Performance of Dense Eigensolvers on BlueGene/Q

Inge Gutheil, Jan Felix Münchhalfen, and Johannes Grotendorst

Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany http://www.fz-juelich.de/ias/jsc

Abstract. Many scientific applications require the computation of about 10-30% of the eigenvalues and eigenvectors of large dense symmetric or complex hermitian matrices. In this paper we will present performance evaluation results of the eigensolvers of the three libraries Elemental, ELPA, and ScaLAPACK on the BlueGene/Q architecture. All libraries include solvers for the computation of only a part of the spectrum. The most time-consuming part of the eigensolver is the reduction of the full eigenproblem to a tridiagonal one. Whereas Elemental and ScaLAPACK only offer routines to directly reduce the full matrix to a tridiagonal one, which only allows the use of BLAS 2 matrix-vector operations and needs a lot of communication, ELPA also offers a two-step reduction routine, first transforming the full matrix to banded form and thereafter to tridiagonal form. This two-step reduction shortens the reduction time significantly but at the cost of a higher complexity of the back transformation step. We will show up to which part of the eigenspectrum the use of the two-step reduction pays off.

Keywords: eigenvalue and eigenvector computation, libraries: Elemental, ELPA, ScaLAPACK, BlueGene/Q

1 Introduction

Many scientific applications in the fields of materials science and quantum chemistry require the computation of eigenvalues and eigenvectors of dense real symmetric or complex hermitian matrices. For example, in DFT (Density Functional Theory) calculations on modern supercomputers [1], typical sizes of those matrices are about 50000 x 50000 and 10-30% of the eigenvalues and eigenvectors have to be computed. As the computation of eigenvalues and eigenvectors is of complexity N^3 where N denotes the problem size, these computations require highly parallel algorithms to speed up the computation on modern computers with thousands of cores and the possibility to start even more threads than cores.
The oldest parallel library for dense linear algebra which is still in use in many applications is ScaLAPACK [2]. It is a pure MPI library requiring a two-dimensional block cyclically distributed matrix as input. In the last years two new libraries have been developed, Elemental [3], a complete C++ framework for dense linear algebra and ELPA [4] which is an add-on to ScaLAPACK for the solution of real symmetric and complex hermitian eigenproblems. Both offer the choice to compute all or some of the eigenvalues and eigenvectors.
Whereas ScaLAPACK and Elemental only offer the one-step reduction of a full matrix

to tridiagonal form, ELPA also includes a routine for two-step reduction, first to banded and then to tridiagonal form. This reduction allows the use of highly optimized BLAS 3 matrix-matrix operations for the first step, thus reducing significantly the execution time for the reduction, which still is the bottleneck of all dense eigensolvers. The gain in the reduction phase however goes with a loss in the back transformation phase, as the back transformation now is also done in two steps. The complexity of the back transformation depends on the number of eigenvectors to be computed, thus the gain in the reduction phase will pay off, if only a part of the eigenvectors are to be computed.

2 The BlueGene/Q architecture

The BlueGene/Q architecture is built from IBM PowerPC® A2 compute cards, each card with 16 cores at 1.6 GHz and with 16 GB DDR3 Memory. Each core has two 4-way SIMD units which each can deliver 4 results per cycle. 32 compute cards form a node card, meaning each node card consists of 512 cores and has a total amount of 512 GB of main memory. 16 node cards form a midplane, two midplanes together with one or two I/O drawers form one rack. Thus each rack consists of 16384 cores and has a peak performance of 0.2 Petaflops. Each core allows 4-way hyperthreading, so up to 64 MPI processes or for example 16 MPI processes with up to 4 threads (OpenMP or pthreads) per MPI-process can be started on each compute card.
The BlueGene/Q called JUQUEEN [5] installed at Jülich Supercomputing Centre in May 2012 now consists of 28 racks (7 rows à 4 racks), which means that the full machine has 28672 compute cards (458752 cores) and thus a peak performance of 5.9 Petaflops.

