Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2010/04-3

03/April/2010

Condensed Forms for the Symmetric Eigenvalue Problem on Multi-threaded Architectures

P. Bientinesi, F. Igual, D. Kressner, M. Petschow and E. Quintana-Orti

RWTH AACHEN UNIVERSITY

AICES

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.

List of AICES technical reports: http://www.aices.rwth-aachen.de/preprints

Condensed Forms for the Symmetric Eigenvalue Problem on Multi-threaded Architectures

Paolo Bientinesi, Matthias Petschow*

Francisco D. Igual, Enrique S. Quintana-Ortí †

Daniel Kressner‡

April 3, 2010

Abstract

We investigate the performance of the routines in LAPACK and the *Successive Band Reduction* (SBR) toolbox for the reduction of a dense matrix to tridiagonal form, a crucial preprocessing stage in the solution of the symmetric eigenvalue problem, on general-purpose multi-core processors. In response to the advances of hardware accelerators, we also modify the code in the SBR toolbox to accelerate the computation by off-loading a significant part of the operations to a graphics processor (GPU). Performance results illustrate the parallelism and scalability of these algorithms on current high-performance multi-core and many-core architectures.

1 Introduction

We consider the solution of the *symmetric eigenvalue problem* $AX = X\Lambda$, where $A \in \mathbb{R}^{n\times n}$ is a *dense* symmetric matrix, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the eigenvalues of A, and the j-th column of the orthogonal matrix $X \in \mathbb{R}^{n\times n}$ is an eigenvector associated with λ_j [9]. Given the matrix A, the objective is to compute its eigenvalues or a subset thereof and, if requested, the associated eigenvectors as well. Many scientific and engineering applications lead to large eigenvalue problems. Examples come from computational quantum chemistry, finite element modeling, multivariate statistics, and density functional theory. There, problems become particularly challenging when a significant fraction of the eigenvalues and eigenvectors needs to be computed [12]. From here on we will concentrate on the case that all the eigenvalues and eigenvectors are desired.

*AICES, RWTH Aachen, Germany. {pauldj, petschow}@aices.rwth-aachen.de

†Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, Spain. {figual,quintana}@icc.uji.es

‡Seminar für angewandte Mathematik, ETH Zürich, Switzerland. kressner@math.ethz.ch.

Efficient algorithms for the solution of symmetric eigenvalue problems usually consist of three stages. Matrix A is first reduced to a symmetric tridiagonal matrix $T \in \mathbb{R}^{n \times n}$ by a sequence of orthogonal similarity transforms: $Q^T A Q = T$, where $Q \in \mathbb{R}^{n \times n}$ is the matrix representing the accumulation of these orthogonal transforms. A tridiagonal eigensolver as, e.g., the $\texttt{MR}^3$ algorithm [7, 4] is then applied to matrix T to accurately compute its eigenvalues and, optionally, the associated eigenvectors. Finally, when the eigenvectors of A are desired, a back-transform has to be applied to the eigenvectors of T. In particular, if $TX_T = X_T\Lambda$, with $X_T \in \mathbb{R}^{n \times n}$ representing the eigenvectors of T, then $X = QX_T$. Both the first and last stage cost $O(n^3)$ floating-point arithmetic operations (flops) while the second stage based on the $\texttt{MR}^3$ algorithm only requires $O(n^2)$ flops. (Other algorithms for solving tridiagonal eigenvalue problems, such as the QR algorithm, the Divide & Conquer method, etc. [9] require $O(n^3)$ flops in the worst case.)

In this paper we re-evaluate the performance of the codes in LAPACK [1] and the *Successive Band Reduction* (SBR) toolbox [5] for the reduction of a symmetric matrix A to tridiagonal form. LAPACK routine SYTRD employs Householder reflectors, enhanced with WY representations [8], to reduce A directly to tridiagonal form. Only half of its operations can be performed in terms of calls to Level-3 BLAS kernels, resulting in a poor use of the memory hierarchy. To overcome this drawback, the SBR toolbox first reduces A to an intermediate banded matrix B, and subsequently transforms B to tridiagonal form. The advantage of this two-step procedure is that the first step can be carried out using BLAS-3 kernels, while the cost of the second step is negligible provided a moderate band width is chosen for B.

A similar study was performed by B. Lang in [11]. The conclusions from that work were that the SBR toolbox could significantly accelerate the computations of the reduction to tridiagonal form compared to the approach in LAPACK. However, if the orthogonal transforms have to be accumulated, then the SBR routines were not competitive. Our interest in this analysis is motivated by the increase in the number of cores in general-purpose processors in the last years and the recent advances in hardware accelerators like graphics processors (GPUs). In particular, we aim at evaluating how the use of multiple cores in these architectures affects the performance of the codes in LAPACK and the SBR toolbox for tridiagonal reduction and back-transform. Note that, because of the efficient formulation and practical implementation of the $\texttt{MR}^3$ algorithm, the reduction to tridiagonal form and the back-transform are currently the most time-consuming stages in the solution of large symmetric eigenvalue problems.

The main contribution of this paper is a practical demonstration that the use of GPUs turns SBR into a competitive approach for both the reduction to tridiagonal form and the accumulation of transforms. This changes the main message that was presented in [11].

The rest of the paper is organized as follows. In Sections 2 and 3 we review, respectively, the routines in LAPACK and SBR for the reduction of a dense matrix to tridiagonal form. We also propose a modification of the code in the SBR toolbox to accelerate the initial reduction to banded form using a GPU.

Section 4 offers experimental results of the LAPACK and SBR codes on two Intel-based workstations and an NVIDIA Tesla C1060 GPU. Finally, Section 5 summarizes the conclusions of our study.

2 The LAPACK Routine SYTRD

Routine SYTRD[1] is based on the classical approach of reducing A to tridiagonal form by a series of Householder reflectors H_1, H_2, ..., H_{n-2}. Each Householder reflector is an orthogonal matrix of the form $H_j = I - \beta_j u_j u_j^T$, where $\beta_j \in \mathbb{R}$, $u_j \in \mathbb{R}^n$ with the first j entries zero, and I denotes here and in the following the identity matrix of appropriate order. The purpose of each reflector H_j is to annihilate the entries below the subdiagonal in the j-th column of $A_{j-1} = H_{j-1}^T \cdots H_2^T H_1^T A H_1 H_2 \cdots H_{j-1}$.

