



A performance study of the PLAPACK and ScaLAPACK Eigensolvers on HPCx for the standard problem

Elena Breitmoser, Andrew G. Sunderland

EPCC, University of Edinburgh, UK

Daresbury Laboratory, Warrington, UK

Abstract

This report compares the performance of two packages available on HPCx, PLAPACK and ScaLAPACK, for the solution of the symmetric, standard eigenvalue problem. Both PLAPACK and ScaLAPACK provide parallel eigensolvers based on the QR algorithm for dense real symmetric matrices. These packages also provide newly developed algorithms: PLAPACK contains an eigensolver based upon the Multiple Relatively Robust Representations algorithm (MR³) which has much reduced memory requirements as compared to other solvers and is therefore particularly suited for large matrices. ScaLAPACK v1.7 contains an eigensolver based on the Divide and Conquer (D&C) method (PDSYEVD).

This is a Technical Report from the HPCx Consortium.

Report available from <http://www.hpcx.ac.uk/research/hpc/HPCxTR0406.pdf>

© HPCx UoE Ltd 2003

Neither HPCx UoE 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	Introduction	1
2	Background	1
2.1	HPCx	2
2.2	The symmetric, standard eigenvalue problem	2
3	CRYSTAL	3
4	Eigensolvers	3
4.1	The PLAPACK Eigensolvers	3
4.1.1	PLAPACK on HPCx	4
4.1.2	Largest problem size on HPCx with MR ³	4
4.2	ScaLAPACK Eigensolvers	5
4.2.1	The Divide and Conquer (D&C) Eigensolver	5
4.2.2	ScaLAPACK on HPCx	5
5	Performance results for CRYSTAL Matrices	6
5.1	Comparison of MR ³ , QR and D&C	6
5.2	Time Breakdown for PLAPACK's MR ³ and ScaLAPACK's D&C	10
6	Summary	12
7	References	13

1 Introduction

In this report we investigate the performance of both parallel eigensolvers provided for the standard problem by PLAPACK (Parallel Linear Algebra Package) [1] and by ScaLAPACK (Scalable Linear Algebra Package) [7] on Phase 1 and Phase 2 of HPCx [1].

The eigenvalue problem can be found in a variety of scientific contexts and establishing an efficient solver on HPCx is often central to obtaining good parallel performance on HPCx. The underlying algorithms and portability of a range of parallel eigensolver packages available on HPCx have been discussed in a recent report [**Error! Reference source not found.**]. This report also provides more information about the eigenvalue problem. Here we will concentrate on the performance of the eigensolvers available within PLAPACK and ScaLAPACK.

In Section 2 we provide some background information about the HPCx computing system, and we define the symmetric, standard eigenvalue problem, which can be solved by PLAPACK or ScaLAPACK. The eigensolvers available within PLAPACK and the Divide and Conquer (D&C)- and QR-algorithm based ScaLAPACK eigensolvers are introduced in Section 4. A direct comparison of the four eigensolvers using typical data from the CRYSTAL software [13] is given in Section 5. In Section 5.2 we give a detailed breakdown of the contribution of the three parts of the MR³ and the D&C solvers to the total run time. Our findings are summarized in 6.

The full set of results can be found in the Tables shown in the APPENDIX A.

Terminology

We refer to the PLAPACK solver based on the Multiple Relatively Robust Representation algorithm as MR³, PMR³ or MRRR. If not otherwise stated the QR-algorithm based eigensolver refers to the PLAPACK solver. The ScaLAPACK eigensolvers investigated in this report are PDSYEVD (**p**arallel **d**ouble precision **s**ymmetric **e**igenvalue/**v**ector problem **D**&**C**), which is based on the D&C-algorithm, and PDSYEV, which is based on the QR-algorithm.

2 Background

HPCx is a high-performance computing system for parallel computing. The *University of Edinburgh*, the *Council for the Central Laboratory for the Research Councils* and IBM run the HPCx service on behalf of EPSRC. Researchers from the UK can apply for access to HPCx in order to run their computer codes on the machine.

We will introduce Phase 1 and Phase 2 of HPCx and recall the symmetric, standard eigenvalue problem in the following Subsections.

2.1 HPCx

The Phase 1 machine was composed of IBM's *p690 high-end compute nodes* and consisted of 1280 processors in total. To improve message passing efficiency over the colony switch, each physical 32-way compute node was logically partitioned into four 8-way *LPAR* nodes. The user could therefore view the system as 160 8-way shared-memory processor (SMP) compute nodes.

In April 2004 the Phase 2 HPCx service replaced the Phase1 service. The logical partition now matches the physical layout and LPARs consist of 32-way SMPs. The differences between Phase 1 and Phase 2 are highlighted in the Table 1.

Table 1: Phase2 vs Phase1 Hardware Comparison.

	HPCx – Phase 1 (pre April '04)	HPCx – Phase 2 (post April '04)
Nodes	40 Power4 p690 Frames 1.3 GHz Processors 4 × 8 way <i>LPARS</i> per Frame 1280 processors in total	50 Power4+ p690+ Frames 1.7 GHz Processors 1 × 32 way <i>LPAR</i> per Frame 1600 processors in total
Interconnect	SP Switch2 with Colony PCI adaptors	HPS Switch – direct connection to GX bus
Disk	18 Tbytes GPFS	36 Tbytes GPFS
R_{max} Linpack	3.41 Tflop/s	6.19 Tflop/s

The benchmark timings throughout this report are derived from runs on both the Phase 1 and Phase 2 machines. The data presented is labelled to distinguish timings between the two HPCx machines.

2.2 The symmetric, standard eigenvalue problem

The symmetric, real or Hermitian, standard eigenvalue problem consists of computing all or selected eigenvalues and associated eigenvectors of a symmetric matrix $\mathbf{A}_{n \times n}$, that is scalars λ_i and vectors $\mathbf{x}_i \neq 0$ satisfying

$$\mathbf{A} \mathbf{x}_i = \lambda_i \mathbf{x}_i,$$

where n is the dimension of the matrix and $i = \{1, \dots, n\}$.

