# Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations[☆]

T. Auckenthaler[a], V. Blum[b], H.-J. Bungartz[a], T. Huckle[a], R. Johanni[c], L. Krämer[d], B. Lang[d,*], H. Lederer[c], P. R. Willems[d]

[a]*Fakultät für Informatik, Technische Universität München, D-85748 Garching, Germany*
[b]*Fritz-Haber-Institut der Max-Planck-Gesellschaft, D-14195 Berlin, Germany*
[c]*Rechenzentrum Garching der Max-Planck-Gesellschaft am Max-Planck-Institut für Plasmaphysik, D-85748 Garching, Germany*
[d]*Fachbereich C, Bergische Universität Wuppertal, D-42097 Wuppertal, Germany*

**Abstract**

The computation of selected eigenvalues and eigenvectors of a symmetric (Hermitian) matrix is an important subtask in many contexts, for example in electronic structure calculations. If a significant portion of the eigensystem is required then typically direct eigensolvers are used. The central three steps are: Reduce the matrix to tridiagonal form, compute the eigenpairs of the tridiagonal matrix, and transform the eigenvectors back. To better utilize memory hierarchies, the reduction may be effected in two stages: full to banded, and banded to tridiagonal. Then the back transformation of the eigenvectors also involves two stages. For large problems, the eigensystem calculations often are the computational bottleneck, in particular with large numbers of processors. In this paper we discuss variants of the tridiagonal-to-banded back transformation, improving the parallel efficiency for large numbers of processors as well as the per-processor utilization. We also modify the divide-and-conquer algorithm for symmetric tridiagonal matrices such that it can compute a subset of the eigenpairs at reduced cost. The effectiveness of our modifications is demonstrated with numerical experiments.

*Keywords:* electronic structure calculations, eigenvalue and eigenvector computation, blocked Householder transformations, divide-and-conquer tridiagonal eigensolver, parallel

[☆]This work was supported by the Bundesministerium für Bildung und Forschung within the project "ELPA—Hochskalierbare Eigenwert-Löser für Petaflop-Großanwendungen", Förderkennzeichen 01IH08007.

[*]Corresponding author

*Email addresses:* `auckenth@in.tum.de` (T. Auckenthaler), `blum@fhi-berlin.mpg.de` (V. Blum), `bungartz@in.tum.de` (H.-J. Bungartz), `huckle@in.tum.de` (T. Huckle), `rrj@rzg.mpg.de` (R. Johanni), `lkraemer@math.uni-wuppertal.de` (L. Krämer), `lang@math.uni-wuppertal.de` (B. Lang), `lederer@rzg.mpg.de` (H. Lederer), `willems@math.uni-wuppertal.de` (P. R. Willems)

## 1. Introduction

Finding the eigenvalues and eigenvectors of symmetric (Hermitian) matrices is a classic problem in linear algebra, and computationally efficient solutions are enormously important. Since the solution effort for the full spectrum scales as $O(n^3)$ with matrix size $n$, solving an eigenproblem as part of a larger computational task can easily dominate the entire workload, or can even render the solution impossible on the most advanced computational hardware available today. We here discuss two particular improvements to the efficiency of general-purpose eigensolvers, regarding the overall workload and the parallel performance on platforms with thousands of CPUs. The examples in the paper arise in large-scale, all-electron electronic structure theory, i.e., the prediction of materials properties from the atomic scale on upwards, based only on the "first principles" of quantum mechanics. However, all algorithms described in this paper are general, and applicable in any field where the solution of large eigenproblems is needed.

A central task of electronic structure theory is the solution of Schrödinger-like eigenproblems $\hat{H}\Psi_m = E_m\Psi_m$, with $\hat{H}$ a Hamilton operator, $\Psi_m$ a wave function and $E_m$ the corresponding eigenenergy. Most frequently the ground-state total energy of a many-electron system in a given external potential (usually due to classical atomic nuclei at fixed geometry) is sought. The most widely used approach to this problem is Kohn-Sham (KS) density functional theory (DFT) [1], for which a set of effective single-particle eigenvalue/eigenfunction pairs for an eigenproblem $\hat{H}_{\mathrm{KS}}\psi_l = \epsilon_l\psi_l$ must be found. Similar eigenproblems arise in many other contexts in electronic structure theory, for example Hartree-Fock (HF) theory [2], or in the description of optical excitations through the Bethe-Salpeter [3] or Casida [4] equations. In KS-DFT, $\psi_l(\boldsymbol{r})$ are continuous functions in three-dimensional space. In practice, they are usually discretized by inserting a set of basis functions $\{\phi_i(\boldsymbol{r}), i = 1, \ldots, n\}$ such that $\psi_l(\boldsymbol{r}) = \sum_i c_{li}\phi_i(\boldsymbol{r})$. The result is a *generalized* matrix eigenproblem:

$$\mathbf{H}_{\mathrm{KS}}\mathbf{c}_l = \epsilon_l\mathbf{S}\mathbf{c}_l \tag{1}$$

with Hamilton matrix $\mathbf{H}_{\mathrm{KS}}$ and overlap matrix $\mathbf{S}$, where $\mathbf{S} = \mathbf{I}$ for some but not all basis types. In KS or HF theory, only the lowest $k$ out of $n$ possible eigenvalue/-vector pairs are needed. They correspond to those electronic "orbitals" which are "occupied," and which thus contribute to the ground-state electron density $n(\boldsymbol{r}) = \sum_{l=1}^{k}|\psi_l(\boldsymbol{r})|^2$. Equation (1) is only formally a linear problem. In fact, $\mathbf{H}_{\mathrm{KS}} \equiv \mathbf{H}_{\mathrm{KS}}[n(\boldsymbol{r})]$ depends on its own eigenvectors $\mathbf{c}_l$ through $n(\boldsymbol{r})$. One thus seeks a particular Hamiltonian that yields as solutions the same eigenvectors $\mathbf{c}_l$ used to construct it in the first place—a "self-consistent" solution of (1). To find such solutions $\mathbf{H}_{\mathrm{KS}}$ and $\mathbf{c}_l$ iteratively, (1) must be solved successively for ten(s) of times even in the simplest case, self-consistency for a single, fixed nuclear geometry. Hundreds of thousands (or more) iterations may be desirable for other questions, such as long *ab initio* molecular dynamics simulations, where the nuclear geometry changes with every step.

Most importantly, the effort to solve the matrix problem (1) scales as $O(N^3)$, where $N$ refers to overall *system* size, not necessarily just the matrix size $n$. By contrast, all other steps needed to set up and solve $\mathbf{H}_{\text{KS}}$ can be implemented to scale as $O(N)$; see, e.g., [5, 6, 7, 8, 9, 10, 11]. As the system size increases, the solution of (1) must therefore come to dominate. In practice, the crossover typically occurs for molecule or unit cell sizes of $\approx$ 100–1000 atoms [12, 13, 14, 15, 16] even for favorable basis sets, well within the range of interest for many applications (biomolecules, nano-electronics, defects in solids, etc.). Furthermore, $O(N^3)$ scaling means that any *larger* calculations are rendered infeasible even on the most advanced supercomputers, unless the prefactor can be reduced. Adding to the problem is the fact that solving (1) in parallel requires quite fine-grained, communication-intensive operations. Especially on massively parallel computer systems with thousands of processor cores and using standard implementations, this places an additional practical limitation on the calculations that can be done.

