High Performance Computing in Material Sciences

—

Numerical Experiments with Symmetric Eigensolvers

Wilfried N. Gansterer
Institute for Applied and Numerical Mathematics
University of Technology Vienna
ganst@aurora.tuwien.ac.at

Dieter F. Kvasnicka
Institute for Technical Electrochemistry
University of Technology Vienna
kvasnicka@tuwien.ac.at

Christoph W. Ueberhuber
Institute for Applied and Numerical Mathematics
University of Technology Vienna
christof@uranus.tuwien.ac.at

November 1998

AURORA TR1998-19

The work described in this report was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund.

Abstract

This report describes and analyzes numerical experiments carried out with various symmetric eigensolvers in the context of the material science code WIEN 97. Of particular interest are the performance improvements achieved with a new Level 3 eigensolver. The techniques which lead to a significant speed up are (1) sophisticated blocking in the tridiagonalization step, which leads to a two-sweep algorithm, and (2) computing eigenvectors using inverse iteration on an appropriately chosen band matrix.

The new Level 3 eigensolver improves the locality of data references and leads to performance improvements of up to 200 % (depending on hardware features and the problem size).

Contents

1 Introduction **3**

2 Level 3 Tridiagonalization **5**

2.1 The New Level 3 Algorithm 6

2.2 Existing Tridiagonalization Codes 7

2.3 Successive Band Reduction 8

2.4 Eigenvector Computation 9

2.5 Back-transformation 9

3 The New Algorithm **11**

4 Complexity **14**

5 Experimental Results **16**

5.1 Comparison with Existing Codes 17

5.2 Number of Required Eigenvectors 27

5.3 Choosing the Block Size 31

Conclusion **40**

Outlook **41**

Acknowledgement **42**

References **43**

Chapter 1

Introduction

Haunschmid and Kvasnicka [13] presented a methodology for speeding up the generalized symmetric eigenproblem solver in the material science code WIEN 97. The most important part of these acceleration activities was the conversion from Level 2 (matrix-vector) routines to Level 3 (matrix-matrix) routines without increasing memory requirements. The methods used were hierarchical blocking and utilization of the block packed storage scheme. However, one part of the eigensolver could not be converted to utilize Level 3 BLAS: the tridiagonalization step. This step contains one call to a Level 2 BLAS routine, which is responsible for up to 80 % of the runtime of the standard symmetric eigenproblem.

Gansterer et al. [11] showed how the tridiagonalization step, which is notoriously difficult to be blocked, can be accelerated using mainly Level 3 routines. In [11] an overview is given of existing work on this topic and modifications are suggested.

This report presents experimental results of the implementation of the two-sweep symmetric eigensolver.

Motivation. Reducing a dense symmetric matrix to a condensed form as a first phase in the solution of symmetric eigenproblems is a standard technique. The motivation is that iterative algorithms require far fewer flops when applied to the condensed form instead of to the original dense matrix.

A real, symmetric $n \times n$ matrix A can be factorized as

$$A = RBR^T = QTQ^T = Z\Lambda Z^T \tag{1.1}$$

with R, Q, and Z being orthogonal matrices. B is a band matrix with band width[1] $2b+1$, T is a tridiagonal matrix, and Λ is a diagonal matrix. The diagonal

[1] Since B is a symmetric matrix, the number of subdiagonals and the number of superdiagonals are identical. b turns out to be the maximal possible block size in B.

elements of Λ are the eigenvalues of the (similar) matrices A, B, T, and Λ. The column vectors of Z are the eigenvectors of A. The band width of the matrices A, B, T, and Λ is $2n-1$, $2b+1$, 3, and 1, respectively.

LAPACK routines (see Anderson et al. [1]) reduce a symmetric dense matrix A to a symmetric tridiagonal matrix T by performing a series of Householder transformations. A bisection algorithm is used to compute selected[2] eigenvalues of T, which are also eigenvalues of A. The eigenvectors of T are found by inverse iteration. These eigenvectors have to be transformed into the eigenvectors of A using the transformation matrix Q.

The new method does not compute the tridiagonal matrix T directly, but derives the band matrix B first. This step can be adapted much better to modern computer architectures than a one-sweep tridiagonalization. Speedups of the two-sweep tridiagonalization of up to ten can be achieved.

The eigenvectors can be computed by inverse iteration applied to A, B, or T. In numerical experiments it turned out that inverse iteration applied to B is most effective. The optimum band width of B depends on the computer architecture and the number of required eigenvectors.

The new algorithm includes two special cases.

(1) If $R = I$ then $B = A$, and inverse iteration is performed on the dense matrix. This variant may be favorable if only a few eigenvectors with known eigenvalues have to be computed.

(2) If $R = Q$ then $B = T$, and inverse iteration is performed on the tridiagonal matrix. This variant coincides with the LAPACK algorithm.

Organization of this Report. Chapter 2 presents a new algorithm for solving the symmetric eigenvalue problem and existing implementations. The new algorithm computes not only eigenvalues, but also eigenvectors, which are of major importance for the program package WIEN 97. Chapter 3 shows how these methods have been implemented.

Chapter 4 presents an analytical model for describing the execution time of the algorithm used and shows how the band width and number of required eigenvectors influence the execution time.

In Chapter 5 the influence of the band width and the number of required eigenvectors is described. Results of numerical experiments carried out on various computer systems are presented.

[2] If *all* eigenvalues are required, LAPACK provides other algorithms.

Chapter 2

Level 3 Tridiagonalization of Symmetric Matrices

Close examination of the tridiagonalization routines `LAPACK/dsytrd` (using conventional, unpacked storage) and `LAPACK/dsptrd` (using packed storage) as well as the newly developed block packed version `dshtrd` (see Haunschmid, Kvasnicka [13]) shows that calls to the Level 2 BLAS routine `BLAS/dsymv` (matrix-vector product) requires nearly all of the computation time needed for the tridiagonalization of large matrices on computers where memory throughput limits floating-point performance.[1]

From the mathematical background it follows that inherent data dependencies prevent the reduction of a column, as long as the column reduced a step before was not applied to the whole matrix in a matrix-vector product (see Gansterer et al. [11]).

The blocked LAPACK approach puts the main emphasis on accumulating several elimination steps. b rank-2 updates (each of them implemented in a Level 2 BLAS routine) are aggregated to form one rank-$2b$ update and hence one Level 3 BLAS routine can be applied. However, this approach does not take into account memory access patterns in the updating process. We tried to introduce blocking in a much stricter sense, namely by localizing data accesses.

[1] It is impossible to exploit the data cache in matrix-vector operations if the memory requirements of the matrix exceed the cache size. If the floating-point units of the processor are running faster than the main memory can deliver required data and store the results (which is the case on most modern computer architectures), floating-point performance is limited by memory throughput.

2.1 The New Level 3 Algorithm

Partitioning matrix A into column blocks, and further partitioning each column block into quadratic submatrices makes it possible to process the block columns and their submatrices the same way that the single columns and their elements are processed in the original unblocked method: all *blocks*[2] below the subdiagonal *blocks* are eliminated, which leaves a *block* tridiagonal matrix, i. e., a *band* matrix B (when considered elementwise).

Further examination shows that the first elimination sweep can be organized such that the subdiagonal blocks in the block tridiagonal matrix are of upper triangular form. Hence the band width of B turns out to be 2b+1 and not $4b-1$.

The resulting algorithm (details are found in Gansterer et al. [11]) consists of the following steps.

1. Compute the transformation matrix for a subblock of the matrix. (It is exactly here where the possibility for blocking is won.) The Householder vectors for the transformation of the columns are independent of each other.

   The Householder vectors for the columns of the subblock of the matrix A are stored in the columns of the (rectangular) matrix Y.

2. Compute another matrix W for the representation $I - WY^T$ of the update using the matrix Y.

3. Perform a rank-$2b$ update using Y and W. The matrix Y is stored in columns of A which have well-known zero values. The matrix W will be overwritten and has to be computed again during backtransformation.

4. Iterate Steps 1 to 3 over the entire matrix.

5. Compute the eigenvalues and eigenvectors of the band matrix B, using appropriate algorithms.

6. Transform the eigenvectors of the band matrix B to the eigenvectors of the input matrix A, using the update matrices W and Y. In this step the matrix W is computed a second time. Storing W would require too much storage overhead (roughly $n^2/2$ elements).

Variations of this algorithm are possible.

[2]Replace *block* by *element* (or *elementwise*) to get the original, unblocked algorithm.

- The QR factorization can be used to compute the transformation matrix of Step 1. This implies that, for the generation of W, a LAPACK routine is applicable.

- Compute a matrix G instead of W to replace $I - WY^T$ by $I - GG^T$. This version requires more floating-point operations as well as more coding effort. It has the advantage, however, that only one matrix, G, has to be stored instead of two matrices, Y and W.

- Compute a small triangular matrix U to replace $I - WY^T$ by $I - YUY^T$. The cost of storing the U matrices is negligible (about as much as the cost of storing the band matrix B).

2.2 Existing Tridiagonalization Codes

Many computer programs for solving eigenvalue problems do not concentrate on an efficient tridiagonalization. A reason for this shortcoming could be that in many applications *all* eigenvalues have to be computed. In this case the time required for the tridiagonalization step can be less than for solving the symmetric tridiagonal eigenproblem. Much research work has already been done in the solution of the tridiagonal eigenproblem (see, for example, Dhillon [9]).

The blocking implemented in LAPACK is only a blocking in the matrix update. This blocking was developed in the late eighties and the early nineties. The Householder reflectors were collected and applied to the rest of the matrix using Level 3 BLAS calls, which are very efficient on modern computer systems. One matrix-vector product could not be removed, however, and this is responsible for up to 80 % of the total run time on computers using fast cache memory (see Haunschmid, Kvasnicka [13]). The concept of the LAPACK blocking is based on Schreiber, van Loan [15], Schreiber and Dongarra et al. [10]. Papers about parallelization are, for instance, Berry et al. [2] or Bischof et al. [8]. The remaining matrix-vector product limits the success of parallelization efforts. The new algorithm will overcome these limitations when transformed into a parallel version.

