programming Issues	7
3.3.2	Comments	7
3.4	BFG: A New Implementation of a Parallel Jacobi Eigensolver	7
3.4.1	Comments	8
3.5	PLAPACK (Parallel Linear Algebra Package)	8
3.5.1	Programming Issues	8
3.5.2	Comments	8
3.6	PARPACK (Parallel ARPACK)	8
3.6.1	Comments	9
3.7	PRISM (Parallel Research on Invariant Subspace Methods)	9
3.7.1	Comments	9
3.8	PJAC (Parallel JACobian Eigensolver library)	9
3.9	HJS	9
3.9.1	Comments	10
3.10	PINEAPL (Parallel Industrial NumERical Applications and Portable Libraries)	10
4	<i>Availability of the presented Eigensolvers on HPCx</i>	10
5	<i>Summary and Conclusions</i>	11
6	<i>References</i>	11

1 Introduction

The Eigenvalue problem can be found in a variety of scientific contexts. There are several algorithms around to help solve it, which differ in many ways. The underlying algorithms determine how time-consuming the calculation will be, how well it can be parallelized, how memory-intensive it will be and whether an orthogonal set of Eigenvectors can be provided.

The standard algorithms to solve the real symmetric tri-diagonal Eigenproblem are bisection and inverse iteration, the QR algorithm and the Divide & Conquer method, as discussed in Section 2.1.2. The latest addition is the MR³ algorithm [1], which is currently incorporated into the PLAPACK and LAPACK libraries. The resulting implementation allows problems of unprecedented size to be solved in a reasonable amount of time. The largest dense problem [1] solved to date is a matrix of size 128,000 x 128,000 on 256 processors with 2Gb of memory per processor. The calculation took about 8.5 hours of CPU time¹.

This report provides an overview of existing parallel Eigensolvers. Not all of the methods discussed are available on HPCx. Following reports will investigate the performance of the more relevant of these Eigensolvers on HPCx. The following Section introduces the Eigenvalue problem. Section 3 gives an overview of parallel Eigensolvers. The availability of all Eigensolvers presented will be discussed in Section 4. We summarize our findings in the final Section.

2 Background to the Eigenproblem

2.1 The Real Symmetric Eigenproblem

Many scientific application codes under development involve the computation of the standard real symmetric Eigensystem problem. In particular, this diagonalization stage in parallel quantum chemistry codes often consumes a significant percentage of overall run time. It is therefore particularly important that efficient parallel algorithms and implementations are used. Computational chemists generally require all or a subset of the spectrum and its associated Eigen-subspaces. For most HPC applications, the solution to the Eigenproblem is limited by available processing time and computer memory.

2.1.1 Definition

The Standard Eigenvalue Problem is described as

$$\mathbf{Ax} = \lambda\mathbf{x},$$

¹ Their results were obtained on a cluster of Linux workstations. Each node in the cluster consisted of a dual Intel® Pentium4 processor (2.4 GHz). The nodes were connected via a high performance network from Myricom.

where \mathbf{A} is a dense real or Hermitian matrix and λ is an Eigenvalue corresponding to Eigenvector \mathbf{x} .

The General Eigenvalue Problem is described as

$$\mathbf{Ax} = \lambda \mathbf{Bx},$$

where \mathbf{A} and \mathbf{B} are dense real (typically symmetric) or Hermitian matrices, \mathbf{B} being typically positive definite and constant throughout the whole calculation, and λ is an Eigenvalue corresponding to Eigenvector \mathbf{x} . The general Eigenvalue problem can always be reduced to the standard problem if \mathbf{B} is Hermitian positive definite. In the following we describe the complex case. For the real case, the Hermitian conjugate has to be replaced by a transpose.

- As \mathbf{B} is Hermitian positive definite, we can always express \mathbf{B} in terms of the Choleski decomposition, specifically $\mathbf{B}=\mathbf{LL}^+$, where \mathbf{L} is a lower triangular matrix.
- Hence, we have $\mathbf{L}^{-1}\mathbf{A}\mathbf{x}=\mathbf{L}^+\mathbf{x}\lambda$, therefore, $(\mathbf{L}^{-1}\mathbf{A}(\mathbf{L}^{-1})^+)\mathbf{L}^+\mathbf{x}=\mathbf{L}^+\mathbf{x}\lambda$.
- Let $\mathbf{A}'=\mathbf{L}^{-1}\mathbf{A}(\mathbf{L}^{-1})^+$ and $\mathbf{x}'=\mathbf{L}^+\mathbf{x}$, then $\mathbf{A}'\mathbf{x}'=\mathbf{x}\lambda'$, which is the standard Eigenvalue problem.

