



Scalable Eigensolvers on HPCx: Case Studies

Ian Bush, Andrew Sunderland
CCLRC Daresbury Laboratory, Warrington, WA4 4AD, UK.

Gavin J. Pringle
EPCC, University of Edinburgh, EH9 3JZ, UK.

Email: i.j.bush@dl.ac.uk, a.g.sunderland@dl.ac.uk, g.pringle@epcc.ed.ac.uk

Abstract

Good parallel eigensolver performance is essential for many application codes on HPCx. This report firstly briefly summarises the latest algorithms available for solving the real dense or Hermitian symmetric eigenproblem and then studies their effectiveness in five application code case studies on HPCx. For the codes covered, timings are reported for a range of processor counts using typical datasets. Drawing upon these findings, recommendations are made for code developers and code users on HPCx.

This is a Technical Report from the HPCx Consortium.

Report available from <http://www.hpcx.ac.uk/research/publications/HPCxTR510.pdf>

© UoE HPCx Ltd 2005

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
1.1	Solving the Symmetric Eigenvalue Problem	3
1.1.1	Definition	3
1.1.2	Algorithms for Computing Eigenvalues and Eigenvectors	4
1.1.3	Algorithms for Solving the Symmetric Tridiagonal Eigenproblem	5
2	<i>Scaling Eigensolvers in Application Codes on HPCx</i>	6
2.1	GAMESS-UK	6
2.2	PRMAT	9
2.3	PLATO	12
2.4	SIESTA	13
2.5	PDVR3DRZ	14
3	<i>Conclusions</i>	16
4	<i>Acknowledgements</i>	17
5	<i>References</i>	17

1 Introduction

Eigenvalue and eigenvector computations arise in a wide range of scientific and engineering applications. For example, in quantum chemistry the computation of eigenvalues may be required in order to calculate electronic energy states whilst in engineering the eigenvalues of a system may represent the natural frequencies of vibration of a structure. For large-scale application codes such as these, a highly efficient algorithm for calculating the eigenvalues and eigenvectors of a matrix in parallel is essential if fast, scalable performance is to be achieved.

1.1 Solving the Symmetric Eigenvalue Problem

Many scientific application codes under development involve the computation of the symmetric real or Hermitian eigenvalue 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 applications the solution of the eigenproblem is limited by available processing time and computer memory. A summary of methods for tackling the symmetric eigenvalue problem is given in [1], however it is useful to give a brief overview here of the characteristics and computational properties of the specific methods discussed in the later sections of this report.

1.1.1 Definition

The Standard Eigenvalue Problem is described as

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

where \mathbf{A} is a matrix and λ is the eigenvalue corresponding to eigenvector \mathbf{x} .

The generalized eigenvalue Problem has a number of forms. The most common in scientific computing is

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

where \mathbf{A} and \mathbf{B} matrices, and again λ is an eigenvalue corresponding to eigenvector \mathbf{x} . This generalized eigenvalue problem can always be reduced to the standard problem if \mathbf{B} has an inverse.

As noted above in scientific applications \mathbf{A} is typically hermitian, and this should be assumed from now on unless otherwise stated. For the generalized case it is also usual that \mathbf{B} is hermitian and positive definite. These two properties allow this form of the generalized eigenvalue problem to be readily reduced to the standard problem whilst retaining the hermiticity of the matrix to be solved (see [1] for details). Consequently we now focus of the standard hermitian (or real symmetric, as all real symmetric matrices are hermitian) problem.

The eigenvalues and eigenvectors of a hermitian matrix have a number of useful properties that are summarized here:

1. The eigenvectors of a hermitian matrix form a complete set, i.e. for an order N matrix there are exactly N distinct normalized eigenvectors
2. All the eigenvalues are real
3. For non-degenerate eigenvalues the corresponding eigenvectors are orthogonal
4. For degenerate eigenvalues the corresponding eigenvectors may always be chosen to be orthogonal. In scientific computing orthogonal eigenvectors are usually the most convenient choice

1.1.2 Algorithms for Computing Eigenvalues and Eigenvectors

The solution to the real or hermitian 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 transformation to find the eigenvectors for the full problem from the eigenvectors of the tridiagonal problem.

Steps 1. and 3. are common to all the algorithms discussed here. For step 1., employing Householder transformations to reduce the original matrix to tridiagonal form costs about $4/3n^3$ operation. Back-transformation in Step 3. requires $2n^3$ operations. The cost of solving the tridiagonal eigenproblem in Step 2. varies from $O(n^2)$ to $O(kn^3)$ ($4 < k < 12$) according to the method used and the numerical characteristics of the matrix. Moreover, Steps 1 and 3 can exploit highly-optimised matrix-multiply operations whereas Step 2 cannot. Therefore the tridiagonal eigensolve usually represents the computational bottleneck in the solution of eigenproblems and it is this stage of the calculation that has received the most attention from developers of eigensolver algorithms. Four alternative approaches to the tridiagonal eigensolve are listed above and their parallel implementations are discussed in more detail in the next section.

1.1.3 Algorithms for Solving the Symmetric Tridiagonal Eigenproblem

The QR Algorithm

The QR method [2] [3] is an iterative process by which the off-diagonal elements of the tridiagonal matrix are eliminated. At each iteration i a shift is applied to the tridiagonal matrix $(T_i - \sigma_i I)$ in order to accelerate convergence. The QR algorithm takes $O(n^2)$ operations for finding all the eigenvalues of a tridiagonal matrix and, on average, $6n^3$ operations to find the eigenvectors. In ScaLAPACK [4] the routine PxSTEQR2 computes all eigenvalues and, optionally, all eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The *simple* ScaLAPACK driver routine PxSYEV can be used on full real symmetric or hermitian symmetric matrices to compute all three stages of the eigensolve (described above) with PxSTEQR2 as the underlying routine used for solving the tridiagonal matrix.