The papers mentioned above do not deal with the seemingly unavoidable matrix-vector product. Without major algorithmic modifications, it is impossible to convert this particular Level 2 BLAS call into Level 3 BLAS calls because of data dependencies (see Gansterer et al. [11]).

Newly developed methods replace the conventional one-sweep technique by two-sweep algorithms. The first sweep reduces the dense input matrix to a band matrix, the second sweep reduces the band matrix to a tridiagonal matrix. First attempts in this direction can be found in Bischof [5]. This paper concentrates on reducing band matrices to narrower banded form. The methods presented there can be easily adapted to dense matrices.

Bischof et al. [6, 7] developed the SBR (successive band reduction) toolbox which provides routines which were used in a first implementation of some of the band reduction steps in the new algorithm. This toolbox uses QR factorization. The SBR toolbox is publicly is available at `http://www.mcs.anl.gov/Projects/-PRISM/lib/software.html`.

2.3 Successive Band Reduction

The SBR toolbox consists of several subroutines for band reduction, tridiagonalization, the generation of the transformation matrix and repacking to and from banded storage as well as testing and timing routines which makes the task of finding the best parameters for a given machine easier.

The most important subroutines in the SBR toolbox are

`SBR/dsyrdb`, which uses blocked Householder transformations to reduce an $n \times n$ symmetric dense matrix to a symmetric band matrix.

`SBR/dsygtr`, which overwrites a matrix A with the orthogonal matrix R that reduces A to banded form. The matrix A is symmetric, the matrix R, however, is not symmetric, so a full matrix is needed for storage.

`SBR/dsbrdb`, which uses blocked Householder transformations to reduce a symmetric band matrix to narrower banded form. This routine is used in multisweep reduction methods.

`SBR/dsbrdt`, which uses Householder transformations to reduce a symmetric band matrix to tridiagonal form.

`SBR/dsy2sb`, which copies a symmetric band matrix from conventional storage (with either the upper or lower triangle explicitly stored) to LAPACK lower banded storage. The matrix may be repacked in place.

`SBR/dsb2sb`, which copies a symmetric band matrix from LAPACK (upper or lower packed) banded storage to lower banded storage. The matrix may be repacked in place.

The computational SBR routines are also capable of omitting parts of the matrix update if some elements of the Householder reflector are very small. This can be used for speeding up the total execution time if the matrix is diagonal dominant and the accuracy requirements are not too high (for example, in early steps of an iteration).

Unfortunately important subroutines necessary for our requirements are missing.

- There is no routine available which is equivalent to `LAPACK/dormtr`. This routine applies the orthogonal transformation matrix Q to the eigenvectors of the band matrix B in order to transform them to eigenvectors of the original matrix A. It is not required to compute the matrix Q explicitly when using this routine. The computation of the orthogonal matrix Q for only a few eigenvectors would introduce too much overhead.
- There is no routine for computing the eigenvectors of a band matrix. Inverse iteration on the band matrix should be faster than the code implemented in LAPACK (which is inverse iteration on the tridiagonal matrix plus back-transformation) if only a few eigenvectors are required (which is the case in the program package WIEN 97).

2.4 Eigenvector Computation

One way to compute eigenvectors is to compute them directly from the band matrix by inverse iteration. If the bandwidth is not too large, this scheme does not incur too much overhead.

Unfortunately, LAPACK only contains routines which compute eigenvectors of tridiagonal symmetric or upper Hessenberg matrices.

However, EISPACK (Smith et al. [17]) contains a routine, `EISPACK/bandv`, which computes the eigenvectors of a band matrix (in double precision data format) given its eigenvalues. To be sure that the eigenvectors are the same as in the corresponding (and more modern) routine `LAPACK/dstein` some modifications have to be applied, for example, the initialization of the eigenvectors before the iteration or the criteria for detecting *clustered* eigenvectors.

2.5 Back-transformation

It is possible to use the routine `LAPACK/dormtr` for the back-transformation. The operation performed by this routine is

$$C \leftarrow QC,$$

where the eigenvectors are stored in the columns of the matrix $C \in \mathbb{R}^{n\times k}$.

$$Q = H_1 H_2 \cdots H_{n-2} \in \mathbb{R}^{n\times n}$$

is the orthogonal matrix used in the band reduction of A. Q is never computed explicitly. Only the Householder vectors u_i are stored, which form the Householder matrices $H_i = I - u_i u_i^\top$.

In the special case of the first reduction sweep

$$R = \hat{H}_1 \hat{H}_2 \cdots \hat{H}_{n-b-1}$$

is a matrix composed of the (smaller) matrices $\hat{H}_i = I - \hat{u}_i \hat{u}_i^\top$. Only the $\hat{u}_i$ are stored explicitly. The first b elements of each eigenvector (stored in C_1) are not affected by the back-transformation, and the trailing elements (stored in C_2) can be computed as

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & R \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix},$$

or, using the routine `LAPACK/dormtr`:

$$C_2 \leftarrow R\, C_2.$$

Chapter 3

The New Algorithm

The following steps are implemented in the new Level 3 eigensolver `dsyxev` (see also Fig. 3.1).

1. Reduce the dense input matrix A to the band matrix B using the SBR routine `SBR/dsyrdb`.

2. Duplicate B by copying it to condensed format using `SBR/dsy2sb`.

3. Reduce B to the tridiagonal matrix T using either `SBR/dsbrdt` or the LAPACK routine `LAPACK/dsbtrd`.

4. Compute the desired eigenvalues using `LAPACK/dstebz`.

5. Compute the corresponding eigenvectors of B using inverse iteration implemented in the (modified) EISPACK routine `EISPACK/bandv`.

6. Transform the eigenvectors of B back to the eigenvectors of A using the routine `LAPACK/dormtr`.

This sequence of steps makes it possible to utilize Level 3 BLAS for all computations performed on the whole matrix. So it is possible to say that the new algorithm is "fully blocked".

Step 1 performs most of the floating-point operations. The blocking parameter, and hence the bandwidth of B, should be chosen not only to speed up Step 1, but also to avoid a significant loss of efficiency in Steps 3 and 5.

The advantage of Step 2 is two-fold: it boosts performance by increasing data locality, and it ensures that the band matrix B is still available after generating the tridiagonal matrix T. B is needed for the computation in Step 5.

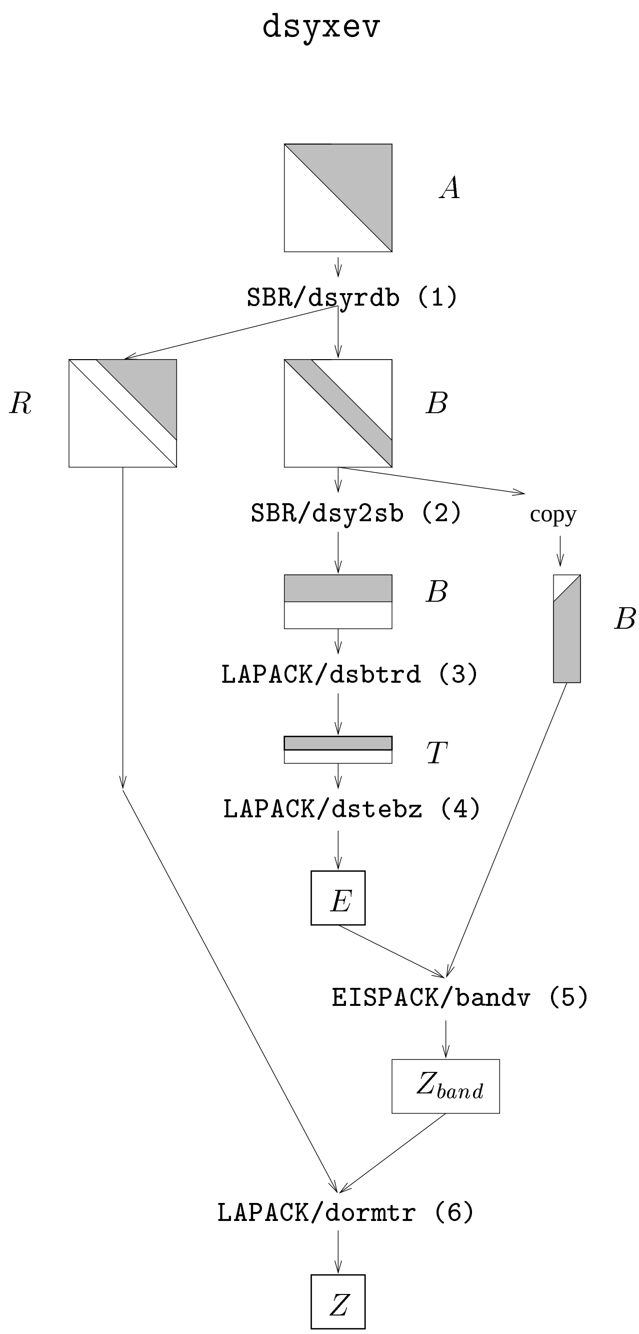

Figure 3.1: Subprogram call tree for the new Level 3 eigensolver `dsyxev`. The eigenvalues E and eigenvectors Z are computed. Dark shaded parts of the matrices are changed, blank parts of the matrices are either not referenced or used as temporary storage. R is the matrix which reduces A to block form B, the tridiagonal form of A is called T.

Step 3 may be blocked, too. The algorithm `SBR/dsbrdt` requires a higher leading dimension of the matrix for use as work space, twice the bandwidth of the band matrix, at the most. On some architectures higher performance is achieved by the unblocked version of this algorithm `LAPACK/dsbtrd`, especially if the band matrix has small band width.