A matrix \mathbf{A} is labelled *dense* when most or all matrix elements are non-zero; otherwise it is *sparse*. This report concentrates upon the eigensolution of real

symmetric dense systems, often encountered in computational chemistry codes such as Crystal (see below).

3 CRYSTAL

The CRYSTAL [7] code uses a periodic Hartree-Fock or density functional Kohn-Sham Hamiltonian and various hybrid approximations in the calculation of wavefunctions and properties of crystalline systems. In a typical large-scale calculation, around 50% of compute time involves the diagonalization of symmetric, real or Hermitian Hamiltonian matrices, where all eigenvalues and eigenvectors are required. Various of these “Crystal” matrices, representing several different calculations, have been extracted for the tests outlined in this report. The code is developed within a long-standing collaboration between the Theoretical Chemistry Group at the University of Torino, Italy, and the Computational Materials Science Group at Daresbury Laboratory, and is widely distributed internationally.

Details of the code and its applications can be found on the CRYSTAL web pages [7].

4 Eigensolvers

4.1 The PLAPACK Eigensolvers

PLAPACK currently supports a parallel implementation of the QR algorithm as well as a beta version of the newly developed MR³ (Multiple Relatively Robust Representations) algorithm. This is the first time the latter algorithm is available in a parallel version. All the eigensolvers in PLAPACK solve the dense, symmetric, real, standard eigenvalue problem and give orthonormalized eigenvectors. Within PLAPACK the user can build additional routines designed to reduce the generalized problem to the standard one and then apply any of the eigensolvers. They are implemented both in C and Fortran, use the MPI library for communications and BLAS (LAPACK) for numerical routines¹.

The eigensolver based on the QR algorithm has been available with PLAPACK for some time. It uses workspace of the order $O(n^2)$ and has to execute arithmetic operations of the order $O(n^3)$; where n being the dimension of the matrix to be solved. A major drawback of the QR algorithm is that it is hard to adapt to the case when only a subset of eigenpairs is desired [7]. The QR algorithm worked well (using CRYSTAL data on 16 and 32 processors) except for two cases (on 64 and 128 processors) where a bug was located in the original version provided by the developers. The developers have been made aware of this problem and have provided the authors with working routines.

The recent development of the MR³ algorithm, [7], offers the advantage of using only $O(n)$ of workspace and $O(n^2)$ operations for the tridiagonal eigensolve. The parallel

¹ The layering of the PLAPACK infrastructure is shown in Fig.1 of [6].

MR³ algorithm (PMR³) has not yet been officially released as part of PLAPACK but it is available on request from the developers. It is possible to calculate any subset of eigenpairs. In this report we will refer to it as either PMR³ or MR³ (or MRRR in the Figures) but the parallel version is used throughout.

We will investigate the performance of both the QR and the MR³ algorithm on HPCx. For this purpose, we will use varying matrix sizes and run the solvers on different numbers of processors. We expect to find an optimum number of processors for each matrix size. When additional processors are used, the array to be solved may be distributed so thinly over so many processors that the inter-processor communications will dominate timings. In other words, each processor will spend less time in executing its calculation than in having to communicate with the other processors in order to exchange information.

The study of the eigensolver based on the QR algorithm and the one based on the MR³ algorithm should demonstrate that much time can be saved during the tridiagonal eigensolution by using the latter algorithm as it requires $O(n)$ fewer operations than the former. In addition, less memory is required, which allows larger problems to be solved.

4.1.1 PLAPACK on HPCx

We installed PLAPACK version R3.2, PLAPACKR32, in 32-bit mode. This version was supplied to Elena Breitmoser by Paolo Bientinesi, and is needed to run the MR³ algorithm, (PLAPACKR30 contains the QR method only). At the time of writing, we also encountered problems using version PLAPACKR321, which were then confirmed by the developers on their machines. It is not possible to use PLAPACK in the 64-bit mode on HPCx, at present, and a working version cannot be expected in the near future [12]. The optimisation flag used to build the library on HPCx and used for the PLAPACK results reported is: `-O3`. Recently, PLAPACK has been installed with the same optimization flags as ScaLAPACK (see Section 4.2.2), namely `-O3 -qarch=prw4 -qtune=pwr4`. An initial investigation showed that the performance differences between the two versions are negligible.

Note that PLAPACK is currently not installed centrally on HPCx.

When compiling `-bmaxdata:0x80000000` and `-bmaxstack:256000000` has to be used to allow the maximum heap and stack sizes available on HPCx: this allows the solution of as large matrices as possible.

4.1.2 Largest problem size on HPCx with MR³

The largest matrix size we could solve with PLAPACK using the MR³ algorithm on 128 processors was $n=50,000^2$ (as opposed to $n\approx 30,000$ on 128 processors for ScaLAPACK). The total run time was 1hr 40 minutes. The reduction of the matrix to tridiagonal form took 71.85 min, the MR³ algorithm 4.87 min and the back transformation 16.12 min. We used an example, which has a random distribution of eigenvalues as described in Chapter 4.1 of [7]. The next bigger array size of $n=60,000$ we tried required too much memory in the MR³-routine `eig.c`. Please note

² Compare this with the results obtained in [7] on a cluster of Linux workstations, where a larger problem size of $n=64,000$ was solved on less processors, i.e. 64, and a size of $n=128,000$ could be solved on 256 processors .

that these results apply to the eigensolver stage alone. A realistic scenario will also include a code's additional memory needs apart from the eigensolver.

4.2 ScaLAPACK Eigensolvers

ScaLAPACK [2] has drivers for solving standard and generalized dense symmetric or dense Hermitian eigenproblems. The routines PDSYEV (QR algorithm) and PDSYEVD (D&C method [9] and below) calculate all the eigenvalues and optionally eigenvectors. PDSYEVX is designed to calculate all or a subset of eigenvalues and (optionally) the corresponding eigenvectors. Unlike PDSYEV and PDSYEVD, PDSYEVX is provided in the PESSL library supplied by IBM [11]. However it is not investigated in this report as our test problems require the computation of all eigenpairs. 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.

