1 SCIENTIFIC HIGHLIGHT OF THE MONTH

The ELPA Library – Scalable Parallel Eigenvalue Solutions for Electronic Structure Theory and Computational Science

Andreas Marek[1], Volker Blum[2,3], Rainer Johanni[1,2](†), Ville Havu[4], Bruno Lang[5], Thomas Auckenthaler[6], Alexander Heinecke[6], Hans-Joachim Bungartz[6], Hermann Lederer[1]

[1]Rechenzentrum Garching der Max-Planck-Gesellschaft am Max-Planck-Institut für Plasmaphysik, D-85748 Garching, Germany

[2]Fritz Haber Institute of the Max Planck Society, D-14195 Berlin, Germany

[3]Mechanical Engineering and Materials Science Department, Duke University, Durham, NC 27708, USA

[4]COMP, Department of Applied Physics, Aalto University, Finland

[5]Fachbereich C, Bergische Universität Wuppertal, D-42097 Wuppertal, Germany

[6]Fakultät für Informatik, Technische Universität München, D-85748 Garching, Germany

Abstract

Obtaining the eigenvalues and eigenvectors of large matrices is a key problem in electronic structure theory and many other areas of computational science. The computational effort formally scales as $O(N^3)$ with the size of the investigated problem, N, and thus often defines the system size limit that practical calculations cannot overcome. In many cases, more than just a small fraction of the possible eigenvalue/eigenvector pairs is needed, so that iterative solution strategies that focus only on few eigenvalues become ineffective. Likewise, it is not always desirable or practical to circumvent the eigenvalue solution entirely. We here review some current developments regarding dense eigenvalue solvers and then focus on the ELPA library, which facilitates the efficient algebraic solution of symmetric and Hermitian eigenvalue problems for dense matrices that have real-valued and complex-valued matrix entries, respectively, on parallel computer platforms. ELPA addresses standard as well as generalized eigenvalue problems, relying on the well documented matrix layout of the ScaLAPACK library but replacing all actual parallel solution steps with subroutines of its own. The most time-critical step is the reduction of the matrix to tridiagonal form and the corresponding backtransformation of the eigenvectors. ELPA offers both a one-step tridiagonalization (successive Householder transformations) and a two-step transformation that is more efficient especially towards larger matrices and larger numbers of CPU cores. ELPA is based on the MPI standard, with an early hybrid MPI-OpenMPI implementation available as well. Scalability beyond 10,000 CPU cores for problem sizes arising in the electronic structure theory is demonstrated for current high-performance computer architectures such as Cray or Intel/Infiniband. For a matrix of dimension 260,000, scalability up to 295,000 CPU cores has been shown on BlueGene/P.

1 Introduction

Finding the eigenvalues and eigenvectors of a large matrix is a frequent numerical task throughout science and engineering. In electronic structure theory, efficient strategies to address the eigenvalue problem have long been a central pursuit.[1]. Most practical solutions of many-electron problems begin with a self-consistent solution of an effective independent-particle problem (e.g., Hartree-Fock[4, 5] or Kohn-Sham theory[6]) that can be discretized into matrix form. If this problem is solved at each step towards self-consistency, a large number of eigenvalue/eigenvector pairs of the Hamilton matrix are needed. The strategies applied to find them (or to avoid having to find them) vary widely, and are often tailored to a specific class of problem. They range from robust, standard algebraic solutions (e.g., in the (Sca)LAPACK library[7]) via iterative strategies of many kinds (e.g., Refs. [1, 2, 8, 9, 10, 11] and many others; best suited if only a small fraction of the eigenvalues and eigenvectors of a large matrix are needed), shape constraints on the eigenfunctions,[12], contour integration based approaches,[13] and many others, all the way to $O(N)$ strategies (e.g., Refs. [14, 15, 16] and many others) which attempt to circumvent the solution of an eigenvalue problem entirely.

The basic numerical task reads as follows: Find $\boldsymbol{\lambda}$ and $\boldsymbol{c}$ such that

$$\boldsymbol{A}\boldsymbol{c} = \boldsymbol{B}\boldsymbol{c}\boldsymbol{\lambda}\,. \tag{1}$$

Here $\boldsymbol{A}$ and $\boldsymbol{B}$ are $N \times N$ matrices and $\boldsymbol{c}$ is the $N \times N$ matrix of eigenvectors for the pair $(\boldsymbol{A}, \boldsymbol{B})$. The diagonal matrix $\boldsymbol{\lambda}$ contains the eigenvalues λ_i (i=1,...,N). If the matrix $\boldsymbol{B}$ is equal to unity, we have a standard eigenvalue problem to solve. Otherwise, the form of Eq. (1) defines a generalized eigenvalue problem. For many cases of practical interest the matrix $\boldsymbol{A}$ is symmetric and has real-valued entries or $\boldsymbol{A}$ is Hermitian having complex-valued entries. The same holds for $\boldsymbol{B}$, which is in addition non-singular and often even positive definite. Then the problem can be transformed to standard form in the course of its solution. In electronic structure theory, the matrix $\boldsymbol{A}$ is usually called the Hamilton matrix $\boldsymbol{H}$. The matrix $\boldsymbol{B}$ is called the overlap matrix $\boldsymbol{S}$, and it is different from unity for the case of a non-orthonormal basis set. The eigenvalues in $\boldsymbol{\lambda}$ are the eigenenergies of the system, and $\boldsymbol{c}$ describes the quantum-mechanical eigenstates of the system in terms of the underlying basis set.

If more than just a few eigenvalue-eigenvector pairs of the matrix $\boldsymbol{A}$ are needed, it is often desirable to perform a direct, algebraic solution of Eq. (1) instead of using an iterative solver. The computational effort to obtain the full set of eigenvalues and eigenvectors scales as $O(N^3)$. Thus, if a single solution on a typical compute node at the time of writing takes several tens of seconds for N=10,000 (a manageable effort and typical time scale as outlined in this paper), a few tens of thousands of seconds will be required for N=100,000, and so on. Clearly, the effort can be somewhat reduced if only k eigenvalue/eigenvector pairs out of the maximum possible number N are needed ($k \leq N$). Still, the $O(N^3)$ bottleneck will impair a straightforward solution of electronic structure problems involving more than (say) 10,000 atoms, but for an accurate description of such systems (especially their electronic structure), quantum-mechanical effects can often still not be simply ignored. Strategies that circumvent the algebraic eigenvalue

[1] See, e.g., Refs. [1, 2] and many later references. A sweeping overview of approaches to the eigenvalue problem in the past century is given in Ref. [3].

problems are an active field,[14, 15, 16] but the $O(N^3)$ "wall" for the general case is still a reality for many applications today.

One way to alleviate the problem is the efficient use of massively parallel compute platforms, which are increasingly available for routine computations. For instance, general-purpose computer architectures can (at the time of writing) offer a few 100,000 CPU cores in the biggest computer installations (see, e.g., the "Top 500" list of supercomputers [17]). By the same measure, smaller installations with hundreds or thousands of CPU cores are now widespread. With such an installation and an ideally scalable eigenvalue solver library, our (hypothetical) N=100,000 problem would be back in the manageable range.

In addition to sheer problem size, there are of course much broader scenarios in which scalable eigenvalue solutions are decidedly desirable. For some problem sizes where the eigenvalue solution is well within the manageable range, traditional $O(N^3)$ diagonalization could even be the fastest available strategy to deal with Eq. (1). However, eigenvalue/eigenvector pairs may be required many times in an individual simulation, along with other computational steps (e.g., in *ab initio* molecular dynamics). Here, parallel execution is obviously beneficial, but only if *all* computational steps scale – including the solution to Eq. (1).

Fortunately, there are now a number of active developments to address parallel eigenvalue solutions for dense matrices and a large number of eigenvalue/eigenvector pairs, including some focused on general-purpose parallel computers,[7, 18, 19, 20, 21, 22, 23, 24, 25] some geared towards multicore architectures (e.g., the PLASMA project [26]) or towards GPUs (Ref. [27] and references therein; most prominently, the MAGMA project [28]).

In this highlight, we review some of these efforts and then focus particularly on the "ELPA" (Eigenvalue soLvers for Petascale Applications) library[22, 29] that facilitates the direct, massively parallel solution of symmetric or Hermitian eigenvalue problems. The roots of ELPA go back to the FHI-aims[30, 31, 32] all-electron electronic structure package. In that code, ELPA is a cornerstone for the parallel efficiency of routine calculations involving up to several thousand atoms,[33, 34] and/or calculations employing (ten)thousands of CPU cores at a time.

The ELPA library relies on the same block-cyclic matrix layout that is provided by the widely used parallel linear algebra library ScaLAPACK. Thus, the ELPA subroutines can function as drop-in enhancements in existing ScaLAPACK-based applications. ELPA also relies on the conventional serial linear algebra infrastructure of LAPACK [35] and the basic linear algebra subroutines (BLAS) that are available in numerous vendor-specific implementations (e.g., IBM's ESSL or Intel's Math Kernel Library, MKL). For particularly performance-critical areas, however, ELPA provides custom linear algebra "kernels" of its own.