Step 6 is done by the routine `LAPACK/dormtr` (see Section 2.5).

Level 2 BLAS calls are never performed for the whole matrix, but are frequently used inside current blocks. Extensive time measurements make clear which blocking parameters to use at a given computer (see Chapter 5).

The SBR routines as well as the EISPACK routine are available only for real single and double precision data, but not for complex data. Routines for complex data remain to be developed.

Chapter 4

Complexity

The total execution time T of the new Level 3 eigensolver consists of five parts:

$T_1 \sim c_1(b)n^3$ for reducing the symmetric matrix A to a band matrix B,

$T_2 \sim c_2(b)n^2$ for reducing the band matrix B to a tridiagonal matrix T,

$T_3 \sim c_3kn$ for computing k eigenvalues of T,

$T_4 \sim c_4kb^2n$ for computing the corresponding k eigenvectors[1] of B ($n \gg b$), and

$T_5 \sim c_5kn^2$ for transforming the eigenvectors of B into the eigenvectors of A.

The parameters c_1 and c_2 depend on the semibandwidth b. The parameter c_1 *decreases* in b, whereas c_2 *increases* in b due to an increasing number of operations to be performed. c_3, c_4, and c_5 are independent of the problem size.

The first reduction step is the only part of the algorithm requiring an $O(n^3)$ effort. Thus, a large semibandwidth b of B seems to be desirable. However, b should be chosen appropriately not only to speed up the first band reduction (T_1), but also to make the tridiagonalization (T_2) and the eigenvector computation (T_4) as efficient as possible.

For example, on an SGI Power Challenge the calculation of $k = 200$ eigenvalues and eigenvectors of a symmetric 2000×2000 matrix requires a total execution time $T = 90$ s. T_1 is 57 s, i. e., 63 % of the total time. With increasing matrix size this percentage increases.

[1] If eigenvalues are clustered and reorthogonalization is required, an additional term $\hat{c}_4k^2n$ dominates.

Chapter 5 describes experiments which help to select the optimum bandwidth b which leads to

$$T_{optimum} = \min_b T(b, n, k),$$

i. e., the shortest execution time for a given computer system and varying n and k. The way $T_{optimum}$ depends on the computer hardware is described in Section 5.1. The influence of k is described in Section 5.2, and the influence of b in Section 5.3. In all numerical experiments the problem size was varied in a wide range.

Chapter 5

Experimental Results

This chapter shows that the overall runtime of the symmetric eigensolution can be reduced significantly by using the new codes. In all numerical experiments 10 % of the eigenvalues and eigenvectors have been computed (with the exception of Section 5.2). This value is representative for what really happens in the material science package WIEN 97. Sometimes fewer eigenvectors are required, which makes the new routines even more beneficial.

The columns of the tables in this chapter are

n: matrix size;

k: number of eigenvalues and eigenvectors computed;

b: block size used in the newly developed routine `dsyxev`;

LAPACK: execution time of `LAPACK/dsyevx` (unpacked storage mode);

new Level 3: execution time of the new eigensolver `dsyxev` using Level 3 operations in the tridiagonalization step;

WIEN 97: execution time of the routine `dshevx` which uses the block packed storage mode and is implemented in WIEN 97;

dsyrdb: execution time of the routine `SBR/dsyrdb`, i. e., the blocked band reduction of the dense input matrix;

bandv: execution time of the routine `EISPACK/bandv`, i. e., the computation of the eigenvectors of the band matrix;

dormtr: execution time of the routine `LAPACK/dormtr`, i. e., the transformation of the eigenvectors of the band matrix into the eigenvectors of the input matrix;

dsbrdt: execution time of the routine `SBR/dsbrdt`, i. e., the blocked transformation from the band matrix into a tridiagonal matrix;

dsbtrd: execution time of the routine `LAPACK/dsbtrd`, i. e., the unblocked transformation from the band matrix into a tridiagonal matrix.

The computer systems used in the numerical experiments are listed in Table 5.1.

System	Processor	Peak Perf.	Memory Levels	OS BLAS	Compiler Arguments
DEC Alpha-Server 2100	Alpha 21064a 275 MHz	275	16 KB 4 MB 512 MB	OSF1 V3.2 dxml	Digital DFAO 4.0 +O3 -pipeline -transform_loops
SGI Power Challenge	MIPS R8000 90 MHz	360	64 KB 4 MB 768 MB	IRIX 64 6.2 complib. sgimath	Ragnarok 6.2 -mips4 -O3 -n32
SGI Power Challenge XL	MIPS R10000 195 MHz	390	32 KB 2 MB 4 GB	IRIX 64 6.2 complib. sgimath	f77 MIPS -mips4 -O3 -n32
HP 9000 K-460	PA-8000 180 MHz	720	1 MB 384 MB	HP-UX 10.20 libblas.1205.a	f77 B.10.20.02 +O2 +U77
Quant-X Alpha	Alpha 21164a 500 MHz	1000	8 KB 96 KB 2 MB 512 MB	Linux PHIPAC[1]	Digital DFAO 4.0 +O3 -pipeline -transform_loops

Table 5.1: Hardware platforms.

5.1 Comparison with Existing Codes

In this section the runtime and the efficiency of the new eigensolver routine `dsyxev` is compared with the eigensolver `LAPACK/dsyevx` and the (LAPACK based) routine `dshevx`, which is using block packed storage.

[1] PHIPAC (see Bilmes et al. [3]) is a program package for automatic generation of optimized matrix-matrix multiplication routines.

Different acceleration factors have been achieved. On the Silicon Graphics Power Challenge (using the MIPS R8000 processor running at 90 MHz) a speedup by a factor of two was achieved (compared to the optimum runtime achieved by routines written in LAPACK-style). On the other computer systems (SGI Power Challenge XL, Alpha and HP workstations, etc.) a speedup of between 20 and 40 % compared to the routines used in WIEN 97 (see Haunschmid and Kvasnicka [13]) was observed. Compared to original LAPACK routines, even higher improvement rates were achieved.

To achieve the same goal, the new eigensolver `dsyxev` requires more floating-point operations than the routines `LAPACK/dsyevx` and `dshevx`[2] (see Fig. 5.1).

- The reduction of the input matrix to a tridiagonal matrix in two sweeps requires more work than a one-sweep reduction, as implemented in LAPACK.
- The computation of the eigenvectors is performed on a band matrix instead of a tridiagonal matrix. The amount of work increases with the band width (see Chapter 4).

The new Level 3 algorithm shows a much better floating-point performance than the LAPACK or LAPACK based algorithms. The reason for this performance improvement is the utilization of Level 3 BLAS for all operations showing an $O(n^3)$ behavior (see Chapter 4).

Fig. 5.2 shows performance degradation for the LAPACK routine and—to a lesser extent—the LAPACK based routine used in WIEN 97 when the matrix size exceeds the cache size. The new Level 3 routine `dsyxev`, however, even shows a performance improvement for larger matrices. The picture remains the same for other computer systems as can be seen, for instance, in Fig. 5.4 or 5.6.

[2] `LAPACK/dsyevx` and `dshevx` perform the same number of floating-point operations.

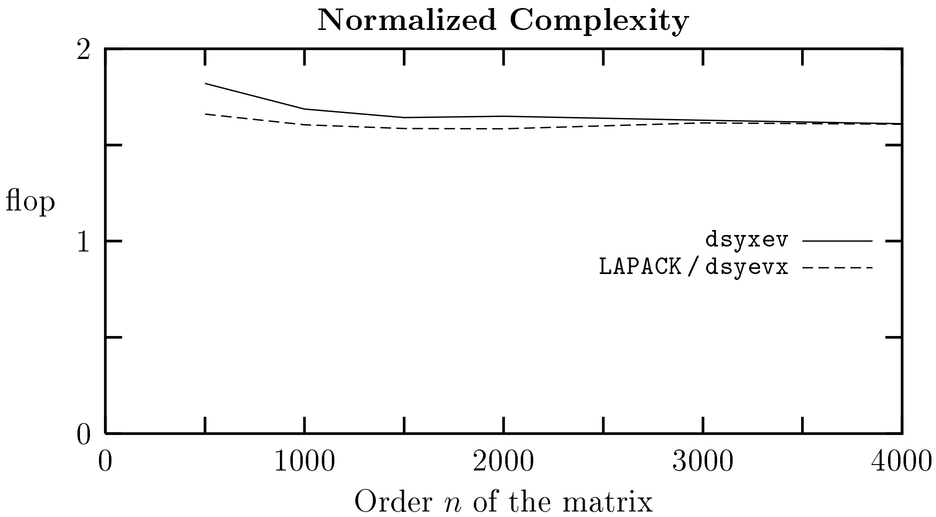

Figure 5.1: Normalized number $\mathrm{op}(n)/n^3$ of floating-point operations.

n	k	b	LAPACK	new Level 3	WIEN 97
500	50	6	2 s	2 s	2 s
1000	100	6	17 s	12 s	12 s
1500	150	6	74 s	39 s	51 s
2000	200	10	239 s	89 s	134 s
3000	300	12	945 s	286 s	529 s
4000	400	12	2432 s	660 s	1432 s

Table 5.2: Comparison of execution times on an SGI Power Challenge.

