2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops

New Algorithm for Computing Eigenvectors of the Symmetric Eigenvalue Problem

Azzam Haidar[1], Piotr Luszczek[1], and Jack Dongarra[1,2,3]

[1]University of Tennessee Knoxville
[2]Oak Ridge National Laboratory
[3]University of Manchester

Abstract

We describe a design and implementation of a multi-stage algorithm for computing eigenvectors of a dense symmetric matrix. We show that reformulating the existing algorithms is beneficial in terms of performance even if that doubles the computational complexity. Through detailed analysis, we show that the effect of the increase in the asymptotic operation count may be compensated by a much improved performance rate. Our performance results indicate that using our approach achieves very good speedup and scalability even when directly compared with the existing state-of-the-art software.

Index Terms

symmetric eigenvalue problem; eigenvectors; dynamic runtime scheduling

1. Introduction and Motivation

Two-stage algorim was applied in the context of the symmetric eigenvalue problem and achieved significant performance gains [1], [2], [3], [4], [5]. However, there was no attempt made to compute the eigenvectors using the two-stage approach. This work is, to our knowledge, the first attempt to provide this missing piece of functionality by addressing a variety of algorithmic challenges as well as by engineering an implementation that outperforms any of the widely available software.

The two-stage method is effective method, for obtaining eigenvalues of a symmetric (or hermitian) matrix, in comparison to the classic one-stage algorithm due to the lack of increase in computational complexity (for the highest order term): $O(4/3n^3)$, and because it recasts memory-bound operations as compute-bound kernels. Thus, the challenge is shifted from dealing with scarcity of bandwidth to obtaining a device with higher computational power – clearly, a desirable situation if one considers the current technology trends [6]. This recasting alone produces appreciative increase in performance [1]. Further tuning contributes even more to the performance gains by reordering and merging the computational steps [3]. The drawback, that is commonly raised against the two-stage method, is the introduction of an additional similarity transformation matrix, that doubles the computational complexity of the back-transformation procedure. Worse still, the back-transformation is performed with computate-bound routines that run at high percentage of the peak performance. They are essentially based on the dense matrix-matrix multiplies. Seemingly then, the use of the two-stage algorithm is discouraged due to these challenges. Overcoming them is the motivation of the research presented here.

2. Related Work

Solving the symmetric eigenvalue problem continues to be an active research field. Recently, many researchers took interest in this area and have developed various strategies with a number of efficient software implementations. LAPACK [7] and ScaLAPACK [8] are considered robust pieces of open source software for shared- and distributed-memory systems, respectively. Hardware vendors provide well-tuned versions o LAPACK and ScaLAPACK. Recent work has concentrated on accelerating the individual components of the solvers, and, in particular, the reduction to the tridiagonal form, which is the most time consuming phase. A new type of algorithm that departs from the standard one-stage reduction algorithms was introduced. The idea behind this recent technique is to split the reduction phase into two or more stages, recasting the expensive memory-bound operations that occur during the panel factorization into compute-bound operations to benefit from the gains in peak performance of modern processors. One of the first uses of a two-stage reduction occurred in the context of out-of-core solvers for generalized symmetric eigenvalue problems [9]. Then, a multi-stage method was used to reduce a matrix to the tridiagonal, bidiagonal and Hessenberg forms [10]. Consequently, a framework called Successive Band Reduction (SBR) was developed [11], [12] to provide the benefits of the approach as a software library. Communication bounds for such reductions have been established under the Communication Avoiding framework [5]. The authors also show a model-driven optimization of a communication-optimal algorithm that is based on the two-stage approach [13]. A multi-stage approach has also been applied to the Hessenberg reduction [14], [15]. A rekindled interest in tile algorithms was also recently seen when applied to the two-stage tridiagonal [3], [1]

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.130

and bidiagonal reductions [4]. The first stage in these implementations is implemented using high performance kernels and asynchronous execution while the second stage is implemented based on cache-aware kernels and a task coalescing technique [3]. Recently, a distributed-memory eigensolver library called ELPA [16] was developed for codes used in electronic structure calculations. It includes one-stage and two-stage tridiagonalization algorithms, the corresponding eigenvector transformation, and a modified divide-and-conquer routine that is capable of computing the entire eigenspace or a portion thereof.

Perhaps the most closely related research is our work on the parallelization of the bi-diagonal reduction on multicore processors [17]. The one and only similarity to this paper is the use of a two-stage approach for a runtime scheduler that results in extra floating-point operations. The differences far more pronounced when this work is scrutinized in more detail. In particular, in the current paper we develop and thoroughly analyze the performance model of the computational stages (see Section 4) and use it to guide the algorithmic optimizations in Section 5 and implementation tuning in Section 6. Clearly, most if not all of the computationally crucial kernels had to be written from scratch to take full advantage of the symmetry of the eigen-value problem, which includes both a new implementation and tuning to achieve optimal cache use for a smaller memory and floating-point count of these kernels. The scheduling strategy and the prioritization of tasks has to account for a much more skewed timing characteristics where the compute-intensive tasks become twice as short (as determined by the flop-count) while the latency- and bandwidth-bound tasks continue running into the same memory system bottlenecks: limited transfer rate and very long fetch delays. This finally leads to the much greater influence of the Amdahl fraction [18] that deals with the memory-bound portion of the computation which in turn makes it more challenging to achieve good scaling and performance for a wide range of core counts and matrix sizes – the computational intesity of cache-resident kernels was much higher in our previous work [17].