4.2.1 The Divide and Conquer (D&C) Eigensolver

A divide and conquer algorithm recursively reduces an instance of a problem into one or more smaller instances of the same problem. The application by Tisseur and Dongarra, [9], of Cuppen's D&C algorithm [10] to the parallel computation of the tridiagonal symmetric eigenvalue problem is an important new development. The general approach is to undertake an initial partition of the problem into subproblems compute the individual subproblems, and then combine the results together using rank-one updates. Unlike the bisection and inverse iteration approach costly re-orthogonalizations are not required when the eigenvalues are not well separated. The method is well suited to calculations where all the eigenvalues and eigenvectors are required. The back transformation stage of the ScaLAPACK D&C implementation has also been redesigned in order to take advantage of highly optimised BLAS3 operations. The latest version (v1.7) of ScaLAPACK includes the D&C-based eigensolver by Tisseur and Dongarra [9] in the routine PDSYEVD. One drawback associated with the divide and conquer implementation is the extra $2n^2$ memory required.

4.2.2 ScaLAPACK on HPCx

ScaLAPACK v1.7 is installed in both 32- and 64-bit, on HPCx in the /usr/local/lib directory. The optimisation flags used to build the library on HPCx are: -O3 -qarch=pwr4 -qtune=pwr4.

ScaLAPACK requires the BLACS library for its communications. This is also available to users from /usr/local/lib on HPCx.

5 Performance results for CRYSTAL Matrices

For the performance study we use input matrices **A** of different sizes n , on a different number of processors obtained from the CRYSTAL code [7]. The matrices are produced during a Hartree-Fock calculation and their eigenvalues tend to be tightly clustered. In this report we present results for $n=3888, 7194, 12354$ and 20480 on 16, 32, 64, 128, 256 and 512 processors on Phase 1 and Phase 2. For the PLAPACK solvers, a maximum of 256 processors was used throughout this report, since no performance gain could be established when using more than 128 processors for the matrix sizes investigated. The numerical values employed in all diagrams shown here can be found in the Tables of Appendix A.

5.1 Comparison of MR^3 , QR and D&C

In this Section we compare the results obtained with the MR^3 algorithm and the QR algorithm from PLAPACK with those obtained with the QR algorithm (PDSYEV) and D&C algorithm (PDSYEV) from ScaLAPACK.

The results for $n=3888$ are shown in Table 2 of APPENDIX A and in Figure 1 and Figure 2 for Phase 1 and Phase 2, respectively. The $n=3888$ matrix is a rather small problem on larger arrays of processors, especially in the case of the MR^3 algorithm which is designed for large problem sizes. The reduction step is very communication intensive, which becomes more pronounced as the number of processors increases. Generally, it appears that the increase in overhead for communication offsets the benefits of the extra processors for this problem size. The total time needed to solve the $n=3888$ problem is dominated by the reduction of the matrix. With an increasing number of processors, the time spent in the reduction routine, and hence the total run time, remains almost constant.

Here the optimum number of processors is 32 for the MR^3 algorithm, 64 for the QR algorithm, 128 for PDSYEV and 265 for PDSYEV on Phase 1. On Phase 2 the respective optimal array sizes are 32, 16, 128 and 128. The ScaLAPACK implementation of the QR method performs significantly faster than the PLAPACK version. If a much larger problem size is solved, the scaling pattern is generally much better for the MR^3 -based solver (Compare the results for $n=12354$ or $n=20480$ shown below).

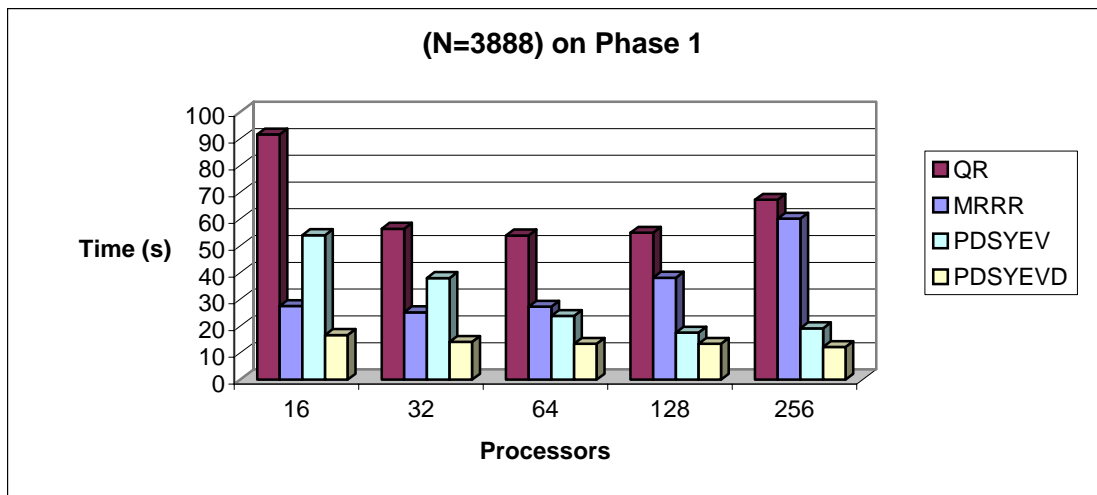


Figure 1: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEVD) algorithms, from ScaLAPACK, on Phase 1 for n=3888.