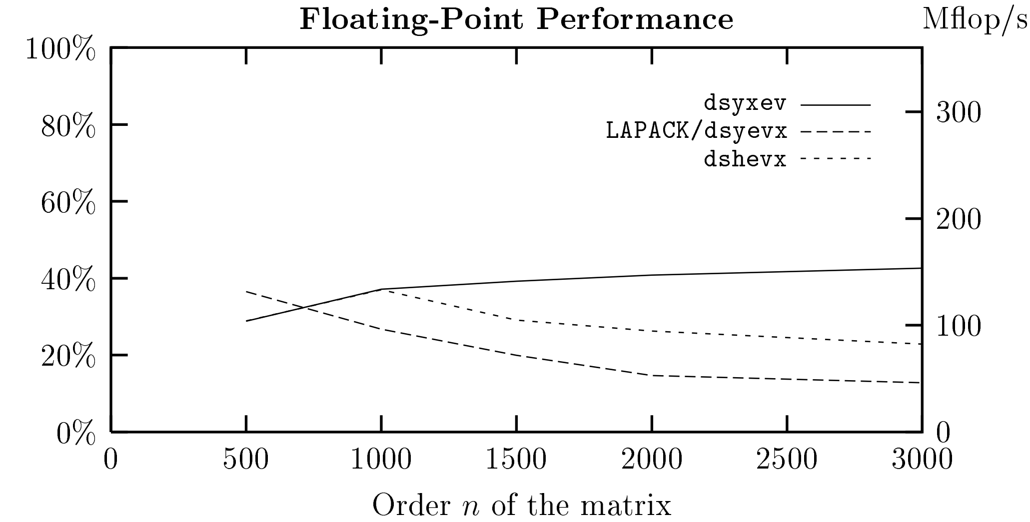

Figure 5.2: Performance of symmetric eigensolvers on an SGI Power Challenge.

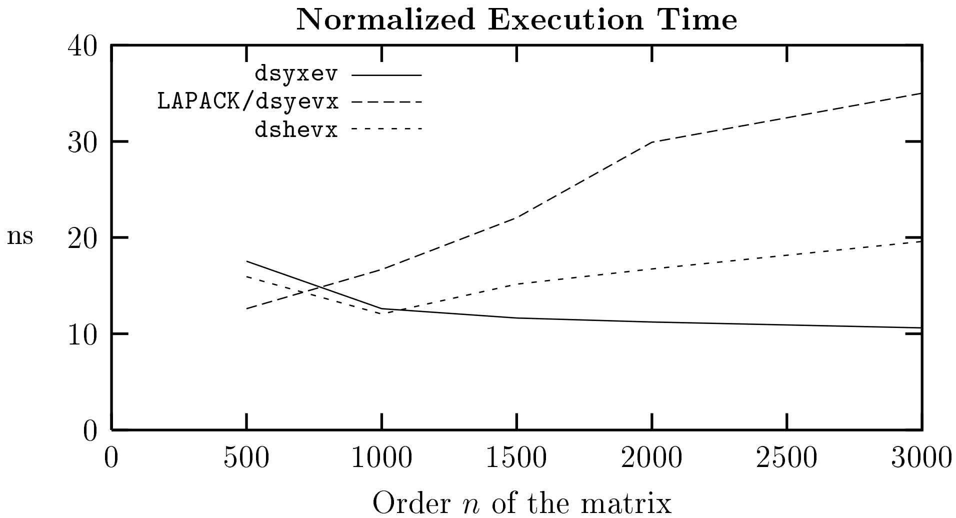

Figure 5.3: Normalized execution time $T(n)/n^3$ of symmetric eigensolvers on an SGI Power Challenge.

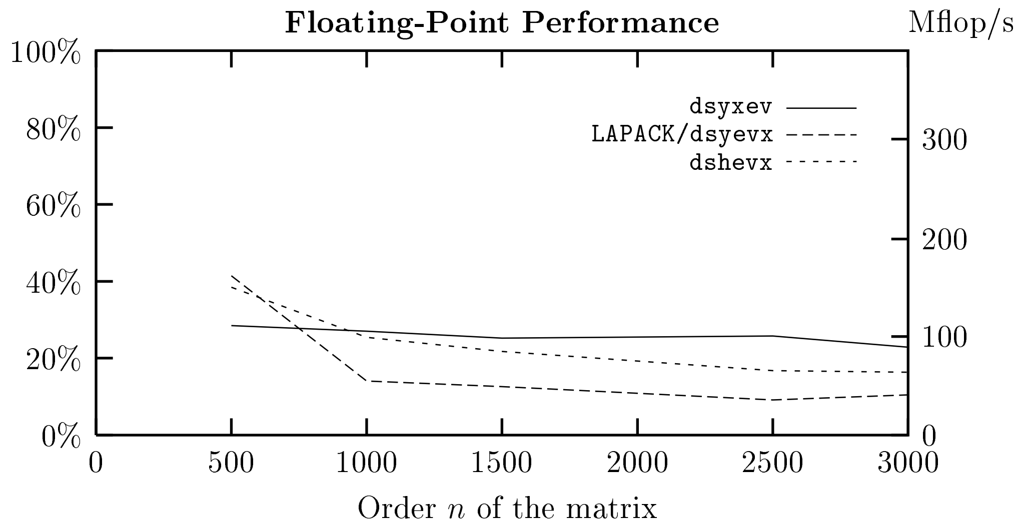

Figure 5.4: Performance of symmetric eigensolvers on an SGI Power Challenge XL.

n	k	b	LAPACK	new Level 3	WIEN 97
500	50	4	1 s	2 s	1 s
1000	100	6	29 s	16 s	16 s
1500	150	12	108 s	56 s	62 s
2500	250	12	694 s	256 s	378 s
3000	300	12	1068 s	493 s	683 s

Table 5.3: Comparison of execution times on an SGI Power Challenge XL.

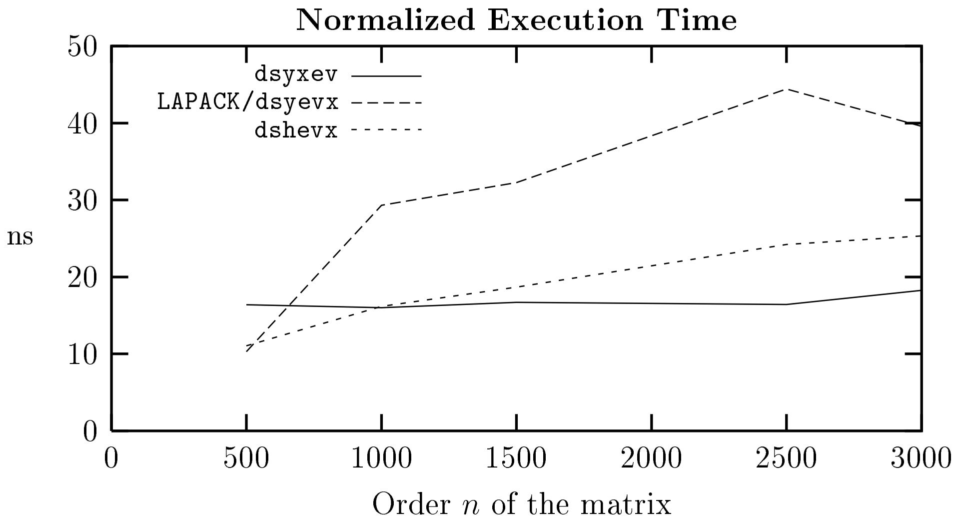

Figure 5.5: Normalized execution time $T(n)/n^3$ of symmetric eigensolvers on an SGI Power Challenge XL.

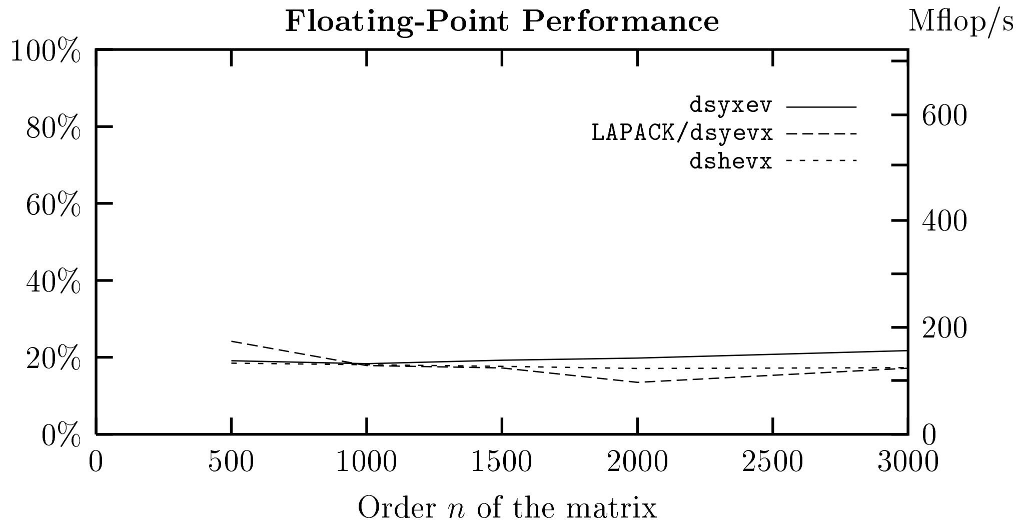

Figure 5.6: Performance of symmetric eigensolvers on an HP K460.

n	k	b	LAPACK	new Level 3	WIEN 97
500	50	6	1 s	2 s	2 s
1000	100	12	12 s	12 s	12 s
1500	150	12	43 s	40 s	42 s
2000	200	12	129 s	92 s	102 s
3000	300	12	351 s	280 s	349 s

Table 5.4: Comparison of execution times on an HP K460.