Bisection and Inverse Iteration

The bisection method is a root-finding algorithm which works by repeatedly dividing an interval in half and then selecting the subinterval in which the solution exists. Givens [3] [6] applied the method to the tridiagonal eigenproblem to find all or just a subset of the eigenvalues. When k is the number of eigenvalues desired, nk operations are required. Thus the bisection method could be much faster than the QR method when $k < n$. It can be highly accurate, but may be adjusted to run faster if lower accuracy is acceptable.

Inverse iteration [7] is a method based on power iteration that can be used to calculate the k corresponding eigenvectors from the k eigenvalues computed by bisection.

Bisection and inverse iteration perform well ($O(nk)$ operations) if the eigenvalues are well separated. However, when the problem involves large clusters of tightly packed eigenvalues, Gram Schmidt orthogonalizations are required to separate eigenvectors. This operation does not parallelize particularly well and can add up to $O(nk^2)$ operations.

Both the Peigs (Parallel Eigensystem Solver) [8] and the ScaLAPACK package provide drivers for using bisection and inverse iteration to obtain all or a subset of eigenpairs. The ScaLAPACK expert² driver PxYEVX calls PxSTEBZ for bisection and PxSTEIN for inverse iteration. The most recent version of the PeIGS simple driver PxSPEV and expert driver PxSPEVX calls an adapted version of PxSTEIN based on the Berkeley algorithm developed by Parlett, Dhillon and Fann [9].

Divide-and-Conquer Algorithm

The divide-and-conquer approach divides the triangular matrix into two parts, solves the eigenproblem for each part recursively and combines the solution to each sub-

¹ The driver is referred to as simple as the eigensolver always calculates all Eigenvalues and (optionally) all Eigenvectors

² The routine is referred to as expert as the user may define a subset of eigenpairs to be obtained.

problem to give a solution to the original problem [10]. ScaLAPACK v1.7 includes the divide-and-conquer based driver PxSYEVD. This approach parallelises very well and can be several times faster than either of the two previous methods described above. One drawback is the large amount of workspace required (between $2n^3$ and $3n^3$).

The Multiple Relatively Robust Representation Algorithm

MR³ is an approach developed recently by Dhillon [11]. This algorithm applies a shift σ to T near each cluster of eigenvalues and computes an LDL^T factorization of (T - σ I). A suitable choice of σ eliminates the need for costly re-orthogonalizations when the eigenvalues are tightly clustered. The method is based upon the property that small relative changes in entries of L and D cause small relative changes in each eigenvalue of T. The algorithm requires $O(nk)$ operations, where k is the number of eigenpairs to be found. This computational requirement is much lower than the other methods listed here. The MR³ algorithm also has the added advantage of requiring only $O(n)$ workspace, thereby increasing the maximum size of system that can be solved. A serial version is available as the routine xSTEGR in LAPACK, and a beta-version parallel implementation is available in PLAPACK [12] upon request. The performance of the routine in PLAPACK has already been examined in an earlier HPCx Technical Report [13]. The upcoming release of ScaLAPACK [14] will contain a parallel version of MR³. Therefore users who already use ScaLAPACK eigensolvers should be able to switch to MR³ with very little disruption.

2 Scaling Eigensolvers in Application Codes on HPCx

2.1 GAMESS-UK

GAMESS-UK [15] represents a typical established electronic structure code, comprising some 800K lines of Fortran that permits a wide range of computational methodology to be applied to molecular systems. Two of the most commonly used methods, Hartree-Fock (HF) and Density Functional Theory (DFT), are iterative methods that require the diagonalization of at least one matrix at each step of the algorithm. These diagonalizations are the limiting factor in the scalability of the method.

GAMESS-UK has two parallel implementations of HF and DFT. The first, and earlier, implementation uses the Global Array (GA) toolkit [16] from the Environmental Molecular Sciences Laboratory at PNNL, in Richland, Washington. This provides a portable "shared-memory" programming interface for distributed-memory computers. Within this implementation the PeIGS diagonalizer is used, and Lapi [17] [18] is used for communication internally. The alternative is a distributed memory implementation that uses ScaLAPACK to diagonalize the matrices, and MPI for communication. This second implementation allows the user to specify which ScaLAPACK diagonalizer is to be used, either PDSYEV or PDSYEVD.

