New Multi-Stage Algorithm for Symmetric Eigenvalues and Eigenvectors Achieves Two-Fold Speedup[⋆]

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 eigenvalues and eigenvectors of a dense symmetric matrix. We show that reformulating the algorithms is beneficial even if that doubles the computational complexity. Through detailed analysis we show that the effect of increase in operation count may be compensated by much improved performance rate. Our performance results indicate that using our approach achieves very good speedup and scalability over existing state-of-the-art software.

1 Introduction and Background

Exisiting approaches are for values only [7,15,21,22,23] The reason quoted in literature was extra computational cost. We are attempting to test the hypothesis that even with the additional cost we would be able to achieve faster execution provided that we reformulate the computations in terms of highly parallel and cache-friendly kernels.

To solve a Hermitian (symmetric) eigenproblem of the form $Ax = \lambda x$, and find its eigenvalues λ and eigenvectors Z such that, $A = Z\lambda Z^H$, we usually need three phases [2,13,24]: (1) We first need to reduce the matrix to tridiagonal form using an orthogonal transformation Q such that $A = QTQ^H$, where T is a tridiagonal matrix (called a "reduction phase"). Note that, when orthogonal transformations are applied to generate T, the eigenvalues of T are the same as those of the original matrix A. (2) Compute eigenpairs (λ, E) of the tridiagonal matrix (called a "solution phase"); (3) Back transform eigenvectors of the tridiagonal matrix to the original matrix Z=Q*E (called a "back transformation phase"). Due to the computational complexity and the data access patterns, it has been well known that phase 1 is considerably more time consuming than the other two phases combined. Thus, in this paper, we will focus improving both: the phase of reduction to tridiagonal matrix and the back transformation phase.

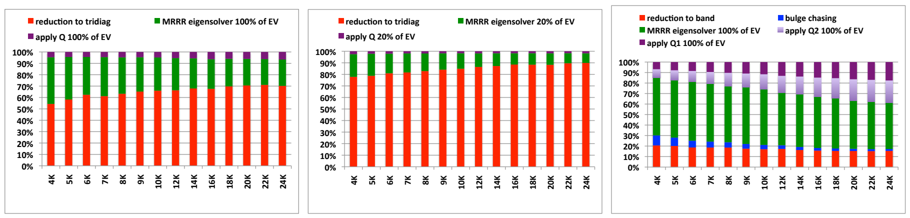
⋆ This research was supported by NSF.

2 Related Work

Solving the symmetric eigenvalue problem is an active research field. Recently, many researchers have been interested in this area and have developed various strategies with a number of software implementations. The robust and conventional software LAPACK [5] and ScaLAPACK [10] are for shared-memory and distributed-memory systems, respectively. Hardware vendors in general provide well tuned and optimized LAPACK and ScaLAPACK versions. Recent work has concentrated on accelerating separate components of the solvers, and in particular, the reduction to tridiagonal form, which is the most time consuming phase, and also the eigensolver. A new type of algorithm that challenges the standard one-stage reduction algorithms has been introduced. The idea behind this new technique is to split the reduction phase into two or more stages, recasting expensive memory-bound operations that occur during the panel factorization into compute-bound operations. One of the first uses of a two-stage reduction occurred in the context of out-of-core solvers for generalized symmetric eigenvalue problems [14]. Then, a multi-stage method was used to reduce a matrix to tridiagonal, bidiagonal and Hessenberg forms [20]. Consequently, a framework called Successive Band Reduction (SBR) was developed [8, 9]. Communication bounds for such type of reductions have been established under the Communication Avoiding framework [7]. A multi-stage approach has also been applied to the Hessenberg reduction [18, 19]. Tile algorithms have also recently seen a rekindled interest when applied to the two-stage tridiagonal [16, 23] and bidiagonal reductions [22]. Their first stage 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 [16]. Recently, a distributed-memory eigensolver library called ELPA [6] was developed for electronic structure codes. It includes one-stage and two-stage tridiagonalization alborithms, the corresponding eigenvector transformation, and a modified divide and conquer routine that can compute the entire eigenspace or a portion thereof.

