Reduction to condensed forms for symmetric eigenvalue problems on multi-core architectures

P. Bientinesi[1], F.D. Igual[2], D. Kressner and E.S. Quintana-Orti[2]

Research Report No. 2009-13
March 2009

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

[1]AICES, RWTH Aachen University, 52074 Aachen, Germany
[2]Depto. de Ingenieria y Ciencia de Computadores, Universidad Jaume I, 12071 Castellón, Spain

Reduction to Condensed Forms for Symmetric Eigenvalue Problems on Multi-core Architectures

Paolo Bientinesi[1], Francisco D. Igual[2], Daniel Kressner[3], and Enrique S. Quintana-Ortí[2]

[1] AICES, RWTH Aachen University, 52074–Aachen, Germany; `pauldj@aices.rwth-aachen.de`.
[2] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain; {`figual,quintana`}`@icc.uji.es`.
[3] Seminar für angewandte Mathematik, ETH Zürich, Switzerland; `kressner@math.ethz.ch`.

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. The target architecture is a current general purpose multi-core processor, where parallelism is extracted using a tuned multi-threaded implementation of BLAS. Also, in response to the advances of hardware accelerators, we modify the code in SBR to accelerate the computation by off-loading a significant part of the operations to a graphics processor (GPU). Our results on a system with two Intel QuadCore processors and a Tesla C1060 GPU illustrate the performance and scalability delivered by these architectures.

1 Introduction

We consider the solution of the *symmetric eigenvalue problem*

$$AX = X\Lambda, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is a 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 jth column of the orthogonal matrix $X \in \mathbb{R}^{n\times n}$ is the 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. Applications leading to eigenvalue problems are ubiquitous in science and engineering, and large-scale eigenproblems of the form (1) appear, e.g., in computational quantum chemistry, finite element modeling, and multivariate statistics, to name only a few. Particularly challenging eigenvalue problems arise in density functional theory, where a significant fraction of the eigenvalues and eigenvectors of a potentially large symmetric matrix need to be computed [10].

Efficient algorithms for the solution of (1) usually consist of the following three stages: the original matrix A is first reduced to a (symmetric) tridiagonal matrix $T \in \mathbb{R}^{n \times n}$ by a sequence of orthogonal similarity transforms: $Q^T AQ = T$, where $Q \in \mathbb{R}^{n \times n}$ is the matrix representing the accumulation of these orthogonal transforms. The $\texttt{MR}^3$ algorithm [7] or the (parallel) $\texttt{PMR}^3$ [4] algorithm is then applied to the tridiagonal matrix T to accurately compute its eigenvalues and, optionally, the associated eigenvectors. Finally, when the eigenvectors of A are desired, we need to apply a back-transform 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] are not competitive as they require $O(n^3)$ flops for the second stage.)

In this paper we re-evaluate the performance of the codes in LAPACK [1] and the *Successive Band Reduction* (SBR) toolbox [5, 6] for the reduction of a full symmetric matrix A to tridiagonal form. The LAPACK routine SYTRD employs a simple algorithm based on Householder reflectors [9], enhanced with WY representations [8], to reduce A directly to tridiagonal form. Unfortunately, only half of its operations can be performed in terms of calls to BLAS-3 kernels, resulting in a poor use of the memory hierarchy. To overcome such drawback of SYTRD, the SBR toolbox first reduces A to an intermediate band matrix B, and subsequently reduces 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 becomes negligible provided a moderate band width is chosen for B.

Our interest in this study is motivated by the increase in the number of cores in general-purpose processors and by the recent advances in more specific hardware accelerators such as graphics processors (GPUs). In particular, we aim at evaluating how the presence of multiple cores in these new architectures affects the performance of the codes in LAPACK and SBR for tridiagonal reduction. 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 the symmetric eigenvalue problem.

The rest of the paper is organized as follows. In Sections 2 and 3 we review the routines in LAPACK and SBR for the reduction of a dense matrix to tridiagonal form. In the latter section, we also propose a modification of the code in SBR to accelerate the initial reduction to band form using a GPU. Section 4 offers experimental results of these implementations on a workstation with two Intel Xeon QuadCore processors (8 cores) and an NVIDIA Tesla C1060 GPU. Finally, Section 5 summarizes the conclusions of our study.