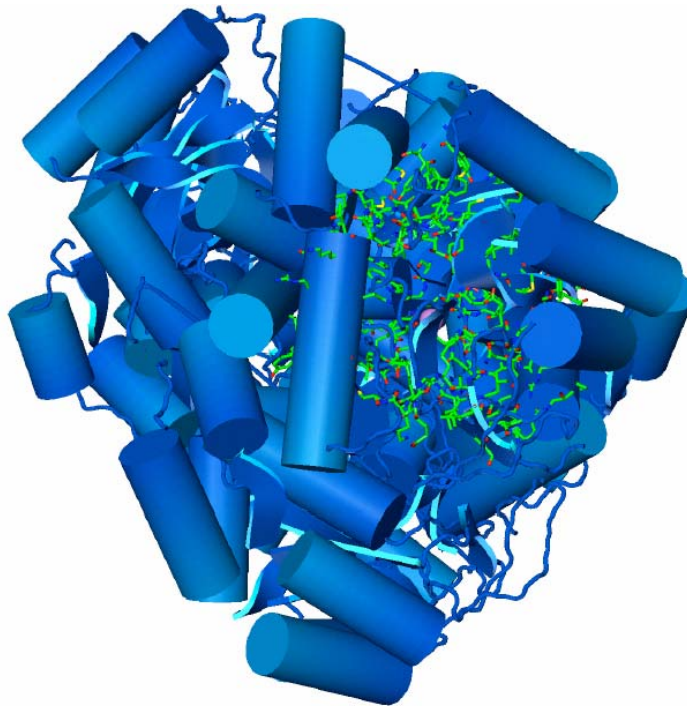


Figure 1. The Isocitrate Lyase Molecule

To study the behaviour of the diagonalization routines within the application two problems were studied. Both were clusters from the isocitrate lyase molecule, an enzyme which is associated with antibiotic resistant types of tuberculosis. This molecule is shown in Figure 1. The smaller cluster, 12Å in diameter, requires matrices of order 3970 to be diagonalized, while the larger (16Å) uses matrices of order 9636. The smaller case allows comparisons of the ScaLAPACK and PeIGS diagonalizers, but the 16Å case is too large to be run by the GA code. We also compare the Lapi implementation of PeIGS with that based upon MPI. For the MPI case this was performed by storing the matrices on disk and diagonalizing them in a stand alone code as GAMESS-UK does not provide an option to use this method.

Figure 2 shows the performance of the various methods for the 12Å (order 3970) case. It can be seen that the ScaLAPACK routine PDYSEVD is by far the best performing, being roughly four times faster than PDSYEV and up to twenty times faster than PeIGS running on Lapi. As for the other routines PDSYEV is the second best, but PeIGS using MPI for message passing is almost equivalent, while PeIGS over Lapi is very much the worst.

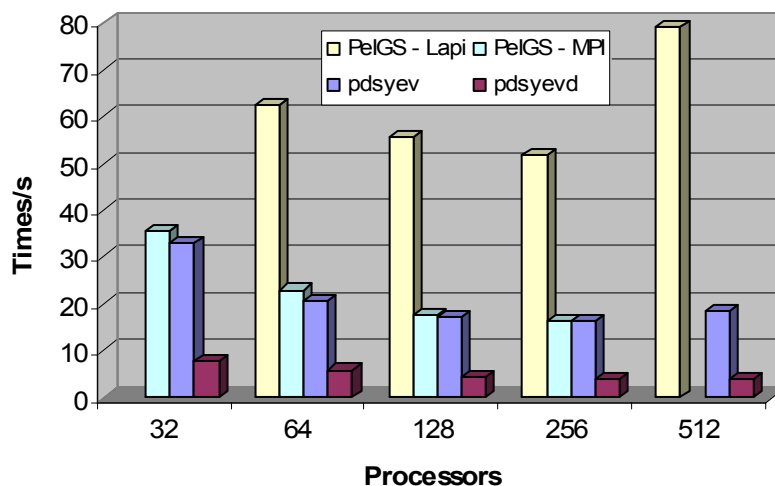


Figure 2. Parallel Eigensolver performance in GAMESS-UK

None of the routines scale well for large processor counts, and in fact by 512 processors the time is increasing in all cases measured. The scaling does improve for the larger case, shown in Figure 3, but it is still far from perfect, and in fact the best scaling routine is also the fastest, PDSYEVD.

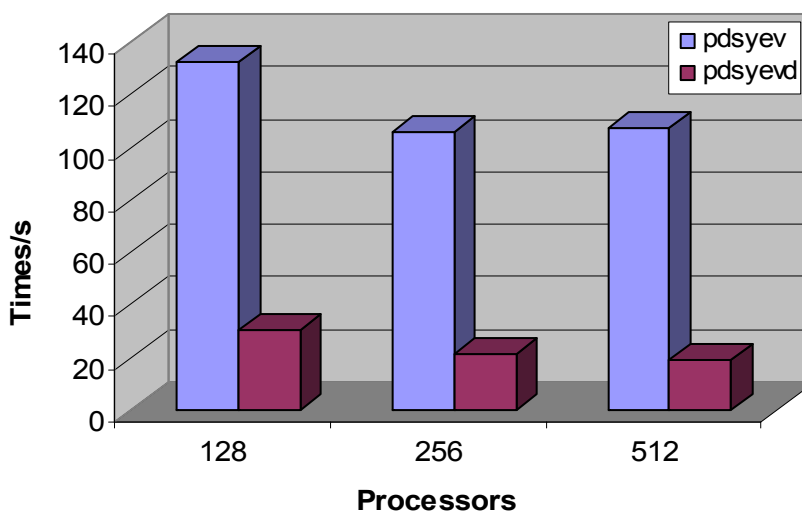


Figure 3. Performance comparison of ScaLAPACK eigensolvers in GAMESS-UK

The above discussion suggests that for GAMESS-UK the best diagonalizer to use is PDSYEVD, as it is both the fastest and least badly scaling. However, it has been found on occasions to give erroneous results and so the default in the code is PDSYEV.

2.2 PRMAT

The new parallel R-matrix program package PRMAT enables very large electron-atom and electron-ion scattering calculations of importance in many applications to start to be addressed on HPCx [19]. In particular, very large-scale calculations (500-2000 channels) are required to treat scattering of atoms in the iron peak region of the periodic table. Electron-atom collision data such as this is essential for understanding the behaviour of plasmas. Important applications include identifying forbidden lines corresponding to the excitation of Ni⁺ seen in observations of the Orion nebula (NGC 1976, shown below in Figure 4) and determining plasma diagnostics of impurities in plasma fusion tokamaks.