In its present form, ELPA is a standalone library, distributed under the lesser GNU General Public License. It is the product of significant development work of a consortium led by the Compute Center of the Max Planck Society (Garching, Germany), and several other entities, initially funded by the German Ministry of Research and Education (BMBF). The chosen license ensures the compatibility of ELPA with both open-source and proprietary implementations. Many of its present users come from a wide range of other electronic structure packages (examples include cp2k, VASP, Quantum Espresso, and others). Perhaps the largest documented application of ELPA is the computation of molecular rotation-vibration spectra using the computer program

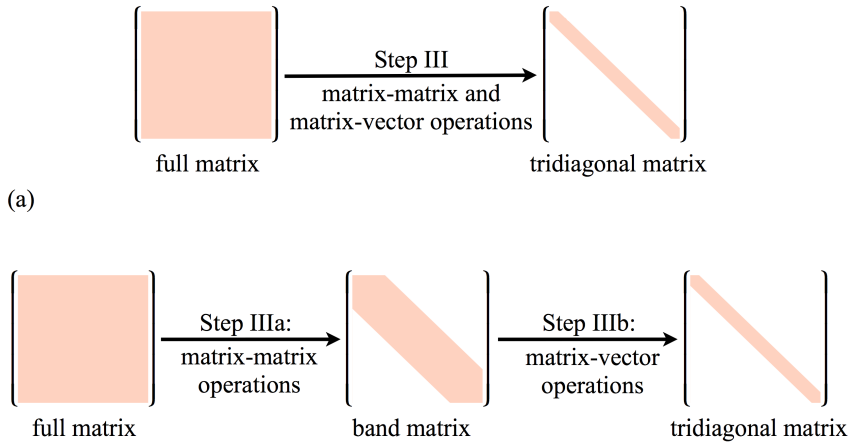


Figure 1: Schematic illustration of the (**a**) one-step vs. (**b**) two-step [38, 22] tridiagonalization schemes employed in ELPA, steps (III) vs. (IIIa) and (IIIb) in the text. The two-step version allows to make full use of matrix-matrix products and sparse matrix-vector products but gives rise to one extra backtransformation step of the eigenvectors (step (V) vs. steps (Va) and (Vb) in the main text).

TROVE [36], allowing them to handle matrix sizes beyond 200,000. ELPA is also included directly in Linux distributions such as Debian or Fedora.

2 The Eigenvalue Problem

The steps to the direct (algebraic) solution of Eq. (1) are conceptually simple and well known (e.g., Ref. [37]). All that is needed are well defined matrix transformations of the problem into a form where its eigenvalues and eigenvectors can easily be found (tridiagonal form), followed by the back-transformation of the eigenvectors to the original standard or generalized problem. We define each step here for a complete notation. Assuming for simplicity that we are dealing with a symmetric eigenvalue problem and the matrices have real-valued entries we have:

(I) Cholesky decomposition of $\boldsymbol{B}$:

$$\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T\,, \tag{2}$$

where $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{L}^T$ is its transpose.

(II) Transformation of Eq. (1) to standard form:

$$\tilde{\boldsymbol{A}}\tilde{\boldsymbol{c}} = \tilde{\boldsymbol{c}}\boldsymbol{\lambda} \tag{3}$$

$$\tilde{\boldsymbol{A}} = \boldsymbol{L}^{-1}\boldsymbol{A}(\boldsymbol{L}^{-1})^T \tag{4}$$

$$\tilde{\boldsymbol{c}} = \boldsymbol{L}^T\boldsymbol{c}\,. \tag{5}$$

Steps (I) and (II) are obviously extra steps for the generalized eigenvalue problems. The next steps are common to generalized and standard eigenvalue problems.

(III) Reduction of $\tilde{\boldsymbol{A}}$ to tridiagonal form (schematically illustrated in Fig. 1a):

$$\boldsymbol{T} = \boldsymbol{Q}\tilde{\boldsymbol{A}}\boldsymbol{Q}^T \tag{6}$$

where $\boldsymbol{Q} = \boldsymbol{Q}_n ... \boldsymbol{Q}_2 \boldsymbol{Q}_1$ and $\boldsymbol{Q}^T = \boldsymbol{Q}_1{}^T \boldsymbol{Q}_2{}^T ... \boldsymbol{Q}_n{}^T$ are the successive products of Householder matrices, which reduce one column or row of the original matrix $\tilde{\boldsymbol{A}}$ and its successive refinements at a time. The elegance of the Householder transform lies in the fact that each step can be written only in terms of a single vector $\boldsymbol{v}_i$: $\boldsymbol{Q}_i = \boldsymbol{I} - \beta_i \boldsymbol{v}_i \boldsymbol{v}_i^T$, where $\boldsymbol{I}$ is the identity matrix and $\beta_i = 2/(\boldsymbol{v}_i^T \boldsymbol{v}_i)$ is just a normalization factor. The individual Householder matrices $\boldsymbol{Q}_i$, of course, never need be formed explicitly. If Householder transforms are applied to a matrix only from one side (as it is the case in computing a QR decomposition of the matrix), they can be *blocked*, allowing almost all arithmetic to take place in highly efficient matrix-matrix operations.[37] In the one-step reduction to tridiagonal form this is prevented by the fact that $\boldsymbol{v}_i$ is determined from the matrix $\tilde{\boldsymbol{A}}_{i-1} = \boldsymbol{Q}_{i-1} ... \boldsymbol{Q}_1 \tilde{\boldsymbol{A}} \boldsymbol{Q}_1{}^T ... \boldsymbol{Q}_{i-1}{}^T$, which requires *two-sided* application of the preceding transforms. By using an implicit representation[35, 7] $\tilde{\boldsymbol{A}}_{i-1} =: \tilde{\boldsymbol{A}} - \boldsymbol{V}_{i-1} \boldsymbol{W}_{i-1}{}^T - \boldsymbol{W}_{i-1} \boldsymbol{V}_{i-1}{}^T$ (where the $i-1$ columns of $\boldsymbol{V}_{i-1}$ just contain the Householder vectors $\boldsymbol{v}_1, ..., \boldsymbol{v}_{i-1}$, and $\boldsymbol{W}_{i-1}$ is a suitable matrix with $i-1$ columns) and building $\boldsymbol{A}_i$ only after a certain number of steps, it is possible to do at least these "explicit builds" with matrix-matrix operations. However, a matrix-vector product with $\tilde{\boldsymbol{A}}_{i-1}$ is still required in each step to determine the next column $\boldsymbol{w}_i$ of $\boldsymbol{W}$, leading to one half of the overall operations to be confined to matrix-vector operations.

If only the forward transformation from $\tilde{\boldsymbol{A}}$ to $\boldsymbol{T}$ were required, it would always be more effective to employ a two-step reduction to tridiagonal form, as shown in Fig. 1b:[38, 22]

(IIIa) Reduction of $\tilde{\boldsymbol{A}}$ to band form:

$$\boldsymbol{D} = \boldsymbol{P} \tilde{\boldsymbol{A}} \boldsymbol{P}^T. \tag{7}$$

(IIIb) Reduction of the (sparse banded) matrix $\boldsymbol{D}$ to tridiagonal form:

$$\boldsymbol{T} = \boldsymbol{O} \boldsymbol{D} \boldsymbol{O}^T. \tag{8}$$

Step (IIIa), the reduction to banded form,[38] relies almost exclusively on efficient matrix-matrix operations.[39, 40] A parallel version has been described in Ref. [22]. Step (IIIb), the reduction of the banded matrix to tridiagonal form, can then be done using Householder transforms, but these now act on a small band of $\boldsymbol{D}$ and constitute much less computational effort than the original step (III).

(IV) Solution of the tridiagonal eigenvalue problem:

$$\boldsymbol{T} \hat{\boldsymbol{c}} = \hat{\boldsymbol{c}} \boldsymbol{\lambda} \tag{9}$$

There are numerous options available for the solution of the tridiagonal problem, e.g., (i) the implicit QL/QR method,[41] (ii) a combination of bisection and inverse iteration,[42, 43] (iii) the divide-and-conquer approach,[44, 45, 46] (iv) and the historically newest "Multiple Relatively Robust Representations" (MRRR) based algorithm.[47, 48, 49, 50] For the purposes of this paper, we note that ELPA relies upon an efficient implementation of the divide-and-conquer approach, the steps of which are reviewed in Ref. [22].

(V) Backtransformation of the k required eigenvectors to the form of the standard eigenvalue problem, $\tilde{\boldsymbol{c}}$. In the case of the one-step reduction technique, step (III) above, only a single backtransformation step is required:

$$\tilde{\boldsymbol{c}} = \boldsymbol{Q}^T \hat{\boldsymbol{c}}\,. \tag{10}$$

In the case of a two-step reduction to tridiagonal form, however, both steps lead to separate backtransformation steps and thus to additional numerical operations (to be weighed against the computational savings of steps (IIIa) and (IIIb)):

(Va) Backtransformation to the intermediate, sparse banded form of the problem:

$$\boldsymbol{c}^{\mathrm{SB}} = \boldsymbol{O}^T \hat{\boldsymbol{c}}\,, \tag{11}$$

(Vb) and backtransformation to the standard eigenvalue problem,

$$\tilde{\boldsymbol{c}} = \boldsymbol{P}^T \boldsymbol{c}^{\mathrm{SB}}\,. \tag{12}$$

(VI) In the case that the original eigenvalue problem was a generalized one (in quantum mechanics, for a non-orthonormal basis set), the eigenvectors need to be transformed to the original generalized basis set one more time, as defined in Eq. (5).

The point of writing down steps (I)-(VI) explicitly is to de-mystify them (if needed), since the basic algorithm is indeed straightforward. The key challenges for an efficient solution arise in the details. Even on a single CPU, it is well known that the use of matrix-matrix operations packaged in computer- and/or vendor-specific BLAS subroutines is greatly preferable over directly implemented products, or indeed over simpler matrix-vector operations. On multiple CPU cores, the simple fact that matrix sizes grow in memory as N^2 shows that it is essential to distribute all matrices and their intermediates over the available cores and nodes. At this point, it becomes important to utilize storage patterns that avoid any communication between different CPU cores if possible, and to minimize the remaining (unavoidable) communication between CPU cores.

It should be mentioned that the above sequence is not the only one for computing the eigensystem via an intermediate banded matrix; in particular, the backtransformation in step (Va) can be replaced with different computations. For very small intermediate bandwidths it might be more efficient to replace steps (IIIb) through (Va) with a band divide and conquer method delivering eigenvalues and eigenvectors of the banded matrix [51]. A narrow-band reduction algorithm coupled with a band divide and conquer approach was also described and successfully implemented in the new EigenExa library.[21, 25] If orthogonality is not an issue then the eigenvectors can also be computed as the null spaces of shifted banded matrices [52, 53]. Finally, even the tridiagonalization of the banded matrix (steps (IIIb) and (Va)) may be subdivided further[39, 40, 54].