Solving or even circumventing the solution of (1) is thus an active research field, with many new and original contributions even in the most recent literature [13, 14, 16, 17, 18, 19, 20, 21, 22, 23]. Among the strategies pursued in electronic structure theory, one finds:

(i) For specific problem classes—eigenspectra with a gap between occupied and unoccupied electronic states—the solution of (1) can be avoided altogether; see, e.g., [10, 12, 24, 25, 26]. Where this works, $O(N)$ scaling for systems up to even millions of atoms has been demonstrated [27]. However, $O(N)$ strategies are not generally applicable. For example, the requirement for a spectral gap excludes all metals. Likewise, any calculation where the effective single-particle band structure $\epsilon_l$ itself is of interest of course requires a solution of (1).

(ii) Another active research area are *iterative* solution strategies (Lanczos, Davidson[28]-based, locally optimal preconditioned gradient [29, 30], many others [31, 32]), for cases where only a small fraction $k/n$ of all possible eigenpairs are needed (e.g., large, systematically convergeable basis types such as plane waves or real-space grid functions). For example, a well preconditioned blocked Davidson scheme is superior even for augmented plane wave based all-electron DFT, with $k/n \approx$ 3–10% [16]. However, iterative strategies usually still require at least one $O(N^3)$ operation [19, 13, 30, 32, 33, 16] (subspace diagonalization, orthonormalization, matrix inversion at the outset, etc.).

(iii) A straightforward path to push the crossover point between (1) and all other, $O(N)$ type operations out is to reduce $n$, the basis size itself. Obviously, in a system with $k$ occupied states to be treated, $n \geq k$ sets a lower bound. Nonetheless, $k/n$ is often already in the range 10–40% for localized Gaussian-type basis sets of quantum chemistry [2, 13]. Other choices, e.g., Slater-type or numerically tabulated atom-centered orbitals (NAOs) (see [15] for references and discussion), can yield essentially converged accuracy with even more compact basis sets that still remain generically transferable. Finally, a recent localization-based filtering approach [14, 23] contracts a large, generic basis set into a system-dependent *minimal* basis ($n = k$), prior to solving (1). The $O(N^3)$ bottleneck is then reduced to the minimum cost $O(k^3)$, at negligible accuracy loss.

In practice, (iii) is attractive because, for many relevant system sizes, (1) then does not yet dominate the overall computational effort. Therefore, robust, conventional eigensolver strategies such as implemented in LAPACK [34] can be used, avoiding system-specific complications such as the exact form of the eigenspectrum, or the choice of an optimal preconditioning strategy [35, 16]. Even for (ii) and (i), though, a conventional diagonalization of some kind may still be required or is a necessary fallback.

In general, the solution of (1) proceeds in five steps: (A) Transformation to a dense standard eigenproblem (e.g., by Cholesky decomposition of $\mathbf{S}$), $\mathbf{H}_{\mathrm{KS}}\mathbf{c}_l = \epsilon_l \mathbf{S}\mathbf{c}_l \rightsquigarrow \mathbf{A}\mathbf{q}_{\mathbf{A}} = \lambda \mathbf{q}_{\mathbf{A}}$, $\lambda \equiv \epsilon_l$; (B) Reduction to tridiagonal form, $\mathbf{A} \rightsquigarrow \mathbf{T}$; (C) Solution of the tridiagonal problem for $k$ eigenvalues and -vectors, $\mathbf{T}\mathbf{q}_{\mathbf{T}} = \lambda \mathbf{q}_{\mathbf{T}}$; (D) Back transformation of $k$ eigenvectors to dense orthonormal form, $\mathbf{q}_{\mathbf{T}} \rightsquigarrow \mathbf{q}_{\mathbf{A}}$; (E) Back transformation to the original, non-orthonormal basis, $\mathbf{q}_{\mathbf{A}} \rightsquigarrow \mathbf{c}_l$. Figure 1 shows the overall timings of these operations on a massively parallel IBM BlueGene/P system, for one specific example: the electronic structure of a 1003-atom polyalanine peptide (small protein) conformation in an artificially chosen, fixed $\alpha$-helical geometry. The example is set up using the "Fritz Haber Institute *ab initio* molecular simulations" (FHI-aims) all-electron electronic structure package [15, 11], at essentially converged basis set accuracy for DFT (*tier* 2 [15]). For (1), this means $n = 27\,069$. The number of calculated eigenpairs is $k = 3\,410$, somewhat more than the theoretical minimum $k_{\min} = 1\,905$, one state per two electrons. Steps (A)–(E) were performed using only subroutine calls as in the ScaLAPACK [36] library where available, as implemented in IBM's system-specific ESSL library, combined as described briefly in [15, Sect. 4.2]. The reason is that ScaLAPACK is arguably the industry standard for (massively) parallel linear algebra. ScaLAPACK provides the driver routine `pdsyevd`, which calls `pdsytrd`, `pdstedc`, and `pdormtr` for tridiagonalization, solution of the tridiagonal eigenproblem and back transformation respectively. `pdstedc` is based on the divide-and-conquer (D & C) algorithm, tridiagonalization and back transformation are done using Householder transformations and blocked versions thereof [37, 38]. The back transformation was done only for the needed eigenvectors.

Our point here are some key conclusions, in agreement with reports in the wider literature [17, 13, 39]. What is most apparent from Fig. 1 is that even for this large matrix size, the calculation does not scale beyond 1 024 cores, thus limiting the performance of any full electronic structure calculation with more processors. By timing steps (A)–(E) individually, it is obvious that (B) the reduction to tridiagonal form, and then (C) the solution of the tridiagonal problem using the D & C approach dominate the calculation, and prevent further scaling. For (B), the main reason is that the underlying Householder transformations involve matrix–vector operations (use of BLAS-2 subroutines and unfavorable communication pattern); the magnitude of (C) is more surprising (see below). By contrast, the matrix multiplication-based transformations (A), (D), and (E) either still scale or take only a small fraction of the overall time.

In the present paper, we assume that step (A) already has been completed, and step (E) will not be considered, either. We present a new parallel imple-

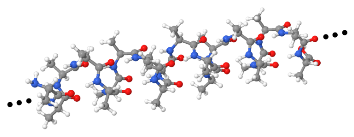
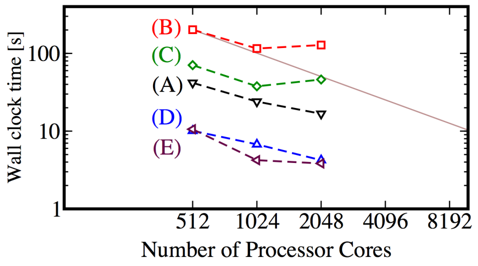

Figure 1: Left: Segment of the $\alpha$-helical Polyalanine molecule $\text{Ala}_{100}$ as described in the text. Right: Timings for steps (A)–(E) of a complete eigenvalue/-vector solution for this molecule, $n = 27\,069$, $k = 3\,410$, as a function of the number of processor cores. The calculation was performed on an IBM BlueGene/P system, using a completely ScaLAPACK-based implementation. Step (C) was performed using the divide-and-conquer method.

mentation based on the two-step band reduction of Bischof et al. [40] concerning step (B), tridiagonalization; Sect. 2.1, with improvements mainly for step (D), back transformation; Sect. 2.2. We also extend the D & C algorithm, thus speeding up step (C); Sect. 3. Some additional optimization steps in the algorithmic parts not specifically discussed here (reduction to banded form, optimized one-step reduction to tridiagonal form, and corresponding back transformations) will be published as part of an overall implementation in [41]. These routines are also included in recent production versions of FHI-aims. For simplicity we will present only the real symmetric case; the complex Hermitian case is similar.