2.1.2 Eigensolver Approach

The solution to the real dense symmetric Eigensolver problem usually takes place via three main steps

1. Reduction of the matrix to tri-diagonal form, typically using the Householder Reduction.
2. Solution of the real symmetric tri-diagonal Eigenproblem via one of the following methods:
 - Bisection for the Eigenvalues and inverse iteration for the Eigenvectors,
 - QR algorithm,
 - Divide & Conquer method (D&C),
 - Multiple Relatively Robust Representations (MR³ algorithm).
3. Back substitution to find Eigenvectors for the full problem.

2.1.3 Other Methods

- The *Jacobi* method, dating back to 1846, works on the dense symmetric matrix itself, thereby dispensing of the need to reduce the problem to tri-diagonal form. It performs a sequence of plane rotations each of which

eliminates an off-diagonal element. The resulting matrix converges to the diagonal matrix of Eigenvalues. Though slower in serial, typically by a factor of three to five, it scales well and its memory requirements scales inversely with the number of processors.

- The *Symmetric Invariant Subspace Decomposition Algorithm* (SYISDA) is a divide-and-conquer approach which scales and shifts the Eigenvalue spectrum of the given matrix so that its Eigenvalues are mapped to the interval $[0,1]$. The mean Eigenvalue is mapped to $\frac{1}{2}$. An appropriate polynomial algorithm is then applied that modifies the Eigenvalues in order to yield a matrix that is significantly easier to solve than the original problem. Then an iterative scheme is applied until all Eigenvalues of the transformed matrix converge. This allows the computation of two subspaces that can then be used to decouple the original matrix into two sub-problems. The process then continues recursively with each of the two sub-problems.

2.1.4 Complexity

For an $n \times n$ matrix, the reduction and back-transformation phases require $O(n^3)$ arithmetic operations each. This is also the case for the algorithms for the tri-diagonal Eigenproblem, including the QR method, inverse iteration and the D&C method. A recent parallelisation of the MR³ algorithm by Dhillon & Parlett has a reduced $O(n^2)$ complexity for the tri-diagonal Eigenproblem.

2.1.5 Overheads

The methods that reduce the symmetric dense matrix to triangular form generally require $O(n^2)$ additional workspace in order to perform efficiently. The exception is the MR³ algorithm which requires only $O(n)$ workspace. The Jacobi method requires $O(n^2)$ workspace when the symmetric matrix is dense.

2.1.6 Parallelisation Issues

The complexity of the parallelisation and its runtime depends on the algorithms chosen for the Eigensolver and how the matrix is distributed over the processors. Inter-processor communication is always a potential bottleneck of any computation. For example, in ScaLAPACK [7] the use of *storage blocking* causes a load imbalance when the block size of the matrix is increased. An implementation with algorithm blocking but no storage blocking like HJS scales better as the block size increases [2]. Another expensive parallel operation occurs when reorthogonalization is needed (as with tightly clustered eigenvalues) where the Gram-Schmidt procedure is generally used [2,3]. A better approach is a relatively new method as described in [4]. Software based on this new algorithm is available in the LAPACK and the PLAPACK libraries.

2.2 The Nonsymmetric Eigenproblem

- Eigenvectors of nonsymmetric matrices are not, in general, orthogonal and, indeed, may not span the full space. If you need an orthogonal invariant subspace then we would advise computing the Schur vectors instead. (See the LAPACK Users' Guide for details [5].)

3 Parallel Eigensolver Software

Currently there are several parallel Eigensolvers available. Each of the following subsections presents a different Eigensolver, describing the kind of problems it is able to solve, the programming issues that are specific to the solver, and which algorithm is used.

3.1 ScaLAPACK

ScaLAPACK has drivers for solving standard and generalized dense symmetric or dense Hermitian Eigenproblems. The routines PDSYEV and PDSYEV (D&C method [6]) calculate all the Eigenvalues and optionally Eigenvectors. PDSYEVX is designed to calculate all or a subset of Eigenvalues and (optionally) the corresponding Eigenvectors. There are also sets of routines for solving the nonsymmetric Eigenproblem. To use ScaLAPACK, the BLACS, PBLAS, LAPACK and BLAS libraries are all required (these are all available on HPCx). ScaLAPACK is public domain software that can be downloaded from the University of Tennessee and Oak Ridge National Laboratory [7]. Precompiled versions of the ScaLAPACK library are available for several platforms (it is also straightforward to compile from source). The underlying algorithms used are Householder Reduction, inverse iteration, back substitution, the QR method and the D&C method.