2 The LAPACK Routine SYTRD

The LAPACK routine SYTRD 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 hereafter the square identity matrix of the appropriate order. The purpose of each H_j is to annihilate the entries below the subdiagonal in the jth 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 SYTRD proceeds as follows. Let b denote the algorithmic block size and assume that the we have already computed the first $j-1$ columns/rows of T. Consider the following partitioning

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

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 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 in tridiagonal form 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) half of this matrix is updated.

The simultaneous computation of U and W along with the reduction in Step 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 Step 1 lays in the formation of W. 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 SYMV kernel, is the most expensive one, requiring roughly $2(n-j)^2b$ flops. Step 2 also requires $2(n-j)^2b$ flops, but is entirely performed by the BLAS-3 kernel SYR2K for the symmetric rank-$2b$ update. The overall cost of 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 workspace is needed to store the scalars β_j, but this requires only $O(n)$ entries and is thus negligible. If the eigenvectors are requested, the back-transform QX_T is computed using this data in $2n^3$ flops. With the compact WY representation [9] 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. SBR includes routines for the reduction of dense

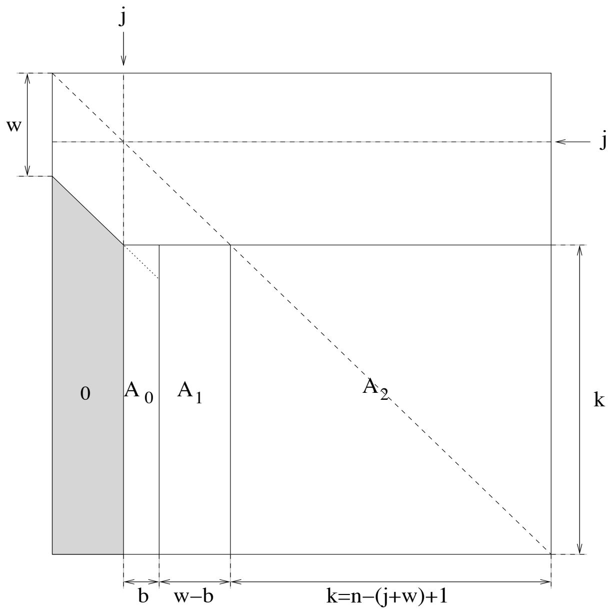

Fig. 1. Partitioning of the matrix during one iteration of routine SYRDB for the reduction to band form.

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 result as the reduction of a dense matrix to tridiagonal form using the 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 band form

Suppose that the first $j-1$ columns of the matrix A have already been reduced to band 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 band matrix are obtained as follows:

1. Compute the QR factorization of $A_0 \in \mathbb{R}^{k\times b}$, $k = n - (j + b) + 1$:
$$A_0 = Q_0 R_0, \tag{2}$$
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. The cost of this first step is $2b^2(k - b/3)$ flops.
2. Construct the factors of the compact WY representation [9] of the orthogonal matrix $Q_0 = I_k + WTW^T$, with $W \in \mathbb{R}^{k\times b}$ and $T \in \mathbb{R}^{k\times k}$ upper triangular. The cost of this step is about kb^2 flops.
3. Apply the orthogonal matrix to $A_1 \in \mathbb{R}^{k\times w-b}$ from the left:
$$A_1 := Q_0^T A_1 = (I_k + WTW^T)^T A_1 = A_1 + W((A_1^T W)T^T)^T. \tag{3}$$
By performing the operations in the order specified in the rightmost expression of (3), the cost of this step becomes $4kb(w - b)$ flops. In case the bandwidth equals the block size ($w = b$), A_1 comprises no columns and, therefore, no operation is performed in this step.
4. Apply the orthogonal matrix to $A_2 \in \mathbb{R}^{k\times k}$ from both the left and right:
$$\begin{aligned} A_2 := Q_0^T A_2 Q_0 &= (I_k + WY^T)^T A_2 (I + WY^T) && (4)\\ &= A_2 + YW^T A_2 + A_2 WY^T + YW^T A_2 WY^T, && (5)\end{aligned}$$
with $Y = WT$. In particular, during this step only the lower (or the upper) triangular part of A_2 is updated. In order to do so, (5) is computed as the following sequence of (BLAS) operations:
$$\begin{aligned} &(\textsc{symm}) && X_1 := A_2 W, && (6)\\ &(\textsc{gemm}) && X_2 := \frac{1}{2} X_1^T W, && (7)\\ &(\textsc{gemm}) && X_3 := X_1 + Y X_2, && (8)\\ &(\textsc{syr2k}) && A_2 := A_2 + X_3 Y^T + Y X_3^T. && (9)\end{aligned}$$
The major cost of this step is in the computation of the symmetric matrix product (6) and the symmetric rank-$2k$ update (9), each with a cost of $2k^2b$ flops. On the other hand, the matrix products (7) and (8) only require $2kb^2$ flops each. Therefore, the overall cost of Step 4 is approximately $4k^2b + 4kb^2$, which is higher than the cost of the remaining Steps 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 with n, the global cost of the reduction of a full matrix to band form is $4n^3/3$ flops. Furthermore, the bulk of the computation is performed in terms of BLAS-3 operations SYMM and SYR2K in (6) and (9), so that high performance can be expected from routine SYRDB in case a tuned implementation of 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) band matrix can be explicitly constructed by accumulating the involved Householder reflectors at a cost of $4n^3/3$ flops. Once again, compact WY representations help in casting this computation almost entirely in terms of calls to BLAS-3 kernels.