The routine proceeds as follows. Let b denote the algorithmic block size and assume that we have already computed the first $j-1$ columns/rows of T. Consider

$$H_{j-1}^T \cdots H_2^T H_1^T A H_1 H_2 \cdots H_{j-1} = \left(\begin{array}{c|c|c} T_{00} & T_{10}^T & 0 \\ \hline T_{10} & A_{11} & A_{21}^T \\ \hline 0 & A_{21} & A_{22} \end{array}\right),$$

where $T_{00} \in \mathbb{R}^{j-1 \times j-1}$ is in tridiagonal form and $A_{11} \in \mathbb{R}^{b \times b}$. With this partitioning, all entries of T_{10} are zero except for the one in its top right corner. Then, the following operations are computed during the current iteration of SYTRD:

1. The current panel $\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$ is reduced to tridiagonal form by a sequence of b orthogonal transforms $H_j, H_{j+1}, \ldots, H_{j+b-1}$. Simultaneously, two matrices $U, W \in \mathbb{R}^{(n-j-b+1) \times b}$ are built such that

$$H_{j+b-1}^T \cdots H_{j+1}^T H_j^T \left(\begin{array}{c|c|c} T_{00} & T_{10}^T & 0 \\ \hline T_{10} & A_{11} & A_{21}^T \\ \hline 0 & A_{21} & A_{22} \end{array}\right) H_j H_{j+1} \cdots H_{j+b-1}$$
$$= \left(\begin{array}{c|c|c} T_{00} & T_{10}^T & 0 \\ \hline T_{10} & T_{11} & T_{21}^T \\ \hline 0 & T_{21} & A_{22} - UW^T - WU^T \end{array}\right),$$

   where T_{11} is tridiagonal and all entries of T_{21} are zero except for its top right corner.

2. The submatrix A_{22} is updated as $A_{22} := A_{22} - UW^T - WU^T$ where, in order to exploit the symmetry, only the lower (or the upper) triangular part of this matrix is updated.

[1] We omit the first letter, denoting the precision. For example, SYTRD refers to the LAPACK routines DSYTRD (double precision) and SSYTRD (single precision)

The simultaneous computation of U and W along with the reduction in Operation 1 is needed to determine the first column of the unreduced part, which defines the Householder reflector. While U simply contains the vectors $u_j, u_{j+1}, \ldots, u_{j+b-1}$ of the Householder reflectors $H_j, H_{j+1}, \ldots, H_{j+b-1}$, more work is needed to determine W. In fact, the bulk of the computation in Operation 1 lays in the formation of this matrix. For each reduced column in the panel, a new column of W is generated. This requires four panel-vector multiplications and one symmetric matrix-vector multiplication with the submatrix A_{22} as operand. The latter operation, computed with the BLAS-2 kernel SYMV, is the most expensive one, requiring roughly $2(n-j)^2 b$ flops. Operation 2 also requires $2(n-j)^2 b$ flops, but is entirely performed by the BLAS-3 kernel SYR2K for the symmetric rank-$2b$ update. The overall cost of performing the reduction $A \rightarrow T$ using routine SYTRD is therefore $4n^3/3$ flops provided that $b \ll n$.

Note that there is no need to construct the orthogonal factor $Q = H_1 H_2 \cdots H_{n-2}$ explicitly. Instead, the vectors u_j defining the Householder reflectors H_j are stored in the annihilated entries of A. Additional work-space is needed to store the scalars β_j, but this requires only $n-2$ entries and is thus negligible. If the eigenvectors are requested, the back-transform QX_T is computed by the LAPACK routine ORMTR in $2n^3$ flops without ever forming Q. Using the compact WY representation [6], this operation can be performed almost entirely in terms of calls to BLAS-3 kernels.

3 The SBR Toolbox

The SBR toolbox is a software package for symmetric band reduction via orthogonal transforms. The package includes routines for the reduction of dense symmetric matrices to banded form (SYRDB), and the reduction of banded matrices to narrower banded (SBRDB) or tridiagonal form (SBRDT). Accumulation of the orthogonal transforms and repacking routines for storage rearrangement are also provided in the toolbox.

In this section we describe the routines SYRDB and SBRDT which, invoked in that order, produce the same effect as the reduction of a dense matrix to tridiagonal form via LAPACK routine SYTRD. For the SBR routine SYRDB, we also describe how to off-load the bulk of the computations to the GPU.

3.1 Reduction to banded form

Assume that the first $j-1$ columns of the matrix A have already been reduced to banded form with bandwidth w. Let b denote the algorithmic block size, and assume for simplicity that $j+w+b-1 \leq n$ and $b \leq w$; see Figure 1. Then, during the current iteration of routine SYRDB, the next b columns of the banded matrix are obtained with the following sequence of *operations*:

1. Compute the QR factorization of $A_0 \in \mathbb{R}^{k \times b}$, $k = n - (j+w) + 1$:

$$A_0 = Q_0 R_0, \tag{1}$$

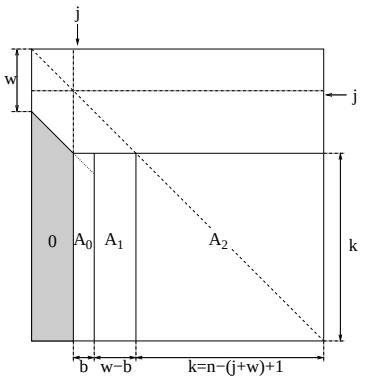

Figure 1: Partitioning of the matrix during one iteration of SYRDB for the reduction to banded form.

where $R_0 \in \mathbb{R}^{b\times b}$ is upper triangular and the orthogonal factor Q_0 is implicitly stored as a sequence of b Householder vectors using the annihilated entries of A_0 plus b entries of a vector of length $n-2$. The cost of this first operation is $2b^2(k-b/3)$ flops.

2. Construct the factors of the compact WY representation of the orthogonal matrix $Q_0 = I + WSW^T$, with $W \in \mathbb{R}^{k\times b}$ and $S \in \mathbb{R}^{k\times k}$ upper triangular. The cost of this operation is about kb^2 flops.