3 Recent Developments in Parallel Eigenvalue Solvers

For the serial (single CPU core) case, highly efficient solution strategies to steps (I)-(VI) have long been available in the framework of libraries like LAPACK[35], its predecessors, and highly optimized basic linear algebra subroutines (BLAS)[55, 56, 57] in optimized libraries such as ATLAS[58], the (no longer maintained) GotoBLAS[59], a successor project, OpenBLAS,[60, 61] or vendor-specific implementations such as e.g., IBM's ESSL or Intel's MKL. Beyond single-CPU core implementations, shared-memory implementations of these libraries also exist using

the OpenMP framework, or for GPU architectures. However, a different approach is required for the large, distributed-memory computer installations that are commonly used for high-performance computing today.

With the ScaLAPACK library[7], a comprehensive linear algebra library (far more than just eigenvalue solvers) for distributed-memory machines emerged in the 1990s, directly from the same roots as LAPACK and BLAS. In ScaLAPACK, most communication is carried out by an intermediate layer of Basic Linear Algebra Communication Subroutines (BLACS). ScaLAPACK makes extensive use of matrices distributed across processors in a block-cyclic layouts (see Figure 2 and the associated discussion below). In many ways, ScaLAPACK thus provides a clearly defined standard for parallel linear algebra and is commonly employed as the reference implementation against which other developments are benchmarked. High-performance linear algebra libraries from vendors such as Intel (MKL) or IBM (ESSL) feature interfaces compatible with ScaLAPACK.

A similarly broad effort from the same era is the PLAPACK library[19, 18, 20]. The (self-described) distinction from ScaLAPACK was the adoption of an object-based coding style, yielding improved execution times over ScaLAPACK for certain operations.[20] This picture was not uniform, however; for example, a 2003 eigenvalue solver benchmark[62] showed that the implementation in ScaLAPACK on the particular computer system used (an IBM Power4 installation) was generally faster.

A third, long-running and broad effort for parallel linear algebra is the Portable, Extensible Toolkit for Scientific Computation (PETSc).[63, 64, 65] However, regarding eigenvalue solvers, the emphasis in the PETSc environment is currently placed on iterative eigenvalue solvers, e.g., in the Scalable Library for Eigenvalue Problem Computations (SLEPc) [66] which is based on PETSc. Many more iterative solvers for large-scale sparse eigenvalue problems are available, e.g., via the Anasazi package[67] in the Trilinos software project[68]. However, the focus of the present paper is on direct methods.

When the ELPA effort began (in 2007), a key motivation was the need to extend the capabilities of the then-current parallel eigenvalue solvers to architectures like IBM's BlueGene/P, where several thousand CPU cores needed to be utilized efficiently and concurrently. As documented in Ref. [22], this effort met with some difficulty, possibly simply because existing libraries had not yet been designed with installations at hand that featured such large processor numbers. This situation has changed for the better in recent years, with several different promising projects taking on the dense eigenvalue problem.

In particular, the Elemental library [23] has emerged as a C++ based framework for parallel linear algebra, which includes components for full and partial symmetric and Hermitian Eigenvalue problems. To date, Elemental includes only the one-stage reduction to tridiagonal form, (III) above. On the other hand, the algorithm employed for the solution of the tridiagonal problem, (IV), is a rather new parallel implementation of the MRRR algorithm by Petschow and coworkers.[24]

A recent detailed benchmark compared components from ScaLAPACK, Elemental, and the ELPA library on the BlueGene/Q architecture [69], including the important aspect of hyper-threading (the possible use of more than one MPI task per CPU core) on that machine. In short,

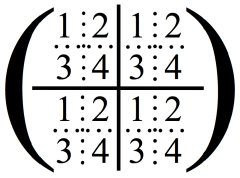


Figure 2: Schematic visualization of a block-cyclic distribution of a matrix on a 2×2 processor grid (four processors in total) that would be used in ELPA. In the example, the matrix is divided into several primary blocks (full lines), and each primary block is divided into four sub-blocks or "tiles" (dotted lines). The number in each tile indicates the CPU number on which it is stored and handled. The width and height of the tiles is given by a fixed parameter n_{blks} with typical values between 16 and 64. n_{blks} thus indirectly determines the overall number of primary blocks.

ELPA and Elemental showed the best performance on this architecture, with a slight edge for the two-step reduction approach (IIIa) and (IIIb) as implemented in ELPA.

A second new effort is the EigenExa effort pursued at the RIKEN Advanced Institute for Computational Science in Japan.[25, 21] As mentioned above, this library incorporates a narrow-band reduction and a band divide and conquer method, thus modifying steps (III)-(V) above. Another very recent benchmark[70] for the K computer, a 640,000 core, SPARC64 processor based distributed-memory computer, shows very promising results for this new approach.

The above discussion gives a flavor of, but certainly cannot fully cover all developments targeted at "traditional" architectures. In addition, we note again the ongoing efforts towards the creation of dedicated libraries for new multicore or GPU based computer architectures, including the PLASMA project[26] and the MAGMA project.[28] An overview of eigensolver-related developments towards distributed multi-GPU installations can be found in Ref. [27], including a direct performance comparison between the GPU-based solution and the traditional distributed-memory solutions implemented in ScaLAPACK and ELPA.

As the broader message, several independent, promising parallel dense linear-algebra based eigenvalue solver developments are now available to cope with the proliferation of new massively parallel distributed computers as well as with new architectural models, particularly GPUs. The remainder of this highlight reviews the underlying model and performance achieved by one of them, the ELPA library, in greater depth.

4 The ELPA Library

4.1 Data Layout and Basic Setup

In parallel linear algebra, the first key decision concerns the layout of the matrices involved (the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as well as many others needed in the intermediate steps). In line with what is arguably the industry standard today for dense matrices, ELPA employs block-cyclic distributions for its data (matrices and eigenvectors), a mode of distribution summarized in detail in the ScaLAPACK manual [7]. An example of a block-cyclic distribution for a 2×2 processor grid (four CPUs total) is shown in Fig. 2. In short, a matrix is divided into a number of

primary blocks. Each primary block is distributed across all available CPU cores. The width and height of the individual matrix sub-blocks (or "tiles") located on an individual CPU, n_{blks}, is an important parameter for parallel load-balancing and efficiency (on standard distributed parallel compute platforms, typical values for good performance for the most time-critical operations range between 16 and 64[22]). Choosing this layout has the additional advantage that it is easy to insert ELPA into existing code that is already set up for ScaLAPACK.

In Fig. 2, it is important to note that the number of tiles in each sub-block is chosen to match the number of available processors. The number of primary blocks then follows from the chosen tile width, n_{blks}. The primary blocks can be arranged so as to greatly restrict the necessary communication in certain operations. In Fig. 2, this is evident for the example of a matrix transposition. This would require only the processor pair (2,3) to communicate.

In practice, ELPA supports rectangular processor grids to split each matrix block into sub-blocks:

$$N_{\mathrm{CPUs}} = N_{\mathrm{cols}} \cdot N_{\mathrm{rows}} \quad , \tag{13}$$

with N_{CPUs} the number of CPU cores available, and N_{cols} and N_{rows} the number of columns and rows into which each matrix block is split. As mentioned above, N_{cols}, N_{rows} and n_{blks} determine the number of primary matrix blocks used. Each processor is associated with a single pair of column/row numbers in each primary block.

The above layout corresponds exactly to the pioneering ScaLAPACK and Basic Linear Algebra Communication Subroutines (BLACS) libraries. Beyond the layout, the ELPA subroutines themselves do not rely on either library. Instead, all communication between different processors is handled by direct calls to a Message Passing Interface (MPI) library, another standard part of any parallel computing system today.

For its communication, ELPA relies on two separate sets of MPI communicators, connecting either the processors that handle the same rows or the same columns of the distributed matrix blocks (row communicators and column communicators, respectively). This choice corresponds closely to what is needed in matrix multiplications and helps avoid any unnecessary communication between all processors at once. Additionally, we note that a (near-)square layout of processor rows and columns can give better performance but is not required.

Once the input matrices are distributed in block-cyclic fashion and the column and row communicators are set up, the actual eigenvalue problem can be solved by ELPA.

4.2 ELPA 1

The main linear algebra steps (I)-(VI) outlined in Sect. 2 can be solved in essentially a straightforward way. The implementation of these basic steps is here referred to as "ELPA 1" (corresponding to a source code file "elpa1.F90"). Importantly, the reduction to tridiagonal form (III) is carried out in vector steps, i.e., each Householder vector is determined one by one. Since this steps constitutes the dominant workload and show limited scalability, its "two-step" alternative (IIIa), (IIIb) (and therefore also (Va), (Vb)) is available in an additional library ("ELPA 2") below. We first comment briefly on the most important algorithmic choices made in "ELPA 1".

4.2.1 Overall Choices

Perhaps the most significant overall choice made in ELPA is to keep things simple. The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are assumed to be real symmetric or complex Hermitian, and only these choices are supported. The matrices are assumed to have been correctly set up on input (a correct assumption in most practical programs that use ELPA), i.e., any redundant layers of consistency checks at the outset of each step are avoided.

The actual programming hierarchy is flat, i.e., parallel linear algebra and MPI communication are conducted outrightly and step by step in the individual subroutines. Any auxiliary linear algebra subroutines called from the ELPA routines are serial themselves, typically LAPACK, LAPACK-like and BLAS subroutines that are available in efficient, highly optimized version in computer- or vendor-specific libraries.

Finally, the necessary communication between individual processors is concentrated in as few steps as possible, making use of the efficient separate column and row MPI communicators described above.

4.2.2 Cholesky Decomposition

In the case of a generalized eigenvalue problem, the Cholesky decomposition of the overlap matrix $\boldsymbol{B}$ is typically not the computationally dominant step. Although linear-scaling computational approaches to the Cholesky decomposition exist [71], ELPA implements the standard algorithm, working only on the matrix tiles which are on the diagonal or above. Intermediate serial steps are carried out using standard LAPACK linear algebra, including actual Cholesky factorizations of single blocks on the diagonal. Communication is restricted to individual processor rows and columns as usual in ELPA, with the exception of a special operation which transposes vectors of individual rows and columns. As mentioned above, block-cyclic arrangements can here be chosen such that only processor pairs that interchange two blocks need to communicate, i.e., no all-to-all communication is needed.