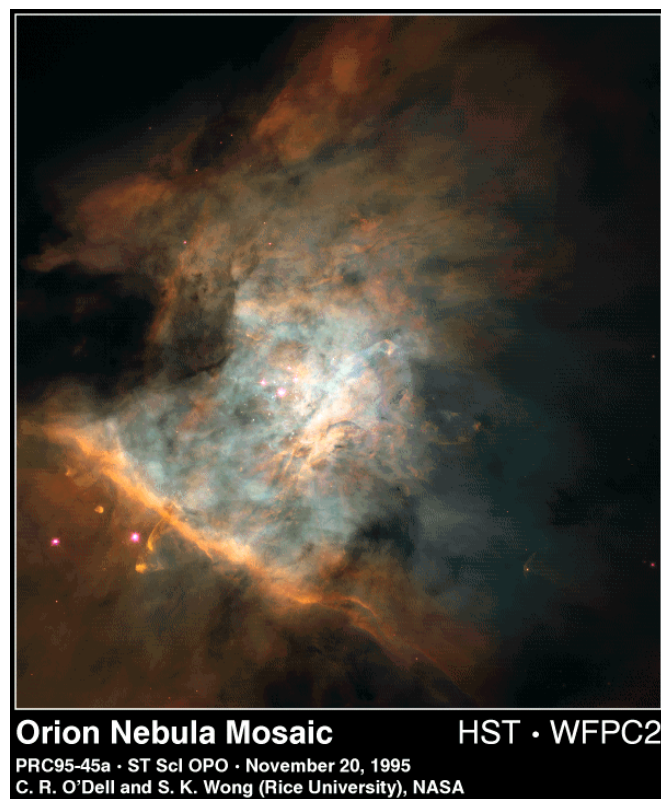


Figure 4. The Orion Nebula (photographed from the Hubble Space Station).

R-matrix theory partitions configuration space into an *Internal Region*, where exchange effects between the scattered electron and target nucleus are included in the calculation, and an *External Region* where these effects can be neglected. The basis for the calculations in both regions is the non-relativistic Schrödinger equation, which describes the scattering of an electron by an N-electron atom or ion.

In both regions the Hamiltonian matrices representing the scattering system are diagonalized and are used to solve the scattering equations at all impact energies. In the External Region, outside the target electron cloud, the scattered electron

moves in the field of the target, and a set of hundreds, or even thousands, of coupled second order differential equations must be solved at each scattering energy to calculate the cross section or collision strength. A variant of the Burke-Baluja-Morgan R-Matrix propagator method is used to solve the coupled second-order differential equations defining the external-region scattering. R-matrices at successively larger radial distances are obtained using Green's functions defined within finite radial sectors.

The External Region calculation proceeds in two separate stages, using parallel programs RMX and PRM. Collectively these programs comprise the PRMAT package. Firstly, in RMX, the (symmetric) Hamiltonian matrix for each sector is distributed across all processors and diagonalized in parallel to obtain the Sector Green's functions. These are then stored as files to disk. In PRM, the processors are reconfigured into three functional groups. A first group of processors prepares the initial R-Matrices at the internal region boundary. A second group is arranged as multiple systolic pipelines with each sector delegated to one processor in each pipeline. The Sector Green's functions are read from disc and the R-matrices are then propagated along each pipeline to a specific radius. A final group of processors representing the Asymptotic Region calculate collision strengths and output results to disk. This second stage of the calculation scales very well up to large processor counts on HPCx as more processor pipelines are added.

Initially the parallel diagonalizations for the symmetric sector Hamiltonian matrices in RMX were undertaken using the PeIGS routine PDSPEV. The initial performance of the PRMAT package on the Phase1 HPCx machine is demonstrated below in Figure 5 for an Fe²⁺ calculation with three sector Hamiltonians of dimension 10032, six sector Hamiltonians of dimension 5280 and 7515 scattering energies. It can be seen in that as the processor counts increase, the time spent in RMX is increasing relative to PRM, to the point where RMX accounts for nearly half the overall run-time on 256 processors.

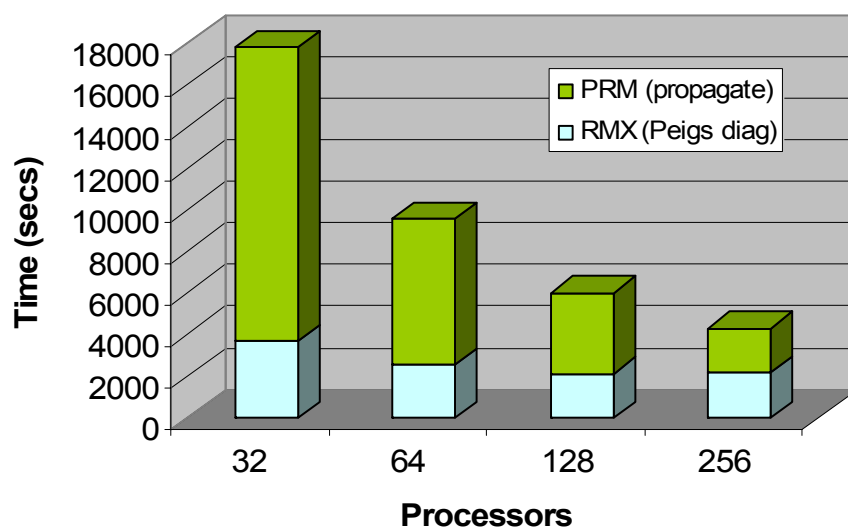


Figure 5. Initial Performance of PRMAT on HPCx Phase 1

In order to improve the performance of the initial stage of the calculation the RMX95 program was developed. This incorporates some Fortran95 features and bases the eigensolves around the parallel divide-and-conquer routine in ScaLAPACK v1.5 (PDSYEVD). This procedure involves distributing the data required to create the sector Hamiltonian matrices in 2D block-cyclic fashion rather than in column distributed fashion. The performance of the ScaLAPACK-based PRMAT on Phase1 of HPCx is shown in Fig. 4. On 64 processors, diagonalizations occurred around 4 times faster on 256 processors than the PeIGS-based version, whilst on 256 processors the ratio was 2.2. The proportion of run-time spent in the improved RMX program therefore reduced to less than a third of overall time on 256 processors.