3. Research Contributions

Besides the software development efforts that we investigate to accomplish an efficient implementation, we highlight three main contributions related to the algorithm's design:

- **Mapping computation to hardware via both dynamic and static scheduling.** We developed our algorithm in a way that facilitates mapping of computational tasks to the strengths of the available hardware components, and taking appropriate care of the data reuse. Our algorithm also uses techniques to mix between dynamic and static scheduling to extract both efficient scaling and performance. The impact of these techniques may be observed during either the bulge chasing stage or the eigenvectors update. The former operates on a small amount of data of the size $n_b \times n$, where n_b and n are the width of the band and the size of the matrix, respectively. Most operations in this stage are memory-bound and the parallelism is limited. Hence, it is better to let this stage run on a small number of cores, which increases data locality, rather than to let all the cores work, which increases the data coherence traffic. The latter applies the Householder reflectors generated by the bulge chasing stage in a complicated fashion and resolves the overlap between the reflector blocks. We combine the computation splitting (a technique based on the available number of resources and on the size of the Level 2 cache) with hybrid task scheduling. This combination is the main factor, that is used to determine the block size required for data reuse, and the way in which parallelism is extracted for high performance. Section 6 provides further details about these techniques.
- **High performance fine-tuned, memory-cognizant, and compute-bound tasks.** Our goal from the onset was to use modern hardware efficiently by providing plentiful parallelism that relies on splitting the computation into tasks that either increase computational intensity or reduce data movement. Two main issues should be taken into consideration here. First, the task splitting and determination of granularity is essential for obtaining high performance and scalability. Moreover, the data reuse among the CPU-cores should also be taken into consideration in order to minimize communication and achieve good execution rate. To that end, we developed new fine-tuned and memory-cognizant BLAS-like kernels for use during the second stage of the TRD reduction, i.e., for the bulge chasing procedure. Another set of kernels was developed for the update of the eigenvectors by the transformation matrices. This is in contrast to the kernels that have been developed by Luszczek et al. [1] only for the first stage of the reduction to the tridiagonal form.
  Sections 5 and 6 provide more information about these new fine-tuned and cache-friendly numerical kernels.
- **Examining the trade-off between higher performance and extra computation.** A judicious determination of this trade-off reduces overall execution time, which we believe will become increasingly important for the current and the future hardware designs. We employ an advanced optimization strategy, which consists of aggregating multiple applications of Householder reflectors occurring within a single data block. This removes the communications overhead as well as enhances the memory reuse while at the same adds a small extra cost. We show in Section 7 how this allows to obtain high performance algorithms that significantly outperform any of the currently available alternatives.

Routine	Method	TRD	Gen. Q	Eig of T	Update Z
EVD	D&C	$4/3n^3$	0	$4\sim8/3n^3$	$4n^3$
EVR	MRRR	$4/3n^3$	0	$O(n^2)$	$4n^3$
EV	QR	$4/3n^3$	$2^4/3n^3$	$\approx 6n^3$	0

Table 1. Three most common methods for symmetric eigenvalue problem.

Reduction	Operations	Sandy Bridge
TRD	4× SYMV	45 Gflop/s
BRD	4× GEMV	26 Gflop/s
HRD	10× GEMV	13 Gflop/s

Table 2. Operation types and their counts for three two-sided reductions.

4. Algorithmic Complexity Study

In this paper, we make design choices that significantly affect the algorithmic complexity of our algorithm by increasing it almost two-fold. This section details this aspect of our work.

Computing eigenvalues of a dense Hermitian matrix A using an LAPACK algorithm proceeds by reducing A to a triangular form:

$$A = Z\Lambda Z^H. \tag{1}$$

Computation of eigenvectors then proceeds

$$T = E\Lambda E^H \tag{2}$$

with matrices Q_1 and Q_2 from the two stages:

$$Z = Q_1Q_2E = (I - V_1T_1V_1^H)(I - V_2T_1V_2^H)E. \tag{3}$$

where (V_1, T_1) and (V_2, T_2) represent the Householder reflectors generated during the first and second reduction stages, respectively. Table 1 presents the computational complexity that results from using one of the common methods when the single stage approach is taken ($Q_2 \equiv I$).

Our model for execution time allows us to ascertain the validity of the two-stage approach for the case when both eigenvalues and eigenvectors are calculated. In the one-stage approach we essentially have two components – first for the eigenvalues and second for the eigenvectors – each of which has cubic complexity:

$$t_{1\text{-}s} = \frac{4}{3}\frac{n^3}{\beta} + 2\frac{n^3}{\alpha p}f \tag{4}$$

where α is the execution rate of xGEMM measured in flop/s, β is the execution rate for xGEMV, and f is the fraction of the number of desired eigenvectors ($0 < f \leqslant 1$). For the two-stage approach, we need to account for both

Parameter	AMD Magny-Cours	Intel Sandy Bridge
α	10 Gflop/s	20 Gflop/s
β	40 MB/s	80 MB/s
p	12	8

Table 3. Sample values of the parameters used in the complexity formulas.

stages that result in, first, symmetric band form, and, later, tridiagonal form:

$$t_{2\text{-}s} = \frac{4}{3}\frac{n^3}{\alpha p} + 6D\frac{n^2}{\alpha p'} + 4\frac{n^3}{\alpha p}f \tag{5}$$

where D is the size of band after the first stage and p' is the level of parallelism available in the second stage (bulge chasing): $p' \leqslant \min(D, p)$.