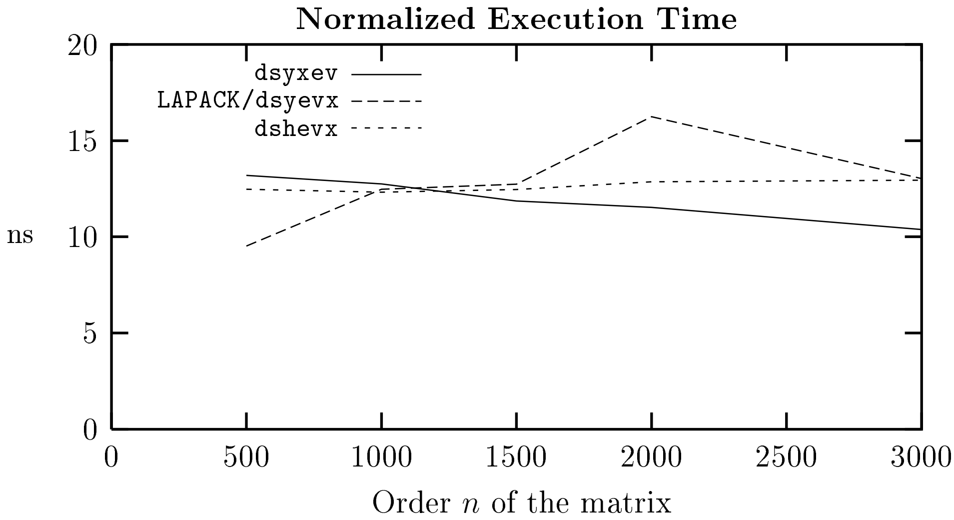

Figure 5.7: Normalized execution time $T(n)/n^3$ of symmetric eigensolvers on an HP K460.

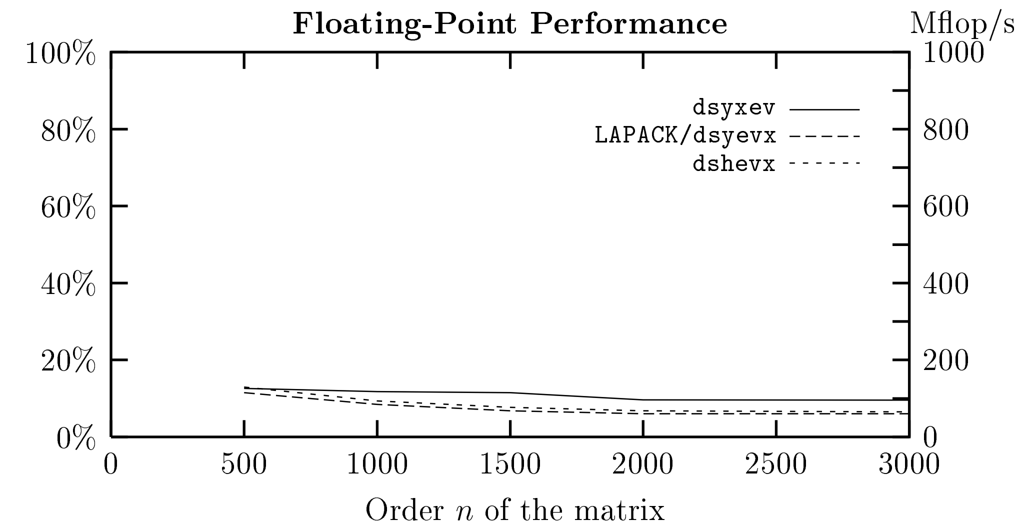

Figure 5.8: Performance of symmetric eigensolvers on an Alpha 21164 processor.

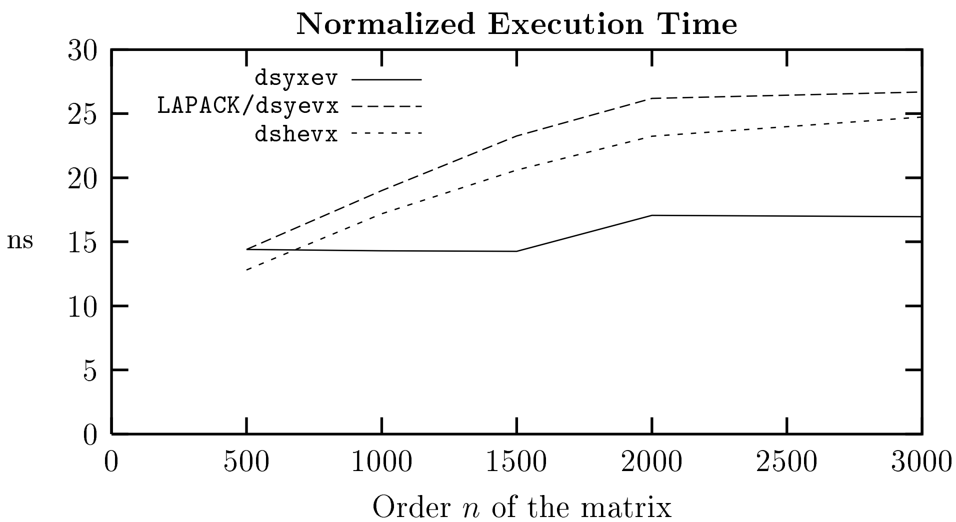

Figure 5.9: Normalized execution time $T(n)/n^3$ of symmetric eigensolvers on an Alpha 21164 processor.

n	k	b	LAPACK	new Level 3	WIEN 97
500	50	12	2 s	2 s	2 s
1000	100	12	19 s	14 s	17 s
1500	150	12	78 s	48 s	69 s
2000	200	12	209 s	136 s	186 s
2500	250	12	417 s	270 s	381 s
3000	300	12	720 s	458 s	667 s
4000	400	12	1821 s	1137 s	1647 s
5000	500	12	3560 s	2401 s	n/a

Table 5.5: Comparison of execution times on an Alpha 21164 processor.

5.2 Number of Required Eigenvectors

On an SGI Power Challenge (with MIPS R8000 processors) the new Level 3 eigensolver shows the best run time efficiency even if all eigenvalues and eigenvectors of large matrices are required. Table 5.6 shows execution times for the LAPACK routine, the new Level 3 routine, and the block packed routine which is used in the program package WIEN 97.

The block size b for the Level 3 routine `dsyxev` has been optimized for varying matrix size and varying number of required eigenvectors. It can be seen clearly that, for an increasing number of required eigenvectors, the optimal block size decreases. This effect is less pronounced for larger matrices than for smaller ones, where even for very few required eigenvectors small block sizes are required.

`LAPACK/dsyevx` and `dshevx` switch to different code if all eigenvectors are required. So there is an additional line in Table 5.6 for $k = n - 1$ to show the execution time behavior caused by this switch.

Figs. 5.10 to 5.13 show that the advantage of the new Level 3 eigensolver increases with increasing matrix size, as compared with conventional algorithms. However, Figs. 5.10 to 5.12 also show that the time per eigenvector has a minimum at $k \sim n/3$ for the Level 3 routine `dsyxev`. In the LAPACK algorithm, every additional eigenvector is relatively cheap to compute.

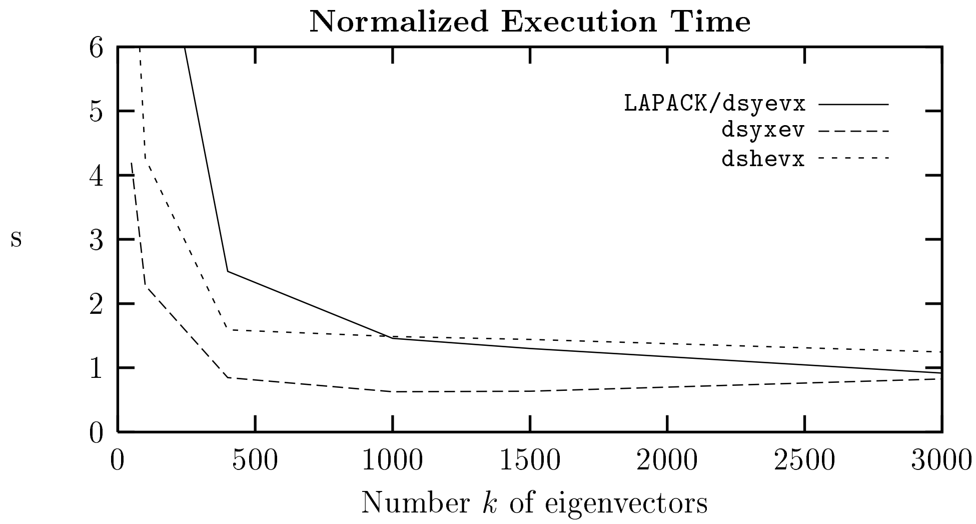

Figure 5.10: Normalized execution time (time in seconds/eigenvector) for varying number of eigenvectors on one processor of an SGI Power Challenge. Matrix size is 3000.

n	k	b	LAPACK	new Level 3	WIEN 97
1000	50	6	18 s	11 s	10 s
	100	6	18 s	13 s	11 s
	999	6	52 s	110 s	90 s
	1000	6	97 s	110 s	103 s
1500	50	12	83 s	30 s	45 s
	100	10	86 s	36 s	49 s
	400	8	113 s	73 s	80 s
	1000	6	135 s	191 s	212 s
	1499	4	172 s	348 s	323 s
	1500	3	413 s	344 s	459 s
2000	50	12	252 s	65 s	121 s
	100	10	234 s	76 s	132 s
	400	8	269 s	131 s	184 s
	1000	6	351 s	305 s	488 s
	1500	5	410 s	512 s	672 s
	1999	5	469 s	787 s	805 s
	2000	4	1005 s	765 s	1168 s
3000	50	16	937 s	209 s	429 s
	100	12	916 s	228 s	426 s
	400	12	1001 s	338 s	636 s
	1000	6	1460 s	626 s	1490 s
	1500	6	1954 s	952 s	2167 s
	2999	4	2760 s	2484 s	3740 s
	3000	4	3317 s	2485 s	3780 s
4000	3999	n/a	11090 s	n/a	13349 s
	4000	8	8115 s	4439 s	9234 s

Table 5.6: Comparison of execution times on an SGI Power Challenge (MIPS R8000 processor).