3. Apply the orthogonal matrix to $A_1 \in \mathbb{R}^{k\times w-b}$ from the left:

$$A_1 := Q_0^T A_1 = (I + WSW^T)^T A_1 = A_1 + W(S^T(W^T A_1)). \qquad (2)$$

By computing this operation in the order specified in the rightmost expression of (2), the cost becomes $4kb(w-b)$ flops. In case the bandwidth equals the block size ($w = b$), A_1 comprises no columns and, therefore, no computation is performed.

4. Apply the orthogonal matrix to $A_2 \in \mathbb{R}^{k\times k}$ from both the left and the right sides with $Y = WS^T$:

$$\begin{aligned} A_2 &:= Q_0^T A_2 Q_0 = (I + WY^T)^T A_2 (I + WY^T) && (3)\\ &= A_2 + YW^T A_2 + A_2 WY^T + YW^T A_2 WY^T. && (4) \end{aligned}$$

In particular, during this computation only the lower (or the upper) triangular part of A_2 is updated. In order to do so, (4) is computed as the

following sequence of calls to BLAS-3 kernels:

$$\begin{aligned}
(\text{SYMM}) \quad X_1 &:= A_2 W, && (5)\\
(\text{GEMM}) \quad X_2 &:= \frac{1}{2} X_1^T W, && (6)\\
(\text{GEMM}) \quad X_3 &:= X_1 + Y X_2, && (7)\\
(\text{SYR2K}) \quad A_2 &:= A_2 + X_3 Y^T + Y X_3^T. && (8)
\end{aligned}$$

The major cost of Operation 4 is in the computation of the symmetric matrix product (5) and the symmetric rank-$2b$ update (8), each with a cost of $2k^2b$ flops. On the other hand, the matrix products (6) and (7) only require $2kb^2$ flops each. Therefore, the overall cost of this operation is approximately $4k^2b + 4kb^2$. This is higher than the cost of the preceding Operations 1, 2, and 3, which require $O(kb^2)$, $O(kb^2)$, and $O(\max(kb^2, kbw))$ flops, respectively. In summary, provided that b and w are both small compared to n, the global cost of the reduction of a full matrix to banded form is $4n^3/3$ flops. Furthermore, the bulk of the computation is performed in terms of the BLAS-3 kernels SYMM and SYR2K in (5) and (8), so that high performance can be expected in case a tuned BLAS is used.

The orthogonal matrix $Q_B \in \mathbb{R}^{n \times n}$ for the reduction $Q_B^T A Q_B = B$, where $B \in \mathbb{R}^{n \times n}$ is the (symmetric) banded matrix, can be explicitly constructed by accumulating the involved Householder reflectors at a cost of $4n^3/3$ flops. Once again, the compact WY representation helps in casting this computation almost entirely in terms of calls to BLAS-3. The SBR toolbox implements this functionality in routine SYGTR.

3.2 Reduction to banded form on the GPU

Recent work on the implementation of BLAS and the major factorization routines for the solution of linear systems [2, 3, 13] has demonstrated the potential of GPUs to yield high performance on dense linear algebra operations which can be cast in terms of matrix-matrix products. In this subsection we describe how to exploit the GPU in the reduction of a matrix to banded form, orchestrating the computations carefully to reduce the number of data transfers between the host and the GPU.

During the reduction to banded form, Operation 4 is a natural candidate for being computed on the GPU, while, due to the kernels involved in Operations 1 and 2 (mainly narrow matrix-vector products), these computations are better suited for the CPU. Operation 3 can be performed either on the CPU or the GPU but, in general, $w - b$ will be small so that this computation is likely better suited for the CPU. Now, assume that the entire matrix resides in the GPU memory initially. We can then proceed to compute the reduced form by repeating the following three steps for each column block:

1. Transfer A_0 and A_1 back from GPU memory to main memory. Perform Operations 1, 2, and 3 on the CPU.

2. Transfer W and Y from main memory to the GPU.

3. Compute Operation 4 on the GPU.

Proceeding in this manner, upon completion of the algorithm most of the banded matrix and the Householder reflectors are available in the main memory. Specifically, only the diagonal $b \times b$ blocks in A remain to be transferred to the main memory.

The construction of Q_B that produces the reduction to banded form can also be easily done in the GPU, as this computation is basically composed of calls to BLAS-3 kernels.

3.3 Reduction to tridiagonal form

Routine SBRDT in the SBR toolbox is responsible for reducing the banded matrix B to tridiagonal form by means of Householder reflectors. Let Q_T denote the orthogonal transforms which yield this reduction, that is $Q_T^T B Q_T = T$. On exit, the routine returns the tridiagonal matrix T and, upon request, accumulates these transforms, forming the matrix $Q = Q_B Q_T \in \mathbb{R}^{n \times n}$ so that $Q^T A Q = Q_T^T (Q_B^T A Q_B) Q_T = Q_T^T B Q_T = T$.

The matrix T is constructed in routine SBRDT one column at the time: at each iteration the elements below the first subdiagonal of the current column are annihilated using a Householder reflector; the reflector is then applied to both sides of the matrix, resulting in a bulge which has to be chased down along the band. The computation is cast in terms of BLAS-2 operations at best (SYMV and SYR2 for two-sided updates, and GEMV and GER for one-sided updates) and the total cost is $6n^2w + 8nw^2$ flops.

If the eigenvectors are desired, then the orthogonal matrix Q_B produced in the first step (reduction from full to banded form) needs to be updated with the orthogonal transforms computed during the reduction from banded to tridiagonal form building $Q_B Q_T$. This accumulation requires $O(n^3)$ flops and can be reformulated almost entirely in terms of calls to BLAS-3 kernels, even though this reformulation is less trivial than for the first step [5]. Furthermore, the matrix $Q = Q_B Q_T$ still needs to be applied as part of the back-transform stage, to obtain $X := QX_T$, adding $2n^3$ flops to the cost of building the matrix containing the eigenvectors of A.

We do not propose to off-load the reduction of the banded matrix to tridiagonal form on the GPU as this is a fine-grained computation which does not lend itself to an easy implementation on this architecture. However, the accumulation of the orthogonal factor produced by this step and the back-transform merely require operations alike the matrix-matrix product, which can be efficiently computed on the GPU.