Clearly, the one stage algorithm does not scale: $\lim_{p\to\infty} t_{1\text{-}s} = 4/3n^3/\beta$ as well as the two-stage one: $\lim_{p\to\infty} t_{2\text{-}s} = 6Dn^2/(\alpha p')$. And for large problem sizes, two stage approach is superior: $\lim_{p\to\infty} \frac{t_{1\text{-}s}}{t_{2\text{-}s}} = \frac{\alpha p/\beta + 3/2 f}{1+3f}$ considering the fact that the quantity $\alpha p/\beta$ may easily exceed a few orders of magnitude even for a single socket multicore system – typical values are given as an example in Table 3 which are based on the operation types and counts included in Table 2. The question then remains in what range of problem sizes n the two-stage algorithm is viable or for each n $t_{1\text{-}s} = t_{2\text{-}s}$. By substitution in (4) and (5) we obtain

$$n(\alpha, \beta, D, f, p) = \frac{9\beta D}{2\alpha p - 3f\beta - 2\beta} \tag{6}$$

which from the theoretical stand-point allows for a wide range of problem sizes to benefit from our two-stage algorithm.

We do not concern ourselves here with the complexity of the second stage of the two-stage algorithm, the bulge-chasing procedure, because involves only a low-order terms. In particular, the estimate of the operation count is $n^2\left(1 + \frac{i_b}{n_b}\right)$ where i_b and n_b are internal and external blocking factors.

4.1. Comparison with the Bi-Diagonal Reduction

The analysis presented above could apply with an appropriate scaling to our bi-diagonal reduction code [17]. However, we did not apply this model before and therefore it should be considered as a new contribution of this paper.

Furthermore, we would like to show how the symmetric eigenvalue problem is more sensitive to the hardware specification and the software implementation in comparison with the SVD algorithm that we have succesfully parallelized before [17].

To simplify the exposition we can assume that the 3-stage process of computing eigen-values and eigen-vectors of a symmetric matrix has the complexity:

$$\frac{4}{3}n^3 + O(n^2) + 2n^3 + 2n^3 \tag{7}$$

where the first compent represents reduction to block tri-diagonal form, the second one – bulge-chasing to achieve tri-diagonal form, and the third one – computation of the corresponding eigen-vectors. In comparison, our SVD algorithm that computes both singular values and singular vectors [17] has the complexity:

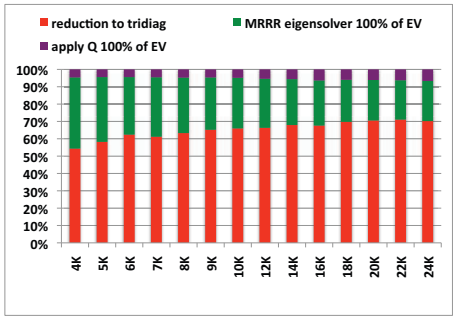
$$\frac{8}{3}n^3 + O(n^2) + 4n^3 + 4n^3 \tag{8}$$

which stems from the lack of symmetry in the SVD code and causes the floating-point instruction count to double.

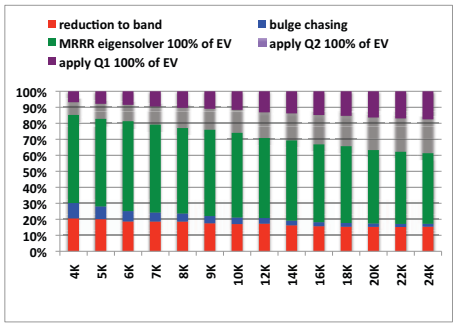
The $O(n^3)$ components are amenable to parallelization because they are based on compute-intensive tasks that readily benefit from the increased number of cores. The $O(n^2)$ component, on the other hand, is strictly memory-bound and not only suffers due to the lack of memory bandwidth but also is sensitive to the memory latency since it involves small vectors. The square component is the Amdahl fraction [18] of both algorithms but our previously parallelized SVD code [17] has disproportionally larger amount of readily parallizable workload. In comparison, the eigenvalue implementation, that we present here, is much more sensitive to insufficient parallelization and below we present ways of dealing with this problem. This of course is compounded by the fact that our multi-stage algorithm practically doubles the number of floating-point operations when compared with the traditional implementations.

5. Two-Stage Asynchronous Algorithm for Tridiagonal Reduction

Due to its computational complexity and data access patterns, the reduction to the tridiagonal form is challenging to implement and optimize. Two approaches exist. The standard one-stage approach from LAPACK [7], whereby the block Householder transformations are used to directly reduce the dense matrix to tridiagonal form. A newer algorithm that we refer to as *two-stage approach*, whereby a block Householder of transformations are used to first reduce the matrix to a band form, and then, in the second stage, bulge chasing procedure is used to reduce the band matrix to the tridiagonal form [3]. The one-stage approach is memory-bound because each reflector relies on a symmetric matrix-vector multiplication with the trailing submatrix, which cuases the entire trailing submatrix to be loaded into main memory without any regard to data reuse in the cache hierarchy. The resulting performance is bound by the main memory bandwidth for larger matrices and exhibits a moderate speedup for smaller matrices that fit entirely in one of the cache levels. Figure 1a shows the percentage of the total time for each of the three components of the eigenvector routine using the standard one-stage reduction approach when all the eigenvectors are requested. When only a portion of eigenvectors are needed, the reduction to the tridiagonal form consumes even greater portion of the execution time while the share of other phases diminishes. The figure indicates that the reduciton to the tridiagonal form requires 90% of the total computation time when only eigenvalues (or a small portion of eigenvectors) are needed and over 60% of execution time when all the eigenvectors are requested. This, aside from the promising analysis from Section 4, was the main motivation for our work, to extend the two-stage algorithm for computing both eigenvalues and eigenvectors. To ease the development on the multicore architectures we used the software infrastructure provided by the PLASMA