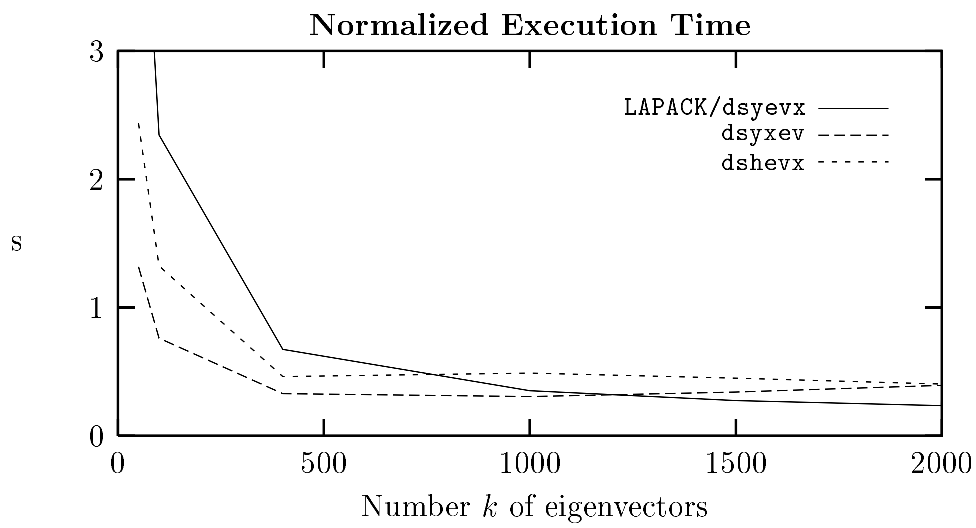

Figure 5.11: Normalized execution time (time in seconds/eigenvector) for varying number of eigenvectors on one processor of an SGI Power Challenge. Matrix size is 2000.

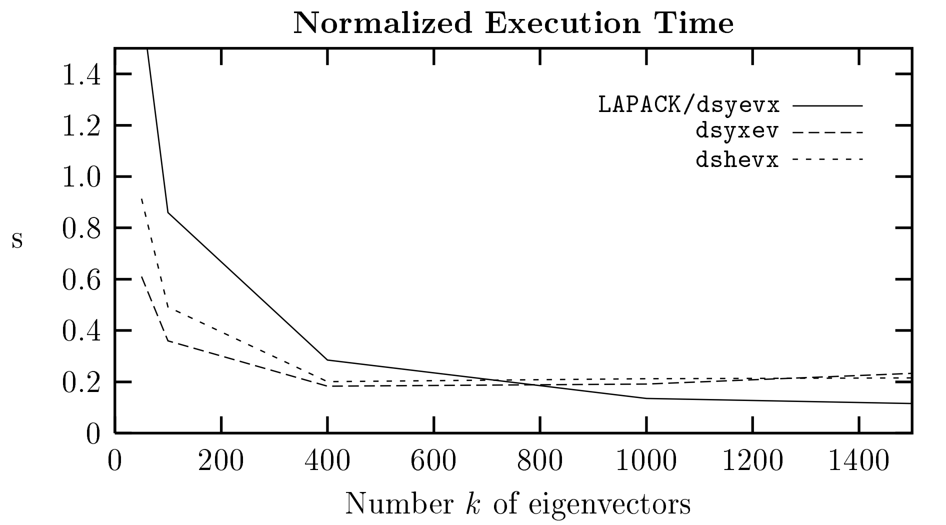

Figure 5.12: Normalized execution time (time in seconds/eigenvector) for varying number of eigenvectors on one processor of an SGI Power Challenge. Matrix size is 1500.

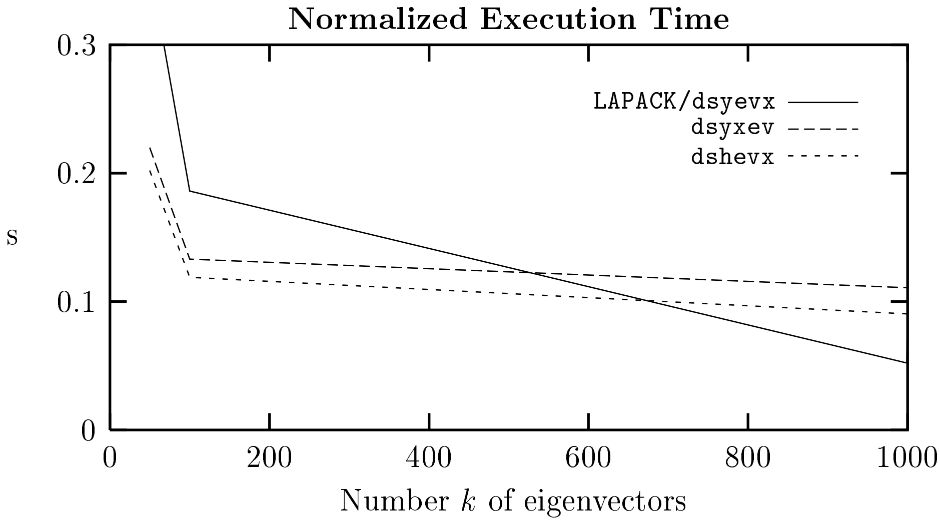

Figure 5.13: Normalized execution time (time in seconds/eigenvector) for varying number of eigenvectors on one processor of an SGI Power Challenge. Matrix size is 1000.

5.3 Choosing the Block Size

To find the optimal blocking parameter b for a given computer system, a given matrix size n, and a given number k of required eigenvectors numerical experiments have to be carried out.

The optimum blocking parameter must neither be too small nor too large (see Tables 5.7 to 5.12 and Figures 5.14 to 5.19).

- Small values of b incur more memory traffic and thus longer execution times for the band reduction of the dense input matrix using `SBR/dsyrdb`. This routine performs most of the work. So even small differences in the execution time of this routine are important for the overall runtime.

- Large values of b lead to an improved cache utilization in `SBR/dsyrdb`. However, large values of b lead to higher execution times needed for the tridiagonal reduction of the band matrix using `SBR/dsbrdt` or `LAPACK/dsbtrd`. Even more important, large values of b lead to much larger execution times needed for the inverse iteration on the band matrix using `EISPACK/bandv`. This EISPACK routine is faster than `SBR/dsyrdb`, but its runtime variations due to a varied blocking parameter are of the same order of magnitude.

Once an optimal blocking parameter has been found, qualitative predictions for other parameters can be made:

- For increasing matrix size n, the optimum blocking parameter b increases, too. The band reduction of the dense input matrix is the only step which requires $O(n^3)$ floating-point operations (see Chapter 4). So this step becomes more important for increasing matrix sizes. Choosing a large blocking parameter reduces the execution time of the band reduction.

- If an increased number of eigenvalues are required, the optimum block size decreases. In this case the band reduction is getting less important for the overall runtime.

- Hardware features may have a significant influence on the optimum choice of the blocking parameter. Some values of the blocking parameter b may lead to better performance than others, if they are adapted to the size of the cache lines. This makes multiples of four chosen for b beneficial on some computer systems (see Fig. 5.20).

  Note, Fig. 5.20 shows the execution time for block sizes $b = 6, 7, \ldots, 32$, whereas in Fig. 5.14 to 5.19 block sizes are $b = 6, 8, 10, 12, 16, 20, 24, 32$.

n	k	b	dsyxev	dsyrdb	dsbtrd	bandv	dormtr
4000	400	6	1686 s	1425 s	44 s	114 s	93 s
		8	1401 s	1134 s	39 s	127 s	92 s
		10	1348 s	1066 s	31 s	149 s	92 s
		12	1239 s	931 s	31 s	175 s	92 s
		16	1205 s	834 s	29 s	239 s	93 s
		20	1226 s	772 s	26 s	326 s	92 s
		24	1296 s	734 s	28 s	432 s	92 s
		32	1946 s	689 s	31 s	1123 s	93 s

Table 5.7: Execution times on an SGI Power Challenge XL. The execution time of the routine `dsyxev` is the sum of the other times in the same line.

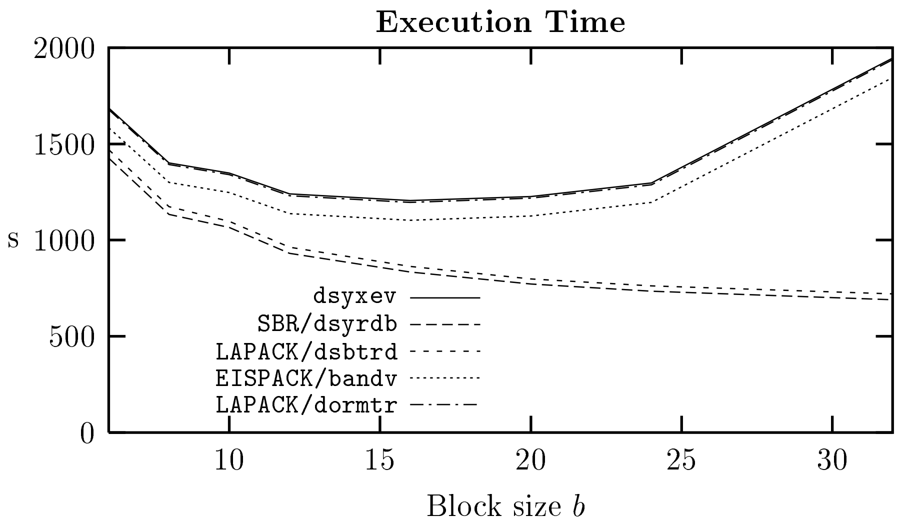

Figure 5.14: Execution time for varying block size. Matrix size is 4000. Accumulative representation.

n	k	b	`dsyxev`	`dsyrdb`	`dsbtrd`	`bandv`	`dormtr`
3000	300	6	599 s	484 s	24 s	50 s	34 s
		8	515 s	394 s	22 s	61 s	33 s
		10	501 s	372 s	17 s	73 s	33 s
		12	472 s	327 s	17 s	88 s	33 s
		16	475 s	295 s	16 s	124 s	33 s
		20	498 s	275 s	14 s	169 s	33 s
		24	541 s	263 s	15 s	223 s	33 s
		32	671 s	248 s	17 s	367 s	33 s

Table 5.8: Execution times on an SGI Power Challenge XL. The execution time of the routine `dsyxev` is the sum of the other times in the same line.