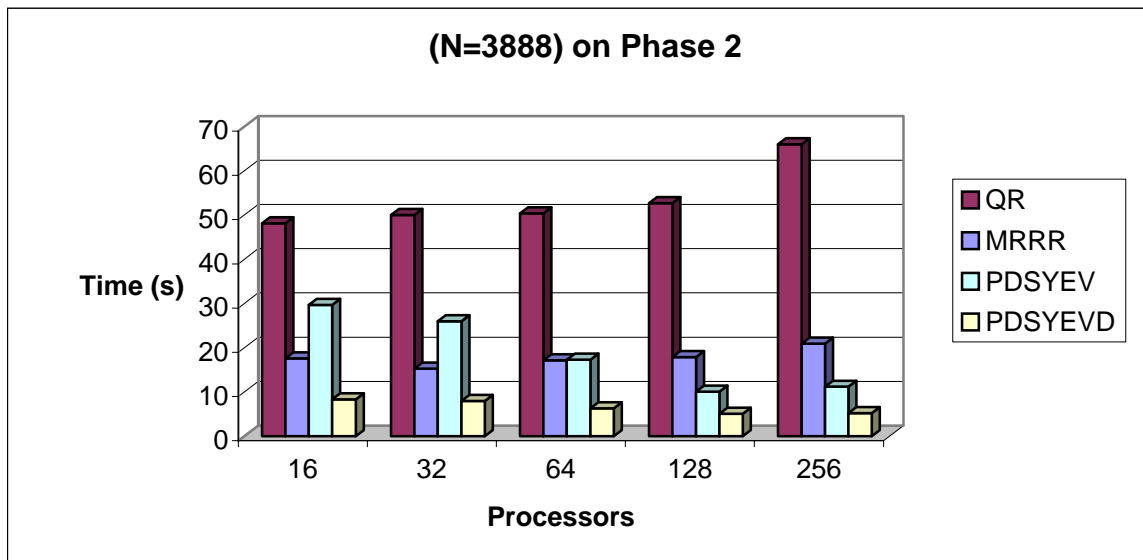


Figure 2: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEVD) algorithms, from ScaLAPACK, on Phase 2 for n=3888.

The results for n=7194 are shown in Table 3 of APPENDIX A, and Figure 3 and Figure 4 show the diagrams for Phase 1 and Phase 2, respectively. This medium problem size already shows an improvement in the scalability of the PLAPACK codes in comparison to a matrix size of n=3888. Here, the optimum number of processors is 64 and 128 for the MR³ algorithm and 128 and 128 for the QR algorithm on Phase 1 and Phase 2, respectively. PDSYEV continues scaling well up to 256 processors on both machines, whereas for PDSYEVD the code is fastest on 64 and 256 processors on Phase 1 and Phase 2, respectively.

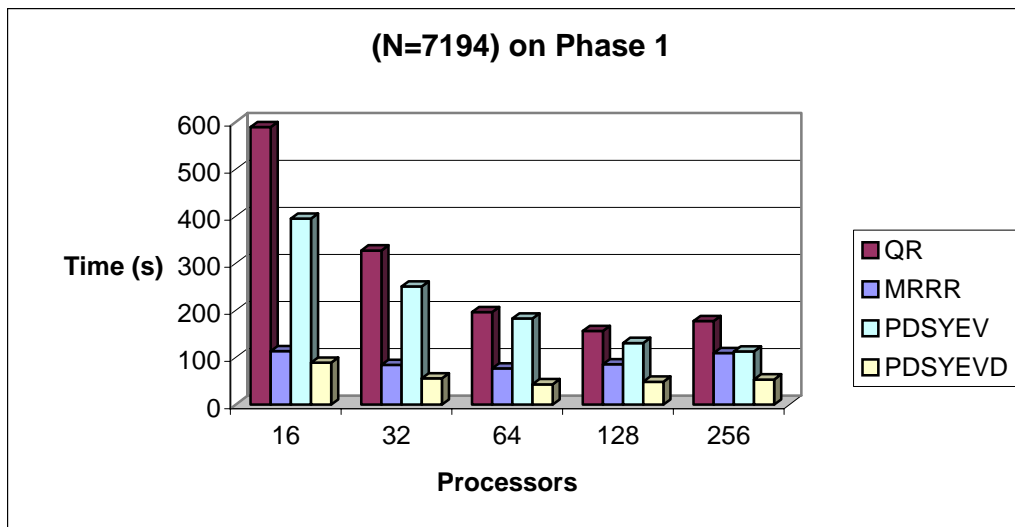


Figure 3: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEV D) algorithms, from ScaLAPACK, on Phase 1 for n=7194.

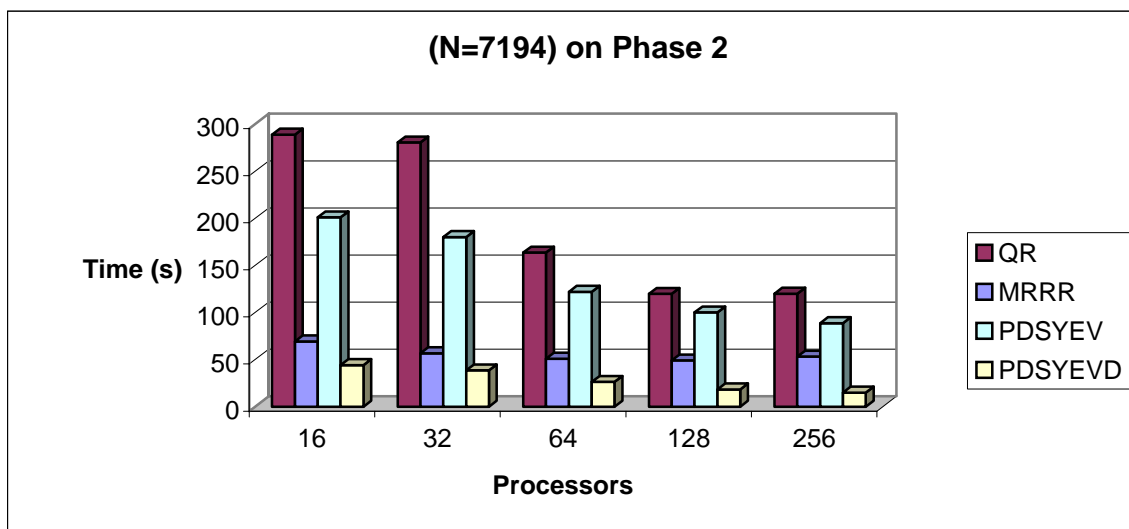


Figure 4: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEV D) algorithms, from ScaLAPACK, on Phase 2 for n=7194.