(a)



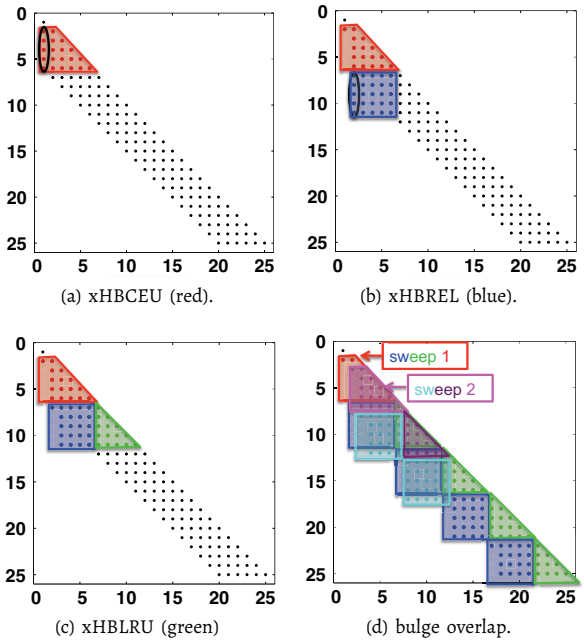
(b)

Figure 1. (a) The percentage of the time spent in each kernel of the eigensolver using the standard one stage approach to compute the tridiagonal form; (b) the corresponding percentages for the two-stage approach.

project [19]. The techniques used here are similar to the one developed previously for multicore processors and eigenvalue-only calculation [3]. In the following we strive for completeness as we begin by briefly describing the first stage (the reduction from dense to band), then we explain in detail the reduction from band to the tridiagonal form and its fine-grain scheduling techniques.

5.1. The First Stage: Reduction to the Band Form

The two-stage approach overcomes the memory bandwidth limitations of the one-stage approach. The first stage (the reduction to band) is compute-intensive and may be performed efficiently using optimized kernels from Level 3 BLAS. In particular, it relies on tile algorithms [20]. The matrix is split into tiles, whereby data within a tile is contiguous in memory and thus avoids the cache and TLB misses associated with strided access. The computation is then broken into tasks and proceeds that are organized into a directed acyclic graph (DAG) [21]. The nodes represent tasks and the edges are the data dependences. Restructuring linear algebra algorithms as a sequence of tasks that operate on blocks of data removes the bottleneck of BSP (or fork-join stages) barriers [22], [23] and increases data locality. This required implementations of new computational kernels [3], [2], [1], [4], [24] to be able to operate within the new algorithm and on the new



(a) xHBCEU (red). (b) xHBREL (blue). (c) xHBLRU (green) (d) bulge overlap.

Figure 2. Kernel execution of the TRD algorithm during the second stage.

data structures.

5.2. The Second Stage: Reduction to the Tridiagonal Form

There are numerous shortcomings of the basic bulge chasing procedure that triggered development of the memory-aware numerical kernels and the scheduling techniques used here. The most problematic aspect of the standard procedure is the element-wise elimination [3]. We developed a bulge chasing that uses a very similar algorithm but differs in using a column-wise elimination. Our modification adds a small amount of extra work but allows the use of the Level 3 BLAS kernels to compute the transformations and to apply them in the form of the orthogonal matrix Q_2 which is the result of computation in this phase. Below we present a brief description of the column-wise bulge chasing approach as well as the technique used for task scheduling and the enhancement of data locality. The bulge chasing algorithm consists of three new kernels. The first kernel is called xHBCEU and it triggers the beginning of each bulge chasing sweep by annihilating the extra non-zero entries within a single column by calling the LAPACK's xLARFG function. This is shown in Figure 2a. The kernel then applies the computed elementary Householder reflectors to both sides of the appropriate symmetric data block (red triangle), that is loaded into the cache memory. The second kernel, xHBREL, continues the application from the right derived from the previous kernel, either xHBCEU or xHBLRU. This subsequently generates triangular bulges, new non-zero entries, as shown in Figure 2b, which must be annihilated by early enough in order to avoid the excessive growth of this fill-in structure. Note that the triangular bulges created by the annihilation process of the sweep i overlap with those of the sweep $i+1$, because of only one column shift to the right and one row down. It is shown in Figure 2d, as the lower triangular portion of the cyan squares (the bulge created in sweep $i+1$) overlaps with the lower triangular portion of the blue squares (corresponding to the bulges created by the previous sweep i). Thus, during the annihilation of sweep i, if we eliminate each of the triangular bulges (the lower blue triangular of Figure 2b) with a call to xHBREL for sweep i, then, at the next step, the annihilation of sweep $i+1$ creates a triangular bulge which will overlap with the one previously eliminated and cause appearance of fill-in in the overlapped region. We can reduce the computational cost by only eliminating the non-overlapped region of bulge, i.e., its first column, instead of eliminating the entire triangular bulge created for sweep i. The remaining columns can be delayed for elimination during the subsequent annihilation sweeps. On the one hand, we can avoid an excessive growth of bulges – a bulge created once will expand dramatically if not chased down the diagonal. On the other hand, our delayed annihilation allows us to reduce the extra computation. We designed our cache friendly xHBREL kernel to take advantage of the fact that the created bulge (the blue block) remains in cache and so right after the annihilation the first column occurs, the left update is immediately applied to the remaining columns of the blue block. The third kernel, xHBLRU, continues the application of reflectors from the left to the green block of Figure 2c. Since, the green block is remaining in cache, hence the kernel proceeds with the application from the right to the symmetric portion. The annihilation resulting from each sweep comprises a single call to the first kernel followed by a repetitive calls to a repetition of the second and the third kernels. The implementation of this stage is done by using either a dynamic or a static runtime environment that we developed [19]. In our opinion, this stage is one of the main challenges of the overall algorithm as it poses challenges of tracking the data dependences. The annihilation from the subsequent sweeps will generate computational tasks, which will have partially overlapped data between tasks from previous sweeps as seen in Figure 2d. We have used our data translation layer (DTL) [1], and supplemented it with functional dependencies [3] to handle the dependences, and to provide sufficient information to the runtime to achieve the correctness in scheduling.