In addition to synthetic testcases, we show benchmarks for two large, real-world problems from all-electron electronic structure theory: first, the $n = 27\,069$, $k = 3\,410$ polyalanine case of Fig. 1, which will be referred to as Poly27069 problem in the following, and second, an $n = 67\,990$ generalized eigenproblem arising from a periodic Pt(100)-"(5×40)", large-scale reconstructed surface calculation with 1 046 heavy-element atoms, as needed in [42]. In the latter calculation, the large fraction of core electrons for Pt (atomic number $Z = 78$) makes for a much higher ratio of needed eigenstates to overall basis size, $k = 43\,409 \approx 64\%$, than in the polyalanine case, even though the basis set used is similarly well converged. This problem will be referred to as Pt67990. Benchmarks are performed on two distinct computer systems: The IBM BlueGene/P machine "genius" (with a maximum of 16 384 cores) used in Fig. 1, and a Sun Microsystems-built, Infiniband-connected Intel Xeon (Nehalem) cluster with individual eight-core nodes. We note that for all standard ScaLAPACK or PBLAS calls, i.e., those parts not implemented by ourselves, the optimized ScaLAPACK-like implementations by IBM (ESSL) or Intel (MKL) were employed.

## 2. Efficient tridiagonalization and back transformation

The two-step band reduction of Bischof et al. [40] is a promising approach for an efficient parallel tridiagonalization of dense symmetric matrices. In this

Table 1: Comparison of the flop-count of one-step and two-step tridiagonalization. The $n_b$ in the one-step tridiagonalization stands for the internal blocking parameter.

| | Reduction | Back transformation |
|---|---|---|
| One-step tridiag. | $\frac{4}{3}n^3$ (50% BLAS-2) $+ O(n^2 n_b)$ | $2kn^2$ |
| Two-step tridiag. | $\frac{4}{3}n^3 + O(n^2 b)$ | $4kn^2$ |

approach the full matrix $\mathbf{A}$ of size $n$ is first reduced to banded symmetric form with (semi)bandwidth $b$. In a second step the banded matrix $\mathbf{B}$ is brought to tridiagonal form. This procedure has the advantage that the vast majority of the operations can be done with highly efficient BLAS-3 routines instead of memory bandwidth limited BLAS-2 routines. The drawback of the two-step approach is the additional computational effort for the second reduction step and its back transformation respectively. While the reduction from banded to tridiagonal form is comparatively cheap ($6bn^2$ flops), the additional costs for the back transformation are significant ($2kn^2$ flops). This holds especially true for a high number $k$ of desired eigenvectors.

For the parallel reduction to banded form and the corresponding back transformation of eigenvectors we used well established algorithms based on Householder transformations. We refer to [40] and [43] for further information.

In Sect. 2.1 we will briefly revisit the bulge-chasing algorithm for reducing banded symmetric matrices to tridiagonal form [44]. In Secs. 2.2 and 2.3 we will present new parallelization strategies for the tridiagonal-to-banded back transformation of eigenvectors.

### *2.1. Reduction from banded to tridiagonal form*

For the reduction of banded matrices to tridiagonal form there exist specialized algorithms which exploit the banded structure of the matrix. In [45], Schwarz introduced an algorithm which successively decreases the bandwidth of the matrix by one, using Givens rotations. In [44] a parallel algorithm based on Householder transformations was presented, where $b-1$ subdiagonal elements are removed at once. This concept was generalized to a successive band reduction [46, 47]. All algorithms necessarily generate intermediate fill-in. To preserve the banded structure, all or some of the fill-in is removed with bulge-chasing techniques.

The bulge-chasing algorithm in [44] takes $n-2$ *stages* to tridiagonalize a symmetric matrix $\mathbf{B} = (b_{ij}) =: \mathbf{B}^{(1)} \in \mathbb{R}^{n \times n}$ having (semi)bandwidth $b$. In the $v$th stage, the $v$th column of the band is brought to tridiagonal form. To this end, the remaining band is treated as a block tridiagonal matrix $\mathbf{B}^{(v)}$ with $\mathbf{B}_{00}^{(v)} = b_{vv} \in \mathbb{R}^{1 \times 1}$ and $\mathbf{B}_{10}^{(v)} = (b_{v+1,v}, \ldots, b_{v+b,v})^T \in \mathbb{R}^{b \times 1}$ containing just the first remaining column, and diagonal blocks $\mathbf{B}_{\beta\beta}^{(v)}$ and subdiagonal blocks $\mathbf{B}_{\beta+1,\beta}^{(v)}$ of size $b \times b$ for $\beta \geq 1$ (except for smaller blocks at the end of the band). Note that because of symmetry only the lower triangle of $\mathbf{B}^{(v)}$ needs consideration.

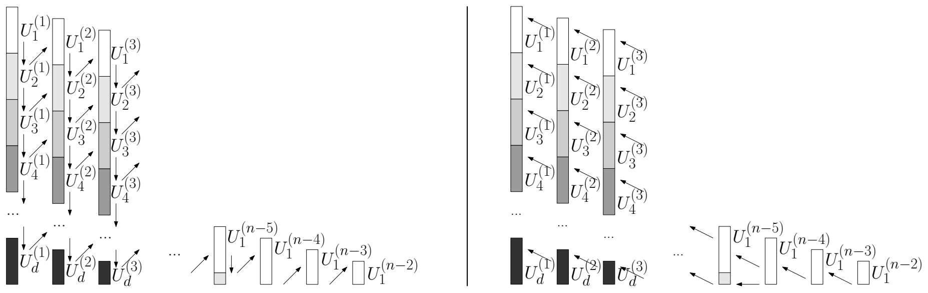

Figure 2: Householder vectors from the bulge-chasing algorithm for the reduction from banded to tridiagonal form [44]. The arrows indicate the order of execution during reduction (left) and back transformation of eigenvectors (right).

The stage is initiated by a length-$b$ Householder transformation, which reduces the first remaining column to tridiagonal form: ${\mathbf{U}_1^{(v)}}^T \mathbf{B}_{10}^{(v)} = (*, 0, \ldots, 0)^T$. This transformation must be applied to the whole first block row and block column, $\bar{\mathbf{B}}_{11} = {\mathbf{U}_1^{(v)}}^T \mathbf{B}_{11}^{(v)} \mathbf{U}_1^{(v)}$, $\bar{\mathbf{B}}_{21} = \mathbf{B}_{21}^{(v)} \mathbf{U}_1^{(v)}$, thereby filling the subdiagonal block completely. To preserve the banded structure for future stages, at least the zeros in the *first column* of $\mathbf{B}_{21}$ must be recovered with a second Householder transformation, and so on. More precisely, in the $\beta$th step of stage $v$, the first column of the subdiagonal block $\mathbf{B}_{\beta,\beta-1}^{(v)}$ has been filled by the preceding transformation, $\bar{\mathbf{B}}_{\beta,\beta-1} = \mathbf{B}_{\beta,\beta-1}^{(v)} \mathbf{U}_{\beta-1}^{(v)}$. The next length-$b$ transformation $\mathbf{U}_\beta^{(v)}$ is chosen to re-eliminate this column, ${\mathbf{U}_\beta^{(v)}}^T \bar{\mathbf{B}}_{\beta,\beta-1}(:,1) = (*, 0, \ldots, 0)^T$. This transformation must be applied to the whole $\beta$th block row and block column, $\bar{\mathbf{B}}_{\beta\beta} = {\mathbf{U}_\beta^{(v)}}^T \mathbf{B}_{\beta\beta}^{(v)} \mathbf{U}_\beta^{(v)}$, $\bar{\mathbf{B}}_{\beta+1,\beta} = \mathbf{B}_{\beta+1,\beta}^{(v)} \mathbf{U}_\beta^{(v)}$, which in turn leads to the transformation $\mathbf{U}_{\beta+1}^{(v)}$. The stage is complete when the end of the band is reached, and then the block decomposition is shifted by one row/column for the next stage. (Thus the rows/columns affected by $\mathbf{U}_\beta^{(v+1)}$ are shifted by one position, as compared to those affected by $\mathbf{U}_\beta^{(v)}$.)