Historically, the primary motivation to include the Cholesky transform into ELPA itself was, in fact, not a performance need, but rather the necessity to circumvent frequent technical problems encountered in third-party implementations of the Cholesky transform.

4.2.3 Transformation to Standard Form

For the transformation to standard form (step (II) above), we require the inverse of the Cholesky factor $\boldsymbol{L}$ and its transpose. As outlined briefly in Ref. [30], using backward/forward substitution to calculate the product of the inverse Cholesky factors with $\boldsymbol{A}$ was found to become a scalability bottleneck.

In ELPA, the inverse of the Cholesky factors is thus calculated outrightly and stored for later use. This is done using a specially written routine for the parallel inversion of an upper triangular matrix, running over the tiles of the block-cyclic distribution one by one. In each step, the LAPACK subroutine for the inversion of a triangular matrix, `dtrtri`, is employed on the lowest remaining block. Using the column and row communicators, the resulting transformations are

broadcast and applied to all other remaining blocks in turn.

The matrix multiplications of $\boldsymbol{A}$ from the left and right, respectively, are performed using a custom-written parallel matrix multiplication routine which allows to take the product of a triangular matrix with a full matrix. The standard parallel BLAS (PBLAS) routine `pdtran` can be used for a matrix transpose operation inbetween.

The context of electronic structure theory is particularly advantageous for this strategy since the Cholesky factors and their inverse do not change during a self-consistency cycle if the basis set and the nuclear coordinates are fixed. Thus, in the iterative self-consistent field case, the inverse Cholesky factors can simply be precomputed once and stored before use in multiple s.c.f. iterations that correspond to the same overlap matrix $\boldsymbol{B}$.

4.2.4 Reduction to Tridiagonal Form by Successive Householder Transforms

This is the most time-critical step of the algorithm. In "ELPA 1", the column by column and row by row based reduction (step (III) above) is implemented in an efficient way, using an implicit representation as described in Sec. 2. In practice, ELPA works on the full matrix $\tilde{\boldsymbol{A}}$, although in the real case, this matrix is in principle symmetric. One important aspect is that the "algorithmic block size" k, i.e., the number of steps between explicit builds of the current matrix $\tilde{\boldsymbol{A}}_i$, does not have to be connected to the memory block size of the block-cyclic distribution, so an independent efficient choice of k is possible ($k = 32$ being a typical value). Eventually, the diagonal and off-diagonal elements of $\boldsymbol{T}$ are stored as vectors, and $\tilde{\boldsymbol{A}}$ is overwritten with the Householder vectors (needed for the backtransformation of the eigenvectors).

In terms of special tweaks in ELPA, it is worth noting that for real symmetric matrices, individual matrix blocks can be used for column and for row operations right after one another, keeping them in the processor cache and thus improving the performance. As before, favorable block-cyclic arrangements can be chosen to minimize communication, e.g., for a matrix transposition, such that only processor pairs that interchange two blocks need to communicate. The overall communication is otherwise again restricted to individual processor rows and columns, respectively. The final storage of the Householder vectors in $\tilde{\boldsymbol{A}}$ happens in groups of $k \leq 32$ rows and columns, each. Overall, this leads to a rather scalable implementation, however still limited by the fact that matrix-vector are carried out in addition to matrix-matrix operations.

4.2.5 Solution of the Tridiagonal Problem

To solve the tridiagonal problem (step (IV) above), ELPA relies on the divide and conquer algorithm. In the context of ELPA, this algorithm has been summarized completely in Ref. [22]. In particular, a simple but very helpful modification is to carry out the solution of the problem only for the needed eigenvectors, which is possible in the final (but most expensive) step of the algorithm.

In ELPA, care was taken to ensure that any trivial parallelism in the algorithm is exploited. Thus, upon each split of the problem into two half-sized problems, the available processors are also split into two subsets that take care of each half-sized problem on their own. This strategy

is pursued recursively until only a single processor is left. From that point on, the remaining partial problem can be solved with the existing, efficient serial implementation of the divide and conquer algorithm in LAPACK.

4.2.6 Backtransformation to Standard Form

In ELPA 1, the reduction to tridiagonal form is carried out in only a single step, i.e., the corresponding backtransformation also requires only a single step, step (V) above. The columns of $\hat{c}$ are transformed with the Householder matrices from the tridiagonalization, in reverse order, and combining n_b of them into a blocked transformation. An important sub-step is the separation of the algorithmic block size n_b – the size of the blocks of matrices that are multiplied in a single BLAS call – and the memory block size of the block-cyclic distribution. While the latter can be relatively small (typically, between 16 and 64), the matrix blocks to be multiplied in the back transformation have a relatively large dimension, normally 256. This value is only reduced if the ratio of the full matrix dimension to the number of processor rows is itself smaller than 256. This modification guarantees an optimally efficient use of the typically highest optimized BLAS operation on a single processor, i.e., a matrix product.

4.2.7 Backtransformation of the Eigenvectors to the Generalized Form

This final step (step (VI)) is carried out as outlined for step (II) above. Again, a special parallel matrix multiplication routine optimized for triangular matrices is employed.

4.3 ELPA 2

The central modifications in "ELPA 2" concern the most compute-intensive parts of the algorithm: The reduction to tridiagonal form and the corresponding backtransformation of the eigenvectors.

As mentioned in Sec. 2, the most important modification is the separation of the reduction into two steps, (IIIa) and (IIIb). The first step reduces the matrix to a banded form, which employs efficient matrix-matrix multiplications. The second step reduces the banded matrix to tridiagonal form, but this time only faces a sparse matrix for the necessary matrix-vector operations. Both steps can be optimized much better than the usual one-step vector-based reduction.

The price to pay is an additional backtransformation step ((Va) and (Vb) above). Thus, it is critical to carry these steps out as efficiently as possible. Since often, only a fraction of the eigenvectors is needed, the overall "ELPA 2" approach lends itself particularly well to scenarios where a sizeable part (say, 10 %-50 %) of the eigenvectors is required. This is often the case when compact basis sets are used to discretize the Kohn-Sham Equations, for final sub-space rotation steps of iterative eigenvalue solvers, for optical spectra where many but not all eigenvalues / -vectors are needed, etc. We also note that, in practice, we find that ELPA 2 is usually superior to ELPA 1 even when all eigenvalues and eigenvector pairs are required, at least for matrix dimensions N of a few thousand and above.

The key technical underpinnings of the two-step reduction in "ELPA 2" have been summarized in detail in Ref. [22]. From a practical point, what is most important is that the overhead from the additional backtransformation step (Va) can be minimized quite efficiently by turning to a transformation involving several Householder vectors at once and a two-dimensional layout of the resulting matrices (allowing parallel scalability beyond what would be possible with a one-dimensional layout). Another key step is the introduction of small so-called "kernel" subroutines which implement this strategy in the computationally most effective way, making sure that operations can be carried out in the processor cache as far as possible. Thus, these kernels can be architecture-dependent in their most efficient variants.

At the time of writing, the ELPA 2 library contains several different kernels for the compute intensive Householder transformations in step (Va). These kernels are optimized for good performance on different platforms and currently support

- IBM Bluegene/P and BlueGene/Q
- X86 SSE: for platforms that support Intel's SSE 4.2 "Single Instruction Multiple Data" (SIMD) vectorization, e.g. Intel Nehalem CPUs
- X86 AVX: for platforms that support Intel's AVX SIMD vectorization, e.g. Intel SandyBridge/IvyBridge CPUs
- AMD Bulldozer
- generic: for routine use on any platform

The generic kernel is written in standard Fortran and ensures that the ELPA library can be used on every system. The other kernels are intended for use on specific architectures and are either written in assembler or with intrinsic machine instructions. In order to use these machine-specific ELPA kernels, a separate compilation (e.g., using the GNU compiler tools) is usually necessary. The ELPA library is distributed with a standard "configure & make" procedure to facilitate this process.

Depending on the respective platform, the kernels are optimized with regard to cache efficiency and – where possible – SIMD vectorization, and make use of machine specific intrinsic or assembler instructions. Compared to the generic kernel, the optimized versions lead to a measurable performance gain. As an example, Table 1 shows a comparison between the generic kernel and the AVX optimized kernel. The measurements were done on an Intel SandyBridge system with Intel Fortran Compiler 14.0 and GNU compiler 4.7.3. Obviously, on a SandyBridge (or IvyBridge) system the optimized version gives about twice better performance than the generic kernel.

In practical use, the architecture-specific kernels are intended as "expert tools". For normal use, the generic ELPA kernel provides good baseline efficiency and is easily compiled with any Fortran compiler. The kernels thus do not constitute a barrier to make ELPA usable in practice. On high-end supercomputers, however, it is possible and worthwhile to carry out specific performance tests once and provide a pre-compiled, optimum kernel in the ELPA library for end users.

Table 1: A comparison of the generic ELPA 2 kernel and the AVX optimized kernel. Measurements were done on two Intel SandyBridge cores. As an example, a real and complex valued matrix with N=5,000 were constructed and 100%, 50%, and 10% of the eigenvectors were computed.

		Generic kernel		AVX kernel	
Number of EVs [%]	matric type	kernel time [s]	GFlops/s	kernel time [s]	GFlops/s
100	real	13.06	9.66	6.75	18.6
50	real	6.48	9.76	3.39	18.5
10	real	1.26	9.9	0.67	18.7
100	complex	60.64	8.2	26.24	19.1
50	complex	30.25	8.3	13.28	18.9
10	complex	6.0	8.3	2.70	18.6

5 ELPA Performance on Reference Platforms

Some performance benchmarks for the ELPA library have already been published on HPC platforms in the form of technical reports. On the former "JUGENE" BlueGene/P installation in Jülich (now decommissioned), strong scaling up to 295,000 CPU cores was shown for a test matrix with N=260,000.[72] Likewise, a whitepaper covering "tier-0" systems of the European "Partnership for Advanced Computing in Europe" (PRACE) high-performance computing (HPC) infrastructure showed excellent scalability of the tested ELPA subroutines for relatively low processor numbers (up to 8,192 processors on BlueGene/P; up to 1,024 processors on an Intel Nehalem based system) and matrix sizes between N=3,888 and 20,480.[73]

In the following, we present performance benchmarks of our own on new, large HPC systems.