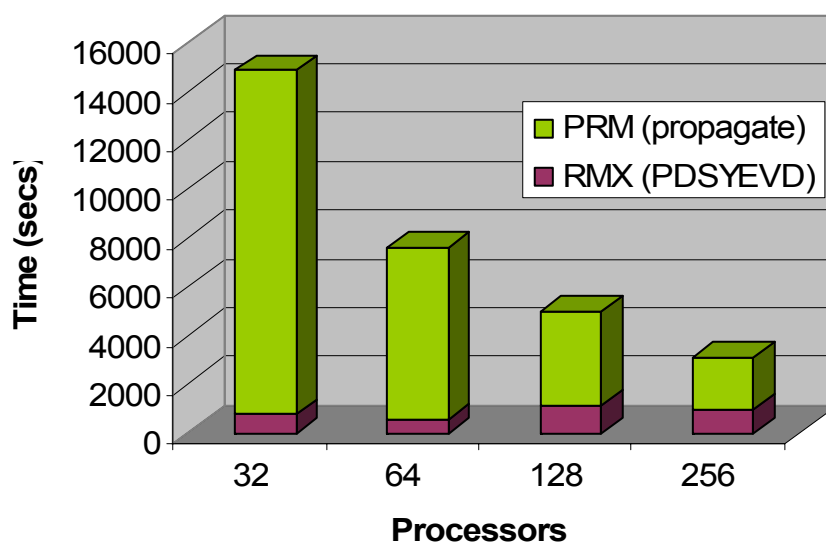


Figure 6. Performance of PRMAT on HPCx Phase1 with PDSYEVD eigensolver

The performance of PDSYEVD improved significantly after the Phase 2 upgrade and the most recent results obtained from HPCx Phase2 by the RMX95 code for the same problem are shown in Figure 7.

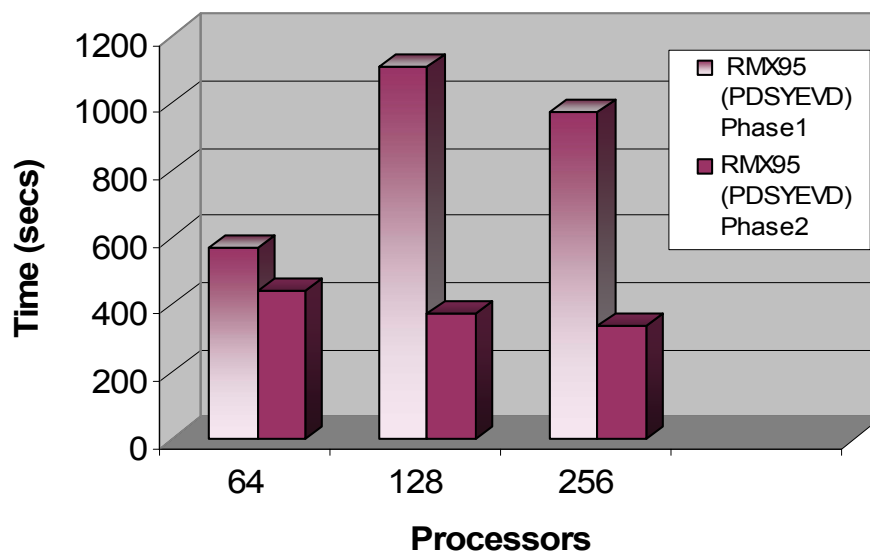


Figure 7. Phase1 vs Phase2 Performance Comparisons for RMX95

2.3 PLATO

PLATO (Package for Linear-combination of Atomic Type Orbitals) [20] is a computational chemistry code, developed by Andrew Horsfield (UCL) and Steven Kenny (Loughborough), partially funded by: HP, Fujitsu Ltd and BNFL and jointly owned by the University of Oxford and HP.

The PLATO suite contains a number of codes. The code optimised for HPCx, between May 2003 and July 2004, is called TB2 and is written in C with MPI. TB2 originally contained the option of employing 4 different serial generalised eigensolvers, specifically, DSPGV or DSPGVX, both bisection method routines, and DSPGVD, a Divide and Conquer routine, from LAPACK, or DSAUPD from ARPACK. ARPACK is suited to problems with sparse matrices and, even though the example test case given to HPCx was only 10% sparse, ARPACK was still faster than DSPGV from LAPACK.

A profiling exercise found that the fastest routine was DSPGVX, followed by the ARPACK routine. The slowest routine was DSPGV. DSPGVD was not considered at the request of the developers. Even when invoking DSPGVX, around 95% of the total execution time was spent in the serial generalised eigensolver routine. We also found that the IBM BLAS were faster than the public domain BLAS, as employed by ARPACK. Replacing LAPACK with ESSL gave no performance gains.

After considering other libraries, namely PESSL, ScaLAPACK, PARPACK, PeIGS, BFG, PINEAPL, PPARSLIB, Aztec, WSMP, SuperLU, we decided to implement parallel version of DSPGVX from ScaLAPACK. When implementing DSPGVX, during the intermediate step of replacing the serial routine DSPGVX with another

serial routine, DSYGVX, we found that this new routine ran 4 times faster, giving an overall speed up of around a factor of 3 at the expense of employing more memory.

The code now employs ScaLAPACK's DSPGVX. Further testing showed that employing PESSL instead of ScaLAPACK did not speed the code up. This is due to the fact that the current PESSL library is based on an old version of ScaLAPACK, whilst the current version of ScaLAPACK employed more advanced algorithms.