3 Multi-Stage Asynchronous Algorithm for Tridiagonal Reduction

Due to its computational complexity and data access patterns, the tridiagonal reduction phase is the most challenging to develop and optimize: both algorithmically and implementation-wise. There are two approaches to the problem: the standard one-stage approach from LAPACK [4], whereby block Householder transformations are used to directly reduce the dense matrix to tridiagonal form, and a newer one, two-stage (or many-stage) approach, whereby block Householder transformations are used to first reduce the matrix to a band form, and in the second stage, bulge chasing technique is used to reduce the band matrix to tridiagonal [16]. The one-stage approach is well known to be memory-bound as each reflector relies on symmetric matrix-vector multiplications with the trailing submatrix. Thus, the entire trailing submatrix needs to be loaded into memory.



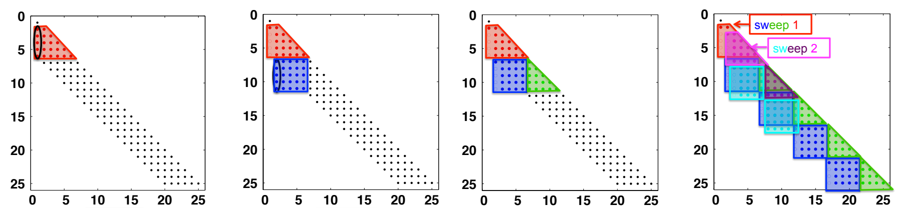
(a) (b) (c)

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

As memory bandwidth is a very scarce resource, this will obviously not scale for larger matrices, and thus will generate a tremendous amount of cache and TLB misses. Figure 1a, shows the percentage of the total time for each of the three components of the eigensolver using the standard one-stage reduction approach when all the eigenvectors are computed, while Figure 1b shows the execution time when only 20% of the eigevectors are computed. These figures show that the reduciton to tridiagonal form requires 90% of the total computing time when only eigenvalues are needed or when only a portion of the eigenvectors and more than 60% of the global time when all the eigenvectors are computed. This was the main motivation for our work, to study and develop an algorithm that compute both eigenvalues and eigenvectors using the two-stage approach, for the multicore architecture using the PLASMA [1] project. The technique used here is similar to the one developed for multicore processors [16]. To put our work into perspective, we start by briefly describing the first stage (reduction from dense to band), then we explain more details about the reduction from band to tridiagonal and its scheduling techniques.

3.1 The First Stage: the Reduction to Band Form

The two-stage approach overcomes the limitations of the one-stage approach that relies heavily on memory-bound operations. 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 [3]. The matrix is split into tiles, whereby data within a tile is contiguous in memory. The general algorithm is then broken into tasks and proceeds using the tile data layout. The tasks are organized into a directed acyclic graph (DAG) [11, 12], with the nodes representing tasks and the edges – the data dependencies between them. Restructuring linear algebra algorithms as a sequence of tasks that operate on blocks of data removes the fork-join bottleneck and avoid idle state while increasing the data locality for each core. This requires implementations of new computational kernels to be able to operate on the new data structures. The details of the implementation of this stage are provided elsewhere [16, 23].