4 Experimental Results

The experimental results are reported for two target platforms representative of recent multithreaded architectures:

- NEHA: A workstation with two Intel Xeon QuadCore CPUs (E5520, *Nehalem*) at 2.27 GHz, with 8 MB L3 cache, 24 GB DDR3 RAM and theoretical peak performance of 145.3 GFLOPS in single precision using the 8 cores (1 GFLOPS = 10^9 flops/second).
  Attached to a PCI-Express Gen2 interface is a NVIDIA Tesla C1060 GPU, with 240 single-precision processor cores running at 1.3 GHz, 4 GB GDDR3 RAM, and featuring a theoretical peak performance of 933 GFLOPS in single precision.

- DUNN: A workstation with four Intel Xeon six-core CPUs (7460, *Dunnington*) at 2.66 GHz, 16 MB L3 cache, 256 GB DDR2 RAM, and theoretical peak performance of 510.7 GFLOPS in single precision using the 24 cores.

GotoBLAS2 version 1.10 and 1.13 were employed for all computations performed on NEHA and DUNN, respectively. NVIDIA CUBLAS 2.3 built on top of the CUDA application programming interface 2.3 together with NVIDIA driver 190.18 were used in our tests on the GPU. Single-precision arithmetic was employed in all experiments, though double precision is the standard in eigenvalue computations. An experimental analysis of the analogous double precision routines offered a similar balance between the benefits of the LAPACK routine(s) versus the SBR two-step alternative on the CPU. On the other hand, current GPUs from NVIDIA are not competitive in double precision, partly due to the much smaller number of double-precision cores and to the lack of an optimized implementation of CUBLAS.

Evaluating the performance of the routines for the reduction to tridiagonal form is an elaborate task, due to the large number of factors that have an influence on it. Among them, we will focus on block size, bandwidth for the SBR toolbox, and number of cores. In the experiments we aim at determining the optimal configuration of these parameters, before proceeding to show a full comparison of the two approaches. For simplicity, we report results for four problem sizes: 2048, 6144, 10240 and 24576, and the best bandwidths w and block sizes b detected in each case; a complete experiment was carried out for many other values.

4.1 Building BLAS-2 and BLAS-3 kernels

We start by analyzing the performance of the kernels involved in the reduction to condensed forms (tridiagonal and banded for the LAPACK and SBR approaches, respectively). For the LAPACK routine SYTRD, these are the symmetric matrix-vector product (SYMV) and the symmetric rank-$2k$ update (SYR2K). For the SBR routine SYRDB, the kernels comprise the symmetric matrix-matrix product (SYMM) and SYR2K. Table 1 reports results on 1, 4 and 8 cores of the Intel Xeon

Nehalem processors in NEHA. For the BLAS-3 kernels (SYMM and SYR2K), we also report their performance on the single GPU in this platform using the implementation in CUBLAS as well as our own implementations, which cast most computations in terms of the general matrix-matrix product [10] (column labeled as "Our BLAS"). Table 2 collects the results of the analogous experiment using 1, 4, 8, 16 and 24 cores of the Intel Xeon *Dunnington* processors in DUNN. The matrix dimensions of SYR2K and SYMM are chosen so that they match the shape of the blocks encountered during the reduction to condensed forms (n is the problem size, while k plays the role of the block size b for SYTRD and that of the bandwidth w for SYRDB). For reference, we also include the performance of the general matrix-vector product (GEMV) and matrix-matrix product (GEMM) kernels.

The performance of SYMV increases with the number of cores and is significantly higher than that of GEMV. When the problem size is $n = 2048$, the matrix fits into the L3 caches of NEHA (8 MB) and DUNN (16 MB), which explains the much higher GFLOPS rate of SYMV. The same does not hold for GEMV as all the matrix needs to be stored and not just half of it (lower or upper triangle).

The two BLAS-3 kernels present different performance behavior depending on the system: both routines increase their GFLOPS rates with the number of cores on NEHA. On DUNN, kernel SYR2K exhibits a performance line much similar to that of NEHA. On the other hand, the performance of kernel SYMM scales with the number of cores on DUNN when the problem size is large relative to it; otherwise, GotoBLAS2 simply reverts to using a single thread in the execution (a clear example of this is the GFLOPS rates attained by this routine for n=10240 and k=32 or 64 on 24 cores, which are identical to those attained using a single core.) This behavior plays a key role in the global performance of the whole reduction process.

The results also illustrate the performance delivered by the GPU for most BLAS-3 kernels, higher than that offered by any of the CPUs evaluated. Although our own implementations of the symmetric BLAS-3 kernels for the GPU deliver a higher GFLOPS rate than those from CUBLAS, they are still quite below the performance of the matrix-matrix product kernel in CUBLAS. The improvement in the performance of the SYMM kernel is of particular relevance for the reduction to tridiagonal form using the SBR routines.

4.2 The LAPACK approach

We next analyze the gains of a multi-core execution of the LAPACK routines SYTRD and, in case QX_T is required, ORMTR. Tables 3 and 4 report the execution time (in seconds) for different values of problem size, block size b and number of cores, on the two platforms.

Consider first the routine that performs the reduction from full to tridiagonal form, SYTRD. Obviously, the block size does not play a role in the performance of it. Increasing the number of cores for the execution yields a reduction in the execution time on NEHA but with moderate speed-up; for example, 3x and 4.2x speed-ups are attained for the largest problem sizes using respectively 4 and 8

	SYMV. $y := Ax + y$ $A \in \mathbb{R}^{n\times n}$ symmetric, $x, y \in \mathbb{R}^n$			GEMV. $y := Ax + y$ $A \in \mathbb{R}^{n\times n}$, $x, y \in \mathbb{R}^n$		
n	1 core	4 cores	8 cores	1 core	4 cores	8 cores
2048	8.26	34.5	60.0	5.15	8.47	8.31
6144	7.80	17.7	21.6	5.07	9.13	10.7
10240	6.69	18.3	22.1	4.70	9.26	11.2
24576	5.75	16.0	21.0	3.16	8.45	10.8