The transformations from the whole reduction and their data dependency are sketched in Fig. 2. One column ($v$) of the figure contains the Householder vectors (their nonzero entries) from one stage $v$ of the described algorithm. The first sweep of transformations (vectors painted white) eliminates elements of the original banded matrix $\mathbf{B}$. All other sweeps (painted in gray and black) remove intermediate fill-in.

## *2.2. Tridiagonal-to-banded back transformation of eigenvectors*

In the tridiagonal-to-banded back transformation all the transformations from the banded-to-tridiagonal reduction have to be applied from the left side to the desired eigenvectors $\mathbf{q}_{\mathbf{T}i}$ of the tridiagonal matrix,

$$\mathbf{q}_{\mathbf{B}i} = \Big( \prod_{v,\beta} \mathbf{U}_\beta^{(v)} \Big) \mathbf{q}_{\mathbf{T}i}, \quad i = 1, \ldots, k. \tag{2}$$

For this task two different approaches have been implemented, which rely on two different parallel data layouts. Of course the Householder transformations are not given as matrices, but as a set of Householder vectors $\mathbf{v}$, where each vector has at most $b$ nonzero elements: $\mathbf{U}_\beta^{(v)} = \mathbf{I} - \tau \mathbf{v}\mathbf{v}^T$.

The work from (2) can run perfectly in parallel, because each eigenvector can be transformed independently. This fact has been used for the first parallelization approach, where the $k$ eigenvectors are distributed uniformly across the $p$ processes. For the second approach [48] the $k$ eigenvectors are seen as a matrix of size $n \times k$, which is distributed in a 2D blocked manner across a 2D processor grid with $p_r$ rows and $p_c$ columns. Each process transforms its local part of the matrix. In the following the costs of the two approaches will be analyzed and compared to each other. According to the data distribution, the approaches will be called 1D- and 2D-parallelization.

For the 1D-approach each process transforms $k/p$ eigenvectors. Thus the computational costs are of order $2kn^2/p$. Additionally every process needs all the Householder vectors from the reduction step, leading to a communication volume of $n^2/2$ words per process. Beside the distribution of Householder vectors no further synchronization is necessary.

For the 2D-approach each process needs only a fraction of the Householder transformations, because only the local part of the eigenvectors has to be transformed. This reduces the costs for the distribution of Householder vectors to $n^2/p_r$ words and guarantees better scalability than the 1D-parallelization. Of course we have to follow the order of the transformations from Fig. 2. Thus an additional synchronization between vertically neighboring processes is needed. After each of the $n/b$ sweeps, a block of size $b \times (k/p_c)$ has to be exchanged. This leads to a communication amount of $2kn/p_c$ words per process. A downside of the 2D-parallelization is its poor load balancing. The lower blocks of the matrix are affected by much more Householder transformations than the upper blocks (see Fig. 2). This leads to load imbalances by a factor of 2 and a computation amount of $4kn^2/p$ for the lowermost processes.

We resolved these load imbalances with a dynamic adaptation of the parallel data layout (Fig. 3). After each sweep, the $b$ uppermost rows are removed from the matrix and the data distribution is adapted accordingly. To avoid memory imbalances, the removed rows have to be distributed uniformly to the whole processor grid. This results in additional communication costs of size $kn/p_c$ for the topmost processes, but the load balancing for computations and distribution of Householder vectors more than compensates for this. Table 2 summarizes the costs of the different parallelization approaches.

Figure 4 shows the strong scalability of the tridiagonal-to-banded back transformation for the Poly27069 and Pt67990 problems on the BlueGene/P and the Intel Nehalem cluster. For the Pt67990 problem, both parallelization strategies perform well, because most time is spent in computations. For the Poly27069 problem, however, the 1D parallelization does not scale beyond 1 024 cores on the BlueGene/P and much less than ideal beyond 256 cores on the Intel Nehalem cluster. The scalability of the 1D parallelization is limited in two ways.

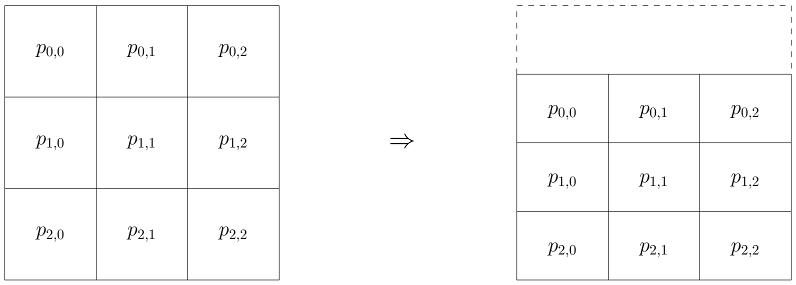

Figure 3: Dynamic adaptation of the 2D data layout.

Table 2: Comparison of different parallelization approaches for the back transformation of eigenvectors (costs per process).

| | computation[flops] | communication[#words] |
|---|---|---|
| 1D | $2kn^2/p$ | $n^2/2$ |
| 2D without load balancing | $4kn^2/p$ | $n^2/p_r \quad + 2kn/p_c$ |
| 2D | $2kn^2/p$ | $n^2/2p_r + 3kn/p_c$ |

First, the cost for the distribution of Householder vectors is constant and does not decrease with $p$; see Tab. 2. Second, the work can be parallelized at most on $k$ processes. The different variants of the algorithm (WY, non-WY) and their behavior will be described in the next subsection.

## *2.3. Non-WY approaches for blocking Householder transformations*

The back transformation of eigenvectors can be performed using WY transformations or variants thereof; cf. e.g. [37, 38]. Here $n_b$ Householder transformations are combined to a blocked Householder transformation, which is represented by matrices $W$ and $Y$ (Fig. 5). While the application of blocked

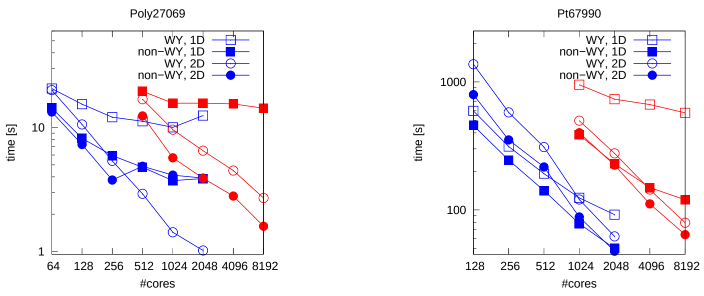

Figure 4: Strong scalability of the tridiagonal-to-banded back transformation of eigenvectors for Poly27069 and Pt67990 (bandwidth $b = 64$). Blue lines: Intel cluster, red lines: BlueGene/P. The WY, 1D times for Poly27069 on BlueGene/P are above the plotting area.

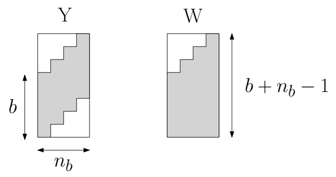

Figure 5: Nonzero structure of the matrices $W$ and $Y$ during the back transformation of eigenvectors.