The results for n=12354 are shown in Table 4 (Table 5 and Table 7 contain the times for 512 processors), APPENDIX A, Figure 5 for Phase 1 and Figure 6 for Phase 2. As the problem size is bigger, the problem scales up to 128 processors for the MR³ algorithm on both Phase 1 and 2, however communication overheads dominate run-time for greater numbers of processors. There is insufficient memory available for the QR method to run on 16 or 32 processors on Phase 1 for n=12354. On Phase 2, the QR method is able to run on 16 processors, as it can access the memory from the complete 32-way node. However, on 32 processors, the memory overheads per processor for the QR method are prohibitively large and it does not run. PDSYEV D is fastest on 512 processors but does not perform significantly better than on 256

9A performance study of the PLAPACK and ScaLAPACK Eigensolvers

processors. There is insufficient memory available for PDSYEVD on 16 processors. Both ScaLAPACK solvers scale better than those from PLAPACK.

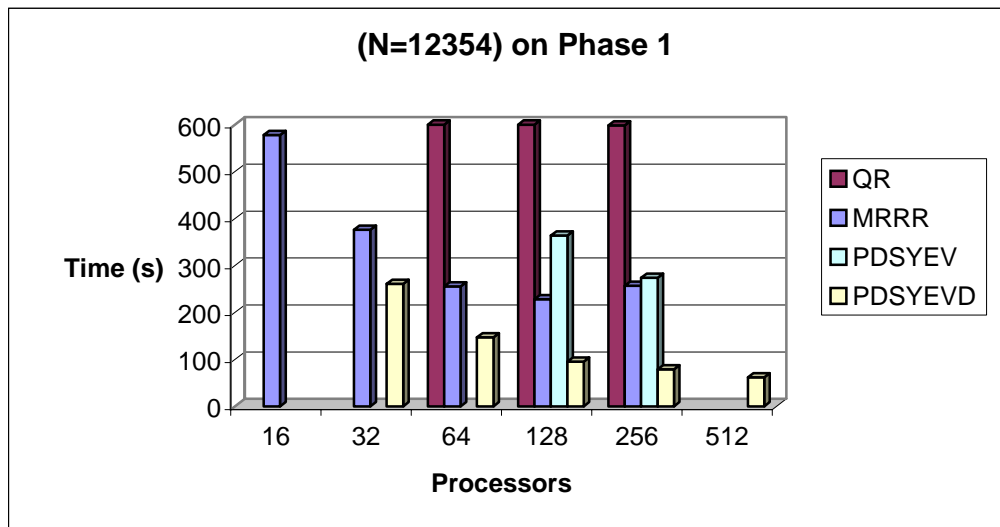


Figure 5: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEVD) algorithms, from ScaLAPACK, on Phase 1 for n=12354.

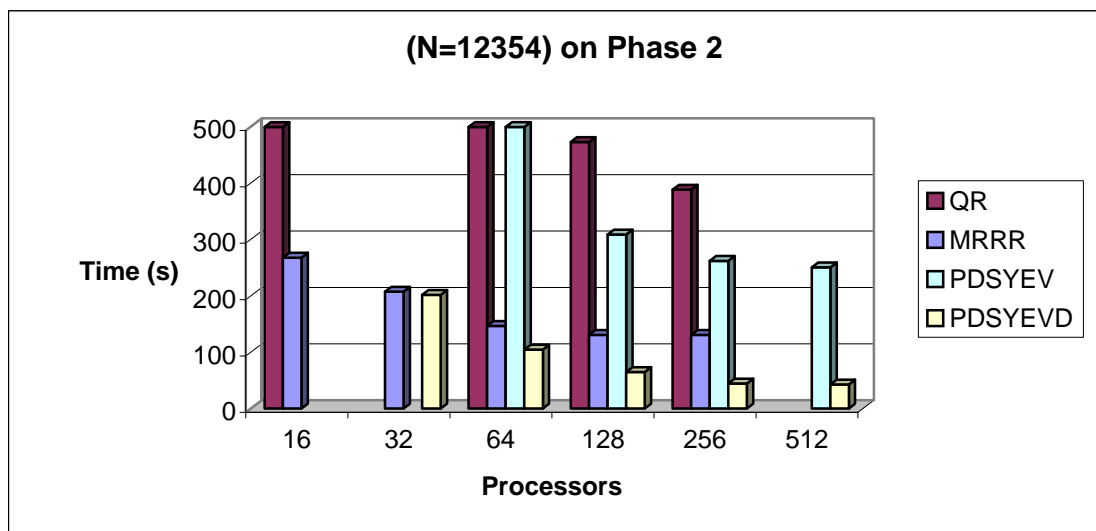


Figure 6: Results from the QR and MR³ algorithms, from PLAPACK, and QR (PDSYEV) and D&C (PDSYEVD) algorithms, from ScaLAPACK, on Phase 2 for n=12354.

We do not provide numbers for the QR and the MR³ eigensolvers for 512 processors since they already showed not to gain any speed-up on a smaller number of processors but actually slowed down.

When using PLAPACK, MR³ always performs much better than QR. On the other hand, ScaLAPACK's D&C: PDSYEVD, is always faster than ScaLAPACK's QR library: PDSYEV. In addition, D&C is always faster than MR³ on HPCx.

The lower memory overheads associated with MR^3 are demonstrated where larger matrix sizes are run on a relatively small number of processors. For $n=12354$, the QR-algorithm based solvers and the D&C algorithm break down on 16, 32 or even 64 processors due to a lack of memory. The MR^3 manages the memory requirements for all these choices. This behaviour can be confirmed for even larger matrix sizes in the next Subsection.