6. The Application of the Orthogonal Matrices Q_1 and Q_2

In this section, we discuss the application of the Householder reflectors generated from the two stages of the reduction to the tridiagonal form. We focus the

discussion from §4 on the two-stage algorithm, whereby the first stage reduces the original Hermitian matrix A to a band matrix B by applying a unitary Q_1 to both sides of A such that we obtain $A = Q_1 B Q_1^H$. In the second stage, the bulge chasing procedure reduces the band matrix B to a tridiagonal form by applying the unitary matrix Q_2 to both sides of B yielding $B = Q_2 T Q_2^H$. This requires the eigensolver iteration to update matrix E of Eq. (2) with the Householder reflectors generated during the reduction phase according to Eq. (3).

From the practical standpoint, the application of the V_2 reflectors is not as straightforward as the application of V_1. To show this, we begin by first describing the complexity and the design of the algorithm for applying V_2. We show the structure of V_2 in Figure 3b. Note that the reflectors being applied represent the annihilation of the band matrix columns, and thus, each one is of length nb – the bandwidth size. A naïve implementation would take each reflector and apply it in isolation to the matrix E. Such an implementation is memory-bound because relies on Level 2 BLAS operations. If we want to group the separate anihilation operations and take advantage of the efficiency of Level 3 BLAS routines, we must pay attention to the overlap between the data they access as well as the fact that their application must follow the specific dependency order dictated by the bulge chasing procedure. For example, for sweep i, the annihilation of the column at position $B_{i,i:i+n_b}$ generates a set of k Householder reflectors $v_i^{(k)}$, each of length n_b represented as column i of the matrix V_2, which is shown in Figure 3b. The columns related to the annihilation of sweep $i+1$, are those presented in column $i+1$ of V_2 and so on with the caveat that each subsequent column is shifted one element below the previous column. Despite the dependences of this bulge chasing procedure, it is possible to group the reflectors $v_i^{(k)}$ from sweep i with those from sweep $i+1$, $i+2$,..., $i+\ell$ and to apply them together in blocked fashion. This grouping is represented by the diamond-shaped region in Figure 3b. While each of those diamonds may be considered as a single block, their application to the E matrix has to follow the order inherited from the bulge chasing stage. For example, applying the green block 4 and the red block 5 of the V_2 in Figure 3b modifies the green block row 4 and the red block row 5, respectively, of the eigenvector matrix E shown in Figure 3c. This allows to observe the overlapping between the regions. The order dictates that block 4 needs to be applied before block 5. We have shown a sample of these dependences by the arrows in Figure 3b. We also represented them by the DAG in Figure 3d. This pattern of dependences allows a very limited number of parallel and pipelined tasks. In practice however, it is possible to compute these efficiently by focusing on the matrix E and on how it is split into blocks of columns over the number of cores as shown in Figure 3c. We can then apply each diamond block independently to the appropriate portion of E. Moreover, this method does not require any data communication between cores reducing cache conflict misses in addition

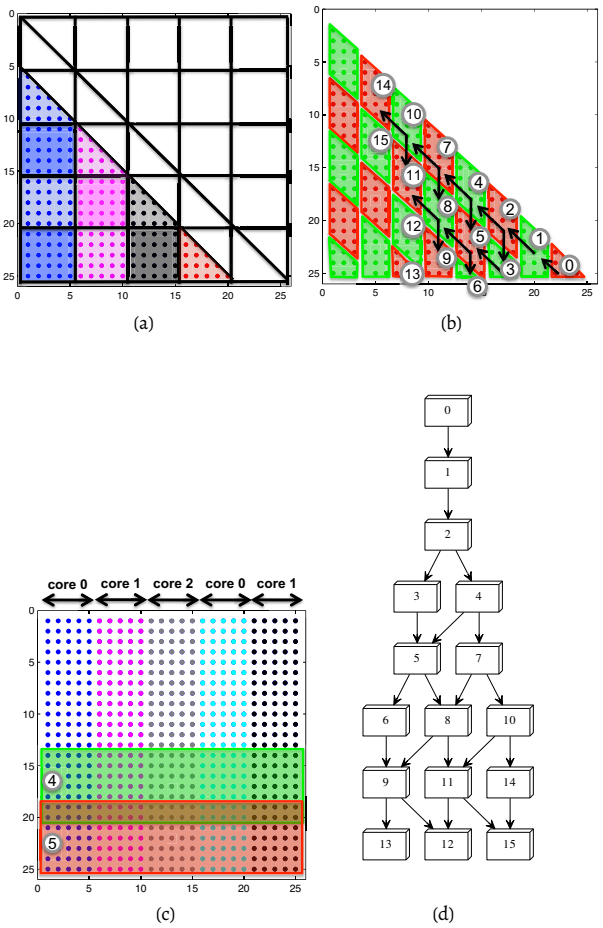


Figure 3. (a) Tiling of V_1, (b) Blocking technique to apply V_2, (c) Distribution of the eigenvectors matrix that create independent fashion of applying Q_2 which increase locality per core, (d) Portion of the DAG showing the dependency of the V's of V_2.

to the cache resuse afforded by groupping. The size of each block of E is small enough so that more than one of them fits in the L2 cache for increased data locality. For example, core 1 applies all the portion of V to the magenta block of Figure 3c, then it moves to its next assigned block – the black block. The draw back is that we had to implement a new kernel that deals with the diamond-shape blocks in a way that increases the cache reuse.

