Solving the Generalized Symmetric Eigenvalue Problem using Tile Algorithms on Multicore Architectures[1]

Hatem LTAIEF [a,2], Piotr LUSZCZEK [b], Azzam HAIDAR [b] and Jack DONGARRA [b]

[a] *KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia*
E-mail: Hatem.Ltaief@kaust.edu.sa

[b] *Innovative Computing Laboratory, University of Tennessee, Knoxville TN USA*
Email: luszczek,haidar,dongarra@eecs.utk.edu

[c] *Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee*

[d] *School of Mathematics & School of Computer Science, University of Manchester*

Abstract. This paper proposes an efficient implementation of the generalized symmetric eigenvalue problem on multicore architecture. Based on a four-stage approach and tile algorithms, the original problem is first transformed into a standard symmetric eigenvalue problem by computing the Cholesky factorization of the right hand side symmetric definite positive matrix (first stage), and applying the inverse of the freshly computed triangular Cholesky factors to the original dense symmetric matrix of the problem (second stage). Calculating the eigenpairs of the resulting problem is then equivalent to the eigenpairs of the original problem. The computation proceeds by reducing the updated dense symmetric matrix to symmetric band form (third stage). The band structure is further reduced by applying a bulge chasing procedure, which annihilates the extra off-diagonal entries using orthogonal transformations (fourth stage). More details on the third and fourth stage can be found in Haidar et al. [Accepted at SC'11, November 2011]. The eigenvalues are then calculated from the tridiagonal form using the standard LAPACK QR algorithm (i.e., DTSEQR routine), while the complex and challenging eigenvector computations will be addressed in a companion paper. The tasks from the various stages can concurrently run in an out-of-order fashion. The data dependencies are cautiously tracked by the dynamic runtime system environment QUARK, which ensures the dependencies are not violated for numerical correctness purposes. The obtained tile four-stage generalized symmetric eigenvalue solver significantly outperforms the state-of-the-art numerical libraries (up to 21-fold speed up against multithreaded LAPACK with optimized multithreaded MKL BLAS and up to 4-fold speed up against the corresponding routine from the commercial numerical software Intel MKL) on four sockets twelve cores AMD system with a 24000×24000 matrix size.

Keywords. Generalized Symmetric Eigenvalue Problem, Tile Algorithms, Tridiagonal Reduction, Bulge Chasing, Dynamic Scheduling for Multicore Systems

[1]This work was supported in part by the Supercomputing Laboratory at King Abdullah University of Science and Technology (KAUST) and by the National Science Foundation under grant No. 0910899.

[2]Corresponding Author: Hatem Ltaief, KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia; E-mail: Hatem.Ltaief@kaust.edu.sa.

Introduction

In this paper, the authors propose to leverage and extend the contributions of the tile two-stage symmetric tridiagonal reduction (TRD) for the standard eigenvalue problem (SSEVP), first introduced in Luszczek et al. [1] and later revisited and improved by Haidar et al. [2], to tackle the challenging generalized symmetric eigenvalue problem on shared-memory multicore architecture. The generalized symmetric eigenvalue problem (GSEVP) [3] is important in a variety of scientific fields that require calculation of energy eigenstates [4,5,6]. It is noteworthy to mention that the GSEVP parallelization techniques have been also studied extensively in the context of distributed memory parallel machines [7].

Using tile algorithms, the TRD algorithm for the GSEVP is broken into fine-grained tasks linked together with data dependencies. This allows to expose the parallelism through a direct acyclic graph (DAG), where nodes are computational tasks and edges represent data dependencies between them. The final condensed form (tridiagonal) requires here a four-stage approach, as opposed to the two-stage approach of the TRD used for the SSEVP. The first stage decomposes the symmetric definite positive right hand side using the Cholesky factorization. The resulting triangular factors are then inverted and applied to the symmetric matrix of the problem. The GSEVP is now transformed to a SSEVP and can further proceed using the two-stage approach described in Haidar et al. [2]. The modified symmetric dense matrix is reduced to band tridiagonal during the third stage and finally, the fourth stage annihilates the extra off-diagonal entries of the band symmetric matrix using a bulge chasing procedure. Last but not least, the eigenvalues are calculated from the condensed tridiagonal form using the LAPACK QR iteration, implemented in the routine DTSEQR. If the eigenvectors are also required, another computational step is necessary for the back transformations, which consists in accumulating all orthogonal transformations (Householder reflectors). The eigenvector calculations will be presented in a companion paper, since the proposed algorithm in this paper (based on the bulge chasing) makes the accumulation of the orthogonal transformations complex and very challenging and therefore, deserves a research paper on its own.