5.2 Time Breakdown for PLAPACK's MR^3 and ScaLAPACK's D&C

All times obtained for the MR^3 - and the D&C-algorithm based solvers can be broken down further to the times needed for the reduction of the matrix to tridiagonal form, the tridiagonal eigensolver and the back transformation. In Figure 7 and Figure 8 (see also Table 8, APPENDIX A) we show how the total time splits between the reduction, the eigensolver and the back transformation for the $n=20480$ case to give an impression of how each stage influences the complete calculation on Phase 2. We present results for the MR^3 algorithm (PLAPACK) and the D&C algorithm (ScaLAPACK) but we do not show results for the QR-algorithms, which are the slower within both libraries. Results for the MR^3 -algorithm on Phase 1 can be found in Table 6, APPENDIX A, no results are available for PDSYEV D on Phase 1. PDSYEV D has not been investigated on Phase 1 due to time constraints. The results for $n=20480$ can be compared with the time breakdown for $n=12354$ to see how the three stages perform with different matrix sizes. The actual execution times for $n=12354$ can be found in Table 5 for Phase 1 and Table 7 for Phase 2, respectively, in APPENDIX A, but are not shown graphically.

For both solvers, it is clear that the stage involving the reduction to tridiagonal form heavily influences the total run time. For the PLAPACK solver the percentage of the time required to reduce the matrix increases with larger arrays of processors as the communications increase, it varies between 76-89% of the total run time for $n=20480$. For the ScaLAPACK solver this percentage is lower and nearly constant between 65-68% of the total run time. The tridiagonal eigensolver itself is the quickest stage for both solvers. Again, the percentage increases with the number of processors for the MR^3 -algorithm from 2.7-4.0%. The contribution of the D&C-algorithm to the total run time is higher, 12.0-15.5%.

As mentioned in a previous section, a possible major drawback of the ScaLAPACK solver is the associated memory overheads. The memory requirements for this size matrix are prohibitive when attempting to run on 16 or 32 processors. PLAPACK requires much less memory and can compute the solution on 16 or 32 processors.

The performance gain ratio between ScaLAPACK and PLAPACK increases on larger arrays of processors, as ScaLAPACK scales better. On 64 processors PLAPACK is only about twice as slow, on 256 processors it is six times slower. It should be noted, that on 64 processors, the MR^3 eigensolver itself is about 60% faster than D&C. This clearly shows the improvement the MR^3 algorithm provides over most traditional solvers. Unfortunately, the reduction routine is not competitive with that of the ScaLAPACK routine.

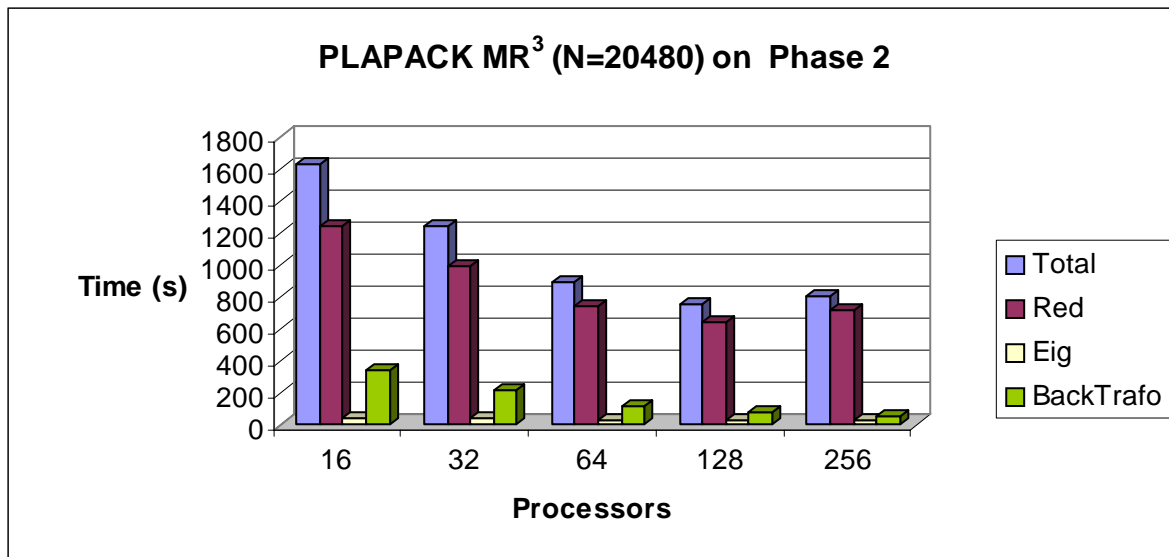


Figure 7: Time breakdown of the various routines for the MR³ algorithm for n=20480 on Phase 2. Shown are the total run time (Total), the time for the reduction to tridiagonal form (Red), the times for the tridiagonal eigensolver (Eig) and for the back transformation (BackTrafo).