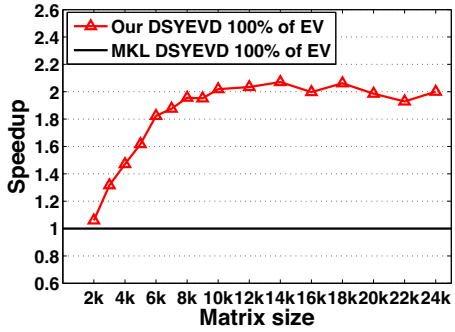
By comparison, the application of V_1 to the resulting matrix, $G = (I - V_2 T_2 V_2^H)E$, may be done simply by using our tile algorithm. For one, there is no overlap between the different parts of V_1. Each tile of a column block of V_1 modifies a different area of the matrix G. In Figure 3a, any tile of the magenta column modifies different area of G and so they can be applied independently. Also, the operations can be merged as shown in Figure 3a, which renders their application a compute-intensive task that may be achieved with efficient BLAS 3 kernels. The elements

of V_1 are stored in a tile fashion as shown in Figure 3a to increase data locality. The application "from the left" has to satisfy only one constraint whereby the $v_{4,3}$ – the black tile (4,3) – has to be applied before the magenta $v_{4,2}$. Similary, the magenta $v_{4,2}$ needs to be aplied before the blue $v_{4,1}$. The parallelism comes from two sources: the matrix G is a set of independent tiles slated for update and, also, the application of blocks of the V_1 is independent. As a result, the design of the tile algorithm generates a large number of independent tasks that can be applied in an asynchronuous manner using either a static or dynamic scheduling.

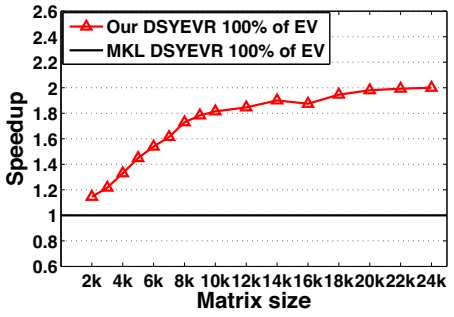
7. Performance Results

Our experiments have been performed on the largest shared memory system that we could access. It is representative of a vast class of servers and workstations commonly used for computationally intensive workloads. We benchmark all implementations on a four-socket system with AMD Opteron™ 6180 SE processor, each comprising 12 cores for the total of 48 cores, running at 2.5 GHz with 128 GiB of main memory. The cores were evenly spread among two physical mother boards. The Level 2 cache size per core was 512 KiB. All computations were performed in double precision arithmetic. The theoretical peak for this machine in double precision was 480 Gflop/s or 10.1 Gflop/s per core. There are a number of software packages that include an eigensolver. We used the latest MKL (Math Kernel Library) [25] version 13.1, which is a commercial software library from Intel that is a highly optimized for Intel processors but remains competitive on AMD hardware especially for small matrix sizes that we need for our tile algorithm implementation. The library includes a comprehensive set of mathematical routines implemented in a way to run well on most x86 multicore processors. In particular, MKL includes the LAPACK-equivalent routines to compute the tridiagonal reduction DSYTRD (with the **QR** iteration), or to find the eigenpairs DSYEVD (divide and conquer – **D&C** – algorithm) and DSYEVR (the Multiple Relatively Robust Representations – **MRRR** – approach).

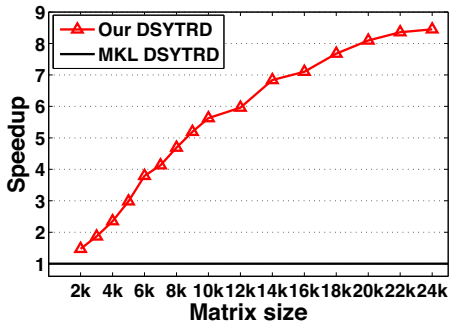
We performed an extensive study with a large number of experimental tests to provide comprehense information for the wide range of possible testing scenarios to take into accounts various runtime effects that might influence the performance and the outcome of comparisons. We computed the eigenpairs of a symmetric eigenvalue problems, by varying the size of matrices from 2000 to 24,000 using the all of the available 48 cores of the machine. As a point of reference against our prior results [1], [2], [3], [4], we report the results of improvements that our two-stage implementation brings to the reduction to the tridiagonal form compared against the one-stage approach. We compare against MKL as it contains a state-of-the-art implementation and is superior in terms of performance when compared with all the available numerical linear algebra libraries. Figure 4c shows the comparison between our implementation versus the DSYTRD routine from



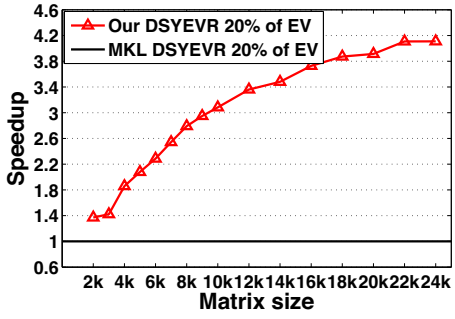
(a)



(b)



(c)



(d)

Figure 4. Speedup comparison versus the MKL librairies for different eigensolver.