All performance results presented in Sections 5.1–5.3 were done on the new HPC system "Hydra", an Intel IvyBridge based X86_64 cluster of the Garching Computing Centre of the Max-Planck-Society (RZG). Each compute node of the system is equipped with two Intel Xeon E-2680v2 "IvyBridge" CPUs (10 cores per CPU), operating at 2.8 Ghz, with a peak performance of 448 GFlop/s. Internode communication is done with an Infiniband FDR-14 network with a maximum bandwidth of 56 Gb/s. The ELPA software is compiled with Intel's Fortran Compiler 13.1 and GNU gcc 4.7.3 where necessary.

The results in Sec. 5.4 were obtained on "Sisu" the Cray XC30 supercomputer of CSC – IT Center for Science in Finland. Each node of Sisu is equipped with two Intel Xeon E5-2670 CPUs with 8 cores per CPU. The system has a proprietary interconnect "Aries" and its theoretical peak performance is 244 TFlop/s. FHI-aims is compiled with Intel's Fortran Compiler 13.1.

5.1 Overall Performance of ELPA

We compare the performance of ELPA to Intel's Math Kernel Library (MKL) 11.0, which implements ScaLAPACK-like interfaces. We use two sets of MKL subroutines: (i) the MKL/ScaLAPACK routines `pdsyevr` and `pzheevr` for real and complex matrices, respectively. These routines are rather new and allow the extraction of only a part of all eigenvectors, as ELPA does. Thus, the

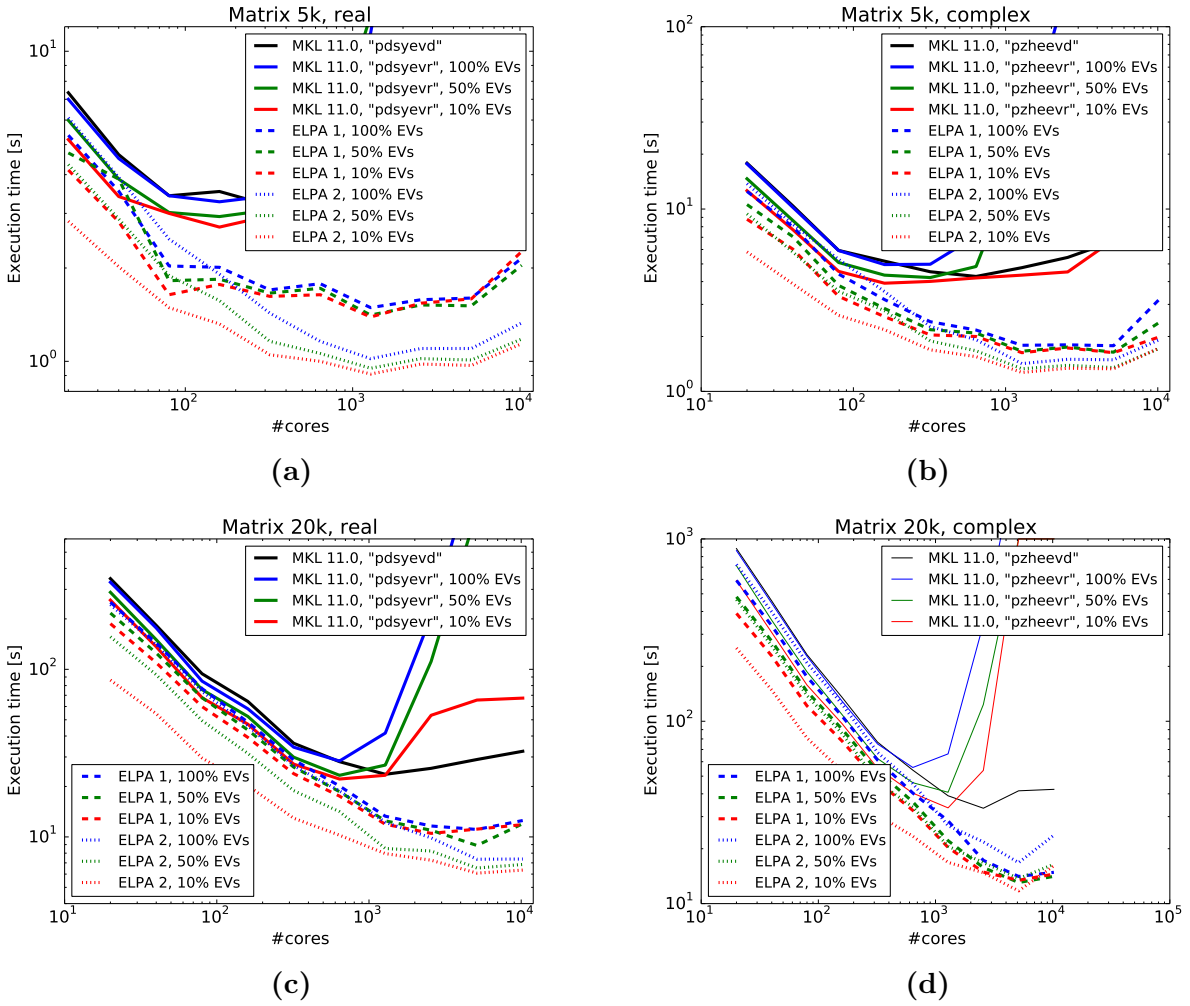
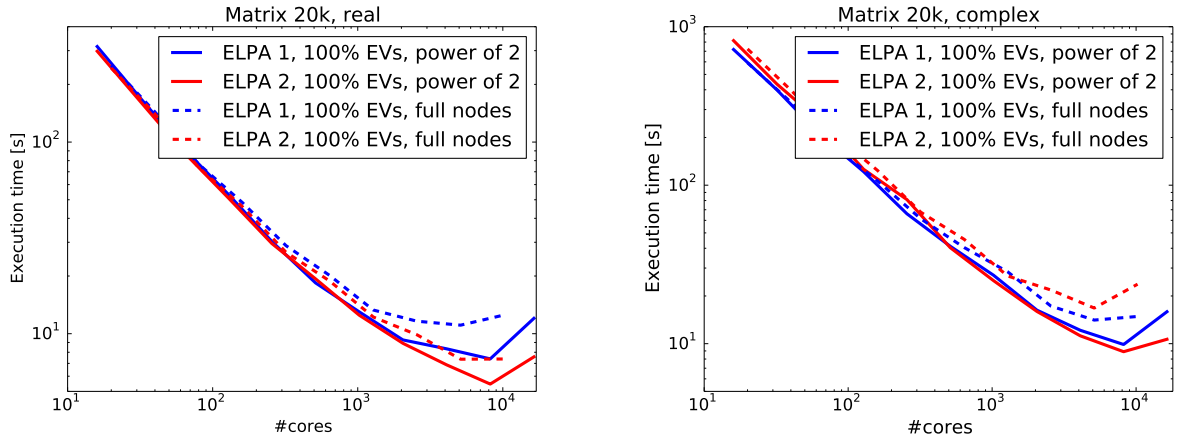


Figure 3: Scaling plots of ELPA and MKL 11.0 on the Intel SandyBridge Cluster. Measurements were done on "full" nodes, of 20 cores (1 node), 40 cores, 80 cores ... up to 10240 cores (512 nodes). Run times are shown for real matrices of size N=5,000 (**a**) and N=20,000 (**c**), and complex matrices for the same matrix sizes (**b** and **d**), respectively. For each matrix, fractions of 100%, 50%, and 10% of the eigenvector spectrum were calculated. Note that the ScaLAPACK/MKL routines `pdsyed` and `pzheevd` always compute all the eigenvectors.

comparison is formally on equal footing. However, as shown in Fig. 3, the `pdsyevr` and `pzheevr` routines in MKL are not yet fully optimized. Thus, we also include (ii) performance results of the routines `pdsyevd` and `pzheevd`, which compute the full set of eigenvectors but show better scalability.

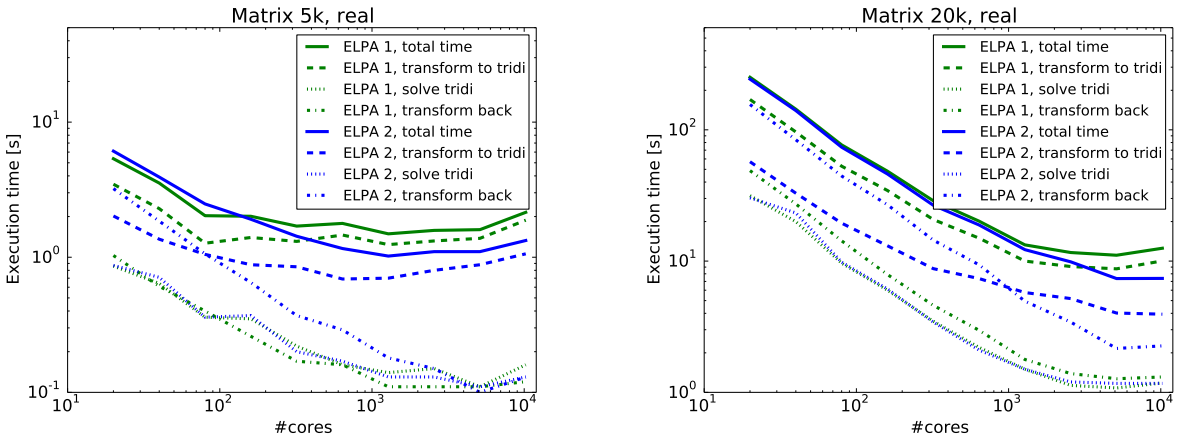
Both ELPA and MKL benchmarks were done with IBM MPI implementation version 1.3 for X86_64 systems. ELPA was built with the kernel routines optimized for AVX SandyBridge and IvyBridge processors.