3.2 Reduction to band form on the GPU

Recent work on the implementation of BLAS and the major factorization routines for the solution of linear systems [2, 3, 11] 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 full matrix to band form, orchestrating the computations carefully to reduce the number of data transfers between the memories of the host and the GPU.

During the reduction to band form, the operations in Step 4 are natural candidates for being computed on the GPU while, due to the kernels involved in Steps 1 and 2 (mainly narrow matrix-vector products), these operations are better suited for the CPU. The operations in Step 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 on 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. Compute Steps 1, 2, and 3 on the CPU.
2. Transfer W and Y from main memory to the GPU.
3. Compute Step 4 on the GPU.

Proceeding in this manner, at the completion of the algorithm most of the band 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.

3.3 Reduction to tridiagonal form

The routine SBRDT in SBR is responsible for reducing the banded matrix B to tridiagonal form by means of Householder reflectors. Let Q_T denote the orthogonal transforms which produce 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 those 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, and the resulting bulge is 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 stage (reduction from full to banded form) needs to be updated by the orthogonal transforms computed during the reduction from banded to tridiagonal form (i.e., Q_T). This update requires $O(n^3)$ flops and can be reformulated almost entirely in terms of calls to level-3 BLAS kernels, even though this reformulation is less trivial than for the first stage [6]. However, the matrix $Q = Q_B Q_T$ still need to be applied as part of the back-transform step, adding $2n^3$ flops to the cost of building the matrix containing the eigenvectors.

We do not propose to off-load the reduction of the band matrix to tridiagonal form on the GPU as this is a fine-grained computation which do not lend itself to an easy implementation on this architecture.

4 Experimental Results

The target platform used in the experiments is a workstation with two Intel Xeon QuadCore CPUs consisting of 8 cores running at 2.33 GHz, with 8 GB DDR2 RAM, and offering a theoretical peak performance of 37.28/18.64 GFLOPS in single/double precision (1 GFLOPS = 10^9 flops/second). The workstation is also equipped with an NVIDIA Tesla C1060 board with 240 single-precision and 30 double precision streaming processor cores running at 1.3 GHz, 4 GB DDR3 RAM, and a theoretical peak performance of 933/78 GFLOPS in single/double precision. The Intel chipset E5410 and the Tesla board are connected via a PCI-Express Gen2 interface with a peak bandwidth of 48 Gbits/second. MKL 10.1 was employed for all computations performed on the Intel cores. NVIDIA

CUBLAS (version 2.0) built on top of the CUDA application programming interface (version 2.0) together with NVIDIA driver (177.73) were used in our tests. Single precision was employed in all experiments, though double precision is the standard in eigenvalue computations. We believe that an experimental analysis of the analogous double precision routines would offer a similar balance between the benefits of the LAPACK routine versus the SBR toolbox two-stage alternative on the CPU. The GPU is not competitive in double precision, partly due to the much smaller number of cores dedicated to this and to the lack of an optimized implementation of BLAS. Investigating the possibility of refinement from single to double precision in the context of eigenvalue problems, so that this architecture becomes a practical alternative, is among our future work.