Intel's MKL library. Asymptotically, our code achieves over 8-fold speedup for the largest matricees tested. This is the result of the effcient implementation of the first stage (reduction to band) which is the compute intensive

stage, and from the careful design and implementation of the second stage that maps both the algorithm and the data to the hardware using cache-friendly kernels and scheduling that takes data locality into account. In Figures 4a and 4b, we illustrate the speedup obtained by our algorithm when computing all the eigenvectors using either the D&C or the MRRR as the tridiagonal eigensolver. We still observe a significant speedup over the optimized MKL solver: our implementation is twice as fast. It is not as high as eigenvalues-only scenario due to the complexity argument presented in §4. Also note, that the time to compute the eigenpairs (λ, Z) of the matrix A, is the sum of the time required for three steps: (1) the time to perform the tridiagonal reduction, which could enjoy as big as 8-fold speedup; (2) the time to compute the eigenvectors of the tridiagonal form, and (3) the time to update these eigenvectors (the back transformation). Since our work is focused on improving and optimizing phases 1 and 3, they are now around 3 times faster than those of the one-stage approach. This makes the phase 2 dominate the execution time as predicted by Amdahl's law [18]. In fact, it now consists of 50% of the new reduced total time, which is shown in Figure 1b. Finally, observe that that the time required to compute the eigenvectors of the tridiagonal matrix (phase 2) is the same as the one for the MKL solver. In the end, reaching a two-fold speedup is worthwhile effort. In Figure 4d, we show the speedup obtained by our algorithm when only 20% of the eigenvectors are needed, which is a scenario that might occur in practice for applications that do not require the full set of eigenvalues. This case is similar and the graph exhibits a familiar trend to the one presented in Figure 4c, with the 4-fold speedup achieved. We would like to highlight the fact that when a portion of the eigenvectors is needed, the cost of our algorithm may be reduced dramatically as both phase 2 and phase 3 require less operation and thus are faster, which is represented the fraction f in Eq. (4) and (5). In concrete terms, our algorithm requires 150 seconds to find $f = 20\%$ of the eigenvectors for a matrix of size 20,000, and 400 seconds when all vectors are requested. Achieving such speedups is one of the initial pieces of motivation to extend the two-stage algorithm for eigenvectors.

7.1. The Effects of the Bandwidth Size

We have previously developed [4] a performance model for the bulge chasing stage of a two-stage algorithm that was helpful in predicting an optimal blocking factor $n_b = 80$. The execution time t_x was modeled as:

$$t_x = \frac{1}{\alpha} \times n^2 \times n_b \tag{9}$$

the communication time t_c was modeled as:

$$t_c = n^2 \times \left(\frac{n_b}{\beta} + \frac{\gamma}{n_b} \right) \tag{10}$$

where parameters α, β, and γ are the same as given in Table 3. The derivation of the model relies on the exactly same assumptions as are applicable to our current implementation. We then proceed in perusing the model for tuning and showing the effects of the current design in the results presented below.

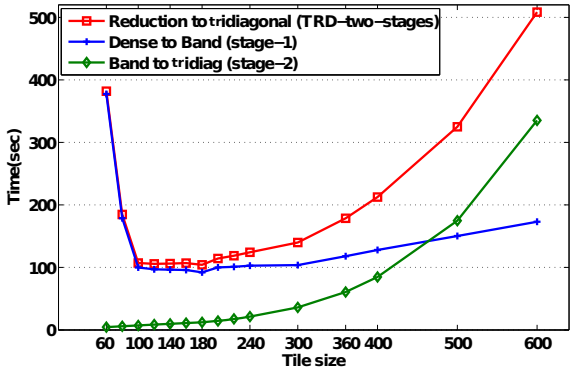


Figure 5. The effect of the tile size on the performance of both stages for a matrix of size 16, 000 using 48 cores of system A.

As the model given by Eqs. (9) and (10) suggests, bandwidth size, which also is the tile size, is an important aspect of tuning the code in order to achieve near optimal performance. Unlike the first stage kernels, in the second stage sensitivity to memory bandwidth is the key due to reliance on Level 2 BLAS operations – their performance depends on how much data can fit in the cache memory. Thus, if the tile size chosen for the first stage is too large, the second stage may encounter significant difficulties in coping with the memory bus latency. Figure 5 illustrates this effect with respect to n_b parameter and how it affects the first stage (reduction to general band) by showing the blue curve with the "+" markers. And for the second stage (the bulge chasing), the green curve with the "diamond" marker. On the one hand, the tile size needs to be large ($120 < n_b < 300$) to extract high performance from the first stage but with limit due to effect it has in the second stage. Evidently, for $n_b > 360$, we lose all the gains obtained from the locality and the cache reuse as the data does not fit into the Level 2 cache and we also lose substantial degree of the parallelism because the number of tiles $n_t = n/n_b$ decreases precipitously. The tile size has to be small enough to extract high performance from the bulge chasing stage, which has to do with the cache containment of tile data. Thus, a judcious trade-off between the stages is offers an optimal choice. In our experiments, we found that $120 < nb < 200$ looks to be the best compromise which is in line with the predictions offered by the performance model.