In Figure 3, we show scaling plots of MKL 11.0, ELPA 1, and ELPA 2 for both real and complex matrices of size N=5,000 and 20,000. The computed fraction of the eigenvector spectrum is 100%, 50%, and 10%, respectively. Throughout the investigated core range (20 to 10,240 cores), ELPA 1 and ELPA 2 both show lower (better) execution times than `pdsyevr` and `pzheevr` in this version of MKL. For low processor counts, i.e. 20 and 40 cores, the time difference to MKL



(a) **(b)**

Figure 4: Comparison of ELPA runs with core counts which correspond to powers of two vs. core counts which correspond to full IvyBridge nodes (20 cores per node), i.e. multiples of 20. Powers of two imply implies that all involved nodes except for one are used with 20 cores per node. Only on the last node, fewer than 20 cores are used. The representative cases of real valued (**a**) and complex valued (**b**) matrices are shown. In both cases, the full eigenvalue spectrum was computed.



(a) **(b)**

Figure 5: Detailed sub-timings of ELPA 1 and ELPA 2 for real valued matrices of size N=5,000 (**a**) and size N=20,000 (**b**), respectively. In both cases, as a typical representation, 100% of the eigenvalue spectrum are computed. Shown are the total time to solution (solid) and the timings of the three major steps transformation to tridiagonal form (dashed), solution in tridiagonal form (dotted), and back transformation (dashed-dotted). Obviously, the transformation to tridiagonal form is the most expensive and also determines the scaling behaviour.

is in the range of 10% to 50%. However, with increasing core count the difference becomes much larger. ELPA 1 and ELPA 2 scale up to almost 5,000 cores in the investigated setups, whereas MKL effectively stops scaling at ≈100 cores (N=5,000) and ≈1,000 cores (N=20,000), respectively. ELPA does not scale beyond ≈10,000 cores. However, the time to solution is still small and does not increase drastically beyond 10,000 cores. For real-world applications, this

behaviour is a critical feature. Even if the eigenvalue solver formally no longer scales, other parts of the calculation will continue to scale. The low overall execution time of ELPA in the limit of large processor counts will help push out the "crossover point", i.e., the number of processors beyond which the eigensolver dominates the entire calculation.

We also show performance results of Intels MKL implementation of the ScaLAPACK `pdsyevd` and `pzheevd` routines for real and complexed valued matrices. These routines always compute all the eigenvectors, which implies that they have to perform worse than ELPA if only a fraction of the eigenvectors is needed. Although neither routine outperforms ELPA, it is interesting to see that these routines currently still show better scalability than the MKL routines that allow one to limit the calculation to only a fraction of the eigenvalue / eigenvector pairs.

5.2 Role of the Block-Cyclic Layout: Powers of Two

As explained in Sec. 4.1 and shown in Fig. 2, ELPA relies on a processor grid that can in principle be rectangular and still maintain a lean communication pattern for certain operations (notably, the transposition of columns and rows). In the example and in the initial implementation of ELPA, this was first done for processor counts that are powers of two. Figure 4 therefore compares the performance of ELPA for processor counts in multiples of 20 with processor counts in powers of two on the same machine. In short, a small performance gain is possible by relying on "power of two" processor counts. This, however, is rather minor for all practical situation in which the overall scalability is not yet exhausted. Only for the largest tested processor counts (well above 1,000), a significant performance benefit from "powers of two" can be exploited. We note that all computations employ full nodes, i.e., only the last node in each "power of two" run is not completely used. Thus, there is no intrinsic difference between both setups other than the processor count. In particular, a simple memory bandwidth effect can be excluded as the reason for the observed, slightly superior scalability observed for "power of two" processor counts.

5.3 ELPA Sub-Timings for Individual Computational Steps

In Fig. 5, we show the sub-timings and the scaling behaviour of the three major computational steps (III)-(V) of Sec. 2, i.e., the transformation to tridiagonal form, the solution of the tridiagonal problem, and the back transformation of the eigenvectors. Fig. 5 shows as a representative case the timings if 100% of the eigenvectors are computed. It is clear from Fig. 5 that the transformation to the tridiagonal form and the corresponding back transformation are still the computationally most expensive parts. In these setups (100% of the eigenvectors), the back transformation in ELPA 2 actually dominates for small processor counts, while the scaling limit is dominated by the behaviour of the transformation to tridiagonal form.

Interestingly, even for 100% of the eigenvalue spectrum (the worst case for ELPA 2), ELPA 2 mostly outperforms ELPA 1 in these benchmarks. This is a direct consequence of the availability and use of the optimized ELPA linear algebra kernels for the back transformation steps.

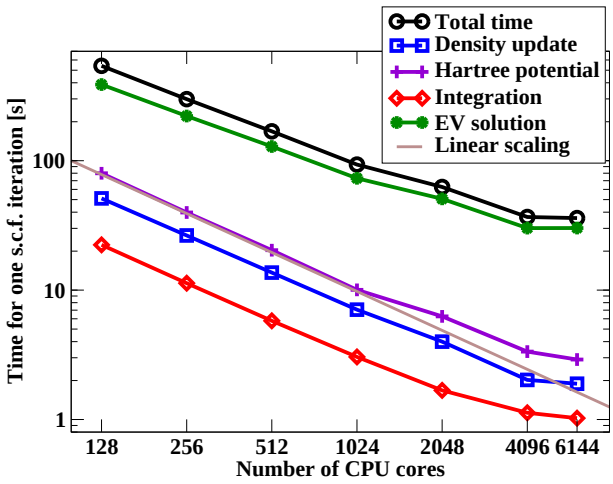


Figure 6: Subtimings for one s.c.f. iteration of a polyalanine molecule having 2,003 atoms computed using FHI-aims on a Cray XC30 supercomputer. The system has 54,069 basis functions and 6,810 eigenvalues and vectors are solved for at each s.c.f. iteration using ELPA 2. The solution time for the eigenproblem is 31 seconds using 6,144 CPU cores.

5.4 ELPA in FHI-aims

To put ELPA into the context of a practical application, we focus on a large large-scale electronic structure problem using the FHI-aims code. Figure 6 shows timings for a single s.c.f. iteration of large polyalanine molecule with 2,003 atoms on a Cray XC30 supercomputer, employing the standard "tight" numerical settings and *tier 2* numeric atom-centered orbital basis set of FHI-aims.[30] Very good scalability is achieved up-to 4,096 CPU cores and going to 6,144 cores gives still some improvement but this is no longer due to ELPA. The fastest time achievable for such a set-up is 37 seconds using 6,144 CPU cores. As is evident from Fig. 6, for this system size the dominant cost for the calculation is the solution of 6,810 eigenvalues and vectors out of the 54,069 possible ones. The other tasks of the s.c.f. cycle are (most importantly) the integration of the Hamilton matrix elements, the computation of the electron density and its gradients, and the calculation of the Hartree potential, all of which happen on a non-uniform real-space grid. These tasks take less than three seconds each for the largest CPU core counts. Here the advantage of ELPA 2 is also obvious since it takes only 31 seconds to solve the eigensystem compared to 128 seconds using ELPA 1 with 6,144 CPU cores.

As Fig. 6 demonstrates, for such a calculation the scalability of the eigensolver is of central importance. The time for one s.c.f. iteration is dominated by the eigensolver consuming about 80% of the iteration time. Use of ELPA on Cray can be encouraged since according to our tests ELPA also outperforms Cray's ScaLAPACK implementation in LibSci on XC30.

6 Hybrid MPI-OpenMP implementation of ELPA

Currently, a hybrid MPI-OpenMP (i.e., distributed-shared memory) version of the ELPA 1 and ELPA 2 library is under development. For a fixed number of available cores, the idea of the

Table 2: Comparison of the plain MPI version of ELPA 1 with the hybrid MPI-OpenMP development version of ELPA 1. Runtimes are shown for different numbers of cores for a matrix with N=5,000. Both real and complex values matrices were investigated and 100% or 50% of the eigenvectors were computed.

			Execution time [s]	
#cores	number of EVs	matrix entries	ELPA-MPI	ELPA-HYBRID
160	100 %	real	2.01	1.83
640	100 %	real	1.78	1.49
2560	100 %	real	1.58	1.26
5120	100 %	real	1.60	1.16
160	50 %	real	1.84	1.72
640	50 %	real	1.72	1.45
2560	50 %	real	1.52	1.22
5120	50 %	real	1.51	1.15
160	100 %	complex	3.19	2.94
640	100 %	complex	2.17	1.93
2560	100 %	complex	1.80	1.53
5120	100 %	complex	1.78	1.50
160	50 %	complex	2.85	2.67
640	50 %	complex	2.09	1.85
2560	50 %	complex	1.75	1.66
5120	50 &	complex	1.64	1.43

hybrid version is to have fewer MPI tasks with a bigger computational workload. The workload for each MPI task is then subdivided to the different OpenMP threads, effectively leading to lower communication in MPI. This means that the hybrid version should become superior to the plain MPI version for large numbers of processor cores, for which a large number of MPI messages is sent. In such a case, it might be possible to achieve improved scaling with a hybrid version of ELPA.

Since the development of the hybrid version of ELPA is ongoing, we here only briefly demonstrate the capability of the present version, labelled 2013.11. Execution times for various scenarios are summarized inTables 2 and 3. All measurements were done on the same machine as the pure MPI runs, the HPC system "Hydra" at RZG (see Section 5).

In particular, we compare – for a subset of core counts – the runtimes of the hybrid version of ELPA with the pure MPI version for a N=5,000 matrix. Results for the N=20,000 matrix (not shown) are qualitatively similar. Already at the time of writing, the hybrid variant of ELPA can yield an additional performance improvement of about 10% to 40% over the plain MPI version. As expected, larger performance benefits are found for higher core numbers, which indeed increases the scalability.

Table 3: Same as Table 2, however, ELPA 2 values are shown.