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

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

3.2 The Second Stage: the Reduction to Tridiagonal Form

There are numerous shortcomings of the standard 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 [16]. We developed a bulge chasing very similar algorithm that differs from the standard one in using a column-wise elimination. Our modification adds a small amount of extra work but it allows the use of the Level 3 BLAS kernels to compute the transformations or to apply them in the form of the orthogonal matrix Q_2 – the result of computation in this phase. Below is our brief description of the column-wise bulge chasing approach as well as the technique used for task scheduling and increasing data locality. The bulge chasing algorithm consists of three new kernels. The first kernel called xHBCEU triggers the beginning of each sweep by annihilating the extra non-zero entries within a single column by calling the xLARFG function. This is shown in Figure 2a. The kernel then applies the computed elementary Householder reflector from the left and the right to the corresponding symmetric data block (red triangle) 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 as shown in Figure 2b, which must be annihilated by an appropriate technique in order to eventually avoid the excessive growth of the fill-in structure. Note that the triangular bulges created by the annihilation process of the sweep $i+1$ overlap with those of the sweep i, by one column shift to the right and to the bottom, as shown in Figure 2d where the reader can see that 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 the one previously eliminated and refill the overlapped region with non-zeros values. As a result, we can reduce the computational cost. Instead of eliminating the triangular bulge created for sweep i, we only eliminate the non-overlapped region of it: its first column. The remaining columns can be delayed to the upcoming annihilation sweeps. In this way, we can avoid the growth of the bulges – a once created bulge will expand dramatically if not chased down the diagonal. Our delayed annihilation allows for reduction of extra computation. Moreover, we designed a cache friendly xHBREL kernel that takes advantage of the fact that the created bulge (the blue block) remains in the cache and therefore it directly eliminates its first column and applies the corresponding left update to the remaining column of the blue block. The third kernel, xHBLRU, continues the application 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. Accordingly, the annihilation of each sweep can be described as, one call to the first kernel followed by a repetitive calls to a cycle 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. This stage is, in our opinion, one of the main challenges algorithms as it is difficult to track the data dependencies. The annihilation of the subsequent sweeps will generate computational tasks, which with partially overlapped data between tasks from previous sweeps (see Figure 2d) – the main challenge of dependence tracking. We have used our data translation layer (DTL) and functional dependencies [16, 23] to handle the dependencies and to provide crucial information to the runtime to achieve the correct scheduling.

4 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 tridiagonal form. The first stage reduces the original Hermitian matrix A to a band matrix by applying a two-sided transformation to A such that $A = Q_1 B Q_1^H$. Similarly, the second stage (bulge chasing) reduces the band matrix B to tridiagonal by applying the transformation from both the left and the right side to B such that $B = Q_2 T Q_2^H$. Thus, when the eigenvectors matrix Z of A are requested, the eigenvectors matrix E resulting from the eigensolver needs to be updated from the left by the Householder reflectors generated during the reduction phase, according to

$$Z = Q_1 Q_2 E = (I - V_1 T_1 V_1^H)(I - V_2 T_2 V_2^H)E, \qquad (1)$$

where (V_1, T_1) and (V_2, T_2) represent the Householder reflectors generated during the first and second reduction stages, respectively. The application of the V_2 reflectors is not as simple as the application of V_1. We begin by first describing the complexity and the design of the algorithm for applying V_2. We represent the structure of V_2 in Figure 3b. Note that these reflectors represent the annihilation of the band matrix, and thus each is of length nb – the bandwidth size. A naïve implementation would take each reflector and apply it to the matrix E. Such an implementation is memory bound, relying on Level 2 BLAS operations and thus results in poor performance. However, if we want to group them to take advantage of the efficiency of Level 3 BLAS operations, we must pay attention to the overlap between them as well as the fact that their application must follow the specific dependency order of the bulge chasing procedure in which they were created. Let us give an example that explain those issues. For sweep i (e.g., the column at position B(i,i):B(i,i+nb)), its annihilation generates a set of k Householder reflectors v_i^k, each of length nb represented in column i of the matrix V_2 depicted in Figure 3b. Similarly, the ones related to the annihilation of sweep $i+1$, are those presented in column $i+1$. They are shifted one element down compared to those of sweep i. After analyzing the dependencies of the bulge chasing procedure as explained by the example above, we notice that we can group the reflectors v_i^k from sweep i with those from sweep i+1, i+2,..., $i+l$ to apply them together using a blocked technique according to the diamond shape region as defined in Figure 3b. While each of those diamonds is considered as one block, their application needs to follow the dependency order. For example, applying the green block 4 and the red block 5 of the V_2's in Figure 3b modifies the green block row 4 and the red block row 5, respectively, of the eigenvector matrix E drawn in Figure 3c, While each of those diamonds is considered as one block, their application needs to follow the dependency order. For example, applying the green block 4 and the red block 5 of the V_2's in Figure 3b modifies the green block row 4 and the red block row 5, respectively, of the eigenvector matrix E drawn in Figure 3c, where we can easily observe the overlapped region. According to the chasing order, block 4 needs to be applied before block 5. We have drawn a sample of those dependencies by the arrows in Figure 3b. For clarity, we also represented them by the DAG in Figure 3d. By studying the pattern of dependencies of this DAG leads us to the conclusion that designing an algorithm based on such schema provides a very limited number of parallel and pipelined tasks. Despite these constraints, it is possible to compute efficiently. Namely, if we design our parallelism based on the matrix E, by splittin E by block of columns over the number of cores as shown in Figure 3c, then we can apply each diamond block independently to each portion of E. Moreover, this method does not require any data communication between cores. The overlap between each application of V's as described above increases the cache reuse. We also define the size of each block of E in a way that it is possible to fit more than one region of it in the L2 cache for increased data locality. For example, core 1 applies all the V's to the magenta block of Figure 3c, then it moves to its next assigned block, the black block. We implemented a new kernel that deals with these diamond shapes in a way that increases the cache reuse.