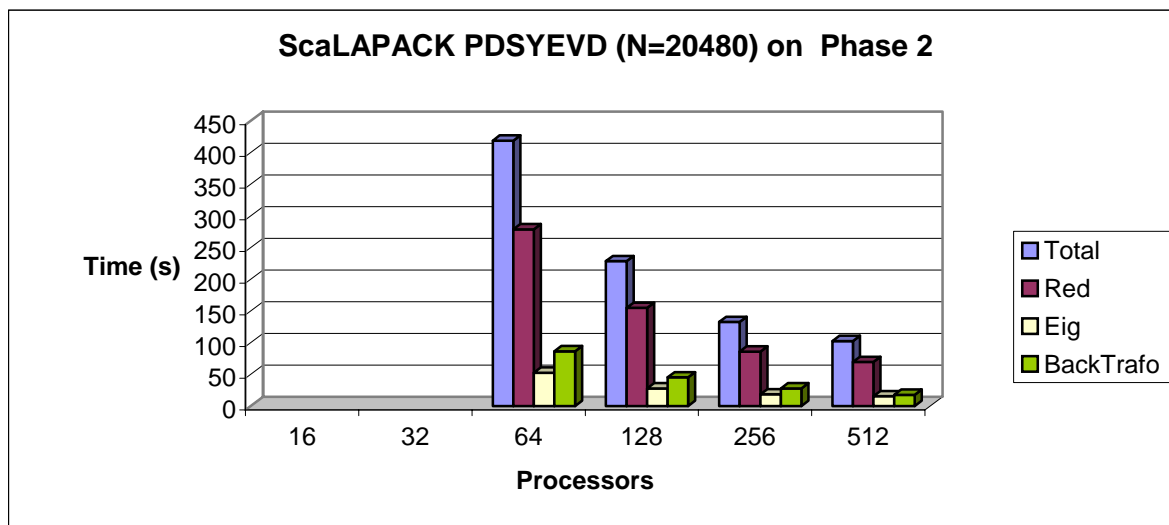


Figure 8: Time breakdown of the various routines for the PDSYEVD solver for n=20480 on Phase 2. Shown are the total run time (Total), the time for the reduction to tridiagonal form (Red), the times for the tridiagonal eigensolver (Eig) and for the back transformation (BackTrafo).

Comparing the results for n=20480 with those obtained for n=12354 on Phase 2 (see Table 7, APPENDIX A), it can be seen that the MR³-eigensolver itself contributes more to the total run time when smaller matrix sizes are used (6.3-8.5%). This increase is much less prominent for the D&C eigensolver (12.9-14.9%). This shows that the MR³-algorithm performs especially well for larger problem sizes. For n=12354 the total run time of the PLAPACK solver is up to three times higher than for the ScaLAPACK solver, increasing with the number of processors. The MR³-

algorithm itself is about two times faster than the D&C-algorithm on up to 32 processors.

6 Summary

We have studied the performance of the PLAPACK eigensolvers based on the MR³ and the QR algorithm, and the ScaLAPACK eigensolvers based on the QR and D&C algorithms for CRYSTAL matrix sizes of $n = 3888, 7194$ and 12354 on 16, 32, 64, 128, 256 and 512 processors on Phase1 and Phase2 of HPCx. We also compare the MR³ and the D&C based eigensolvers for $n=20480$ and show detailed timing breakdowns of the reduction routine, the eigensolver and the back transformation.

For any of our given test cases, the MR³ algorithm is always faster than the PLAPACK QR algorithm. In turn, the ScaLAPACK D&C solver is faster than MR³ for all problem sizes investigated here, although the tridiagonal eigensolve stage of MR³ is competitive for very large matrices. This is in contrast to the tests performed by [7] on a Linux cluster and for much larger matrix sizes. Although the ScaLAPACK QR (PDSYEV) scales well, it is always slower than ScaLAPACK's D&C for the examples tested here.

For small problem sizes, the total run time for the MR³ based solver increases as the number of processors is increased (For $n=3888$ when using more than 32 processors, for $n=7194$ for more than 64 processors and for $n=12354$ for more than 128 processors). This lack of scalability suggests that communication overheads dominate run-time on larger arrays of processors. The overall MR³ time to solution suffers from a poorly performing tridiagonal reduction routine. Some effort to optimize the reduction routine can be expected in the future [12]. The tridiagonal eigensolver stage in MR³ is extremely fast and contributes much less to the overall run time than in ScaLAPACK or PEIGs [14], where PEIGs is another parallel eigensolver package.

Note, that the matrix sizes investigated are characteristic for CRYSTAL but not the ideal test bed for MR³ and the number of processors investigated. The MR³ algorithm aims for much larger matrix sizes (see [7]).

The MR³ algorithm is capable of solving far larger problems than the QR or D&C algorithms on HPCx. This can be seen for problem sizes of $n=12354$ and $n=20480$ where the other algorithms broke down for 16, 32 or even 64 processors. The largest problem size we could solve on HPCx with MR³ was $n=50,000$ on 128 processors³.

Throughout the tests it can be seen all codes run faster on Phase 2 than Phase 1. The largest speed-ups are achieved for small matrix sizes and out of these, for the D&C-solver (a factor of 3.6 for $n=7194$ on 256 processors). Least improvement can be seen for the QR-solver.

³ This result is unrelated to CRYSTAL input matrices.

Acknowledgements

EYB would like to thank the developers of PLAPACK and the MR³ algorithm, Robert van de Geijn and Paolo Bientinesi, for their support and discussions of arising problems and results. AGS would like to thank Ian Bush for providing the CRYSTAL test matrices discussed in this report.

The authors would also like to thank Ian Bush and Gavin Pringle for proof-reading the manuscript.

7 References

1. Parallel Linear Algebra Package, www.cs.utexas.edu/users/plapack/
2. The ScaLAPACK Home Page, www.netlib.org/scalapack/scalapack_home.html
3. The HPCx Service, www.hpcx.ac.uk/
4. A. G. Sunderland, E.Y. Breitmoser, An Overview of Eigensolvers for HPCx, # HPCxTR0312, (2003), www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0312.pdf
5. *CRYSTAL 2003 User's Manual*, V.R. Saunders, R. Dovesi, C. Roetti, R. Orlando, C. M. Zicovich-Wilson, N. M. Harrison, K. Doll, B. Civalleri, I. J. Bush, Ph. D'Arco, M. Llunell, University of Torino, Torino. <http://www.crystal.unito.it> .
6. www.cs.utexas.edu/users/plapack/tutorial/SC98/
7. P. Bientinesi, I.S. Dhillon, R. A. van de Geijn, A Parallel Eigensolver for Dense Symmetric Matrices based on Multiple Relatively Robust Representations, UTCS Technical Report #TR-03026, 2003 www.cs.utexas.edu/users/plapack/papers/pareig.ps
8. Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html .
9. 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).
10. J. J. M. Cuppen. A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem, Numer. Math., 36:177-195, 1981.
11. www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/Libraries.html
12. Paolo Bientinesi, Robert van de Geijn, Private Communication
13. V.R. Saunders, R. Dovesi, C. Roetti, R. Orlando, C.M. Zicovich-Wilson, N.M. Harrison, K. Doll, B. Civalleri, I. Bush, Ph. D'Arco, M. Llunell, CRYSTA2003 Users's Manual, University of Torino, 2003. www.cse.dl.ac.uk/Activity/CRYSTAL
14. www.emsl.pnl.gov/docs/nwchem/doc/peigs/docs/peigs3.html (G.Fann et al).