			Execution time [s]	
#cores	number of EVs	matrix entries	ELPA-MPI	ELPA-HYBRID
160	100 %	real	1.91	1.97
640	100 %	real	1.16	1.13
2560	100 %	real	1.10	0.85
5120	100 %	real	1.10	0.78
160	50 %	real	1.57	1.57
640	50 %	real	1.06	1.00
2560	50 %	real	1.02	0.81
5120	50 %	real	1.01	0.73
160	100 %	complex	3.53	3.33
640	100 %	complex	1.92	1.82
2560	100 %	complex	1.50	1.24
5120	100 %	complex	1.49	0.99
160	50 %	complex	2.77	2.59
640	50 %	complex	1.67	1.58
2560	50 %	complex	1.39	1.19
5120	50 %	complex	1.35	0.93

7 Conclusions

Simulations in science and engineering often require the computation of eigenvalues and eigenvectors, and while many of these eigenvalue problems are most efficiently solved by iterative methods, there remains strong demand for direct solvers achieving high performance on massively parallel machines.

Compared to the implementations available at its inception, the ELPA library provides significantly better scalability, and thus higher performance, than previous implementations of well-known direct methods. This was possible by *(i)* algorithmic changes, such as two-step reduction instead of direct tridiagonalization, and enabling the divide and conquer algorithm to compute partial eigensystems at reduced cost; *(ii)* improved data layouts and communication patterns in most stages of the computation; *(iii)* highly tuned kernels for a performance-critical operation, and building on fewer software layers to reduce overhead.

Obviously, these modifications cannot eliminate the $O(N^3)$ scaling with system size of conventional eigenvalue solvers. They can also not lead to "infinitely good" parallel scaling in the sense that, beyond some limiting processor count, reasonable matrix block sizes to enable efficient matrix block multiplications mean that the workload for some processors must eventually run out.

However, the practical impact of moving the scalability limit much further out is enormous in real-world applications that have other workloads, too. Simply put, processor counts of a few hundred for small to mid-sized molecules in *ab initio* molecular dynamics become accessible. Likewise, more challenging larger computations can be carried out with reasonable efficiency on

processor counts or 1,000 and above, a number that still sets a practical (availability) limit for many routine computations.

As a final point, perhaps the biggest practical challenge for "everyday computations" is that libraries like ELPA do not live in an isolated universe. ELPA relies on standard tools such as MPI libraries that are outside the scope of "normal" parallel application development. Here, differences between competing implementations can be very significant, as can be the effects of simple setup differences for the same implementation or computer architecture.

One of the objectives of presenting benchmarks in this work for the test cases provided with ELPA is to provide reference points that give an indication how ELPA should scale in a "clean" setup. With the efficiency demonstrated here, a significant computational bottleneck for many applications in electronic structure theory and elsewhere is indeed alleviated by the ELPA library.

In general, we stress again the fact that new developments for parallel eigenvalue solutions are now carried out in multiple different contexts and by several independent groups of scientists. For scientists in electronic structure theory and other fields requiring the solution of large eigenvalue problems, this renewed interest in traditional linear-algebra based solutions is excellent news. In this context, the development of ELPA certainly stands as a highly successful example of how research in computational science and engineering can be done on the eve of the exascale era: as an interdisciplinary effort involving mathematicians, computer scientists, and scientists from (in this case) the electronic structure community, to help realize the potential of the computational means that are now available for practical research.

8 Acknowledgements

The ELPA library was developed by a multidisciplinary consortium of scientists from the Fritz Haber Institute of the Max Planck Society, the Garching Computing Center of the Max Planck Society, the University of Wuppertal, the Technical University of Munich, IBM Germany, and the Max Planck Institute for Mathematics in the Natural Sciences. Support by the German Government through BMBF grant 01IH08007 is gratefully acknowledged.

The authors thank Inge Gutheil and coworkers (Forschungszentrum Jülich) for making available an early version of their manuscript on eigensolver performance on the IBM BlueGene/Q high-performance computer architecture.

V.H. acknowledges the computer time provided by CSC – the Finnish IT Center for Science.

The authors would like to acknowledge the contribution of Dr. Rainer Johanni, who is listed as a co-author of this work. Many of the central practical implementation ideas in ELPA were conceived by him. A brilliant computational scientist, Dr. Johanni died after a short but severe illness on June 05, 2012.

A Technical Realization of ELPA

A.1 Access and Build Process

The ELPA library is actively maintained and is distributed with a standard `configure & make` build process. This provides a convenient way to build the library, including the support of one of the different ELPA 2 kernels or to build the hybrid MPI/OpenMP development version (see Sec. 6). Access to the source code is normally provided through repository using the "git" version control system, which is standard software, e.g., on most Linux distributions. The main project web site is http://elpa.rzg.mpg.de/, with git repository access described at http://elpa-lib.fhi-berlin.mpg.de/ .

A.2 Basic Handling

The ELPA library is shipped with a set of easy-to-understand test cases for real and complex matrices. The test cases demonstrate how to

- set up the BLACS distribution,
- create the ELPA specific communicators for rows and columns,
- and call the solver for ELPA 1 and ELPA 2.

Thus, the test cases serve as the most straightforward practical documentation of the library. Simply following the lead of the test cases should result in a working port of existing software to the ELPA environment.

The test cases are also meant as a playground to check the performance of ELPA 1 and ELPA 2, respectively, and to allow the user to get a feeling for the ELPA library. For this reason, one can arbitarily choose for each test case the size of the matrix and the fraction of the eigenvectors to be computed.

After setting up the BLACS data distribution special ELPA specific MPI communicators have to be set up. This is done with the following method:

```
GET_ELPA_ROW_COL_COMMS (MPI_COMM_GLOBAL, MY_PROW, MY_PCOL,
                        MPI_COMM_ROWS, MPI_COMM_COLS)

MPI_COMM_GLOBAL: (integer, input)  global communicator for calculations
MY_PROW:         (integer, input)  row coordinate of the calling process
                                   in the process grid
MY_PCOL:         (integer, input)  column coordinate of the calling
                                   process in the processor grid
MPI_COMM_ROWS:   (integer, output) communicator for communication with
                                   rows of processes
MPI_COMM_COLS:   (integer, output) communicator for communication
                                   with columns of processes
```

Next, the ELPA 1 solver for real and complex valued matrices can be called with the methods

```
SOLVE_EVP_{REAL|COMPLEX} (NA, NEV, A, LDA, EV, Q, LDQ, NBLK, MPI_COMM_ROWS,
                          MPI_COMM_COLS)
```

NA:	(integer, input)	order of matrix A
NEV:	(integer, input)	numbers of eigenvectors to be computed
A(LDA,*):	(real|complex, input)	distributed matrix for which eigenvectors are computed. Distribution as in ScaLAPACK. The full matrix must be set (not only one half as in ScaLAPACK. Destroyed on exit (upper and lower half)
LDA:	(integer, input)	leading dimension of A
EV(NA)	(real, output)	eigenvalues of A, every processor gets complete set
Q(LDQ,*):	(real|complex, output)	eigenvectors of A. Distribution as in ScaLAPACK. Must be always dimensioned to full size (NA,NA) even if only a part of the eigenvalues is needed.
LDQ:	(integer, input)	leading dimension of Q
NBLK:	(integer, input)	blocksize of cyclic distribution, must be the same in both directions
MPI_COMM_ROWS	(integer, input)	MPI communicator for rows
MPI_COMM_COLS	(integer, input)	MPI communicator for columns

The ELPA 2 solver is called with

```
SOLVE_EVP_{REAL|COMPLEX}_2STAGE (NA, NEV, A, LDA, EV, Q, LDQ, NBLK, MPI_COMM_ROWS,
                    MPI_COMM_COLS, MPI_COMM_ALL)

Input and ouput definitions as in ELPA 1

MPI_COMM_ALL:  (integer, input) global communicator (e.g. MPI_COMM_WORLD)
```

References

[1] S. Boys, "Electronic wave functions. II. A calculation for the ground state of the beryllium atom," *Proc. Roy. Soc. (London)*, vol. A201, p. 125, 1950.

[2] R. K. Nesbet, "Algorithm for diagonalization of large matrices," *J. Chem. Phys.*, vol. 43, p. 311, 1965.

[3] G. H. Golub and H. A. van der Vorst, "Eigenvalue computation in the 20th century," *J. Comput. Applied Math.*, vol. 123, p. 35, 2000.

[4] V. Fock, "Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems," *Z. Phys.*, vol. 61, p. 126, 1930.

[5] C. Roothaan, "New developments in molecular orbital theory," *Rev. Mod. Phys.*, vol. 23, p. 69, 1951.

[6] W. Kohn and L. Sham, "Self-consistent equations including exchange and correlation effects," *Phys. Rev.*, vol. 140, p. A1133, 1965.

[7] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997. http://www.netlib.org/scalapack.

[8] E. R. Davidson, "The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices," *J. Comput. Phys.*, vol. 17, p. 87, 1975.

[9] D. Wood and A. Zunger, "A new method for diagonalising large matrices," *J. Phys. A: Math. Gen.*, vol. 18, p. 1343, 1985.

[10] A. V. Knyazev, "Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method," *SIAM J. Sci. Comput.*, vol. 23, pp. 517–541, 2001.

[11] P. Blaha, H. Hofstätter, O. Koch, R. Laskowski, and K. Schwarz, "Iterative diagonalization in augmented plane wave based methods in electronic structure calculations," *J. Comp. Phys.*, vol. 229, pp. 453–460, 2010.

[12] M. J. Rayson, "Rapid filtration algorithm to construct a minimal basis on the fly from a primitive Gaussian basis," *Comp. Phys. Commun.*, vol. 181, pp. 1051–1056, 2010.

[13] E. Polizzi, "Density-matrix-based algorithm for solving eigenvalue problems," *Phys. Rev. B*, vol. 79, pp. 115112–1–6, 2009.

[14] N. D. M. Hine, P. D. Haynes, A. Mostofi, C. Skylaris, and M. C. Payne, "Linear-scaling density-functional theory with tens of thousands of atoms: Expanding the scope and scale of calculations with ONETEP," *Comp. Phys. Commun.*, vol. 180, pp. 1041–1053, 2009.