3 Parallel libraries investigated

The first library for the solution of dense symmetric eigenvalue problems we studied is ScaLAPACK, where the current release is 2.0.2, but we still used 2.0.1. It is mainly written in FORTRAN 77 with a few C-routines hidden from the user. An add-on to ScaLAPACK for eigenvalue problems is ELPA (EigensoLver for Petaflop Applications), written in Fortran 95. Both libraries use the same block-cyclic two-dimensional data distribution. The user has to distribute the matrix according to that distribution and calls the routines with the distributed matrix. ScaLAPACK only offers a pure MPI version whereas in the development version of ELPA an OpenMP/MPI hybrid version is available. ELPA does not use the BLACS for the computational routines but only creates two MPI sub-communicators per process, one for the process row and one for the process column of each MPI process. We used the ELPA development version of November 2012.
A new library for linear algebra with dense matrices is the Elemental C++ framework which uses a two-dimensional data distribution with block size 1. The version we used was 0.75. Elemental also offers a hybrid parallel version.

3.1 Routines tested in the libraries

All routines in all libraries under investigation follow the same three steps for the computation of eigenvalues and eigenvectors of a real dense symmetric matrix: Reduction to

tridiagonal form via orthogonal transformations, solution of the tridiagonal eigenvalue problem, back transformation of the eigenvectors of the tridiagonal matrix to those of the original matrix.

ScaLAPACK 2.0.1 offers four different routines for the solution of the dense symmetric eigenvalue problem. They differ in the reduction routine used and, more important, in the way the tridiagonal eigenvalue problem is solved. PDSYEV and PDSYEVD use the original version PDSYTRD of the reduction to tridiagonal form whereas PDSYEVX and PDSYEVR use a reduction called PDSYNTRD which is usually faster than the original version due to some improvements in communication costs. This reduction always uses a square processor grid for the reduction phase. Even though data has to be redistributed in cases where the original process grid is not square, we found out that this routine is faster than the old one [6]. PDSYEV uses the QR algorithm for the computation of eigenvalues and eigenvectors of the tridiagonal matrix. This method is rather slow although it can be parallelized well. It delivers all eigenvalues and eigenvectors and the eigenvectors are orthogonal to working precision. We did not investigate that routine because it is too slow.
PDSYEVX is the most often used eigensolver routine in ScaLAPACK as it was for a long time the only one allowing to compute only a part of the eigenspectrum. It uses bisection and inverse iteration for the computation of the eigenvalues and eigenvectors of the symmetric tridiagonal matrix. This method can be easily parallelized and works well if the eigenvalues are well separated. The input parameter ORFAC can be used to decide when eigenvalues are treated as clustered and thus the eigenvectors have to be re-orthogonalized. The re-orthogonalization of eigenvectors belonging to clustered eigenvalues is not parallelized and thus it can be very expensive both in terms of compute time and in terms of memory requirements. For our measurements we set ORFAC=$1.0 * 10^{-4}$.
PDSYEVD uses the divide-and-conquer-method for the tridiagonal eigenproblem. It was up to version 1.8 the fastest routine if all eigenvalues and eigenvectors had to be computed. There is no version of this routine allowing to compute only a part of the spectrum.
PDSYEVR is new in release 2.0.1 and uses the MRRR [7][8] algorithm for the tridiagonal eigenproblem. This routine allows to compute only a part of the eigenspectrum and is supposed to be faster than bisection and inverse iteration if eigenvalues are clustered.

The **ELPA** library consists of two routines which differ in the way the full eigenproblem is reduced to tridiagonal form and as a result of that in the back transformation of eigenvectors. The routine ELPA1 uses the well-known one-step reduction to tridiagonal form, similar to the old ScaLAPACK reduction routine, ELPA2 uses a two-step reduction first to band and then to tridiagonal form and a two-step back transformation [9] [10]. Both routines use a modification of the divide-and-conquer method for the solution of the tridiagonal eigenvalue problem which allows to compute only a part of the eigenspectrum.

In the library **Elemental** there is only one routine for the solution of the symmetric or hermitian eigenvalue problem, HermitianEig. For the reduction to tridiagonal form there are two choices, one for general rectangular processor grids and one that uses the largest square processor grid that fits to the number of processors given. As with the ScaLAPACK reduction routines, the routine using a square processor grid is faster than the other even if the original grid is not square. This has been shown in [11]. For the solution of the tridiagonal eigenproblem the MRRR algorithm is used in the implementation PMRRR by M. Petschow [12]. In contrast to the other libraries the distribution block size does not determine the algorithmic block size thus allowing to use different algorithmic block sizes for different computation steps. We did not investigate this possibility.