Householder transformations is very efficient due to the use of BLAS-3 routines, it also creates a lot of computational overhead compared to the non-blocked application of the transformations. Each Householder vector has $b$ nonzero entries. Thus, the non-blocked application of $n_b$ such Householder transformations to $k$ eigenvectors requires $4kbn_b$ flops. Due to the fact that the nonzero entries are shifted in each Householder vector, the matrices $W$ and $Y$ are not of size $b \times n_b$ but $(b + n_b - 1) \times n_b$ (see Fig. 5), bringing the flop count for the application of the WY transformations from $4kbn_b$ to $4k(b + n_b - 1)n_b$.

To overcome this overhead without losing the cache efficiency of BLAS-3 operations, we implemented our own optimized kernel routines [49]. We used a widespread optimization technique, called cache blocking [50], which rearranges the execution order of instructions within an algorithm such that the working set fits into the cache and the number of cache misses is minimized. This means that not all $k$ eigenvectors are transformed simultaneously, but only a subset of $k_b$ eigenvectors. The number $k_b$ should be set such that, first, the working set of $b \times k_b$ words fits into the (L1-)cache and, second, $k_b + 2$ words (a vector of length $k_b$ and one element of the current Householder vector and eigenvector) fit into the registers to reduce the data transfer to the caches. Together with some other optimization techniques such as loop unrolling, the optimized kernel routines achieve very good performance.

Optimized kernels were implemented for the PowerPC 450 (BlueGene/P) and for x86 processors. Figures 4 and 6 compare the runtime of the WY and non-WY approach for the banded-to-tridiagonal back transformation of eigenvectors. The comparison of the different approaches (1D WY, 1D non-WY, 2D WY, 2D non-WY) reveals some platform and problem size dependence in detail, but the overall trends are clear. (i) The current 1D or 2D non-WY implementations almost always outperform the WY approaches, in particular for short Householder vectors (small $b$). (ii) There is presently a cross-over between our 1D and 2D non-WY approaches as the number of cores increases, which can be understood because the 1D approach must stop scaling as the number of cores approaches the number of needed eigenvectors. The distribution of Householder vectors also becomes increasingly expensive in the 1D approach. (iii) Only for the smaller Poly27069 problem on the Intel cluster, there is an unexpected cross-over to the 2D WY version above 256 cores. This indicates that some further optimization can be done for the non-WY case on this platform. We speculate that part of the reason is different behavior of overlapping communication with

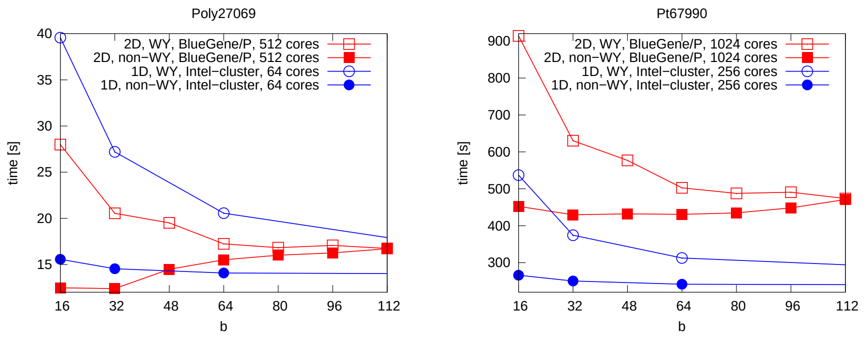

Figure 6: Runtime of the tridiagonal-to-banded back transformation for Poly27069 and Pt67990 using different intermediate bandwidths $b$.

computation (MPI_Isend/MPI_Irecv) on the two machines.

## 3. Partial eigensystems of symmetric tridiagonal matrices

The current release 1.8.0 of the ScaLAPACK library [36] provides three different methods for computing eigenvalues and eigenvectors of a symmetric tridiagonal matrix: `xSTEQR2`, the implicit QL/QR method [51]; `PxSTEBZ` and `PxSTEIN`, a combination of bisection and inverse iteration (*B & I*) [52, 53]; and `PxSTEDC`, the divide-and-conquer (*D & C*) method [54, 55, 56]. LAPACK 3.2.2 [34] and release 3.2 of the PLAPACK library [57] also provide the new MRRR algorithm [58, 17], which will be included in a future ScaLAPACK release as well [59].

B & I is able to compute a subset of $k$ eigenpairs at reduced cost, but for clustered eigenvalues this method may take $O(k^2n)$ operations or/and lose orthogonality. The MRRR algorithm also can compute subsets (at cost $O(kn)$), but the current implementations in rare cases fail to attain satisfying orthogonality; recent research towards improving its robustness is described in [60, 61]. Compared with these two methods, QR/QL and D & C produce very good eigensystems, but both are designed to compute either all or no eigenvectors. In the "all eigenvectors" case, D & C typically is *much* faster than QR/QL, at the cost of $2n^2$ additional workspace. Therefore, in practice D & C is often used to compute all eigenvectors, even if only a subset of them is needed.

In the following we will briefly review the D & C algorithm and show how it can be modified to compute a subset of the eigenvectors at reduced cost.

### *3.1. Outline of the divide-and-conquer algorithm*

To determine the eigendecomposition $\mathbf{T} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$ of a symmetric tridiagonal matrix $\mathbf{T} \in \mathbb{R}^{n\times n}$, D & C proceeds as follows.

**Split** the tridiagonal matrix into two half-sized submatrices:

$$\mathbf{T} = \begin{pmatrix} \mathbf{T}_1 & \\ & \mathbf{T}_2 \end{pmatrix} + \rho\mathbf{w}\mathbf{w}^T, \qquad (3)$$

where $\mathbf{T}_1$ and $\mathbf{T}_2$ are of size $n_1 \approx n/2$ and $n_2 = n - n_1$, resp., and $\mathbf{w} \in \mathbb{R}^n$ has nonzeros only at positions $n_1$ and $n_1 + 1$.

**Solve subproblems**, i.e., call the D & C routine recursively to compute the eigendecompositions $\mathbf{T}_1 = \mathbf{Q}_1\mathbf{\Lambda}_1\mathbf{Q}_1^T$ and $\mathbf{T}_2 = \mathbf{Q}_2\mathbf{\Lambda}_2\mathbf{Q}_2^T$. If the subproblems are "small enough" then QR/QL may be used instead, which terminates the recursion. Note that by (3) we have

$$\mathbf{T} = \mathbf{Q}^{\mathrm{sub}}(\mathbf{\Lambda}^{\mathrm{sub}} + \rho\mathbf{z}\mathbf{z}^T)\mathbf{Q}^{\mathrm{sub}^T}, \tag{4}$$

where $\mathbf{Q}^{\mathrm{sub}} = \operatorname{diag}(\mathbf{Q}_1, \mathbf{Q}_2)$, $\mathbf{\Lambda}^{\mathrm{sub}} = \operatorname{diag}(\mathbf{\Lambda}_1, \mathbf{\Lambda}_2)$, and $\mathbf{z} = \mathbf{Q}^{\mathrm{sub}^T}\mathbf{w}$. Thus the next steps are aimed at computing the eigendecomposition for a rank-one perturbed diagonal matrix, $\mathbf{\Lambda}^{\mathrm{sub}} + \rho\mathbf{z}\mathbf{z}^T = \widehat{\mathbf{Q}}\mathbf{\Lambda}\widehat{\mathbf{Q}}^T$.