3.1.1 Programming Issues

The ScaLAPACK routines use a block-cyclic distribution of the matrices across processors in order to ensure efficient performance and load-balancing.

The memory requirements for the solvers are generally of $O(n^2)$.

3.1.2 Comments

The workspace requirement to ensure orthogonal Eigenvectors is large.

Matrices with tightly clustered Eigenvalues require repeated, i.e. costly, re-orthogonalisations.

3.2 PESSL (*Parallel Engineering and Scientific Subroutine Library*)

PESSL [9] is a library provided by IBM, which has been optimised for IBM platforms, such as HPCx, and should therefore be faster than comparable public domain software, for example ScaLAPACK. PESSL provides solutions for real symmetric and complex Hermitian matrices both for the standard and the positive definite generalized problem. The subroutines include a subset of the ScaLAPACK subroutines, for example PDSYEVX, PDSYGVX and

PDSYGST (PZHEGST). The latter reduces a real symmetric (complex Hermitian) positive definite generalized Eigenproblem to standard form. These are designed in accordance with the proposed ScaLAPACK standard. If these subroutines do not comply with the standard IBM will consider updating them. The update may require modifications to the calling application program. When using PESSL you have to ensure that the faster PESSL routines replace the slower ScaLAPACK routines, although the argument list of the PESSL routines might not match the argument list of the ScaLAPACK routines. The ordering of the libraries when linking is crucial (see [8]).

3.3 *PelGS (Parallel Eigensystem Solver)*

PelGS provides both standard and generalized dense, real and tri-diagonal symmetric Eigensystem problem solvers and requires LAPACK and BLAS. Both expert drivers (selection of Eigenvalues and Eigenvectors) and simple drivers (full set) are available. The software can be requested from the Pacific Northwest National Laboratory [10]. Readers should be aware that PelGS is no longer under active development. The algorithms used are Householder Reduction, inverse iteration (which is optimised) and back substitution.

3.3.1 Programming Issues

PelGS uses a column based distribution of the matrix. A mapping vector records the distribution of columns to processors.

The memory requirements are of $O(n^2)$.

Compilation from the Fortran77 and C source is required.

The inter-processor communications are MPI or Global Array [11] based.

3.3.2 Comments

PelGS is included in the NWCHEM package [12].

PelGS 3.0 contains a version of Dhillon & Partlett's optimisations for the inverse iteration stage of the calculation.

The investigations of Allan and Bush [13] found that it is generally slower than ScaLAPACK, but scales better on parallel machines. Initial investigations on HPCx appear to support these findings.

3.4 *BFG: A New Implementation of a Parallel Jacobi Eigensolver*

BFG solves the standard dense symmetric, real and Hermitian Eigenproblems and requires BLAS and MPI. BFG is an iterative method, based on a Block Jacobi Eigensolver. BFG is written in Fortran90 with MPI for message passing. The library is available upon request from Ian Bush, CCLRC, Daresbury Laboratory, i.j.bush@dl.ac.uk.

3.4.1 Comments

There are interfaces for both column-based (PeIGS) and block-cyclic (ScaLAPACK) data distributions.

BFG is slower in serial than Householder based methods but shows excellent scaling. It is ideal for multiple calls to an Eigensolver, where previous solutions are a fair approximation of subsequent solutions. BFG produces very accurate Eigenvalues and Eigenvectors. The Hermitian version of BFG is not nearly as well optimized yet as the real symmetric version of BFG that employs cache blocking [14].

3.5 *PLAPACK (Parallel Linear Algebra Package)*

PLAPACK solves for dense symmetric standard Eigenproblems. It is written in Fortran and C and requires BLAS and MPI. PLAPACK was developed at the University of Texas [15]. The underlying algorithms are the QR algorithm and the MR³ algorithm.

3.5.1 Programming Issues

The PLAPACK Routines use an Abstract Programming Interface – parallel algorithms are coded with a high level of abstraction. The data is distributed across processors by columns.