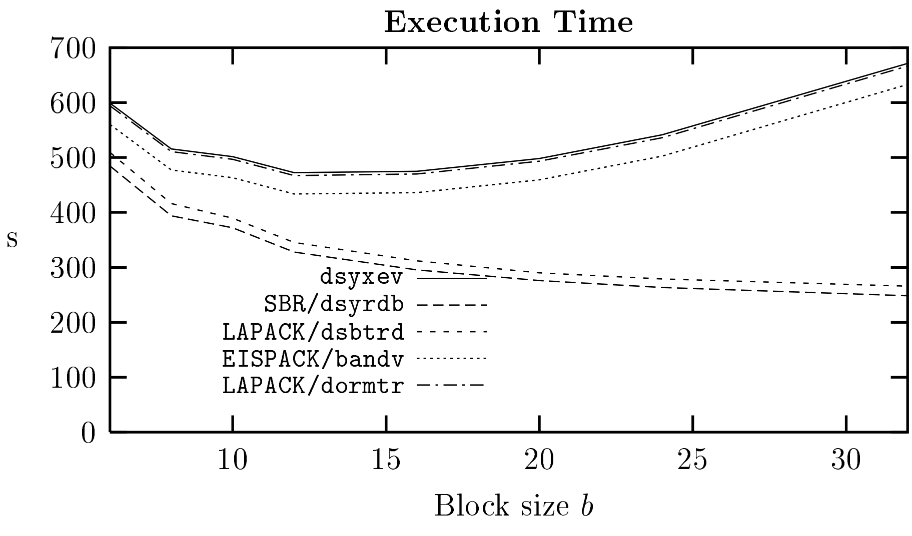

Figure 5.15: Execution time for varying block size. Matrix size is 3000. Accumulative representation.

n	k	b	dsyxev	dsyrdb	dsbtrd	bandv	dormtr
2000	200	6	146 s	106 s	11 s	17 s	9 s
		8	130 s	88 s	9 s	21 s	9 s
		10	131 s	84 s	7 s	27 s	9 s
		12	129 s	76 s	7 s	33 s	9 s
		16	139 s	70 s	7 s	49 s	9 s
		20	154 s	66 s	6 s	69 s	9 s
		24	177 s	64 s	6 s	93 s	9 s
		32	234 s	61 s	7 s	153 s	9 s

Table 5.9: Execution times on an SGI Power Challenge XL. The execution time of the routine **dsyxev** is the sum of the other times in the same line.

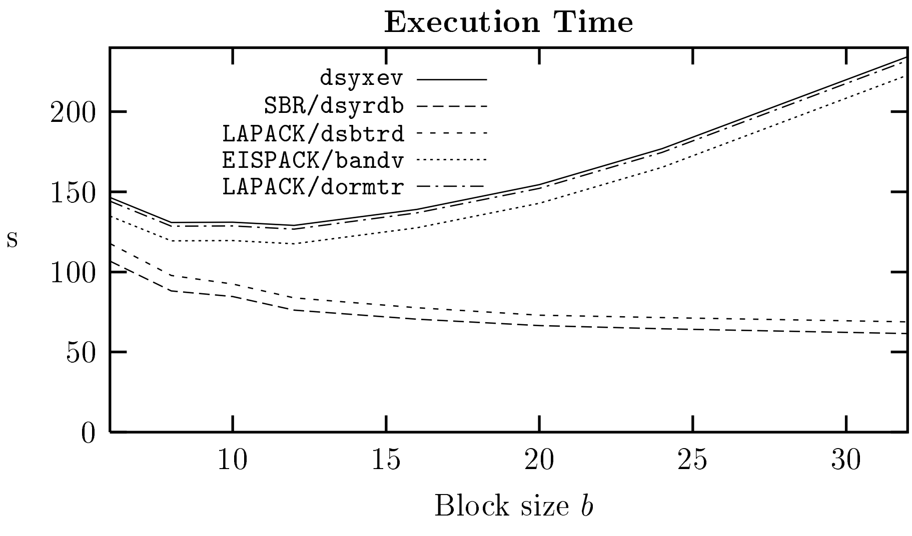

Figure 5.16: Execution time for varying block size. Matrix size is 2000. Accumulative representation.

n	k	b	`dsyxev`	`dsyrdb`	`dsbtrd`	`bandv`	`dormtr`
1500	150	6	56 s	37 s	6 s	8 s	3 s
		8	53 s	32 s	5 s	10 s	3 s
		10	52 s	29 s	4 s	13 s	3 s
		12	53 s	27 s	4 s	17 s	3 s
		16	60 s	25 s	4 s	26 s	3 s
		20	69 s	23 s	3 s	37 s	3 s
		24	82 s	23 s	3 s	50 s	3 s
		32	115 s	23 s	4 s	84 s	3 s

Table 5.10: Execution times on an SGI Power Challenge XL. The execution time of the routine `dsyxev` is the sum of the other times in the same line.

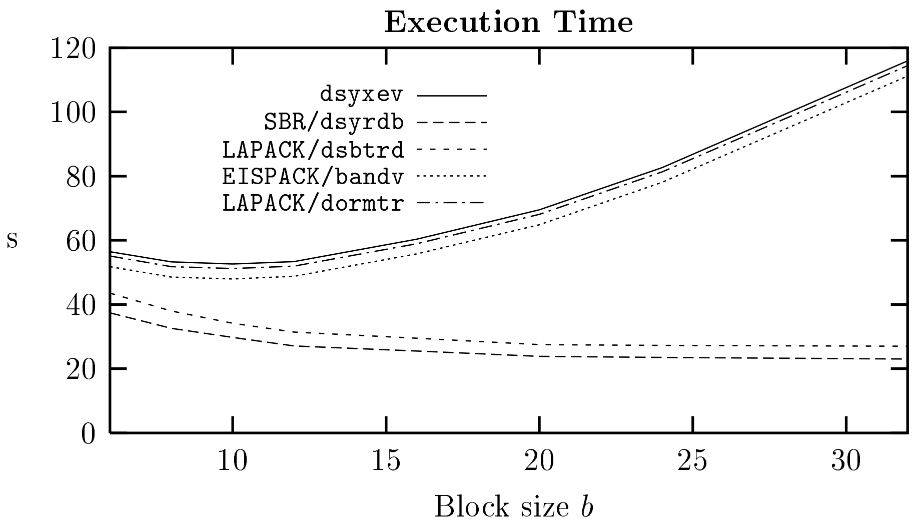

Figure 5.17: Execution time for varying block size. Matrix size is 1500. Accumulative representation.

n	k	b	`dsyxev`	`dsyrdb`	`dsbtrd`	`bandv`	`dormtr`
1000	100	6	17 s	10 s	3 s	3 s	1 s
		8	17 s	9 s	2 s	4 s	1 s
		10	17 s	8 s	2 s	5 s	1 s
		12	18 s	7 s	2 s	7 s	1 s
		16	21 s	7 s	2 s	11 s	1 s
		20	25 s	6 s	2 s	16 s	1 s
		24	31 s	6 s	2 s	21 s	1 s
		32	46 s	6 s	2 s	36 s	1 s

Table 5.11: Execution times on an SGI Power Challenge XL. The execution time of the routine `dsyxev` is the sum of the other times in the same line.

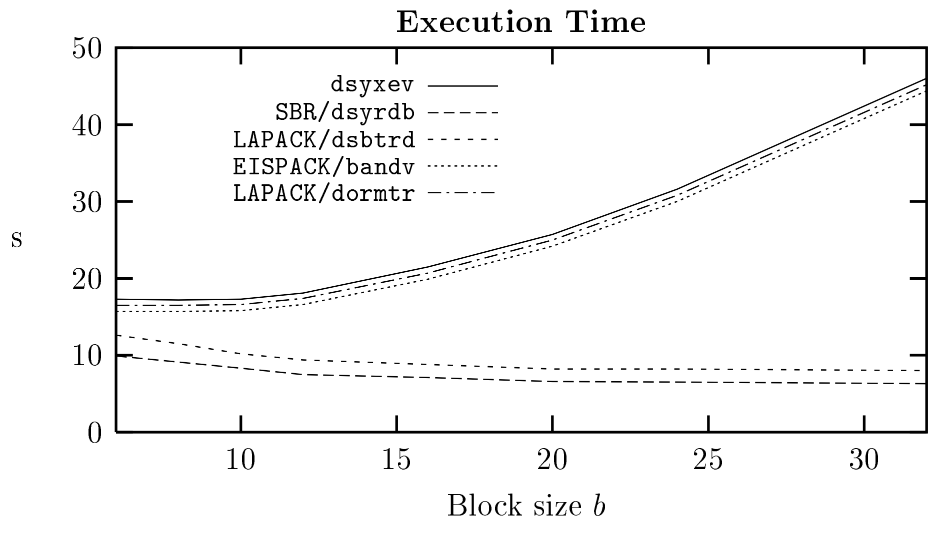

Figure 5.18: Execution time for varying block size. Matrix size is 1000. Accumulative representation.

n	k	b	dsyxev	dsyrdb	dsbtrd	bandv	dormtr
500	50	6	2.5 s	0.9 s	0.6 s	0.6 s	0.1 s
		8	2.7 s	0.9 s	0.6 s	0.9 s	0.1 s
		10	2.8 s	0.8 s	0.4 s	1.2 s	0.1 s
		12	3.1 s	0.8 s	0.4 s	1.6 s	0.1 s
		16	4.1 s	0.7 s	0.4 s	2.5 s	0.1 s
		20	5.2 s	0.7 s	0.3 s	3.7 s	0.1 s
		24	6.6 s	0.7 s	0.4 s	5.1 s	0.1 s
		32	10.1 s	0.7 s	0.4 s	8.6 s	0.1 s

Table 5.12: Execution times on an SGI Power Challenge XL. The execution time of the routine `dsyxev` is the sum of the other times in the same line.