We also considered PLAPACK. To this end, we had to first convert the generalised eigenvalue problem to a Standard eigenvalue problem, however, PLAPACK provides routines to perform this conversion. We found, for our test cases, that PLAPACK's new MR³ algorithm was faster than PLAPACK's QR algorithm, however, ScaLAPACK's DSPGVX still remained the fastest routine.

2.4 SIESTA

Like GAMESS-UK and PLATO, SIESTA [21] solves approximately the electronic structure problem for many body systems. Its main focus is the use of DFT in condensed matter systems. Again, like the previous two code, this is an iterative method that requires the diagonalization of a hermitian matrix at each step.

The code was originally parallelized using ScaLAPACK to perform the diagonalization. However it was observed that the performance and scaling of the diagonalizer was appreciably worse than that observed in comparable applications, and that this performance did not improve with increasing system size. The reason for this was traced to the processor grid used in SIESTA when calling the diagonalizer. The code used a one dimensional processor grid, with the result that on large numbers of processors the maximum blocking factor, which for a one dimensional grid is N/P where N is the order of the matrix and P the number of processors, becomes too small to obtain good performance. A small blocking factor results in

- The algorithm working on very small chunks of data at any one time, and so the performance will be low
- Many short messages being sent, resulting in poor use of the communications network.

The code was modified to use a two dimensional processor grid, for which the maximum blocking factor is roughly $N/P^{1/2}$. In practice the optimal blocking factor is found to be in the range of about 20-150, and using the two dimensional grid the acceptable values for the blocking factor typically include this range. After modification it was found that the code both scaled better and achieved a higher percentage of peak performance. The results are shown in Figure 8.

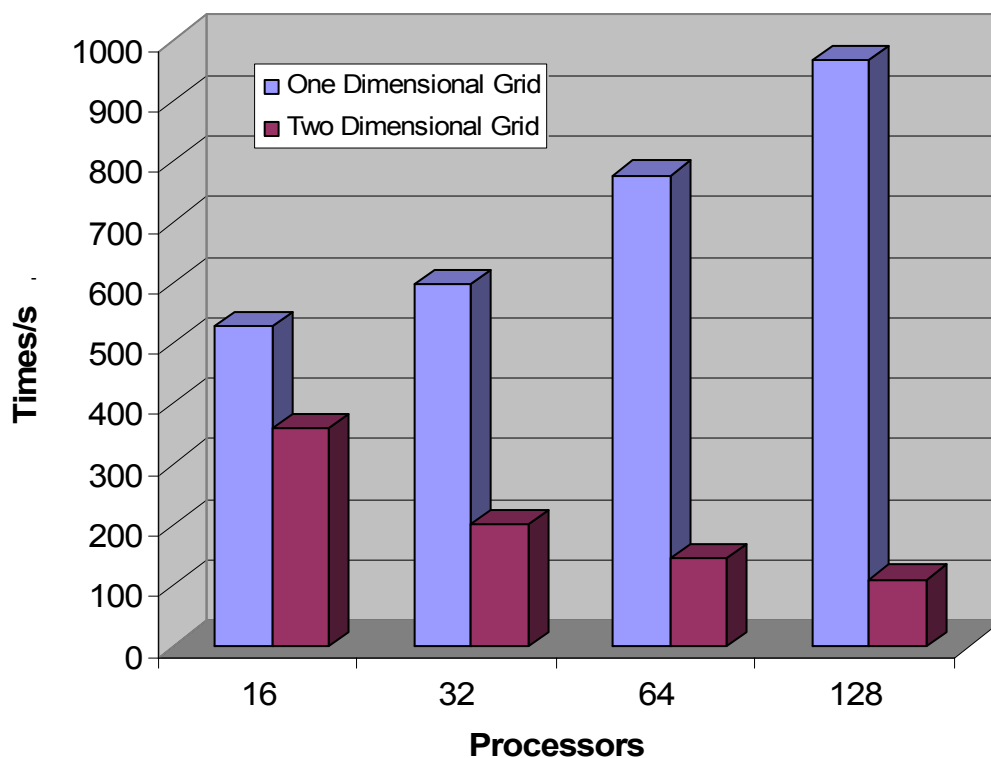


Figure 8. Eigensolver performance in SIESTA on HPCx

2.5 PDVR3DRZ

The code PDVR3DRZ is a parallel implementation of part of the DVR3D program suite [22]. As such it performs high accuracy rotation-vibration nuclear motion calculations on three atom molecules. The electron motion calculations are performed separately and form an input to the program in the form of a potential energy surface. On HPCx the research involved performing high accuracy, near dissociation calculations on H_3^+ in order to match experimental results. These calculations on HPCx have helped to understand better the behaviour of simple molecules at dissociative energies [23]. The findings from the HPCx runs are helping to solve a problem that has been under investigation for over 20 years.

The original code on HPCx used the ScaLAPACK routine PDSYEVX (Bisection & Inverse Iteration) to calculate a subset of the eigenvalues and eigenvectors of the very large Hamiltonian matrices representing the system. Although PDSYEVX is designed to compute only a subset of the eigenpairs, substituting calls to PDSYEVX with calls to PDSYEVD (Divide-and-Conquer) significantly reduced both the run-time, the memory overhead and the orthogonality and relative accuracy of the eigenvectors. Figure x below shows the parallel performance of PDVR3DRZ on HPCx using the PDSYEVD eigensolver for a typical 3D Hamiltonian matrix of order

62507 Also shown is a breakdown of the time spent in the other constituent parts of PDVR3DRZ for the specified processor counts.