The application of V_1 to the resulting matrix discussed above, $G = (I - V_2 T_2 V_2^H)E$, can be done easily using our tile algorithm. First, there is no overlap between the different V_1's. Each tile of a block column of V_1's modifies different area of the matrix G, e.g., for example any tile of the magenta column of Figure 3a modifies different area of G. Thus they can be applied independently. Second, they can be blocked as shown in Figure 3a. Thus their application is also compute-intensive and involves efficient BLAS 3 kernels. The V_1's are stored in a tile fashion as shown in Figure 3a to increase data locality. The only constraint to satisfy is the application from the left meaning that the $v_{4,3}$ (black tile (4,3)) need to be applied before that the magenta $v_{4,2}$. Similary the magenta $v_{4,2}$ needs to be aplied before the blue $v_{4,1}$. The parallelism here comes on both sides, meaning that the matrix G can be viewed as a set of independent tiles to be updated

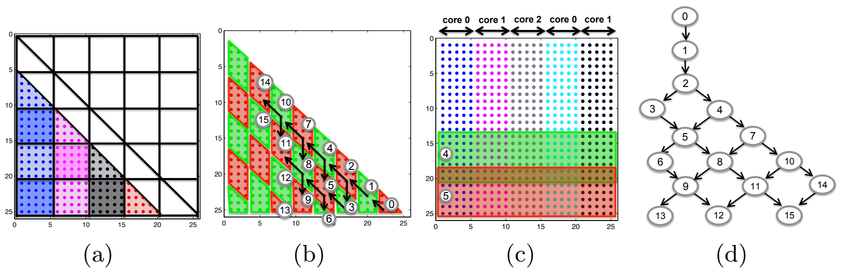


Fig. 3. (a) Tiling of V_1, (b) Blocking technique to apply V_2, (c) Distribution of the eigenvectors matrix that create independent fashion of applying Q_2 which increase locality per core, (d) Portion of the DAG showing the dependency of the V's of V_2.

and also the parallelism can be extracted from applying the V_1's as explained above. As a result, the design of the tile algorithm generates a large number of independent tasks that can be applied in an asynchronous manner using either a static or dynamic scheduler.

5 Algorithmic Complexity Study

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:

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

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 ($0 < f \leq 1$). For the two-stage approach, we need to account for both stages that result in, first, symmetric band form, and, later, tridiagonal form:

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

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): $p' \leq \min(D, p)$.

Clearly, the one stage algorithm does not scale: $\lim_{p\to\infty} t_{\text{1-s}} = 4/3n^3/\beta$ as well as the two-stage one: $\lim_{p\to\infty} t_{\text{2-s}} = 6Dn^2/(\alpha p')$. And for large problem sizes

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 1. Sample values of the parameters used in formulas.

two stage approach is superior: $\lim_{p\to\infty} \frac{t_{\text{1-s}}}{t_{\text{2-s}}} = \frac{\alpha p/\beta + 3/2 f}{1+3f}$ 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 1. The question then remains in what range of problem sizes n the two-stage algorithm is viable or for each n $t_{\text{1-s}} = t_{\text{2-s}}$. By substitution in (2) and (3) we obtain

$$n(\alpha, \beta, D, f, p) = \frac{9\beta D}{2\alpha p - 3f\beta - 2\beta} \qquad (4)$$