When reporting the rate of computation, we consider the cost of the reduction to tridiagonal form (either using LAPACK SYTRD or SBR SYRDB+SBRDB) to be $4n^3/3$ flops for square matrices of order n. Note that, depending on w, this count may be considerably smaller than the actual number of flops performed by SYRDB+SBRDB. We do not build the orthogonal factors/compute the eigenvectors in our experiments. The GFLOPS rate is computed as $4n^3/3$ divided by $t \times 10^{-9}$, where t equals the elapsed time in seconds. The cost of all data transfers between main memory and GPU memory is included in the timings.

Our first experiment evaluates the performance of the major BLAS-3 kernels involved in the reduction to tridiagonal form using the LAPACK and SBR routines: SYMM and SYR2K. For reference, we also evaluate the performance of the general matrix-product kernel, GEMM. Tables 1 and 2 report results on 1, 4 and 8 cores of the Intel Xeon processors and 1 GPU. For the latter architecture, we employ the kernels in CUBLAS and also our own implementations (column labeled as "Own CUBLAS"). The matrix dimensions of SYMM and SYR2K are chosen so that they match the structure of the blocks encountered during the reduction. The matrix dimensions of GEMM mimic the sizes of the operands in SYMM or SYR2K. The results show the higher performance yield by the GPU for most matrix operations and the benefits of using block sizes that are integer multiple of 32 in this hardware. Although our own implementations of the symmetric kernels in CUBLAS deliver a higher GFLOPS rate than that of NVIDIA BLAS, they are still quite below the performance of the matrix-matrix product kernel. In particular, the results in Table 2 illustrate that, depending on the value of k, on the GPU it may be more efficient to call twice the GEMM kernel (updating the whole matrix in Step 4 of routine

SYMM. $C := AB + C$; $A \in \mathbb{R}^{m\times m}$ symmetric, $B, C \in \mathbb{R}^{m\times n}$						
m	n	1 Core	4 Cores	8 Cores	CUBLAS	Own CUBLAS
2048	24	6.6	6.5	8.2	68.6	67.1
	32	8.0	8.8	6.9	89.7	106.5
	64	10.9	16.0	13.8	97.1	183.4
6144	24	6.6	6.0	4.3	71,6	73.1
	32	7.8	7.7	6.0	94.1	129.6
	64	10.7	14.1	11.6	99.1	188.4
10240	24	6.6	5.9	3.7	57.4	68.1
	32	7.8	7.8	6.1	76.0	113.5
	64	10.7	14.2	11.7	76.5	175.8

GEMM. $C := AB + C$; $A \in \mathbb{R}^{m\times k}$, $B \in \mathbb{R}^{k\times n}$, $C \in \mathbb{R}^{m\times n}$						
$m = k$	n	1 Core	4 Cores	8 Cores	CUBLAS	
2048	24	5.6	18.1	25.1	101.0	
	32	6.8	23.5	32.7	177.5	
	64	9.8	34.2	51.8	279.0	
6144	24	5.9	21.1	30.9	134.9	
	32	7.0	26.0	38.7	327.5	
	64	9.9	37.4	60.1	339.3	
10240	24	5.9	21.6	30.7	139.7	
	32	7.0	26.3	34.9	321.9	
	64	9.9	38.1	56.1	346.9	

Table 1. Performance of the BLAS kernel SYMM involved in the reduction to band form and the corresponding matrix product (for reference).

SYRDB) instead of the SYR2K one (updating only one of the triangles), even that this implies doubling the number flops. Besides, in case both the upper and the lower triangular parts of A_2 in 9 are updated, then one can replace the call to SYMM by a call to GEMM. These are strategies that we employ in our implementation of SYRDB for the GPU.

The second experiment compares the LAPACK and SBR codes for the reduction to tridiagonal form using 1, 4 and 8 cores of the CPU or the GPU plus one of the cores of the CPU. Figure 2 reports the GFLOPS for these alternatives. Only the results corresponding to the best block size (b) and bandwidth (w) are reported in the figure. Note that, as we are using the same flop count for the two approaches, a higher GFLOPS rate implies a smaller execution time. The performance behaviour of SYTRD is typical for a routine based on BLAS-2: when the matrix is too large to fit into the cache of the processor, the performance rapidly drops. The GFLOPS rate attained by the SBR does in the CPU is more consistent and shows a good scalability for four cores. However, when the number of cores is increased to 8, there is no performance gain. Finally, the SBR