**Deflate**. If some components $z_i$ of $\mathbf{z}$ are (almost) zero then the corresponding $(\lambda_i^{\mathrm{sub}}, \mathbf{q_T}_i^{\mathrm{sub}})$ are also (within small error bounds) eigenpairs of $\mathbf{T}$. Similarly, if two eigenvalues in $\mathbf{\Lambda}^{\mathrm{sub}}$ are identical (or close) then a zero (or small entry) in $\mathbf{z}$ can be generated by applying a sequence of plane rotations, thus again leading to the first case. Assume that $d$ eigenpairs can be computed cheaply in this way and that these are permuted to the end, that is, $\mathbf{P}^T\mathbf{R}^T\mathbf{z} = (\mathbf{z}_{\mathrm{n}}^T, \mathbf{0})^T$, where $\mathbf{R}^T$ is the product of all rotations, $\mathbf{P}^T$ is the permutation, and $\mathbf{z}_{\mathrm{n}}$ contains the components that cannot be made zero. Then (4) reduces to

$$\mathbf{T} = \mathbf{Q}^{\mathrm{sub}}\mathbf{R}\mathbf{P}\begin{pmatrix} \mathbf{\Lambda}_{\mathrm{n}}^{\mathrm{sub}} + \rho\mathbf{z}_{\mathrm{n}}\mathbf{z}_{\mathrm{n}}^T & \\ & \mathbf{\Lambda}_{\mathrm{d}}^{\mathrm{sub}} \end{pmatrix}(\mathbf{Q}^{\mathrm{sub}}\mathbf{R}\mathbf{P})^T, \tag{5}$$

that is, the size of the rank-one perturbed diagonal eigenproblem is reduced by $d$ (*"deflation"*). Note that $\mathbf{\Lambda}_{\mathrm{d}}^{\mathrm{sub}} \in \mathbb{R}^{d\times d}$ contains the deflated eigenvalues $\lambda_i^{\mathrm{sub}}$ corresponding to a zero in $\mathbf{P}^T\mathbf{R}^T\mathbf{z}$, whereas $\mathbf{\Lambda}_{\mathrm{n}}^{\mathrm{sub}}$ contains the $n-d$ non-deflated eigenvalues.

**Solve the secular equation**. The eigenvalues $\lambda_i$ of the matrix $\mathbf{\Lambda}_{\mathrm{n}}^{\mathrm{sub}} + \rho\mathbf{z}_n\mathbf{z}_n^T$ are given by the solutions of the *secular equation*

$$f(\lambda) = 1 + \rho\sum_{k=1}^{n-d}\frac{\zeta_k^2}{\ell_k - \lambda} = 0, \tag{6}$$

where $\zeta_k$ and $\ell_k$ denote the $k$th component of $\mathbf{z}_{\mathrm{n}}$ and the $k$th diagonal entry in $\mathbf{\Lambda}_{\mathrm{n}}^{\mathrm{sub}}$, respectively. In theory, the corresponding eigenvectors are given by diagonal scalings $\widehat{\mathbf{q_T}}_i = (\mathbf{\Lambda}_{\mathrm{n}}^{\mathrm{sub}} - \lambda_i\mathbf{I})^{-1}\mathbf{z}_{\mathrm{n}}$, but they must be computed in another way to ensure orthogonality.

**Propagate eigenvectors.** According to (5), the eigenvectors of $\mathbf{T}$ are given by

$$\mathbf{Q} = \mathbf{Q}^{\mathrm{sub}}\mathbf{R}\mathbf{P}\widehat{\mathbf{Q}} = \begin{pmatrix} \mathbf{Q}_1 & \\ & \mathbf{Q}_2 \end{pmatrix}\left[\mathbf{R}\mathbf{P}\begin{pmatrix} \widehat{\mathbf{Q}}_{\mathrm{n}} & \\ & \mathbf{I}_d \end{pmatrix}\right].$$

To improve the efficiency of this computation, the matrix in square brackets is further permuted as $\begin{pmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} & 0 & \mathbf{V}_{14} \\ 0 & \mathbf{V}_{22} & \mathbf{V}_{23} & \mathbf{V}_{24} \end{pmatrix}$, where the first and third block columns contain the eigenvectors from $\mathbf{T}_1$ (from $\mathbf{T}_2$, resp.) that were not involved in any rotation, the fourth block column contains the deflated eigenvectors, which need no further processing, and the second block column comprises the rest. Then the update actually involves two matrix–matrix products

$$\mathbf{Q}_1 \cdot (\mathbf{V}_{11}, \mathbf{V}_{12}) \quad \text{and} \quad \mathbf{Q}_2 \cdot (\mathbf{V}_{22}, \mathbf{V}_{23}). \tag{7}$$

Two factors contribute to the mostly excellent performance of the D & C method: The majority of the arithmetic operations takes place in heavily optimized matrix–matrix products, and deflation often leads to a significant reduction of the complexity as compared to the $O(n^3)$ worst case.

The recursive calls of the D & C algorithm correspond to a binary call tree. The LAPACK and ScaLAPACK implementations avoid recursion by traversing this tree in a breadth-first bottom-up order, i.e., by first splitting the matrix into $m = 2^\mu$ small subblocks, which are treated directly, then merging the eigensystems of $m/2$ pairs of these, then $m/4$ pairs of the double-size blocks, and so on. The final step merges two eigensystems of size-$n/2$ blocks. The LAPACK implementation precedes the whole procedure with a check for irreducibility (calling D & C for each irreducible block), whereas the ScaLAPACK implementation skips this step.

## *3.2. Computing a subset of the eigenvectors with D & C*

As pointed out above, a significant portion of the operations is done in matrix–matrix products. Profiling reveals that the final two products (7) at the *root* of the recursion tree can account for up to 70% of the total time, unless extreme deflation takes place. In that case the matrix products contribute only a few percent to the total time.

If only a subset of the eigenvectors is required then only these must be propagated in the two products. This can be accomplished in two ways. Either only the wanted vectors are included in $(\mathbf{V}_{11}, \mathbf{V}_{12})$ and $(\mathbf{V}_{22}, \mathbf{V}_{23})$, thus reducing the size of the matrix products, or unwanted vectors are set to zero before the multiplication. The latter approach reduces memory traffic, but it only works with zero-aware `xGEMM` implementations.

Note that the splitting into irreducible blocks, as done in the LAPACK routine `xSTEDC`, requires an additional preprocessing step. First, we must determine which eigenpairs are required from each block (e.g., eigenpairs $10 : 50$ of the whole matrix may correspond to eigenpairs $3 : 18$ of the first block, eigenpairs $5 : 13$ of the second, and eigenpairs $4 : 19$ of the third block). This functionality is provided by the routine `xLARXI` from our new MRRR implementation [60, 61]. Then the modified D & C algorithm is called for each irreducible block with the respective sub-subset.

Figure 7 shows that this modification yields considerable savings in the serial algorithm unless extreme deflation takes place. Similar speedups can be achieved with the parallel implementation; see Tab. 3. If the matrix is large enough and not too few eigenvectors are sought then the method also scales well up to large numbers of processors; see Tab. 4.

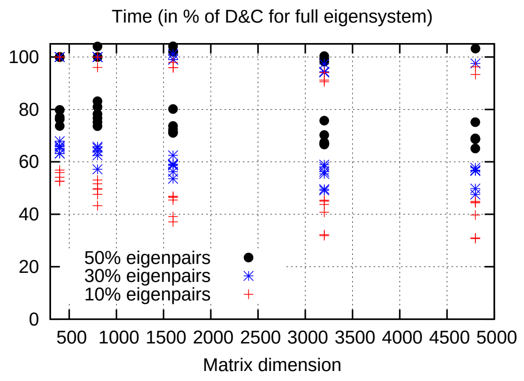

Figure 7: Percentage of the time needed to compute a subset of the eigenvectors, as compared to the computation of the full eigensystem. The data points around 100% are for matrices with extreme deflation, the other points are for matrices with moderate or no deflation. For each of the three matrix types and each subset size, three subsets (at the lower and upper end and around the median of the spectrum) were computed.