APPENDIX A Tables

This appendix contains all the numbers for the diagrams shown in the report. CRYSTAL data are used throughout.

Table 2: Results for all four solvers on Phase 1 and Phase 2 for n=3888.

Algorithm Time(s)	#PEs					Machine
	16	32	64	128	256	
MR ³	27.5	25.2	27.3	38.1	60.2	Phase 1
QR	91.6	56.4	53.9	54.9	67.2	
PDSYEV	51.3	38.0	23.8	17.6	19.2	
PDSYEVD	16.7	14.1	13.4	13.4	12.2	
MR ³	17.6	15.3	17.2	17.9	20.9	Phase 2
QR	48.1	50.0	50.4	52.7	66.0	
PDSYEV	29.7	26.0	17.3	10.1	11.2	
PDSYEVD	8.4	8.0	6.4	5.1	5.2	

Table 3: Results for all four solvers on Phase 1 and Phase 2 for n=7194.

Algorithm Time(s)	#PEs					Machine
	16	32	64	128	256	
MR ³	113.9	84.5	77.0	85.9	109.8	Phase 1
QR	589.9	327.5	196.8	156.6	177.6	
PDSYEV	395.0	251.0	183.0	131.0	103.0	
PDSYEVD	89.1	56.1	43.3	48.7	53.5	
MR ³	69.3	56.8	51.0	49.0	53.2	Phase 2
QR	288.8	280.5	163.6	119.7	120.0	
PDSYEV	201.0	180.0	122.0	100.0	88.6	
PDSYEVD	44.2	38.5	26.3	18.3	15.0	

15A performance study of the PLAPACK and ScaLAPACK Eigensolvers

Table 4: Results for all four solvers on Phase 1 and Phase 2 for n=12354.

Algorithm Time(s)	#PEs					Machine
	16	32	64	128	256	
MR ³	577.7	375.9	255.4	228.3	256.5	Phase 1
QR	--	--	1038.	657.4	598.2	
PDSYEV	--	--	--	364.1	274.2	
PDSYEVD	--	261.0	147.0	95.0	78.8	
MR ³	267.8	207.4	147.1	130.9	131.0	Phase 2
QR	1743.	--	767.6	472.9	388.5	
PDSYEV	--	--	512.8	309.4	262.2	
PDSYEVD	--	201.8	104.6	65.2	44.4	

Table 5: Time breakdown for the results on Phase 1 for a matrix of size n=12354 and the MR³ algorithm and PDSYEVD.

#PEs	Phase1 time(s)							
	MR ³				PDSYEVD			
	Red	Eig	Back Trafo	Total	Red	Eig	Back Trafo	Total
16	428.1	20.7	128.9	577.7				--
32	288.5	17.6	69.8	375.9	165	32.97	64.39	261
64	166.0	15.3	44.1	255.4	79.18	23.75	44.73	147
128	178.2	14.7	35.4	228.2	60.57	14.56	22.32	95.0
256	210.2	14.4	31.1	256.5	46.85	12.6	19.31	78.8
512				--	42.63	8.83	10.37	61.8

16A performance study of the PLAPACK and ScaLAPACK Eigensolvers

Table 6: Time breakdown for the results on Phase 1, a matrix of size $n=20480$ and the MR^3 algorithm. No results available for PDSYEVD.

#PEs	Phase 1 time(s)							
	MR^3				PDSYEVD			
	Red	Eig	Back Trafo	Total	Red	Eig	Back Trafo	Total
16	1770.9	49.34	521.09	2341.				--
32	1225.3	44.12	285.46	1555.				--
64	886.1	44.21	160.26	1091.				--
128	746.46	43.4	92.99	882.9				--
256	802.89	42.27	75.91	921.1				--

17A performance study of the PLAPACK and ScaLAPACK Eigensolvers

Table 7: Time breakdown for the results on Phase 2, a matrix of size $n=12354$ and the MR³ algorithm and PDSYEVD.

#PEs	Phase 2 time (s)							
	MR ³				PDSYEVD			
	Red	Eig	Back Trafo	Total	Red	Eig	Back Trafo	Total
16	178.9	16.8	72.1	267.8	--	--	--	--
32	149.3	13.3	44.8	207.4	136.8	26.1	38.4	201.8
64	11.7	11.6	23.8	147.1	68.0	15.0	21.3	104.6
128	103.9	11.1	16.0	130.9	44.3	9.3	11.1	65.2
256	108.0	11.0	12.0	131.0	30.2	6.6	7.5	44.4
512				--	30.6	6.3	5.6	42.5

18A performance study of the PLAPACK and ScaLAPACK Eigensolvers

Table 8: Time breakdown for the results on Phase 2, a matrix of size $n=20480$ and the MR^3 algorithm and PDSYEVD.

#PEs	Phase 2 time (s)							
	MR^3				PDSYEVD			
	Red	Eig	Back Trafo	Total	Red	Eig	Back Trafo	Total
16	1250.3	43.6	342.6	1637				--
32	995.2	36.1	216.0	1247				--
64	744.9	32.9	118.2	896.1	278.9	52.4	86.4	418.5
128	641.1	31.9	77.7	750.6	154.9	28.0	45.4	228.5
256	715.1	31.7	58.3	805.0	86.0	18.5	27.9	132.6
512				--	69.5	15.5	17.6	102.7