4 Measurements done

As most applications on JUQUEEN should use at least one midplane with 512 compute cards and thus 8192 cores we tried to measure the performance of the eigensolvers on a complete midplane and on one rack. Most of those measurements were done in the pre-production phase of JUQUEEN, later on we only had limited resources for tests. The up to 4-way multi-threading allows to start up to 64 processes per compute card. We therefore tried to use one, two, or four MPI processes per core to see, whether multithreading pays even with pure MPI. In the pre-production phase of JUQUEEN we measured execution times for the computation of the full eigenspectrum on one node card, one midplane, and one rack with matrix sizes ranging from 5000 to 60000. Later on measurements on one node card were done with matrix sizes from 6000 to 50000 by steps of 4000. For ScaLAPACK and ELPA block sizes were chosen to be 32. In private communication with Thomas Auckenthaler, one of the ELPA authors, we learned that for ELPA smaller blocks should be better and so we later used a block size of 16 for ELPA [13]. Elemental had not yet been ported to JUQUEEN in the pre-production phase, thus all measurements were done with limited resources. We chose the default algorithmic block size of 128 which was seen to be optimal on BlueGene/P and on a preliminary BlueGene/Q hardware [11].
Only the ScaLAPACK measurements of the pre-production phase are still used in this presentation. For ELPA we repeated the measurements with the development version of November 2012 and block size 16. This version contains a new routine for the two-step back transformation, which is optimized for the BlueGene/Q vector instructions. The new optimized ELPA library turned out to be much faster than the older version.
ELPA and Elemental both contain hybrid versions, but due to compiler problems we could not compare these versions to the pure MPI implementation. For all measurements we used a test program that constructs a matrix with known eigenvalues and compares the computed ones to the given ones. All measurements included tests for the correctness of the results and the orthogonality of the eigenvectors computed. All measurements of the partial spectrum were done only on one node card.

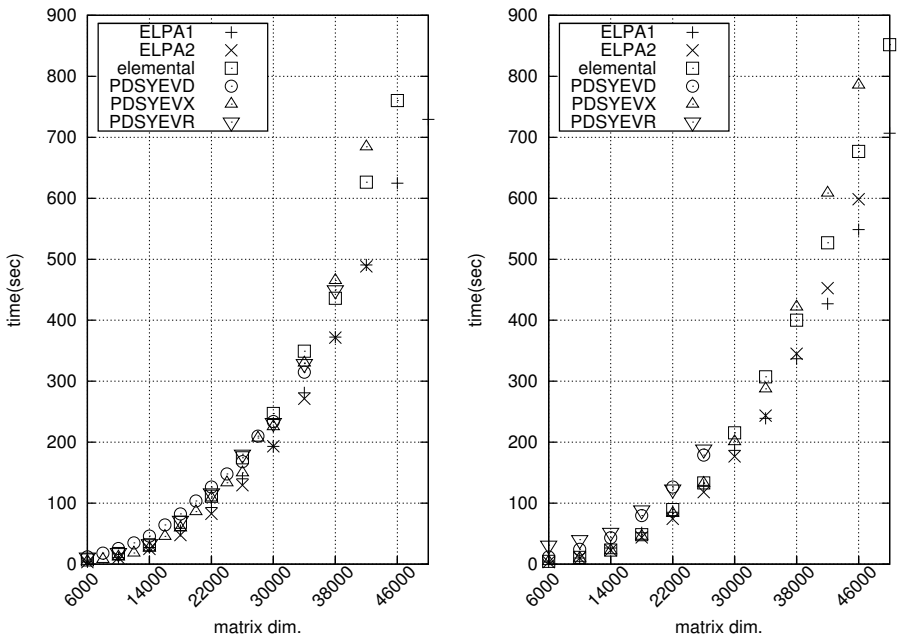

Fig. 1. Comparison of all routines of each library on a node card using 16 (left) and 32 (right) MPI processes per compute card.

5 Scaling results up to one rack of JUQUEEN

Fig. 1 shows the results of the fastest routine for the full eigenspectrum of each library on a node card of JUQUEEN using 16 (left) and 32 (right) MPI processes per compute card. We measured the performance for matrices of sizes between 1000 and 50000. Due to our checking of results and other additional memory consumptions PDSYEVD could only be measured up to $N = 34000$, PDSYEVR up to $N = 29000$ and PDSYEVX with ORFAC=$1.0 * E^{-4}$ up to $N = 42000$. All ScaLAPACK routines investigated performed similarly, there is not much difference between 16 and 32 MPI processes per compute card except for PDSYEVR, which for matrices of size up to 20000 is significantly slower with 32 processes per compute card than with 16 processes. Overall for matrices of sizes up to 25000 the fastest ScaLAPACK routine was PDSYEVX, followed by PDSYEVD for matrices smaller than 6000 and PDSYEVR for matrices larger than 6000.