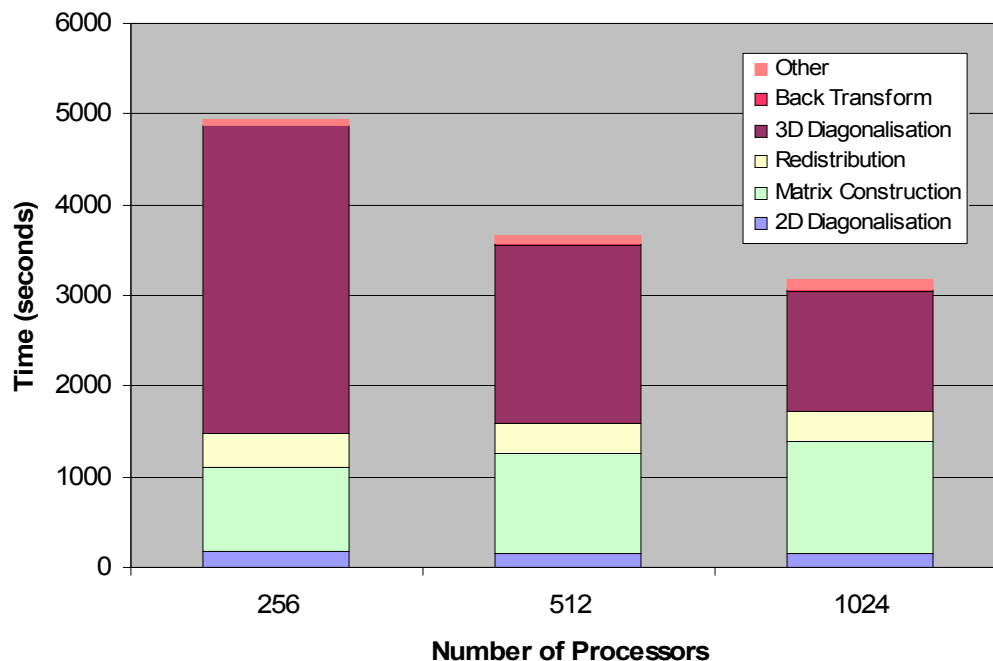


Figure 9. Breakdown of timings for PDVR3DRZ

From Figure 9 it can be seen that the parallel performance of PDSYEVD here is quite good. For example the speed-up between the 512 processor run and the 1024 processor run is around 1.5. However the other parts of the calculation do not scale nearly so well, leading to an overall speed up of only 1.15 between 512 and 1024 processors. In particular, the time spent in Matrix Construction actually increases with processor count. Rather than optimise the fine-grained parallelism in the Matrix Construction routines, the developers looked to take advantage of the coarse-grained parallelism inherent to the method in order to improve overall parallel performance. The properties that lend themselves to an efficient coarse-grained parallelization are:

1. The approach is a variational method and as the size of the calculation is increased the answer converges to a correct one. Running a number of jobs reveals the convergence behaviour and helps determine the level of accuracy of the results obtained.
2. The structure of the Hamiltonian is such that the problem can be broken down into a number of vibrational steps followed by a rotational step. For each J (the angular momentum quantum number) a number of vibrational jobs are run, based on k (another quantum number). These separate

results are then coupled together in a final job to give the final energies and wavefunctions for states of the molecule.

In order to take advantage of these properties, the underlying BLACS context defining the global process grid in the parallel code was split into contexts defining local BLACS-based sub-grids. After splitting the global BLACS grid, the global MPI communicator is split in the same fashion and the parallel code then runs to completion within each sub-grid. All communication takes place within the sub-grids of processes, apart from the global MPI and BLACS finalization call. As the problem size in each of the cases listed above remains constant, the code remains perfectly load-balanced. The overall run-times for solving a system with Hamiltonian matrices of order 80,000 with the optimised code are shown in the table below.

Number of Processors	Number of Blacs Grids	Time Taken for Completing 2 Angular Momentum Quantum Numbers (secs)
512	1	13935
1024	2 (x 512)	6822

Due to these excellent parallel scaling properties the code was recently award a Gold Star under the HPCx Capability Incentives Scheme. The PDVR3DRZ code has also been used to undertake the largest diagonalization to date of a dense real symmetric matrix on HPCx. Using PDSYEVD, a 3D Hamiltonian matrix of order 143541 was diagonalized in 10259 seconds on 1280 processors of HPCx.

3 Conclusions

For the case studies investigated in this report the parallel scaling of eigensolvers is generally limited to around the 256 – 512 processor count, representing about a quarter to half of the maximum parallel job size (1280 processors) permitted on HPCx. This is to be expected as eigensolvers generally have a low compute to communications ratio. However it can be seen from the PDV3DRZ timings that parallel scaling to 1024 processors on HPCx can be achieved with the latest eigensolver algorithms for very large ($n=62,507$) matrices. It should be noted that for eigensolvers *large* matrices are typically much larger than for other dense linear algebra operations. For the applications studied here the use of the ScaLAPACK divide-and-conquer routine PDSYEVD yields significant performance benefits over other ScaLAPACK eigensolvers and PeIGS-based eigensolvers. This can even be the case when only a subset of eigenvalues and eigenvectors are required. Code developers should note the report of very occasional wildly erroneous results associated with PDSYEVD when used in GAMESS-UK. However, no such problems have been reported to date with PDSYEVD usage in PRMAT or PDV3DRZ. One way of increasing the parallel performance of eigensolvers within an application is to take advantage of any inherent parallelism contained within the problems under

investigation. The PDV3DRZ code is a good example of this, where matrices representing different angular momentum quantum numbers can be solved concurrently within BLACS-based partitions of the global processor grid. GAMESS-UK is also able to take advantage of concurrent ScaLAPACK diagonalizations on BLACS sub-grids with certain types of problems. There are clearly similar opportunities for the RMX95 code, whereby sector Hamiltonian matrices would be diagonalized concurrently on BLACS sub-grids of processors. From the SIESTA timings it is clear that ScaLAPACK performance deteriorates significantly when data is distributed along a 1 dimensional processor grid and either very large or very small block sizes for the data distribution should be avoided.