SYR2K. $C := AB^T + BA^T + C$; $A,\ B \in \mathbb{R}^{n \times k}$, $C \in \mathbb{R}^{n \times n}$ symmetric						
n	k	1 Core	4 Cores	8 Cores	CUBLAS	Own CUBLAS
	24	10.7	24.2	36.0	36.1	36.8
2048	32	11.7	29.9	42.9	53.2	53.2
	64	13.6	40.6	57.8	74.4	159.2
	24	10.6	24.2	34.9	40.3	40.3
6144	32	11.9	29.9	43.2	55.9	56.0
	64	13.6	43.8	64.0	78.4	124.2
	24	10.0	20.6	24.5	41.2	41.4
10240	32	10.8	26.6	31.5	56.4	56.4
	64	13.2	43.0	56.5	79.2	114.2
GEMM. $C := AB^T + C$; $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{n \times k}$, $C \in \mathbb{R}^{m \times n}$						
$m = n$	k	1 Core	4 Cores	8 Cores	CUBLAS	
	24	14.7	44.0	86.5	70.7	
2048	32	15.6	50.4	95.3	157.2	
	64	16.8	59.4	112.9	185.5	
	24	15.0	27.7	27.9	71.4	
6144	32	15.9	36.6	36.9	161.0	
	64	17.0	59.3	73.1	185.0	
	24	15.0	27.3	27.7	71.9	
10240	32	15.8	36.0	36.8	159.3	
	64	16.9	58.2	73.3	182.2	

Table 2. Performance of the BLAS kernel SYR2K involved in the reduction to band form and the corresponding matrix product (for reference).

code modified to off-load the bulk of the computation to the GPU clearly outperforms all the executions on the general-purpose CPU. Comparing the SBR routines using 4 cores and the GPU, the speed-up observed for second when solving the larger problem size is 3.7x (18.53 GFLOPS with optimal $w = 96$ on the CPU vs. 68.39 GFLOPS with optimal $w = 32$ on the GPU). This in practice reduces the cost of the first stage in SBR to that of the second stage, as shown in Table 3. The results in that table also show that the acceleration attained by off-loading the computation of the first stage to the GPU is a factor of 12x for the largest problem size and $w = 32$.

5 Concluding Remarks

We have evaluated the performance of existing codes for the reduction of a full dense matrix to tridiagonal form, in the context of the symmetric eigenvalue problem. Our experimental results confirm that the two-stage approach proposed in the SBR toolbox delivers a higher parallel scala-

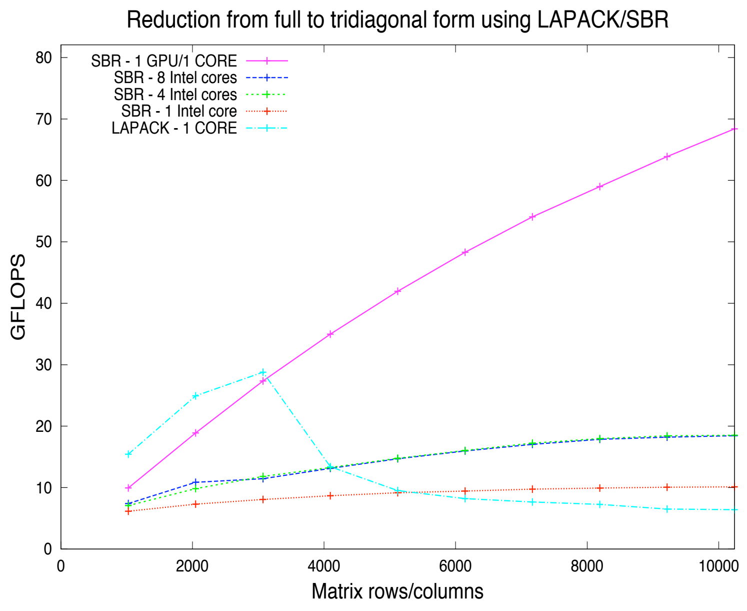

Fig. 2. Performance of the reduction of a dense matrix to tridiagonal form using the routines in LAPACK and the SBR toolbox on a Intel Xeon and a Tesla C1060.

bility than the LAPACK-based alternative. We have also modified the codes in SBR to compute the most-expensive operations during the reduction to band form on a GPU. To increase the performance we employ the highly tuned implementation of the matrix product kernel in CUDA to compute symmetric matrix products and symmetric rank-$2k$ updates. Although this increases the cost of the stage by 33%, we found in our experiments that this is clearly compensated by the higher performance of that particular kernel. In summary, the GPU variant reduces significantly the cost of this initial stage, making it comparable to that of the second stage (reduction of the band matrix to tridiagonal form).