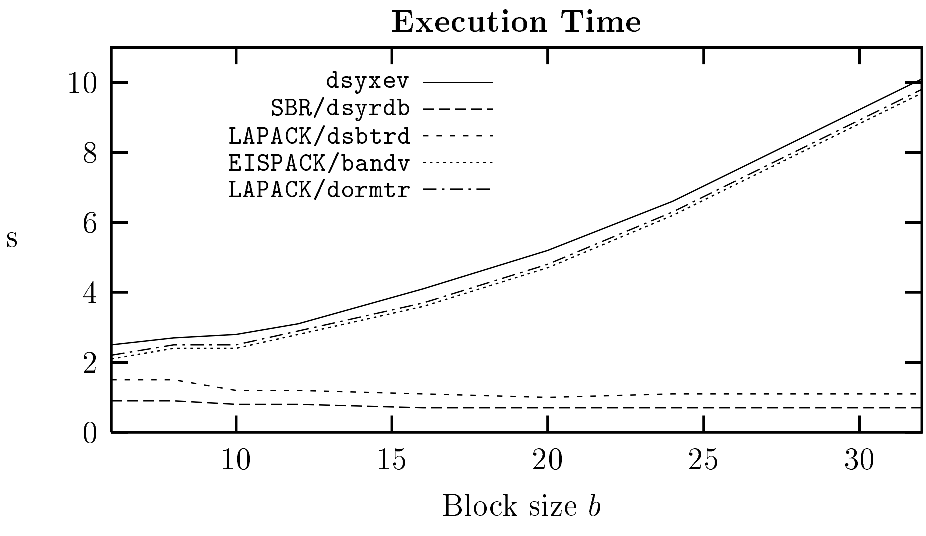

Figure 5.19: Execution time for varying block size. Matrix size is 500. Accumulative representation.

n	k	b	dsyxev	dsyrdb	bandv
4000	400	6	1689 s	1427 s	148 s
		7	1541 s	1265 s	159 s
		8	1437 s	1147 s	172 s
		9	1471 s	1166 s	186 s
		10	1391 s	1071 s	201 s
		11	1337 s	997 s	217 s
		12	1301 s	943 s	236 s
		13	1349 s	972 s	254 s
		14	1309 s	913 s	274 s
		15	1302 s	875 s	296 s
		16	1297 s	847 s	320 s
		17	1337 s	869 s	343 s
		18	1330 s	830 s	371 s
		19	1331 s	803 s	401 s
		20	1335 s	772 s	432 s
		24	1444 s	737 s	568 s
		25	1507 s	764 s	606 s
		32	2359 s	693 s	1494 s
		40	4053 s	690 s	3172 s
		48	6138 s	680 s	5228 s
		56	8575 s	677 s	7652 s
		64	11535 s	675 s	10563 s

Table 5.13: Execution time for varying block size on an SGI Power Challenge XL. Matrix size is 4000. Multiples of four are advantageous for the block size.

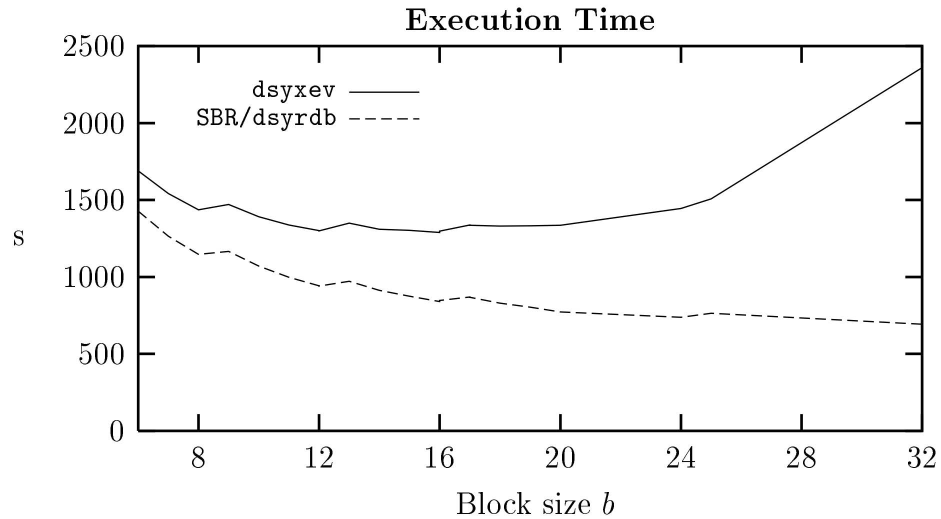

Figure 5.20: Execution time for varying block size on an SGI Power Challenge XL. Matrix size is 4000. Multiples of four are advantageous for the block size.

Conclusion

A new algorithm for the computation of eigenvalues and eigenvectors of symmetric matrices was presented, which is up to twice as fast as existing algorithms. The method used is Level 3 blocking in the tridiagonalization step, not only for the matrix update, as already implemented in LAPACK, but also in steps needed for the generation of update matrices.

The price that has to be paid for this efficiency improvement is that the eigenvectors cannot be computed from the tridiagonal matrix, otherwise serious overhead would occur. The eigenvectors have to be computed by inverse iteration from an intermediate band matrix.

Choosing the band width means making a tradeoff between (i) maximizing the block size to speed up the band reduction of the dense input matrix, and (ii) minimizing the block size to decrease the time needed for inverse iteration. This compromise makes the new method only competitive as long as not all eigenvectors are required. The application of the new Level 3 eigensolver in the material science package WIEN 97 is of the required type: only about 10 % of the eigenvectors are to be computed.

For future computer generations, where the gap between processor speed and memory bandwidth will most likely increase, and some of the current computer systems, the new eigensolver is the most efficient one even if all eigenvectors are required.

Outlook

There is room for improvement of the two most important subalgorithms, (i) the reduction of the dense input matrix to a band matrix, and (ii) inverse iteration for computing eigenvectors.

The block size of the band reduction step in the new eigensolver is coupled with the band width parameter. Experiments show that for large matrices this blocking parameter is often too small to utilize the high performance of Level 3 Blas. Dongarra et al. [10] introduced some other way of blocking the matrix. This blocking technique could be incorporated additionally into the new eigensolver, and would further improve its run time efficiency.

Routines dominated by Level 3 Blas operations, like the Level 3 eigensolver presented in this report, have the potential of speeding up almost linearly on parallel machines. This favorable property will be very useful in the development of a new parallel eigensolver.

Acknowledgement

We would like to thank Karlheinz Schwarz and Peter Blaha for providing us with information on the program package WIEN 97.

In addition, we would like to acknowledge the financial support of the Austrian Science Fund FWF.

Bibliography

[1] E. Anderson et al., LAPACK *Users' Guide*, SIAM Press, Philadelphia, 1995.

[2] M. W. Berry, J. J. Dongarra, Y. Kim, *A Highly Parallel Algorithm for the Reduction of a Nonsymmetric Matrix to Block Upper-Hessenberg Form*, Technical Report CS-94-221, LAPACK Working Note 68, 1994.

[3] J. Bilmes, K. Asanovic, C.-W. Chin, J. Demmel, *Optimizing Matrix Multiply using* PHIPAC*: a Portable, High-Performance, ANSI C Coding Methodology*, Proceedings of the International Conference on Supercomputing, ACM, Vienna, Austria, 1997, pp. 340–347.

[4] C. H. Bischof, B. Lang, X. Sun, *Parallel Tridiagonalization through Two-Step Band Reduction*, Proceedings of the Scalable High-Performance Computing Conference, IEEE, Washington D. C., 1994, pp. 23–27.

[5] C. H. Bischof, *A Summary of Block Schemes for Reducing a General Matrix to Hessenberg Form*, Technical Report ANL/MCS-TM-175, Argonne National Laboratory, 1993.

[6] C. H. Bischof, B. Lang, X. Sun, *A Framework for Symmetric Band Reduction*, Technical Report ANL/MCS-P586-0496, Argonne National Laboratory, 1996.

[7] C. H. Bischof, B. Lang, X. Sun, *The* SBR *Toolbox—Software for Successive Band Reduction*, Technical Report ANL/MCS-P587-0496, Argonne National Laboratory, 1996.

[8] C. H. Bischof, M. Marques, X. Sun, *Parallel Bandreduction and Tridiagonalization*, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, (R. Sincovec, Ed.), SIAM Press, 1993, pp. 383–390.

[9] I. S. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue / Eigenvector Problem*, Ph. D. Thesis, University of California, Berkeley, 1997.

[10] J. J. Dongarra, S. J. Hammarling, D. C. Sorensen, *Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations*, J. Comput. Appl. Math. 27 (1989), pp. 215–227.

[11] W. Gansterer, D. Kvasnicka, C. Ueberhuber, *High Performance Computing in Material Sciences. Higher Level* BLAS *in Symmetric Eigensolvers*, Technical Report AURORA TR1998-18, Vienna University of Technology, 1998.

[12] G. H. Golub, C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[13] E. Haunschmid, D. Kvasnicka, *High Performance Computing in Material Sciences. Maximizing Cache Utilization without Increasing Memory Requirements*, Technical Report AURORA TR1998-17, Vienna University of Technology, 1998.

[14] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, 1980.

[15] R. Schreiber, C. V. Loan, *A Storage-Efficient WY Representation for Products of Householder Transformations*, SIAM J. Sci. Stat. Comput. 10 (1989), pp. 53–57.

[16] R. Schreiber, B. Parlett, *Block Reflectors: Theory and Computation*, SIAM J. Numer. Anal. 25 (1988), pp. 189–205.

[17] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, *Matrix Eigensystem Routines*—EISPACK *Guide*, Lecture Notes in Computer Science, Vol. 6, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1976.