Large workspace overheads associated with the current ScaLAPACK eigensolvers can limit the maximum problem size that users wish to solve, especially when other parts of the application code are consuming large amounts of the available memory. Two future developments may significantly ease this limitation. Firstly, the introduction of the HPCx Phase2A machine which will more than double the amount of memory available per processor and secondly, the inclusion of an MR³ algorithm implementation in the next release of ScaLAPACK [14]. Although initial results from the PLATO code using PLAPACK's implementation of the MR³ algorithm appear unpromising, it should be noted that these timings refer to beta version software, with non-optimal Householder reduction and back transformation implementations.

4 Acknowledgements

The authors would like to thank James Munro from University College London, Joachim Hein and Elena Breitmoser from EPCC, and Cliff Noble from Daresbury Laboratory for their assistance with producing this report.

5 References

- [1] *"An overview of eigensolvers for HPCx"*, Elena Breitmoser, Andy Sunderland, HPCx Technical Report HPCxTR0312, (2003)
www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0312
- [2] *"The QR transformation, parts I and II"*, Computer J., 4:265-271, 332-345, (1961-62)
- [3] *"Numerical Recipes in Fortran 77, 2nd Edition"*, B. P. Flannery, W. H. Press, Saul A. Teukolsky, W. T. Vetterling, pages 449-490, Cambridge University Press (1996)
- [4] *"A look at scalable dense linear algebra libraries"*, Scalable High-Performance Computing Conference. IEEE Computer Society Press, (1992)
- [5] www.netlib.org/scalapack/scalapack_home.html
- [6] *"The numerical computation of the characteristic values of a real symmetric matrix"*, Wallace J. Givens, Technical Report ORNL-1574, Oak Ridge National Laboratory, Oak Ridge, TN, USA, (1954)

- [7] "The calculation of specified eigenvectors by inverse iteration, contribution II/18", volume II of Handbook of Automatic Computation, pages 418-439, Springer-Verlag, New York, Heidelberg, Berlin, (1971).
- [8] <http://www.emsl.pnl.gov/docs/nwchem/doc/peigs/docs/peigs3.html> (G.Fann et al)
- [9] "A new $O(n^2)$ algorithm for the symmetric tridiagonal Eigenvalue/Eigenvector problem", I. Dhillon, Ph.D. thesis, University of California, Berkeley, (1997)
- [10] "A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures", F.Tisseur and Jack Dongarra, SIAM J. SCI. COMPUT, Vol.20, No. 6, pp. 2223-2236 (1999)
- [11] "A Parallel Eigensolver for Dense Symmetric Matrices based on Multiple Relatively Robust Representations", P.Bientinesi, I.S.Dhillon, R.A.van de Geijn, UT CS Technical Report #TR-03026, (2003)
<http://www.cs.utexas.edu/users/plapack/papers/pareig.ps>
- [12] "Plapack: Parallel Linear Algebra Package", The University of Texas at Austin, <http://www.cs.utexas.edu/users/plapack/>
- [13] "A Performance Study of the PLAPACK and ScaLAPACK Eigensolvers on HPCx for the standard problem", E. Breitmoser, A. G. Sunderland, HPCx Technical Report HPCxTR0406.pdf (2004)
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0406.pdf
- [14] "PDSYEV. ScaLAPACK's parallel MRRR algorithm for the symmetric eigenvalue problem", D. Antonelli, C. Vornel, Lapack working note 168, (2005).
<http://www.netlib.org/lapack/lawnspdf/lawn168.pdf>
- [15] Computing for Science, <http://www.cfs.dl.ac.uk/>
- [16] The Global Array Toolkit <http://www.emsl.pnl.gov/docs/global/>
- [17] "IBM General Parallel File System for AIX: Administration and Programming Reference", IBM Document Number SA22-7452-02
http://www.nersc.gov/vendor_docs/ibm/gpfs/am3admst.html.
- [18] "Introducing LAPI and Capturing the Performance of its Point-to-Point Communications on HPCx and the Colony SPswitch2", Christos Kartsaklis, HPCx Technical Report HPCxTR0402.pdf (2004)
http://www.hpcx.ac.uk/research/technical_reports/HPCxTR0402.pdf
- [19] "A parallel R-matrix program PRMAT for electron-atom and electron-ion scattering calculations", A. G. Sunderland, C. J. Noble, V. M. Burke, P. G. Burke, Computer Physics Communications 145 (2002) 311-340
- [20] "Parallelising Plato's Matrix Diagonalizations on HPCx", presentation at the PLATO Workshop, Gavin Pringle (2003)
<http://www.hpcx.ac.uk/research/hpc/presentations/PLATO.pdf>
- [21] "Improved parallel performance of SIESTA for the HPCX Phase2 system", Joachim Hein, HPCx Technical Report, HPCxTR0410 (2004)
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0410.pdf

- [22] “DVR3D: a program suite for the calculation of rotation–vibration spectra of triatomic molecules”, Jonathan Tennyson, Maxim A. Kostin, Paolo Barletta, Gregory J. Harris, Oleg L. Polyansky, Jayesh Ramanlal and Nikolai F, *Comp. Phys. Comm.* 163 p 85-116 (2004)
- [23] “Asymptotic vibrational states of the H_3^+ molecules”, James J Munro, Jayesh Ramanlal, and Jonathan Tennyson, *New J. Phys* 196 (2005)