[15] D. R. Bowler and T. Miyazaki, "$O(N)$ methods in electronic structure calculations," *Rep. Prog. Phys.*, vol. 75, p. 036503, 2012.

[16] J. VandeVondele, U. Borstnik, and J. Hutter, "Linear scaling self-consistent field calculations with millions of atoms in the condensed phase," *J. Chem. Theor. Comput.*, vol. 8, p. 3565, 2012.

[17] http://www.top500.org.

[18] P. Alpatov, G. Baker, C.Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y.-J. Wu, "PLAPACK: Parallel linear algebra package: Design overview." in Proceedings on the Conference on Supercomputing 1997 (SC97).

[19] "PLAPACK." http://www.cs.utexas.edu/users/plapack/.

[20] "PLAPACK: High Performance Through High-Level Abstraction." Technical paper submitted to ICPP'98, retrieved from http://www.cs.utexas.edu/users/plapack/papers/icpp98.ps in December, 2013.

[21] T. Imamura, S. Yamada, and M. Yoshida, "Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system," *Prog. Nucl. Sci. Tech.*, vol. 2, pp. 643–650, 2011.

[22] T. Auckenthaler, V. Blum, H. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. Willems, "Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations," *Parallel Computing*, vol. 37, pp. 783–794, December 2011.

[23] J. Poulson, B. Marker, R. A. van de Geijn, J. A. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Trans. Math. Software*, vol. 39, p. 13, 2013.

[24] M. Petschow, E. Peise, and P. Bientinesi, "High-performance solvers for dense hermitian eigenproblems," *SIAM J. Sci. Comput.*, vol. 35, pp. C1–C22, 2013.

[25] "High performance eigen-solver EigenExa." http://www.aics.riken.jp/labs/lpnctrt/EigenExa_e.html.

[26] "Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA)." http://icl.cs.utk.edu/plasma/.

[27] A. Haidar, R. Solca, M. Gates, S. Tomov, T. Schulthess, and J. Dongarra, "Leading edge hybrid multi-GPU algorithms for generalized eigenproblems in electronic structure calculations," in *ISC 2013, LNCS 7905* (J. M. Kunkel, T. Ludwig, and H. W. Meuer, eds.), pp. 67–80, Springer-Verlag Berlin Heidelberg 2013, 2013.

[28] "Matrix Algebra on GPU and Multicore Architectures (MAGMA)." http://icl.utk.edu/magma/.

[29] http://elpa.rzg.mpg.de.

[30] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, "*Ab initio* molecular simulations with numeric atom-centered orbitals," *Comp. Phys. Comm.*, vol. 180, pp. 2175–2196, 2009.

[31] V. Havu, V. Blum, P. Havu, and M. Scheffler, "Efficient $O(N)$ integration for all-electron electronic structure calculation using numeric basis functions," *J. Comp. Phys.*, vol. 228, pp. 8367–8379, 2009.

[32] http://aims.fhi-berlin.mpg.de.

[33] P. Havu, V. Blum, V. Havu, P. Rinke, and M. Scheffler, "Large-scale surface reconstruction energetics of Pt(100) and Au(100) by all-electron density functional theory," *Phys. Rev. B*, vol. 82, p. 161418(R), 2010.

[34] L. Nemec, V. Blum, P. Rinke, and M. Scheffler, "Thermodynamic equilibrium conditions of graphene films on SiC," *Phys. Rev. Lett.*, vol. 111, p. 065502, 2013.

[35] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Lapack Users' Guide*. SIAM, Philadelphia, PA, 3 ed., 1999. http://www.netlib.org/lapack.

[36] A. Yachmenev, I. Polyak, and W. Thiel, "Theoretical rotation-vibration spectrum of thioformaldehyde," *J. Chem. Phys.*, vol. 139, p. 204208, 2013.

[37] G. H. Golub and C. F. V. Loan, "Matrix Computations," The Johns Hopkins University Press, Maltimore, MD, USA (2012).

[38] C. Bischof, B. Lang, and X. Sun, "Parallel tridiagonalization through two-step band reduction," in *Proc. Scalable High-Performance Computing Conf.*, (Los Alamitos, CA), pp. 23–27, IEEE Computer Society Press, 1994.

[39] C. H. Bischof, B. Lang, and X. Sun, "A framework for symmetric band reduction," *ACM Trans. Math. Software*, vol. 26, pp. 581–601, Dec. 2000.

[40] C. H. Bischof, B. Lang, and X. Sun, "Algorithm 807: The SBR toolbox—software for successive band reduction," *ACM Trans. Math. Software*, vol. 26, pp. 602–616, Dec. 2000.

[41] J. G. F. Francis, "The QR transformation: A unitary analogue to the LR transformation, part I and II," *Computer J.*, vol. 4, pp. 265–271 and 332–345, 1961/62.

[42] J. W. Demmel and K. S. Stanley, "The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers," in *Proc. Seventh SIAM Conf. Parallel Proc. Sci. Comput.*, (Philadelphia, PA), pp. 528–533, SIAM, 1994.

[43] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.

[44] J. J. M. Cuppen, "A divide and conquer method for the symmetric tridiagonal eigenproblem," *Numer. Math.*, vol. 36, pp. 177–195, 1981.

[45] M. Gu and S. C. Eisenstat, "A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem," *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 1, pp. 172–191, 1995.

[46] F. Tisseur and J. J. 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.

[47] I. S. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, University of California at Berkeley, 1997.

[48] P. Bientinesi, I. S. Dhillon, and R. A. V. D. Geijn, "A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations," *SIAM J. Sci. Comput.*, vol. 27, no. 1, pp. 43–66, 2005.

[49] P. R. Willems, *On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and the Bidiagonal SVD*. PhD thesis, Bergische Universität Wuppertal, Fachbereich Mathematik und Naturwissenschaften, Wuppertal, Germany, 2010.

[50] P. R. Willems and B. Lang, "A framework for the MR^3 algorithm: Theory and implementation," *SIAM J. Sci. Comput.*, vol. 35, no. 2, pp. A740–A766, 2013.

[51] A. Haidar, H. Ltaief, and J. Dongarra, "Toward a high performance tile divide and conquer algorithm for the dense symmetric eigenvalue problem," *SIAM J. Sci. Comput.*, vol. 34, no. 6, pp. C249–C274, 2012.

[52] W. N. Gansterer, J. Schneid, and C. W. Ueberhuber, "A low-complexity divide-and-conquer method for computing eigenvalues and eigenvectors of symmetric band matrices," *BIT*, vol. 41, no. 5, pp. 967–976, 2001.

[53] G. König, M. Moldaschl, and W. N. Gansterer, "Computing eigenvectors of block tridiagonal matrices based on twisted block factorizations," *J. Comput. Appl. Math.*, vol. 236, pp. 3696–3703, 2012.

[54] G. Ballard, J. Demmel, and N. Knight, "Avoiding communication in successive band reduction," Tech. Rep. UCB/EECS-2013-131, EECS Department, University of California, Berkeley, Jul 2013.

[55] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Trans. Math. Soft.*, vol. 28-2, 2002.

[56] J. Dongarra, "Basic linear algebra subprograms technical (BLAST) forum standard," *Int. J. High Performance Computing Applications*, vol. 16, pp. 1–111, 2002.

[57] J. Dongarra, "Basic linear algebra subprograms technical (BLAST) forum standard," *Int. J. High Performance Computing Applications*, vol. 16, pp. 115–199, 2002.

[58] R. Clint Whaley *et al.*, "Automatically tuned linear algebra software (ATLAS)." http://math-atlas.sourceforge.net/.

[59] K. Goto, "GotoBLAS2." https://www.tacc.utexas.edu/tacc-software/gotoblas2/.

[60] "Openblas." http://xianyi.github.com/OpenBLAS.

[61] Q. Wan, X. Zhang, Y. Zhang, and Y. Qing, "AUGEM: Automatically generate high performance dense linear algebra kernels on x86 CPUs," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13), Denver, CO*, 2013.

[62] E. Breitmoser and A. G. Sunderland, "A performance study of the PLAPACK and ScaLAPACK eigensolvers on hpcx for the standard problem," 2003. Technical Report from the HPCx Consortium, available from http://www.hpcx.ac.uk/research/hpc/HPCxTR0406.pdf (retrieved December, 2013).

[63] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing* (E. Arge, A. M. Bruaset, and H. P. Langtangen, eds.), pp. 163–202, Birkhäuser Press, 1997.

[64] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2013. http://www.mcs.anl.gov/petsc.

[65] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc users manual," Tech. Rep. ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.

[66] "Scalable library for eigenvalue problem computations (SLEPc)." http://www.grycap.upv.es/slepc/.

[67] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist, "Anasazi software for the numerical solution of large-scale eigenvalue problems," *ACM Trans. Math. Softw.*, vol. 36, pp. 13:1–13:23, July 2009.

[68] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, vol. 31, pp. 397–423, Sept. 2005.

[69] I. Gutheil, J. F. Münchhalfen, and J. Grotendorst, "Performance of dense eigensolvers on BlueGene/Q," in *10th International Conference on Parallel Processing and Applied Mathematics (PPAM)*, 2013. Manuscript submitted for publication.

[70] T. Imamura, "KMATHLIB: High-performance and scalable numerical library for the k computer." http://www.aics.riken.jp/labs/lpnctrt/EigenExa_e.html.

[71] S. Schweizer, J. Kussmann, B. Doser, and C. Ochsenfeld, "Linear-scaling Cholesky decomposition," *Computational Chemistry*, vol. 29, p. 1004, 2008.

[72] R. Johanni, A. Marek, H. Lederer, and V. Blum, "Scaling of eigenvalue solver dominated simulations," in *Jülich BlueGene/P Extreme Scaling Workshop 2011* (B. Mohr and W. Frings, eds.), p. 27, Forschungszentrum Jülich, 2011.

[73] A. Sunderland, "Numerical library eigensolver performance on PRACE Tier-0 systems," 2012.