This paper describes mainly threefold contributions: (1) a new four-stage approach to solve the GSEVP, (2) decoupling the algorithmic block sizes of the various stages and (3) out-of-order scheduling thanks to our dynamic runtime system environment QUARK [8]. And indeed, the problem the authors are trying to solve in this paper is far from trivial because not only the dynamic scheduler is overwhelmed by tracking data dependencies of a large number of tasks, but it has also to carefully deal with the characteristics and the heterogeneity of each stage (compute-bound vs memory-bound) in a systematic manner. For instance, while the Cholesky factorization (first stage) may require a specific tile size as its performance solely depends on the compute intensive matrix-matrix multiplication kernel, the bulge chasing (fourth stage) is a memory-bound procedure as it operates on a small chunk of data around the symmetric diagonal structure and necessitates the aggregation of neighbor tiles to create a *super* tile in order to improve data reuse. The new four-stage algorithm has been successfully integrated within the last release of PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures) [9].

The remainder of this paper is as follows: Section 1 reviews in some detail the mechanisms behind the concepts of block (i.e, LAPACK) and tile algorithms (i.e., PLASMA). Section 2 recalls the different steps toward solving solving the GSEVP and presents an efficient implementation based on tile algorithms. Section 3 describes the dynamic DAG scheduler and runtime environment QUARK. Section 4 shows some performance results. Section 5 summarizes the paper and presents future work.

1. From Block to Tile Algorithms

Block algorithms in LAPACK [10] surfaced with the emergence of cache-based architectures. They are characterized by a sequence of panel-update computational phases. The panel phase calculates all transformations using mostly memory-bound operations and applies them as a block to the trailing submatrix during the update phase. This panel-update sequence introduces unnecessary synchronization points and lookahead is prevented, while it can be conceptually achieved. Moreover, the parallelism in the block algorithms implemented in LAPACK resides in the BLAS library, which follows the expensive fork-join paradigm. Last but not least, the LAPACK library also uses the standard column-major layout from Fortran, which may not be appropriate in the current and next generation of multicore architectures.

A solution to the fork-join bottleneck in block algorithms has been presented in [11,12]. It removes the overhead seen in block algorithms due to the expensive fork-join paradigm. Based on tile algorithms, this new model is currently used in shared memory numerical libraries, such as PLASMA (University of Tennessee Knoxville) [9] and FLAME (University of Texas Austin) [13]. The approach consists of breaking the original matrix into smaller tasks that operate on a smaller block. Figure 1 describes how the column-major matrix is broken into smaller tiles (i.e., a set of b contiguous columns where b is the block size).

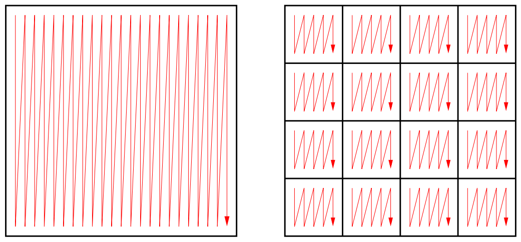

Figure 1. Translation from LAPACK Layout (column-major) to Tile Data Layout

Breaking the matrix into tiles may require a redesign of the standard numerical linear algebra algorithms. Furthermore, tile algorithms allow to bring the parallelism to the fore and expose sequential computational fine-grained tasks to benefit from any dynamic runtime system environments, which will eventually schedule the different tasks across the processing units. The actual framework boils down to scheduling a directed acyclic graph (DAG), where tasks represent nodes and edges define the data dependencies between them. This may produce an out-of-order execution and therefore, permits to remove the unnecessary synchronization points between the panel and update phases noticed in the LAPACK algorithms. Lookahead opportunities become also practical and engender a tremendous amount of concurrent tasks.