On a node card computing all eigenvalues and -vectors ELPA1 is faster than ELPA2. Overall the performance of ELPA is better than ScaLAPACK's and Elemental's performance. All routines show a small speedup if 32 MPI processes per compute card are used.

On a midplane we could see that due to a bug in the communication that we had already seen on BlueGene/P with 1024 MPI processes (see [11]) PDSYEVR took more

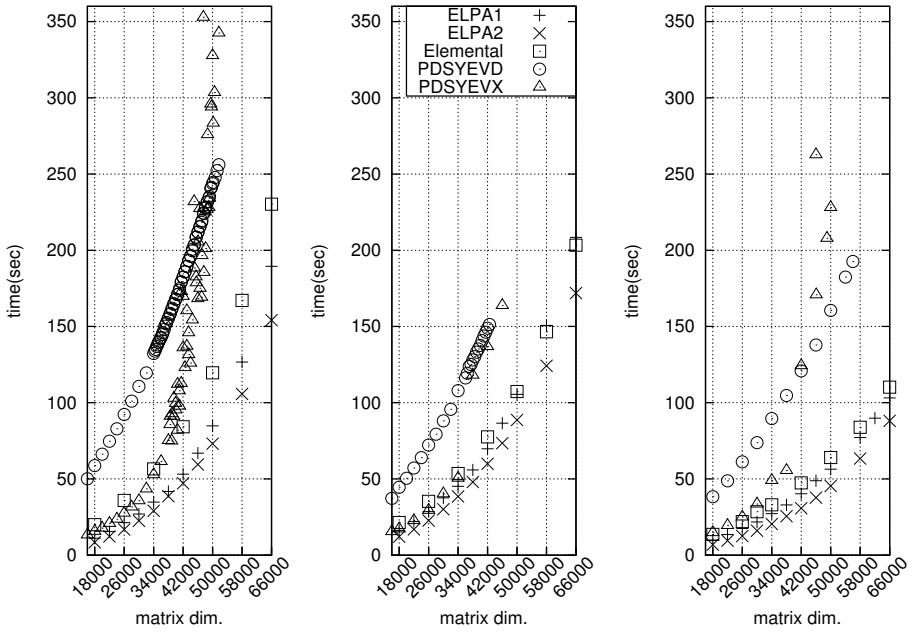

Fig. 2. Comparison of all routines of each library on a midplane left: using 16 MPI processes per compute card, middle: using 32 MPI processes per compute card and right: on one rack using 16 MPI processes per compute card.

than 900 seconds for the smallest matrix of size $N = 6000$ for 16 MPI processes per compute card and thus could not be shown in the figures. We did not further investigate PDSYEVR for the computation of the full eigenspectrum.
Overall the ScaLAPACK routines could no longer compete with the new libraries ELPA and Elemental, and PDSYEVX showed a very high variation in execution times for large matrices as can be seen from Fig. 2. ScaLAPACK and Elemental still showed a small speedup with 32 MPI processes per compute card, whereas both ELPA routines became slower. On one rack ELPA2 again was the fastest routine, ELPA1 and Elemental almost equal and ScaLAPACK slower than the new routines.
For a matrix of size $N = 50000$ Elemental showed a speedup for one midplane compared to a node card of about 8, ELPA1 of about 9, ELPA2 even 10. It is the fastest routine on a midplane. The speedup of the ScaLAPACK routines could not be measured as we could not run the programs with $N = 50000$. For one rack there was a speedup compared to the fastest run on a midplane for all routines except PDSYEVX, Elemental even got a speedup of more than two compared to the run on a midplane with 16 MPI processes per compute card. As the speedup is only slightly more than two, we think that it is due to the fact that we did only one measurement per matrix size.
All new routines scale up to one rack of BlueGene/Q, for Elemental using 32 MPI processes per compute card is faster, for ELPA using only 16 processes per compute card is

faster. Thus we are indeed waiting for a hybrid parallelization to see whether that allows to explore hyperthreading.