Table 3: Time (in seconds) for computing partial eigensystems with D & C.

| | 128-CPU Intel cluster | | 1024-CPU BlueGene/P | |
|---|---|---|---|---|
| | $n = 27\,069$ | $n = 67\,990$ | $n = 27\,069$ | $n = 67\,990$ |
| 1 eigenvector | 7.0 | 78.5 | 7.3 | 69.2 |
| 20% eigenvectors | 7.6 | 81.1 | 7.7 | 70.0 |
| 40% eigenvectors | 10.3 | 100.0 | 9.3 | 79.4 |
| 60% eigenvectors | 13.5 | 138.6 | 10.9 | 102.1 |
| 80% eigenvectors | 16.4 | 184.7 | 12.7 | 128.8 |
| all eigenvectors | 19.4 | 223.9 | 14.3 | 149.1 |
| Poly27069 ($\approx$ 13%) | 7.0 | | 7.3 | |
| Pt67990 ($\approx$ 64%) | | 148.5 | | 105.1 |

Since at least 30% of D & C's work is spent outside the two top-level matrix–matrix multiplications, at most threefold speed-up can be achieved this way. Thus it is natural to ask if subsets can also lead to savings at deeper recursion levels. Unfortunately this seems not to be the case because the secular equation (6) involves all non-deflated eigenvalues, and these are independent from the subset.

## 4. Conclusions

We have presented improvements for two stages in the computation of eigensystems of symmetric matrices.

For the banded-to-tridiagonal back transformation of the eigenvectors, which is necessary if a two-step tridiagonalization has been used, a dynamic 2D data

Table 4: Time (in seconds) for computing partial eigensystems with D & C on a BlueGene/P.

| | 512 CPUs | 1024 CPUs | 2048 CPUs |
|---|---|---|---|
| Poly27069 | 13.5 | 7.3 | 5.7 |
| Pt67990 | 201.4 | 105.1 | 62.4 |

layout reduces the communication volume (as compared to a 1D layout) without incurring significant load imbalance (as would be the case with a static 2D layout). In addition, explicit cache blocking can reduce the computational overhead for the blocked application of the Householder transformations, as compared to WY-based techniques. Combined, these two approaches can yield significant improvements for the parallel efficiency, in particular for large numbers of processor cores, as well as for the per-processor utilization, and they enable further large savings by using a two-step reduction to tridiagonal form.

The solution of the tridiagonal eigenproblem has been addressed, too. The popular divide-and-conquer tridiagonal eigensolver features high speed and accuracy, but it is designed to compute either all or no eigenvectors. We have presented a modification that allows to compute partial eigensystems at reduced cost, on serial and parallel machines.

Figure 8 summarizes our findings and improvements. As can be seen, the vendor-optimized ScaLAPACK implementations (ESSL or MKL) always stop scaling, even for the very large Pt67990 problem. Our present two-step implementation is clearly more efficient than even our optimized one-step implementation [41] for both problems on the Intel cluster. Only on the BlueGene/P, and for the large problem Pt67990, does the one-step implementation remain competitive, due to the large ratio of needed eigenvectors to matrix size, $k/n \approx 64\%$. This indicates that the one-step tridiagonalization would be more beneficial if all eigenpairs were needed on BlueGene/P, as expected due to the higher operations count of the two-step approach, but remarkably, the same is not necessarily true on Intel/Infiniband.

For the measurements in Fig. 8 we used an intermediate bandwidth $b = 64$ on both machines. An optimal choice of $b$ for the whole algorithm (reduction and back transformation) depends on the matrix size $n$, the number of desired eigenvectors $k$, the number of processes $p$, and last but not least on the used hardware. A larger $b$ allows BLAS routines to operate near peak performance and decreases the number of messages sent, but it also increases the runtime of the reduction from banded to tridiagonal form. Experience has shown that $b \sim 50$ is a good choice; see also Fig. 6. The choice of an optimal parameter set will be the subject of further research.

While our research is driven by demands arising in electronic structure theory, the techniques described in this paper are general and applicable to any field where the solution of large symmetric eigenproblems is required.

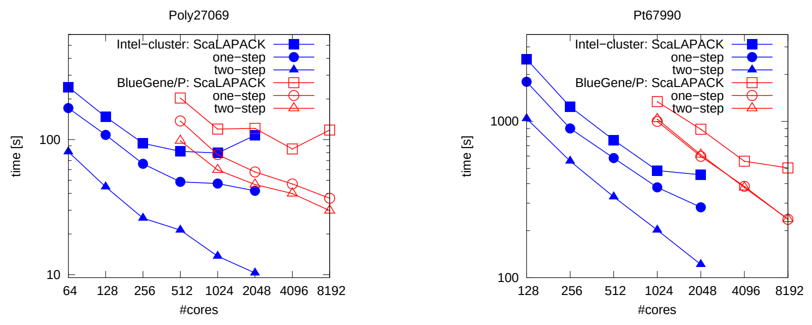

Figure 8: Overall runtime of the symmetric eigenvalue problem. The ScaLAPACK implementation `pdsyevd` is compared to the implementation presented in [41] (one-step) and the two-step approach, presented in this paper. The one-step implementation as well as the two-step approach make use of the improved divide-and-conquer algorithm. For the tridiagonal-to-banded back transformation in each case the fastest implementation was used; see Fig. 4.

## References

[1] W. Kohn, L. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133–A1138.

[2] A. Szabo, N. Ostlund, Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory, Dover, 1996.

[3] M. Rohlfing, S. G. Louie, Electron-hole excitations in semiconductors and insulators, Phys. Rev. Lett. 81 (1998) 2312–2315.

[4] M. Casida, All-electron local and gradient-corrected density-functional calculations of $\mathrm{Na}_n$ dipole polarizabilities for $n = 1$–6, in: P. Chong (Ed.), Recent Advances in Density Functional Methods, Part I, World Scientific, Singapore, 1995, p. 155.

[5] J. M. Perez-Jorda, W. Yang, An algorithm for 3d numerical integration that scales linearly with the size of the molecule, Chem. Phys. Lett. 241 (1995) 469–476.

[6] R. Stratmann, G. Scuseria, M. Frisch, Achieving linear scaling in exchange-correlation density functional quadratures, Chem. Phys. Lett. 257 (1996) 213–223.

[7] M. Challacombe, E. Schwegler, Linear scaling computation of the Fock matrix, J. Chem. Phys. 106 (1997) 5526–5536.

[8] C. Fonseca Guerra, J. Snijders, G. te Velde, E. Baerends, Towards an order-$N$ DFT method, Theor. Chem. Acc. 99 (1998) 391–403.

[9] C. Ochsenfeld, C. A. White, M. Head-Gordon, Linear and sublinear scaling formation of Hartree-Fock-type exchange matrices, J. Chem. Phys. 109 (1998) 1663–1669.

[10] S. Goedecker, Linear scaling electronic structure methods, Rev. Mod. Phys. 71 (1999) 1085–1123.

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

[12] J. M. Soler, E. Artacho, J. D. Gale, A. Garcia, J. Junquera, P. Ordejon, D. Sanchez-Portal, The SIESTA method for ab initio order-$N$ materials simulation, J. Phys.: Condens. Matter 14 (2002) 2745–2779.

[13] M. J. Rayson, P. R. Briddon, Rapid iterative method for electronic-structure eigenproblems using localised basis functions, Comp. Phys. Commun. 178 (2008) 128–134.

[14] M. J. Rayson, P. R. Briddon, Highly efficient method for Kohn-Sham density functional calculations of 500–10000 atom systems, Phys. Rev. B 80 (2009) 205104–1–11.