		SYR2K. $C := AB^T + BA^T + C$ $A, B \in \mathbb{R}^{n\times k}$, $C \in \mathbb{R}^{n\times n}$ symmetric					GEMM. $C := AB^T + C$ $A, B \in \mathbb{R}^{n\times k}$, $C \in \mathbb{R}^{n\times n}$			
n	k	1 core	4 cores	8 cores	CUBLAS	Our BLAS	1 core	4 cores	8 cores	CUBLAS
2048	32	14.0	55.4	91.0	53.2	53.2	15.0	58.6	116.4	157.2
	64	15.5	62.5	116.3	74.4	159.2	16.7	65.9	129.3	185.5
	96	16.5	65.3	122.2	78.0	162.9	17.3	68.4	135.9	192.2
6144	32	13.6	50.3	89.6	55.9	56.0	15.0	59.6	106.9	161.0
	64	15.7	59.8	112.3	78.4	124.2	16.7	66.6	123.1	185.0
	96	16.8	64.4	122.6	81.8	126.3	17.4	69.2	137.8	195.1
10240	32	13.8	51.2	83.9	56.4	56.4	15.0	59.6	116.5	159.3
	64	15.8	60.9	113.8	79.2	114.2	16.7	66.7	130.6	182.2
	96	16.9	65.3	123.7	78.1	116.7	17.5	69.4	137.7	187.3
24576	32	13.9	51.0	89.9	57.4	53.4	14.7	58.3	108.9	156.0
	64	16.0	60.9	113.6	79.1	116.9	16.6	64.6	129.9	186.2
	96	16.5	65.1	123.9	83.4	112.2	17.3	68.8	137.5	189.9

		SYMM. $C := AB + C$ $A \in \mathbb{R}^{n\times n}$ symmetric, $B, C \in \mathbb{R}^{n\times k}$					GEMM. $C := AB + C$ $A \in \mathbb{R}^{n\times n}$, $B, C \in \mathbb{R}^{n\times k}$			
n	k	1 core	4 cores	8 cores	CUBLAS	Our BLAS	1 core	4 cores	8 cores	CUBLAS
2048	32	12.2	29.7	46.9	89.7	106.5	15.0	59.9	99.2	177.5
	64	14.7	45.2	75.4	97.1	183.4	16.7	66.2	105.3	279.0
	96	15.7	49.3	80.4	97.9	189.4	17.4	68.7	133.5	290.1
6144	32	11.9	28.5	42.9	94.1	129.6	15.1	59.5	116.7	327.5
	64	14.5	43.9	71.8	99.1	188.4	16.8	66.5	132.2	339.3
	96	15.6	48.5	77.5	100.4	198.1	17.5	69.3	132.2	338.2
10240	32	11.1	25.3	39.4	76.0	113.5	15.0	59.5	116.7	321.9
	64	13.9	40.5	66.6	76.5	175.8	16.8	66.6	132.4	346.9
	96	15.2	45.3	73.4	77.5	180.0	17.4	69.4	135.8	348.0
20480	32	10.8	24.8	37.9	77.8	110.0	14.7	58.3	108.4	328.0
	64	13.5	39.3	63.3	66.8	176.7	16.6	66.0	131.3	344.9
	96	15.0	44.6	65.7	65.9	179.0	17.3	68.9	135.3	346.0

Table 1: Performance (in GFLOPS) of the BLAS kernels SYMV (top), SYR2K (middle) and SYMM (bottom) and the corresponding matrix-vector and matrix-matrix products (for reference) on NEHA. Peak performance for 1, 4 and 8 cores of this platform are 18.2, 72.6 and 145.3 GFLOPS, respectively.

	SYMV. $y := Ax + y$ $A \in \mathbb{R}^{n \times n}$ symmetric, $x, y \in \mathbb{R}^n$					GEMV. $y := Ax + y$ $A \in \mathbb{R}^{n \times n}$, $x, y \in \mathbb{R}^n$				
n	1 core	4 cores	8 cores	16 cores	24 cores	1 core	4 cores	8 cores	16 cores	24 cores
2048	6.33	30.7	54.6	70.1	63.8	0.93	10.3	5.88	10.4	11.1
6144	2.07	8.98	4.04	7.35	15.2	0.76	2.86	2.00	3.26	4.97
10240	2.09	7.60	3.93	6.34	8.98	0.74	2.72	2.00	3.24	4.30
24576	2.07	7.72	3.89	6.18	8.84	0.63	2.41	2.00	3.24	4.64

		SYR2K. $C := AB^T + BA^T + C$ $A,\ B \in \mathbb{R}^{n \times k}$, $C \in \mathbb{R}^{n \times n}$ symmetric					GEMM. $C := AB^T + C$ $A, B \in \mathbb{R}^{n \times k}$, $C \in \mathbb{R}^{n \times n}$				
n	k	1 core	4 cores	8 cores	16 cores	24 cores	1 core	4 cores	8 cores	16 cores	24 cores
2048	32	13.3	59.4	111.6	199.9	267.7	14.5	55.3	84.5	130.8	142.6
	64	16.4	67.7	126.5	224.9	310.0	17.9	68.0	129.0	215.4	231.1
	96	17.5	71.0	131.4	234.6	325.9	18.8	72.4	139.0	254.8	272.3
6144	32	10.4	45.5	31.09	56.6	117.2	12.7	48.3	42.2	65.1	78.4
	64	14.8	61.3	59.10	105.4	188.4	17.1	66.7	80.9	125.6	135.4
	96	16.4	66.8	83.25	145.6	243.9	18.2	71.8	113.8	182.7	192.5
10240	32	10.1	37.2	29.00	45.8	54.1	12.1	45.7	41.9	64.2	71.4
	64	14.6	56.7	55.92	89.0	102.4	16.7	65.5	80.7	125.7	134.7
	96	16.3	63.9	79.76	128.8	149.8	18.0	71.0	113.7	181.8	202.1
24576	32	9.49	34.9	28.73	43.7	52.2	11.2	42.8	42.0	64.2	76.1
	64	14.6	56.0	55.72	85.5	101.7	16.7	65.5	81.6	126.4	150.3
	96	16.3	63.7	79.39	125.1	147.9	18.1	71.5	115.6	185.0	220.8