8. Conclusions and Future Work

In this paper, we presented a novel implementation of an algorithm that computes eigenvalues and of eigenvectors (or a chosen subset thereof) for symmetric or hermitian matrices. Our algorithm is based on the two-stage approach and thus performs twice as many floating operations when

asked to compute the eigenvectors when compared with the classic one-stage approach. Such a drastic increase in operation count might have been considered a hindrance just a few years back but on modern hardware it has become an advantage due to the overall structure of the code and its design, that is geared towards high flop-counts and comparatively low memory bandwidth. In particular, we attribute this to the formulation of the algorithm in terms efficient kernel routines and we show their benefit both theoretically as well as in a practical setting. With a two-fold increase in the operation count, we were still able to achieve two-fold speedup over the current of state-of-the-art software packages that achieve the highest speed on the tested hardware. Because of good scalability properties of our algorithm, we believe that our approach lends itself well to distributed memory implementations and we plan to pursue this direction in the future. Similarly, we consider a future implementation on hardware accelerators such as GPUs or coprocessors to be good candidates for validation of our approach.

References

[1] P. Luszczek, H. Ltaief, and J. Dongarra, "Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures," in *IPDPS 2011: IEEE International Parallel and Distributed Processing Symposium*. Anchorage, Alaska, USA: IEEE Computer Society, May 16-20 2011, pp. 944–955.

[2] H. Ltaief, J. Kurzak, and J. Dongarra, "Parallel band two-sided matrix bidiagonalization for multicore architectures," *IEEE TPDS*, vol. 21, no. 4, April 2010.

[3] A. Haidar, H. Ltaief, and J. Dongarra, "Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 8:1–8:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063394

[4] H. Ltaief, P. Luszczek, and J. Dongarra, "High Performance Bidiagonal Reduction using Tile Algorithms on Homogeneous Multicore Architectures," *ACM Transactions on Mathematical Software*, vol. 39, no. 3, pp. 16:1–16:22, May 2013. [Online]. Available: http://dx.doi.org/10.1145/2450153.2450154

[5] G. Ballard, J. Demmel, and I. Dumitriu, "Communication-optimal parallel and sequential eigenvalue and singular value algorithms," EECS University of California, Berkeley, CA, USA, Technical Report EECS-2011-14, February 2011, LAPACK Working Note 237.

[6] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," Department of Computer Science and Engineering, University of Notre Dame, Tech. Rep. TR-2008-13, September 28 2008.

[7] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: SIAM, 1992, http://www.netlib.org/lapack/lug/.

[8] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[9] R. G. Grimes and H. D. Simon, "Solution of large, dense symmetric generalized eigenvalue problems using secondary storage," *ACM TOMS*, vol. 14, pp. 241–256, September 1988. [Online]. Available: http://doi.acm.org/10.1145/44128.44130

[10] B. Lang, "Efficient eigenvalue and singular value computations on shared memory machines," *Parallel Computing*, vol. 25, no. 7, pp. 845–860, 1999.

[11] C. H. Bischof, B. Lang, and X. Sun, "Algorithm 807: The SBR Toolbox—software for successive band reduction," *ACM TOMS*, vol. 26, no. 4, pp. 602–616, 2000.

[12] P. Bientinesi, F. D. Igual, D. Kressner, and E. S. Quintana-Ortí, "Reduction to condensed forms for symmetric eigenvalue problems on multi-core architectures," in *Proceedings of the 8th international conference on Parallel processing and applied mathematics: Part I*, ser. PPAM'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 387–395. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882792.1882839

[13] G. Ballard, J. Demmel, and N. Knight, "Avoiding communication in successive band reduction," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-131, Jul 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-131.html

[14] L. Karlsson and B. Kågström, "Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures," *Parallel Computing*, 2011, dOI:10.1016/j.parco.2011.05.001.

[15] B. Kågström, D. Kressner, E. Quintana-Orti, and G. Quintana-Orti, "Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited," *BIT Numerical Mathematics*, vol. 48, pp. 563–584, 2008.

[16] T. Auckenthaler, V. Blum, H. J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. R. Willems, "Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations," *Parallel Comput.*, vol. 37, no. 12, pp. 783–794, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2011.05.002

[17] A. Haidar, J. Kurzak, and P. Luszczek, "An improved parallel singular value algorithm and its implementation for multicore hardware," in *SC13, The International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, USA, November 17-22 2013.

[18] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings*, vol. 30. Atlantic City, N.J.: AFIPS Press, Reston, VA, APR 18-20 1967, pp. 483–485.

[19] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *J. Phys.: Conf. Ser.*, vol. 180, no. 1, 2009.

[20] E. Agullo, B. Hadri, H. Ltaief, and J. Dongarra, "Comparative study of one-sided factorizations with multiple software packages on multi-core hardware," *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[21] A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov, "The impact of multicore on math software," in *Applied Parallel Computing. State of the Art in Scientific Computing, 8th International Workshop, PARA*, ser. LNCS, B. Kagstrom, E. Elmroth, J. Dongarra, and J. Wasniewski, Eds., vol. 4699. Springer, 2006, pp. 1–10.

[22] L. G. Valiant, "Bulk-synchronous parallel computers," in *Parallel Processing and Artificial Intelligence*, M. Reeve, Ed. John Wiley & Sons, 1989, pp. 15–22.

[23] ——, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, Aug. 1990, dOI 10.1145/79173.79181.

[24] A. Haidar, H. Ltaief, P. Luszczek, and J. Dongarra, "A comprehensive study of task coalescing for selecting parallelism granularity in a two-stage bidiagonal reduction," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*. Shanghai, China: IEEE Computer Society, May 21-25 2012, pp. 25–35.

[25] Intel, "Math Kernel Library," Available at http://software.intel.com/en-us/articles/intel-mkl/.