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

[16] P. Blaha, H. Hofstätter, O. Koch, R. Laskowski, K. Schwarz, Iterative diagonalization in augmented plane wave based methods in electronic structure calculations, J. Comp. Phys. 229 (2010) 453–460.

[17] P. Bientinesi, I. S. Dhillon, R. A. van de Geijn, A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations, SIAM J. Sci. Comput. 27 (1) (2005) 43–66.

[18] W. N. Gansterer, J. Zottl, Parallelization of divide-and-conquer eigenvector accumulation, in: J. C. Cunha, P. D. Medeiros (Eds.), Euro-Par 2005, Springer, Berlin, 2005, pp. 847–856.

[19] Y. Zhou, Y. Saad, M. L. Tiago, J. R. Chelikowsky, Parallel self-consistent-field calculations via Chebyshev-filtered subspace acceleration, Phys. Rev. E 74 (2006) 066704–1–8.

[20] Y. Bai, R. C. Ward, Parallel block tridiagonalization of real symmetric matrices, J. Parallel Distrib. Comput. 68 (2008) 703–715.

[21] E. Polizzi, Density-matrix-based algorithm for solving eigenvalue problems, Phys. Rev. B 79 (2009) 115112–1–6.

[22] J.-L. Fattebert, Accelerated block preconditioned gradient method for large scale wave functions calculations in density functional theory, J. Comp. Phys. 229 (2010) 441–452.

[23] M. J. Rayson, Rapid filtration algorithm to construct a minimal basis on the fly from a primitive Gaussian basis, Comp. Phys. Commun. 181 (2010) 1051–1056.

[24] D. Bowler, T. Miyazaki, M. Gillan, Recent progress in linear scaling *ab initio* electronic structure techniques, J. Phys.: Condens. Matter 14 (2002) 2781–2799.

[25] C.-K. Skylaris, P. D. Haynes, A. A. Mostofi, M. C. Payne, Introducing ONETEP: Linear-scaling density functional simulations on parallel computers, J. Chem. Phys. 122 (2005) 084119.

[26] C. J. Garcia-Cervera, J. Lu, Y. Xuan, W. E, Linear-scaling subspace-iteration algorithm with optimally localized nonorthogonal wave functions for Kohn-Sham density functional theory, Phys. Rev. B 79 (2009) 115110–1–13.

[27] D. Bowler, T. Miyazaki, Calculations for millions of atoms with density functional theory: linear scaling shows its potential, J. Phys.: Condens. Matter 22 (2010) 074207.

[28] E. R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, J. Comput. Phys. 17 (1975) 87–94.

[29] A. V. Knyazev, Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method, SIAM J. Sci. Comput. 23 (2001) 517–541.

[30] F. Bottin, S. Leroux, A. Knyazev, G. Zerah, Large-scale ab initio calculations based on three levels of parallelization, Comp. Mat. Sci. 42 (2008) 329–336.

[31] D. M. Wood, A. Zunger, A new method for diagonalising large matrices, J. Phys. A: Math. Gen. 18 (1985) 1343–1359.

[32] C. Vömel, S. Z. Tomov, O. A. Marques, A. Canning, L.-W. Wang, J. J. Dongarra, State-of-the-art eigensolvers for electronic structure calculations of large-scale nano-systems, J. Comp. Phys. 227 (2008) 7113–7124.

[33] J. Iwata, D. Takahashi, A. Oshiyama, T. Boku, K. Shiraishi, S. Okada, K. Yabana, A massively-parallel electronic-structure calculations based on real-space density functional theory, J. Comp. Phys. 229 (2010) 2339–2363.

[34] E. Anderson, Z. Bai, C. H. Bischof, L. S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. C. Sorensen, LAPACK Users' Guide, 3rd Edition, SIAM, Philadelphia, PA, 1999.

[35] J. Vandevondele, J. Hutter, An efficient orbital transformation method for electronic structure calculations, J. Chem. Phys. 118 (2003) 4365–4369.

[36] L. S. Blackford, J. Y. Choi, A. J. Cleary, E. D'Azevedo, J. W. Demmel, I. S. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. P. Petitet, K. S. Stanley, D. W. Walker, R. C. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, PA, 1997.

[37] C. H. Bischof, C. F. Van Loan, The WY representation for products of Householder matrices, SIAM J. Sci. Stat. Comput. 8 (1) (1987) s2–s13.

[38] R. S. Schreiber, C. F. Van Loan, A storage-efficient WY representation for products of Householder transformations, SIAM J. Sci. Stat. Comput. 10 (1989) 53–57.

[39] A. G. Sunderland, Parallel diagonalization performance on high-performance computers, in: R. Ciegis, et al. (Eds.), Parallel Scientific Computing and Optimization: Advances and Applications, Springer, Berlin, 2009, pp. 57–66.

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

[41] R. Johanni, V. Blum, V. Havu, H. Lederer, M. Scheffler, to be published. (2010).

[42] P. Havu, V. Blum, V. Havu, P. Rinke, M. Scheffler, Large-scale surface reconstruction energetics of Pt(100) and Au(100) by all-electron density functional theory, Phys. Rev. B (RC), accepted for publication.

[43] B. Lang, Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung, Habilitationsschrift, Bergische Universität GH Wuppertal, Fachbereich Mathematik (1997).

[44] B. Lang, A parallel algorithm for reducing symmetric banded matrices to tridiagonal form, SIAM J. Sci. Comput. 14 (6) (1993) 1320–1338.

[45] H. R. Schwarz, Tridiagonalization of a symmetric band matrix, Numer. Math. 12 (1968) 231–241.

[46] C. H. Bischof, B. Lang, X. Sun, A framework for symmetric band reduction, ACM Trans. Math. Software 26 (4) (2000) 581–601.

[47] C. H. Bischof, B. Lang, X. Sun, Algorithm 807: The SBR toolbox—software for successive band reduction, ACM Trans. Math. Software 26 (4) (2000) 602–616.

[48] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, P. R. Willems, Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem, submitted for publication (2010).

[49] R. Wittmann, Development and implementation of purpose-built high performance linear algebra kernels for the use on supercomputers, Bachelor's thesis, Institut für Informatik, Technische Universität München (2010).

[50] M. S. Lam, E. E. Rothberg, M. E. Wolf, The cache performance and optimizations of blocked algorithms, in: Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 1991, pp. 63–74.

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

[52] J. W. Demmel, Applied Numerical Linear Algebra, SIAM, Philadelphia, PA, 1997.

[53] J. W. Demmel, 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., SIAM, Philadelphia, PA, 1994, pp. 528–533.

[54] J. J. M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, Numer. Math. 36 (1981) 177–195.

[55] M. Gu, S. C. Eisenstat, A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem, SIAM J. Matrix Anal. Appl. 16 (1) (1995) 172–191.

[56] F. Tisseur, J. J. Dongarra, A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures, SIAM J. Sci. Comput. 20 (6) (1999) 2223–2236.

[57] R. A. van de Geijn, Using PLAPACK, MIT Press, Cambridge, MA, 1997.

[58] I. S. Dhillon, A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, Ph.D. thesis, University of California at Berkeley (1997).

[59] C. Vömel, ScaLAPACK's MRRR algorithm, ACM Trans. Math. Software 37 (1) (2009) 1–35.

[60] P. R. Willems, On MRRR-type algorithms for the tridiagonal symmetric eigenproblem and the bidiagonal SVD, Ph.D. thesis, Bergische Universität Wuppertal, Fachbereich Mathematik und Naturwissenschaften (2010).

[61] P. R. Willems, B. Lang, A framework for the MR$^3$ algorithm: Theory and implementation, in preparation (2010).