		SYMM. $C := AB + C$ $A \in \mathbb{R}^{n \times n}$ symmetric, $B, C \in \mathbb{R}^{n \times k}$					GEMM. $C := AB + C$ $A \in \mathbb{R}^{n \times n}$, $B, C \in \mathbb{R}^{n \times k}$				
n	k	1 core	4 cores	8 cores	16 cores	24 cores	1 core	4 cores	8 cores	16 cores	24 cores
2048	32	12.5	44.5	67.5	12.2	12.2	14.4	55.1	84.9	130.0	138.6
	64	15.3	55.3	100.8	160.3	15.1	17.8	67.3	128.0	215.9	210.1
	96	16.6	60.3	106.4	147.4	220.8	18.8	71.4	137.2	253.7	269.3
6144	32	8.45	28.9	37.3	8.42	8.42	12.8	48.4	42.2	65.9	82.2
	64	11.8	42.4	64.2	89.8	11.8	17.2	66.8	81.0	126.2	157.0
	96	13.7	50.1	75.9	97.1	179.0	18.3	72.0	114.2	183.9	218.9
10240	32	8.23	28.7	37.0	8.20	8.20	12.1	45.9	41.9	64.3	71.7
	64	11.6	42.3	64.0	102.9	11.6	16.8	65.6	80.8	125.9	140.4
	96	13.5	50.2	76.9	114.8	194.3	18.1	71.1	113.7	182.1	203.6
24576	32	8.07	28.5	36.4	8.02	8.02	11.2	42.9	42.0	64.2	76.1
	64	11.5	42.1	63.3	93.4	11.4	16.8	65.6	81.6	126.2	149.8
	96	13.4	50.1	75.5	100.2	168.6	18.1	71.6	115.2	185.0	220.9

Table 2: Performance (in GFLOPS) of the BLAS kernels SYMV (top), SYR2K (middle) and SYMM (bottom) and the corresponding matrix-vector and matrix-matrix products (for reference) on DUNN. Peak performance for 1, 4, 8, 16 and 24 cores of this platform are 21.3, 85.2, 170.4, 340.8 and 510.7 GFLOPS, respectively.

n	SYTRD: Full→Tridiagonal				ORMTR: Compute QX_T			
	b	1 core	4 cores	8 cores	b	1 core	4 cores	8 cores
2048	32	1.11	0.34	0.23	128	1.22	0.41	0.27
	64	1.11	0.35	0.23	192	1.23	0.41	0.28
	96	1.11	0.36	0.25	256	1.27	0.41	0.27
6144	32	40.4	11.4	9.2	128	28.8	8.60	5.20
	64	39.9	11.3	8.4	192	28.2	8.32	5.09
	96	40.1	11.3	10.4	256	29.0	8.46	5.08
10240	32	156.3	52.4	40.6	128	128.5	36.6	21.1
	64	152.5	51.9	40.5	192	127.4	35.9	21.1
	96	152.6	52.1	40.9	256	126.4	35.3	21.9
24576	32	2522	812.5	590.3	128	1767	488.1	275.2
	64	2453	796.8	600.9	192	1732	471.5	272.0
	96	2444	795.0	582.4	256	1720	466.0	262.7

Table 3: Execution time (in seconds) for the LAPACK routine on NEHA.

n	SYTRD: Full→Tridiagonal						ORMTR: Compute QX_T					
	b	1 core	4 cores	8 cores	16 cores	24 cores	b	1 core	4 cores	8 cores	16 cores	24 cores
2048	32	1.68	0.51	0.37	0.40	0.50	128	1.28	0.56	0.41	0.33	0.32
	64	1.73	0.53	0.38	0.40	0.50	192	1.28	0.53	0.39	0.33	0.31
	96	1.81	0.55	0.39	0.42	0.51	256	1.30	0.53	0.39	0.33	0.31
6144	32	89.5	15.3	33.2	14.6	8.63	128	32.5	12.0	9.05	7.05	6.41
	64	88.9	15.6	33.0	14.8	7.44	192	30.8	11.2	8.27	6.54	5.70
	96	90.6	16.3	33.7	15.3	8.23	256	31.0	11.2	8.22	6.47	5.87
10240	32	429.6	107.0	190.4	111.5	74.2	128	144.4	47.8	35.0	25.7	23.4
	64	419.6	105.8	185.9	109.5	73.5	192	137.0	45.2	32.2	23.9	21.3
	96	423.5	107.6	186.4	110.1	74.1	256	133.4	43.7	30.6	22.6	20.1
24576	32	6037	1591	2780	1742	1255	128	1933	561.5	375.9	255.1	230.3
	64	5779	1529	2683	1681	1202	192	1798	518.3	336.4	229.0	182.4
	96	5790	1534	2665	1668	1194	256	1752	498.5	320.2	211.8	178.7

Table 4: Execution time (in seconds) for the LAPACK routine on DUNN.

cores. The behavior in DUNN is quite inconsistent: the use of 4 cores delivers superlinear gains in some cases (n=6144 yields a 5.81x factor) but, compared with these, 8 cores increase the execution time, and 16 cores roughly match it. 24 cores result in the best timings, though speed-ups are poor; e.g., 2.2x, 3.8x and only 5.7x for 8, 16 and 24 cores, respectively, with n=10240.

Applying the orthogonal transformations by the routine ORMTR requires less time and is, in general, more efficient than the reduction stage. This is to be expected, as most of the computations in ORMTR are performed by BLAS-3 kernels (GEMM), while only half of those in SYTRD are BLAS-3. Representative speed-ups of routine ORMTR in NEHA are 3.7x and 6.6x, attained respectively using 4 and 8 cores for the largest problem size. Efficiency in DUNN is still reasonable for 4 cores, but rapidly drops for 8 and more cores. Note also that this routine benefits from using larger block sizes (e.g., 192 and 256) than the

optimal for SYTRD.

4.3 The SBR approach

We next study the parallelism of the two-step SBR approach: reduction of a general matrix to banded form (SYRDB) and subsequent reduction to tridiagonal form (SBRDT). Also, we include in the analysis the routines that construct the orthogonal factor Q (SYGTR to build Q_B and SBRDT to accumulate $Q = Q_B Q_T$) and compute QX_T (GEMM) in the back-transform. Remember that while the computational cost of the first step is inversely proportional to the bandwidth w, the cost of the second step is directly proportional to it. In other words, a larger bandwidth requires a smaller amount of computation for the first step, transferring more flops to the second step.

Tables 5 and 6 display results for these experiments on the two platforms. For the discussion, consider the following five cases:

1. Reduction of a dense matrix to banded form (Step 1). On both platforms, the usage of a larger number of cores or the increase of the bandwidth results in a smaller execution time. The execution time on the GPU, on the other hand, is quite independent of w and outperforms the Intel-based architectures for all problem sizes except n=2048.