The MR³ algorithm generally requires only $O(n^2)$ operations and $O(n)$ workspace.

3.5.2 Comments

The algorithm MR³ is currently only available by request from the developers. No drivers are available and therefore users have to call individual PLAPACK routines for each stage of the solution.

Small memory overheads mean larger systems can be solved.

The library offers good load balancing and excellent scaling [1].

3.6 *PARPACK (Parallel ARPACK)*

PARPACK [17] solves large-scale symmetric and nonsymmetric, generalized Eigenproblems. It is designed to compute a few Eigenvalues and corresponding Eigenvectors of large sparse systems. PARPACK is written in Fortran77 and requires BLACS and MPI. You can get the software for both the serial and the parallel code via anonymous ftp [16]. More information can be found at [17].

The software is based upon an algorithmic variant of the Arnoldi process (Implicitly Restarted Arnoldi Method or IRAM). When the matrix A is symmetric it reduces to a variant of the Lanczos process (Implicitly Restarted Lanczos Method or IRLM). The 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 factorisation is not required.

3.6.1 Comments

PARPACK is most appropriate for large, sparse or structured matrices, where structured means that a matrix-vector product requires $O(n)$ rather than the usual $O(n^2)$ floating point operations.

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 $n*k$ locations. No auxiliary storage is required. A set of Schur basis vectors for the desired k -dim Eigenspace is computed which is numerically orthogonal to working precision. Numerically accurate Eigenvectors are available on request.

3.7 *PRISM (Parallel Research on Invariant Subspace Methods)*

PRISM solves for dense symmetric Eigensystems and requires LAPACK, BLAS and MPI. PRISM is available from the MCS division of the Argonne National Laboratory [18] and uses the Invariant Subspace Methods SYISDA (D&C approach). It is written in C. There is no Fortran interface available.

3.7.1 Comments

PRISM has a virtual 2D torus wrap (cyclic) data layout [19] that is compatible with HPF data layouts.

PRISM utilises optimised matrix-matrix multiplications for the bulk of the computations, thereby ensuring good FLOP counts

3.8 *PJAC (Parallel JACobian Eigensolver library)*

The PJAC library provides a scalable Eigensolver for symmetric Hermitian matrices to solve the standard problem. Problem sizes ranging from $N = 2$ to 8192 have been solved. Information can be found at the website [20] of the University of Pittsburgh which will also provide the source code on request. PJAC is currently installed on the T3E in Pittsburgh. The algorithm employed is essentially a hybrid of Jacobi methods. The parallel Jacobi method is described by Brent and Luk. Note, that the current implementation requires the size of the input system to be an even multiple of the specified processor count.

3.9 *HJS*

The HJS algorithm that solves real, dense symmetric matrices focuses on the reduction to tri-diagonal form and the back-transformation of Eigenvectors. HJS solves the standard problem. The solver requires BLAS. However, the parallel HJS algorithm developed by Hendrickson, Jessup and Smith [2] from Sandia National Lab, USA, for the Intel Paragon, was created to clarify questions concerning efficiency, and therefore was not intended to be used as a public library routine.

3.9.1 Comments

ScaLAPACK is the most similar package to HJS. The underlying algorithms are Householder tri-diagonalisation, back-transformation and the modified Gram-Schmidt procedure.

A nonsymmetric, outer-product update of a lower triangular matrix needs to be performed. No corresponding BLAS 3 routine is available, which results in performance loss.

To ensure orthogonal Eigenvectors the modified Gram-Schmidt procedure is used, an $O(n^3)$ algorithm. They use a square-torus wrap instead of an arbitrary rectangular block. This restricts the code to runs on a perfect square number of processors. This may explain why the tri-diagonalization is significantly faster than for ScaLAPACK.

Tri-diagonalization is a great deal more costly than the back-transformation. Typically, HJS tri-diagonalization is substantially faster than for ScaLAPACK while the back-transformation is somewhat slower.

3.10 PINEAPL (*Parallel Industrial NumErical Applications and Portable Libraries*)

PINEAPL solves real or Hermitian, symmetric or nonsymmetric, dense, standard Eigenvalue problems. PINEAPL is written in Fortran77 and requires MPI, BLACS and BLAS. Information about PINEAPL can be found at [21]. The library alone is no longer available, however, most of the software is now included in the NAG Parallel Library [22]. Most of the dense solvers in the NAG Parallel Library are based upon the same algorithms as ScaLAPACK or on the one-sided Jacobi method. No drivers equivalent to those for ScaLAPACK are available, though.