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

6 Performance Results

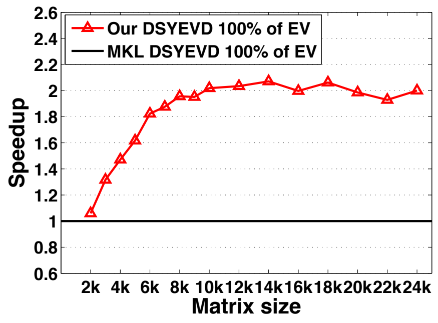
Our experiments have been performed on the largest shared memory system that we could access. It is representative of a vast class of servers and workstations commonly used for computationally intensive workloads. We benchmark all implementations on a four-socket system with AMD Opteron(tm) 6180 SE: 12 cores each (48 cores total), running at 2.5 GHz with 128 GiB of main memory, where the total number of cores is evenly spread among two physical mother boards. The cache size per core is 512 KiB. These computations are done in double precision arithmetic. The theoretical peak for this architecture in double precision is 480 Gflop/s (10.1 Gflop/s per core). There are a number of software packages that include an eigensolver. For comparison, we used the latest MKL (Math Kernel Library) [17] version 13.1, which is a commercial software from Intel that is a highly optimized programming library. It includes a comprehensive set of mathematical routines implemented to run well on x86 multicore processors. In particular, MKL includes the LAPACK-equivalent routines to compute the tridiagonal reduction DSYTRD, 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 give the reader as much information as possible. We computed the eigenpairs of a symmetric eigenvalue problem, varying the size of matrices from 2000 to 24000 using the whole 48 cores of the machine. We report the result of improvements that our two-stage implementation brings to the reduction to tridiagonal form compared against the one-stage approach from the state-of-the-art numerical linear algebra libraries. In particular, Figure 4c, shows the comparison between our implementation versus the DSYTRD routine from Intel's MKL library. It asymptotically achieves more than a 8× speedup. This results from the effcient implementation of the first stage (reduction to band) which is the compute intensive stage, and from the design of the second stage that maps both the algorithm and the data to the hardware using cache friendly kernels and scheduling based on increasing data locality. We illustrate in Figure 4a and Figure 4b the speedup obtained by our algorithm when computing all the eigenvectors using either the D&C or the MRRR as the tridiagonal eigensolver. As expected, an efficient speedup may be oberved here – our implementation is twice as fast as the optimized MKL solver. Note that the time to compute the eigenpairs (λ, Z) of the matrix A, is the sum of the time required for three phases: (1) the time to perform the tridiagonal reduction, (2) the time to compute the eigenvectors of the tridiagonal, 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. Then, phase 2 became the dominant one. It now consists of 50% of new reduced global time. This is shown in Figure 1c. Note that the time required for computing the eigenvectors of the tridiagonal matrix (phase 2) is the same as the one of the MKL solver. Therefore, reaching a two-fold speedup is worthy effort and can be considered sped-up by more than a factor of 2. We depict in Figure 4d the speed-up obtained by our algorithm when only 20% of the eigenvectors are needed. The graph here has a similar trend to the one presented in Figure 4c, it achieves more than 4× speedup. 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. For example, to find 20% of the eigenvectors of a matrix of size 20k, our algorithm requires 150 seconds while it needs 400 seconds when all of them are computed. This is one of the initial pieces of motivation to develop the two-stage algorithm.

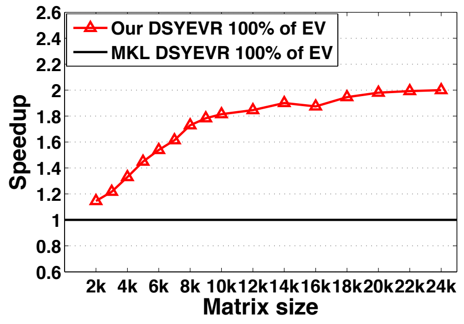
Finally, we demonstrate that our algorithm is very efficient and can achieve two-fold speedup over the well know state-of-the-art optimized librairies. It is also well suitable especially when only the eigenvalues or a portion of the eigenvectors is needed – the results show 4× to 8× speedup. We believe that this achievement makes our algorithm a very good candidate for the current and next generation of machines.