The back-transform stage has not been included in this experimental study. On the one hand, it is only necessary when the eigenvectors are requested; on the other, it can be expressed in terms of efficient BLAS-3 kernels which we expect to deliver high performance and scalability on both a general-purpose processor or a hardware accelerator like a GPU. Future work will certainly consider this stage as well as the refinement of single-precision results to double-precision.

		1st stage: Full→ Band				2nd stage: Band→ Tridiagonal
n	w	1 Core	4 Cores	8 Cores	CUBLAS	1 core
2048	32	1.1	0.8	0.8	0.2	0.4
	96	0.9	0.5	0.5	0.2	0.8
6144	32	33.5	23.8	28.5	2.5	3.7
	96	25.3	11.6	11.7	2.7	7.5
10240	32	155.8	110.4	129.5	10.1	10.3
	96	116.6	51.2	51.6	10.6	25.6

Table 3. Execution time (in seconds) for the two-stage SBR routines.

Acknowledgments

The second and fourth authors were supported by projects CICYT TIN2005-09037-C02-02, TIN2008-06570-C04-01 and FEDER, and P1B-2007-19 of the *Fundación Caixa-Castellón/Bancaixa* and UJI.

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 on Parallel Processing*, 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. Christian H. Bischof, Bruno Lang, and Xiaobai Sun. Algorithm 807: The SBR Toolbox—software for successive band reduction. *ACM Trans. Math. Soft.*, 26(4):602–616, Dec. 2000.
6. Christian H. Bischof, Bruno Lang, and Xiaobai Sun. A framework for symmetric band reduction. *ACM Trans. Math. Soft.*, 26(4):581–601, Dec. 2000.
7. Inderjit S. Dhillon, Beresford N. Parlett, and Christof Vomel. The design and implementation of the MRRR algorithm. *ACM Trans. Math. Soft.*, 32(4):533–560, Dec. 2006.
8. Jack J. Dongarra, Sven J. Hammarling, and Danny 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. R. M. Martin. *Electronic Structure: Basic Tehory and Practical Methods*. Cambridge University Press, Cambridge, UK, 2008.
11. Vasily Volkov and James Demmel. LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley, May 2008.

Research Reports

No.	Authors/Title
09-13	*P. Bientinesi, F.D. Igual, D. Kressner, E.S. Quintana-Orti* Reduction to condensed forms for symmetric eigenvalue problems on multi-core architectures
09-12	*M. Stadelmann* Matrixfunktionen - Analyse und Implementierung
09-11	*G. Widmer* An efficient sparse finite element solver for the radiative transfer equation
09-10	*P. Benner, D. Kressner, V. Sima, A. Varga* Die SLICOT-Toolboxen für Matlab
09-09	*H. Heumann, R. Hiptmair* A semi-Lagrangian method for convection of differential forms
09-08	*M. Bieri* A sparse composite collocation finite element method for elliptic sPDEs
09-07	*M. Bieri, R. Andreev, C. Schwab* Sparse tensor discretization of elliptic sPDEs
09-06	*A. Moiola* Approximation properties of plane wave spaces and application to the analysis of the plane wave discontinuous Galerkin method
09-05	*D. Kressner* A block Newton method for nonlinear eigenvalue problems
09-04	*R. Hiptmair, J. Li, J. Zou* Convergence analysis of Finite Element Methods for H(curl;Ω)-elliptic interface problems
09-03	*A. Chernov, T. von Petersdorff, C. Schwab* Exponential convergence of *hp* quadrature for integral operators with Gevrey kernels
09-02	*A. Cohen, R. DeVore, C. Schwab* Convergence rates of best N-term Galerkin approximations for a class of elliptic sPDEs
09-01	*B. Adhikari, R. Alam, D. Kressner* Structured eigenvalue condition numbers and linearizations for matrix polynomials
08-32	*R. Sperb* Optimal bounds in reaction diffusion problems with variable diffusion coefficient