6 Results for different parts of the spectrum

Fig. 3 shows that even for the computation of the full eigenspectrum the routine with the two-step reduction is not much slower than the one with the one-step reduction on a node card. The parts of the time for reduction and back transformation almost change their roles. Thus it can be expected that if less eigenvectors have to be transformed the time for the routine with the two-step reduction will become much smaller.
As most applications require only a part of the eigenspectrum we also compared the performance of the different libraries and routines for the computation of 5% and 45% of the eigenvalues and eigenvectors. From Fig. 4 it can be seen that for the computation

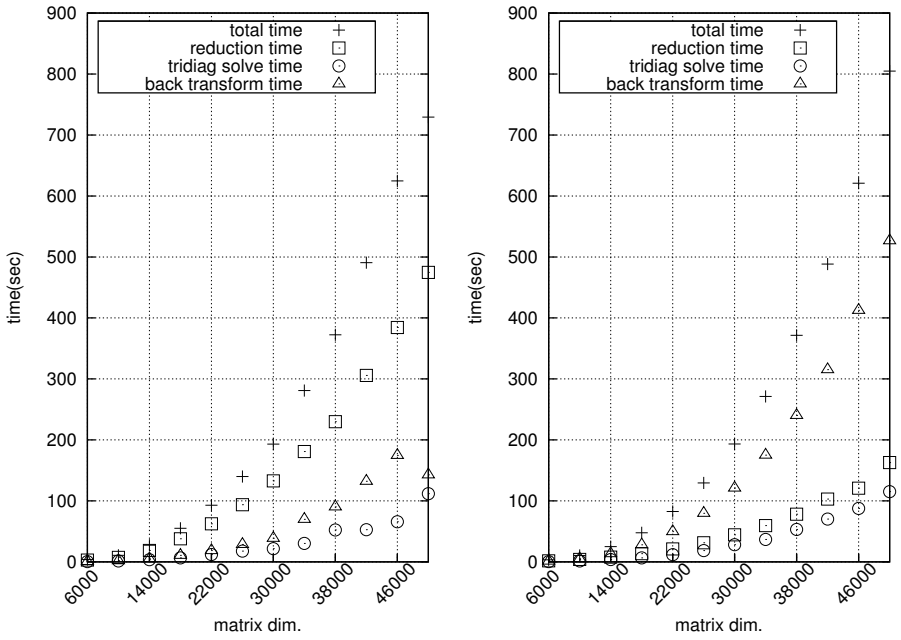

Fig. 3. Comparison of ELPA1 (left) and ELPA2 (right) on one node card using 16 MPI processes per compute card, computation of the full eigenspectrum

of 45% of the spectrum ELPA2 is almost twice as fast as the other routines, for 5% even more than three times as fast.

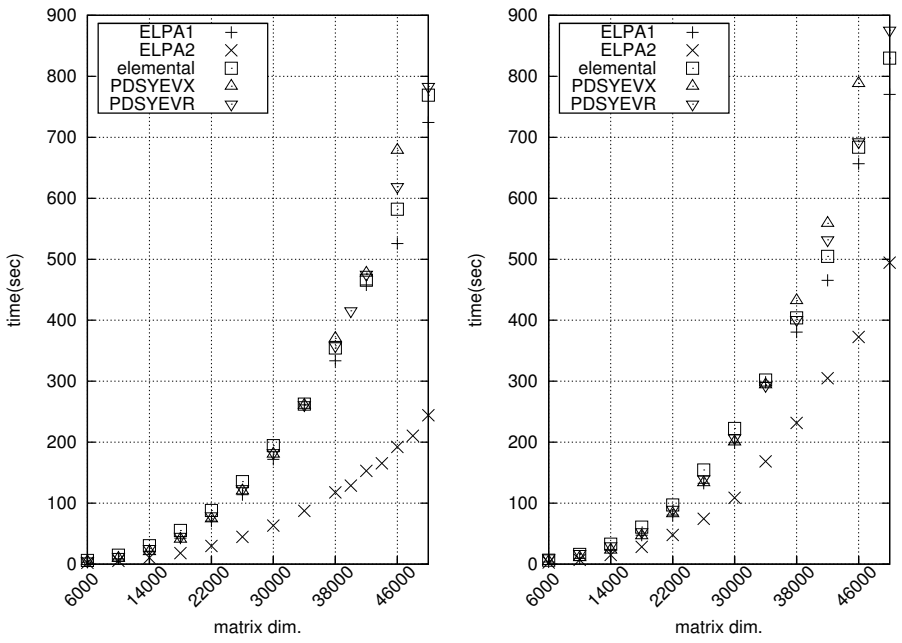

Fig. 4. Comparison of all routines if only 5% (left) and 45% (right) of the spectrum is computed, one node card using 16 MPI processes per compute card