2. Reduction of banded matrix to tridiagonal form (Step 2). Using more than a single core yields no gain. As expected, a larger value of w results into a longer execution time of this step when using one core. For multi-threaded implementations, there is a reduction in the execution time as the bandwidth is increased. However, the execution time is still smaller for the sequential execution even for those values of w.

3. Building the orthogonal factor resulting from Case 1 (Step 1). On the Intel cores, the execution time and parallelism of routine SYGTR is quite similar to those of SYRDB discussed in Case 1. Compared with the results obtained on 8 cores of NEHA, the GPU in this platform accelerates the execution by a considerable factor, between 2.5x–3x.

4. Accumulating the orthogonal transforms corresponding to Case 2 (Step 2). By far, this is the most expensive operation of the five cases in the Intel cores, though it exhibits a certain degree of parallelism, which helps in reducing its weight on the overall process. The speed-up attained by the GPU for the larger problem dimensions is impressive.

5. Back-transform. The cost of this operation is comparable to that in Case 3. The best result is always attained on 8 cores and the GPU yields a notable acceleration.

Note that a study needs to take into account that the choice of bandwidth cannot be done independently for different cases. Therefore, we delay further comments on the data in the previous tables to the next subsection. There, we

n	w	1st step (SYRDB): Full→Band				2nd step (SBRDT): Band→Tridiagonal		
		1 core	4 cores	8 cores	GPU	1 core	4 cores	8 cores
2048	32	0.89	0.34	0.23	0.21	0.37	1.64	1.72
	64	0.81	0.28	0.19	0.20	0.45	1.08	1.03
	96	0.80	0.27	0.19	0.22	0.57	0.90	0.91
6144	32	23.6	8.3	5.2	2.78	3.48	14.93	17.1
	64	20.8	6.2	3.7	2.27	4.88	9.92	10.1
	96	19.9	5.9	3.6	2.29	5.42	8.23	8.91
10240	32	112.6	41.1	26.7	10.81	9.51	41.1	43.3
	64	95.9	29.6	18.7	9.72	11.7	27.5	26.3
	96	90.9	27.3	16.1	10.39	15.1	23.1	25.3
24576	32	1589	569.0	354.3	112.6	54.2	237.3	258.0
	64	1330	404.3	235.5	99.3	72.9	159.2	157.7
	96	1251	370.7	220.5	105.3	96.8	133.3	140.3

n	w	1st step (SYGTR): Build Q_B				2nd step (SBRDT): Accum. $Q = Q_B Q_T$			
		1 core	4 cores	8 cores	GPU	1 core	4 cores	8 cores	GPU
2048	32	0.81	0.33	0.25	0.07	2.31	1.28	1.38	0.76
	64	0.73	0.26	0.20	0.04	1.86	0.83	0.55	0.42
	96	0.70	0.25	0.19	0.03	1.61	0.54	0.36	0.26
6144	32	21.2	7.35	7.04	1.68	65.4	33.0	35.7	6.24
	64	19.0	5.83	3.86	1.77	51.8	22.1	14.5	3.09
	96	18.3	5.62	3.67	1.75	44.3	14.5	9.5	1.74
10240	32	97.5	32.5	22.7	6.81	291.0	150.8	163.4	32.4
	64	87.5	25.6	17.2	6.44	235.1	102.3	66.6	12.6
	96	84.1	24.7	16.5	5.61	203.1	67.2	43.9	6.27
24576	32	1399	456.8	310.7	94.6	4149	2166	2403	101.8
	64	1232	353.0	217.3	88.0	3390	1465	956.6	55.3
	96	1177	377.7	207.6	81.0	2898	969.9	638.8	30.9

	Back-transform (GEMM): Comp. QX_T			
n	2048	6144	10240	24576
8 cores	0.12	3.36	15.0	209.5
GPU	0.07	1.50	6.46	89.3

Table 5: Execution time (in seconds) for the SBR routines on NEHA.

		1st step (SYRDB): Full→Band					2nd step (SBRDT): Band→Tridiagonal		
n	w	1 core	4 cores	8 cores	16 cores	24 cores	1 core	4 cores	8 cores
2048	32	0.89	0.52	0.36	0.74	0.77	0.43	3.73	3.84
	64	0.80	0.41	0.29	0.28	0.65	0.54	2.66	2.62
	96	0.79	0.38	0.27	0.27	0.28	0.78	2.28	2.30
6144	32	30.1	10.7	8.75	20.2	19.7	3.98	33.7	34.8
	64	22.8	7.74	5.60	3.97	15.1	5.48	24.3	23.9
	96	21.2	6.85	4.80	3.57	27.8	8.85	21.1	21.2
10240	32	145.6	44.7	39.4	98.7	97.2	10.7	94.0	96.9
	64	106.5	31.5	23.3	15.8	68.9	18.5	67.8	66.6
	96	97.5	28.0	19.2	13.6	10.2	29.6	59.3	59.1
24576	32	2137	592.2	525.5	1400	1379	85.1	541.5	559.4
	64	1483	401.5	296.5	198.2	950.5	136.0	391.9	384.6
	96	1342	354.3	236.5	166.4	119.2	202.3	345.8	342.0

		1st step (SYGTR): Build Q_B					2nd step (SBRDT): Accum. $Q = Q_B Q_T$			
n	w	1 core	4 cores	8 cores	16 cores	24 cores	1 core	4 cores	8 cores	16 cores
2048	32	0.96	0.54	0.41	0.77	0.79	3.08	8.36	4.07	8.95
	64	0.81	0.41	0.31	0.28	0.60	2.23	3.36	2.49	6.53
	96	0.76	0.37	0.28	0.27	0.27	1.73	2.34	0.72	5.13
6144	32	29.8	11.0	10.3	20.9	20.9	96.5	155.1	100.8	149.9
	64	22.6	8.21	6.64	5.06	15.1	67.2	43.0	61.5	97.5
	96	20.8	7.33	5.65	4.49	3.89	53.3	28.9	14.4	75.8
10240	32	145.1	45.3	44.5	98.2	97.0	468.0	665.9	463.1	616.6
	64	106.6	33.8	27.8	20.4	69.3	323.3	155.2	284.5	382.5
	96	97.1	30.6	23.5	17.9	15.2	254.8	103.0	60.3	296.1
24576	32	2268	619.7	552.9	1469	1446	7194	8471	6724	7469
	64	1531	423.6	323.4	230.9	988.7	4826	1403	4113	4133
	96	1363	376.3	262.8	196.3	151.2	1160	988.2	793.0	3145

	Back-transform (GEMM): Comp. QX_T			
n	2048	6144	10240	24576
24 cores	0.09	1.48	6.56	80.8

Table 6: Execution time (in seconds) for the SBR routines on DUNN.