NAG requires a license, and it is available for universities through CHEST [23].

4 Availability of the presented Eigensolvers on HPCx

ScaLAPACK and PESSL are both available on HPCx for all users. More information can be found in the *User's Guide to the HPCx Service* [8].

Several Eigensolvers are being tested on HPCx but are currently not accessible for all users. They can be made publicly available on request. We recommend contacting the HPCx Helpdesk helpdesk@hpcx.ac.uk if you are interested in using PeIGS, BFG, PLAPACK or PARPACK.

PRISM and PJAC are not available on HPCx at the moment. Further information can be obtained from [19] and [20], or from the HPCx Helpdesk.

NAG, and hence PINEAPL, are currently not available on HPCx.

HJS is not available on HPCx, as it is not intended as a set of public library routines.

5 Summary and Conclusions

In this report we discussed the characteristics, underlying algorithms, possible programming or other issues for the parallel Eigensolvers ScaLAPACK, PESSL, PeIGS, BFG, PLAPACK, PARPACK, PRISM, PJAC, HJS and PINEAPL.

All parallel Eigensolvers presented in this report are for dense systems except PARPACK which is only suitable for sparse systems.

ScaLAPACK solves many kinds of dense Eigenvalue problems. However, if it suits your problem and provides the necessary routines, we would recommend using PESSL instead, as it is optimised for IBM.

It will definitely be worth keeping an eye on the development of the new MR³ algorithm in PLAPACK, promising both less operations and less workspace.

The authors would like to thank Gavin Pringle and Ian Bush for their assistance in compiling this report.

6 References

1. P.Bientinesi, I.S.Dhillon, R.A.van de Geijn, A Parallel Eigensolver for Dense Symmetric Matrices based on Multiple Relatively Robust Representations, UT CS Technical Report #TR-03026, 2003, <http://www.cs.utexas.edu/users/plapack/papers/pareig.ps>.
2. B.Hendrickson, E.Jessup, C.Smith, Toward an efficient parallel Eigensolver for dense symmetric matrices, SIAM J. SCI. COMPUT., Vol.20, No. 3, pp. 1132-1154 (1999).
3. H.Y.Mussa, J.Tennyson, Bound and quasi-bound rotation-vibrational states using massively parallel computers, Computer Physics Communications 128 (2000) 434-445.
4. I.Dhillon, A new $O(n^2)$ algorithm for the symmetric tridiagonal Eigenvalue/Eigenvector problem, Ph.D. thesis, University of California, Berkeley, 1997.
5. http://www.netlib.org/lapack/lug/lapack_lug.html.
6. F.Tisseur and Jack Dongarra, A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures, SIAM J. SCI. COMPUT, Vol.20, No. 6, pp. 2223-2236 (1999).
7. www.netlib.org.
8. *User's Guide to the HPCx Service*, <http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/HPCxuser.html>.
9. <http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2272734.pdf>.
10. <http://www.emsl.pnl.gov/docs/nwchem/doc/peigs/docs/peigs3.html> (G.Fann et al).

11. <http://www.emsl.pnl.gov/docs/global/>.
12. <http://www.emsl.pnl.gov/docs/nwchem/nwchem.html>.
13. Parallel Application Software on High Performance Computers – Parallel Diagonalisation Routines. The CCLRC HPCI Centre at Daresbury Laboratory, 1996.
http://www.dl.ac.uk/TCSC/Subjects/Parallel_Algorithms/diagreport/
14. *Ian Bush, private communication.*
15. <http://www.cs.utexas.edu/users/plapack/>.
16. <ftp://ftp.caam.rice.edu/pub/software/ARPACK>.
17. http://www.caam.rice.edu/~kristyn/parpack_home.html.
18. <http://www-unix.mcs.anl.gov/prism/index.html>.
19. http://www.cs.sandia.gov/~dewomb/torus_wrap_mapping.html.
20. <http://www.psc.edu/general/software/packages/pjac/pjac.html>.
21. <http://www.nag.co.uk/projects/PINEAPL>.
22. The documentation for the NAG Parallel Library can be seen at
<http://www.nag.co.uk/numeric/FD/manual/html/Fdlibrarymanual.asp>.
23. <http://www.chest.ac.uk/>.