7 Conclusions

For the computation of all eigenvalues and eigenvectors of a large full real symmetric matrix the old ScaLAPACK routine PDSYEVX has big problems with clustered eigenvalues. The new ELPA library solves this problem by using the divide-and-conquer algorithm for the computation of the eigenvalues of the tridiagonal matrix. The ELPA implementation is faster than ScaLAPACK's routine PDSYEVD, perhaps due to the fact that the ELPA routine is less flexible than the ScaLAPACK routine, but also due to the usage of pure MPI instead of BLACS. A better implementation of the BLACS could perhaps speed up PDSYEVD. An alternative to ELPA is the new library Elemental which uses the PMRRR algorithm and is very flexible because the algorithmic block size can be chosen without changing the data layout. Also, filling of the C++ class DistMatrix can be easier than distributing a matrix in the block-cyclic two-dimensional way ScaLAPACK and ELPA require.

For the computation of only a part of the spectrum (even more than 50%) the ELPA2 routine is much faster than all the other routines. It is in fact the only one that is significantly faster if only a part of the spectrum is needed compared to the full spectrum. This is mainly because for all other routines the reduction phase is so dominant, that gains in the other parts of the computation have almost no influence on the execution time. This means that for computations where only some of the eigenvalues and -vectors have to

be computed, ELPA2 should be used if possible, especially, if the application already was written to use ScaLAPACK and thus already has the matrix in the way it is also needed for ELPA.

Acknowledgements

The authors thank Jack Poulson, the author of the Elemental library and the ELPA team, especially Thomas Auckenthaler, for their immediate responses to problem reports.

References

1. FLEUR: The Jülich FLAPW code family. Website (May 2013) Available online at `http://www.flapw.de`.
2. Choi, J., Demmel, J., Dhillon, I., Dongarra, J., Ostrouchov, S., Petitet, A., Stanley, K., Walker, D., Whaley, R.: ScaLAPACK: a portable linear algebra library for distributed memory computers–design issues and performance. Computer Physics Communications **97**(1-2) (1996) 1–15
3. Poulson, J., Marker, B., van de Geijn, R.A., Hammond, J.R., Romero, N.A.: Elemental: A new framework for distributed memory dense matrix computations. ACM Transactions on Mathematical Software **39**(2) (February 2013) 13:1–13:24
4. ELPA: Eigenvalue Solvers for Petaflop Applications home page. Website (May 2013) Available online at `http://elpa.rzg.mpg.de`.
5. FZJ-JSC: IBM Blue Gene/Q - JUQUEEN home page. Website (May 2013) Available online at `http://www.fz-juelich.de/ias/jsc/juqueen`.
6. Gutheil, I.: Performance evaluation of scalapack eigensolver routines on two hpc systems. In: 6th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'10). (2010) Available online at `http://juser.fz-juelich.de/record/10376`.
7. Dhillon, I., Parlett, B., Vömel, C.: The design and implementation of the MRRR algorithm. ACM Transactions on Mathematical Software (TOMS) **32**(4) (2006) 533–560
8. Dhillon, I.: A New O(n^2) Algorithm for the Symmetric Tridiagonal Eigenvalue Eigenvector Problem. PhD thesis, University of California, Berkeley (1997)
9. Auckenthaler, T., Blum, V., Bungartz, H.J., Huckle, T., Johanni, R., Krämer, L., Lang, B., Lederer, H., Willems, P.: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. Parallel Computing **37**(12) (2011) 783–794
10. Auckenthaler, T., Bungartz, H.J., Huckle, T., Krämer, L., Lang, B., Willems, P.: Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem. Journal of Computational Science **2**(3) (2011) 272 – 278
11. Gutheil, I., Berg, T., Grotendorst, J.: Performance Analysis of Parallel Eigensolvers of Two Libraries on BlueGene/P. Journal of Mathematics and System Science **2**(4) (2012) 231–236
12. Petschow, M., Peise, E., Bientinesi, P.: High-performance solvers for dense hermitian eigenproblems. SIAM Journal on Scientific Computing (SISC) **In Press, Accepted Manuscript** (2012) – arXiv:1205.2107v2[cs.MS].
13. Münchhalfen, J.: Performance analysis and comparison of parallel eigensolvers on blue gene architectures. Berichte des Forschungszentrums Jülich (Jül-4359) (2013) 65 pp. Available online at `http://juser.fz-juelich.de/record/128657`.