	Reduction to tridiagonal form				
	LAPACK		SBR		
n	NEHA	DUNN	NEHA	NEHA+GPU	DUNN
2048	0.23	0.37	0.6	0.58	0.79
6144	8.4	7.44	8.58	6.26	9.45
10240	40.5	73.5	30.4	20.32	34.3
24576	582.4	1194	308.4	166.8	321.5

	Reduction to tridiagonal form and back-transform				
	LAPACK		SBR		
n	NEHA	DUNN	NEHA	NEHA+GPU	DUNN
2048	0.50	0.68	1.77	1.12	3.14
6144	13.5	13.1	29.0	12.7	44.5
10240	61.6	93.6	116.8	43.8	151.3
24576	845.1	1371.7	1416.7	403.3	1486.3

Table 7: Comparison of the execution time (in seconds) for the the LAPACK and SBR routines on NEHA and DUNN.

elaborate on the optimal combination of the factors that determine the overall performance of this approach.

4.4 Comparing the two approaches

Even though the routines that tackle the symmetric eigenvalue problem are nicely structured as a sequence of steps, these are not independent. As a consequence, the tuning of parameters for each step in general cannot be done separately. For example, the bandwidth has to be kept constant through all the routines involved in the reduction. The block size, instead, can be adjusted for each routine. Additionally, on the multi-core processors, one may choose the degree of parallelism for each routine by fixing the number of threads employed for its execution. As an example, consider the reduction to tridiagonal form of a problem of size $n = 10240$ when performed on DUNN using the SBR routines. For bandwidths $w = 32$, 64 and 96, the best timings for the reduction to banded form using the corresponding SBR routine are 39.4, 15.8, and 10.2 seconds, using 8, 16 and 24 cores, respectively. The cost for the next stage, reduction from banded to tridiagonal form, is minimized when a single core is used, resulting in 10.7, 18.5 and 29.6 seconds for bandwidths 32, 64 and 96, respectively. Overall, the best combination, totaling 33.9 seconds, corresponds to bandwidth 64, using 16 cores for the first step and a single core for the second.

In Table 7, we collect results for an experimental comparison of the two approaches on the three architectures: NEHA, the GPU in this platform for all steps except the reduction from banded to tridiagonal form using the SBR routines (labeled as "NEHA+GPU"), and DUNN. For small and middle problem sizes, LAPACK is the fastest approach. For the largest dimensions, the SBR approach greatly benefits from the acceleration enabled by the GPU, and outperforms LAPACK both in the reduction and back-transform stages.

In the reduction stage, the GPU delivers speed-ups of 1.49x and 1.85x for the two largest problem sizes compared with the best options (SBR or LAPACK) on any of the two Intel-based architectures. When the back-transform is also required, the speedups for these problem sizes become 1.29x and 2.69x.

5 Concluding Remarks

We have evaluated the performance of existing codes for the reduction of a dense matrix to tridiagonal form and back-transform in the context of the symmetric eigenvalue problem. Two modern Intel 8-core and 24-core platforms were employed in this evaluation, representative of current high-end processors.

Our experimental results confirm that the two-stage approach proposed in the SBR toolbox (reduction from full to banded form in the first stage followed by a reduction from banded to tridiagonal form in a second stage) delivers a higher parallel scalability than the LAPACK-based alternative on general-purpose multi-core architectures. However, when the orthogonal factors that define the back-transform have to be constructed and applied in the last stage, this results in a computation time considerably larger than that for LAPACK.

The use of a hardware accelerator like a GPU changes the message: By off-loading the level-3 BLAS operations in the SBR codes to an NVIDIA 240-core GPU, remarkable speed-ups are attained to the point that the SBR toolbox becomes a competitive alternative to the standard LAPACK-based one. The reward did not come effortless, though. Specifically, the gains came from two improvements: 1) a reformulation of the CUBLAS symmetric rank-2k update and the symmetric matrix-matrix product, and 2) a careful modification of the SBR routines to exploit the hardware elements of the hybrid CPU-GPU architecture and to minimize the number of data transfers between the host and the device memory spaces.

Acknowledgments

Bientinesi and Petschow were supported by project 50225798 PARSEMUL from the *German Academic Exchange Service*, and by grant GSC 111 from the *German Research Association*.

Igual and Quintana-Ortí were supported by projects TIN2008-06570-C04-01 and FEDER, and the *Acciones Integradas* DE-2009-0038 from the *Ministerio de Innovación y Ciencia*.

References

[1] E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.

[2] S. Barrachina, M. Castillo, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí. Evaluation and tuning of the level 3 CUBLAS for graphics processors. In *Proceedings of the 10th IEEE Workshop on Parallel and Distributed Scientific and Engineering Computing, PDSEC 2008*, pages CD–ROM, 2008.

[3] S. Barrachina, M. Castillo, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí. Solving dense linear systems on graphics processors. In E. Luque, T. Margalef, and D. Benítez, editors, *Proceedings of the 14th International Euro-Par Conference*, Lecture Notes in Computer Science, 5168, pages 739–748. Springer, 2008.

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

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

[6] Christian Bischof and Charles Van Loan. The WY representation for products of Householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987.

[7] Inderjit S. Dhillon and Beresford N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications*, 387:1 – 28, 2004.

[8] J. Dongarra, S. J. Hammarling, and D. C. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. LAPACK Working Note 2, Technical Report MCS-TM-99, Argonne National Laboratory, Sept. 1987.

[9] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[10] Francisco D. Igual, Gregorio Quintana-Ortí, and Robert van de Geijn. Level-3 BLAS on a GPU: Picking the low hanging fruit. FLAME Working Note #37. DICC 2009-04-01, Universitat Jaume I. Dept. ICC, 2009.

[11] Bruno Lang. Efficient eigenvalue and singular value computations on shared memory machines. *Parallel Comput.*, 25(7):845–860, 1999.

[12] R. M. Martin. *Electronic Structure: Basic Theory and Practical Methods*. Cambridge University Press, Cambridge, UK, 2008.

[13] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.