7 Conclusions and Future Work

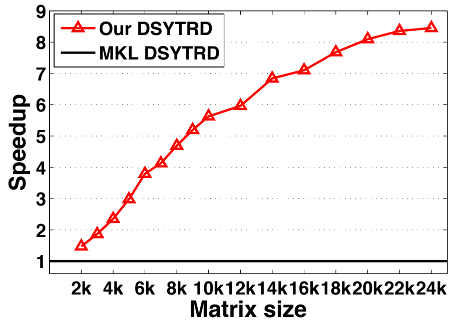
In this paper, we have presented a novel implementation of an algorithm that computes eigenvalues and eigenvectors of a symmetric or hermitian matrix. Our algorithm is based on the two-stage approach and thus performs twice as many floating operations to obtain the eigenvectors when compared with the classic approach that is in common use. Such drastic increase in operation count might have been considered a hindrance a few years back but on modern hardware it is not so. We attribute this to the formulation of the algorithm in terms more efficient kernel routines and we show the benefit both theoretically as well as through practical experiments. Instead of two-fold slow down we were able to achieve two-fold speed-up over the current breed of state-of-the-art software packages that were considered the fastest at the time of this writing. 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.



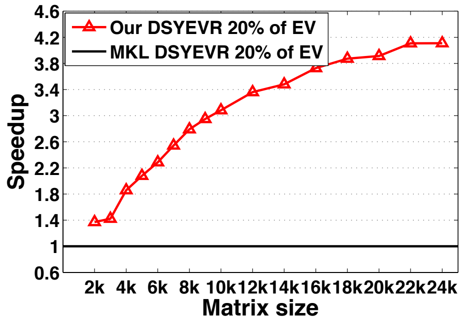
(a)



(b)



(c)



(d)

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

References

1. PLASMA. http://icl.cs.utk.edu/plasma/.
2. J. O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.
3. 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.
4. 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*. SIAM, Philadelphia, PA, 1992. http://www.netlib.org/lapack/lug/.
5. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
6. 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.*, 37(12):783–794, Dec. 2011.
7. G. Ballard, J. Demmel, and I. Dumitriu. Communication-optimal parallel and sequential eigenvalue and singular value algorithms. Technical Report EECS-2011-

14, EECS University of California, Berkeley, CA, USA, February 2011. LAPACK Working Note 237.
8. 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*, PPAM'09, pages 387–395, Berlin, Heidelberg, 2010. Springer-Verlag.
9. C. H. Bischof, B. Lang, and X. Sun. Algorithm 807: The SBR Toolbox—software for successive band reduction. *ACM TOMS*, 26(4):602–616, 2000.
10. 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.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.
11. A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov. The impact of multicore on math software. In B. Kågström, E. Elmroth, J. Dongarra, and J. Wasniewski, editors, *Applied Parallel Computing. State of the Art in Scientific Computing, 8th International Workshop, PARA*, volume 4699 of *LNCS*, pages 1–10. Springer, 2006.
12. E. Chan, E. S. Quintana-Orti, G. Quintana-Orti, and R. van de Geijn. Supermatrix out-of-order scheduling of matrix operations for smp and multi-core architectures. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 116–125, New York, NY, USA, 2007. ACM.
13. G. H. Golub and C. F. V. Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, USA, second edition, 1989.
14. R. G. Grimes and H. D. Simon. Solution of large, dense symmetric generalized eigenvalue problems using secondary storage. *ACM TOMS*, 14:241–256, September 1988.
15. 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*, SC '11, pages 8:1–8:11, New York, NY, USA, 2011. ACM.
16. 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 *SC11: International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WA, USA, November 12-18 2011.
17. Intel. Math Kernel Library. Available at http://software.intel.com/en-us/articles/intel-mkl/.
18. B. Kågström, D. Kressner, E. Quintana-Orti, and G. Quintana-Orti. Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited. *BIT Numerical Mathematics*, 48:563–584, 2008.
19. 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.
20. B. Lang. Efficient eigenvalue and singular value computations on shared memory machines. *Parallel Computing*, 25(7):845–860, 1999.
21. H. Ltaief, J. Kurzak, and J. Dongarra. Parallel band two-sided matrix bidiagonalization for multicore architectures. *IEEE TPDS*, 21(4), April 2010.
22. H. Ltaief, P. Luszczek, and J. Dongarra. High Performance Bidiagonal Reduction using Tile Algorithms on Homogeneous Multicore Architectures. *ACM TOMS*, 2011. Accepted.

23. 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, May 16-20 2011.
24. B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1980.