The next Section recalls the four-stage approach to solve the GSEVP and presents an implementation using tile algorithms.

2. A Four-Stage Approach for the GSEVP

The common way of stating the original GSEVP algorithm [3] is $Ax = \lambda Bx$, A *and* $B \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$, with A being a symmetric or Hermitian matrix ($A = A^T$ or $A = A^H$), B being a symmetric or Hermitian positive definite matrix, λ – an eigenvalue, and x the corresponding eigenvector. The goal is to transform the problem described above to a SSEVP by computing two preliminary steps, the Cholesky factorization $B = L \times L^T$, followed by the applications by the inverted triangular factors $C = L^{-1} \times A \times L^{-T}$. Getting the eigenpairs of the original problem is then equivalent to solving the following equation: $Cy = \lambda y$, $C \in \mathbb{R}^{n\times n}$, $y \in \mathbb{R}^n$, where $y = L^{-T}x$. The matrix C is then reduced to the tridiagonal form $T = Q^T \times C \times Q$ by successively applying orthogonal transformations using a one-stage approach, where the matrix is directly reduced to the condensed form. The eigenvalues of T are then calculated using the QR iteration (other numerical methods are also available) and if the corresponding matrix of eigenvectors X are also required, the previous transformations (from the Cholesky factorization, the tridiagonal reduction and the eigenvalue computations) need to be accumulated accordingly. This is the actual algorithm, as implemented within the state-of-the-art numerical linear algebra library LAPACK [10]. And there are clearly two issues with its current implementation, as pointed out in Section 1: (a) the fork-join paradigm, which prevents asynchronous execution and the parallelism to be explicitly exposed *within* each stage and (b) the unnecessary global synchronization points, which preempt any overlapping computations *between* stages.

The new tile four-stage GSEVP introduced in this paper permits to efficiently overcome those limitations. It computes the tile Cholesky factorization (first stage) of B [11], applies the inverted tile factors to the left and right of the general tile matrix A using a parallel triangular solve (second stage), reduces the resulting matrix symmetric C to symmetric band form (third stage) and finally, proceeds by further reducing the symmetric band structure to the final tridiagonal form T (fourth stage). While the first and second stage do not present any major difficulties, the following two stages, which reduce the symmetric matrix to tridiagonal form using tile algorithms, are the most challenging and have been recently revisited by Haidar et al. [2]. During the third stage, most of the computations are cast into Level 3 BLAS operations using compute-intensive numerical kernels. In the fourth stage, an efficient bulge chasing procedure annihilates the extra off-diagonal elements using optimized memory-bound kernels based on Level 2 BLAS operations. Furthermore, a grouping technique has been implemented for this latter stage, which consists of aggregating fine-grained and memory-aware computational tasks to improve cache reuse. Therefore, this grouping technique permits to efficiently accommodate the overall heterogeneity of the tridiagonal reduction by significantly optimizing the fourth stage (memory-bound), without impeding the performance of the third stage (compute-bound). The obtained tile tridiagonal reduction is able to achieve unprecedented performance. By extending the concepts behind the grouping technique to the first two stages (i.e., Cholesky factorization and parallel triangular solve) in the same way it has been done for the reduction to symmetric band form in [2], the algorithmic block sizes of the various stages can be decoupled. Indeed, the Cholesky factorization may require a specific tile size as its performance solely depends on the compute intensive matrix-matrix multiplication kernel. And since the bulge chasing is a memory-bound procedure as it operates on a small chunk of data around the symmetric diagonal structure, it necessitates the aggregation of neighbor tiles to build a *super* tile in order to improve data reuse and thus, avoiding expensive TLB misses. A layer of abstraction has been implemented, similar to the data dependency layer introduced in Luszczek et al. [1], in order to correctly describe to the runtime environment the new dependencies on the *super* tiles. As a result, the grouping technique permits to sustain the applications overall high performance, as seen in the Section 4. Once the condensed form is obtained, the eigenvalues are then computed using the LAPACK *DTSEQR* routine, which represents only $\theta(n^2)$ flops. The bulge chasing procedure generates many orthogonal transformations, which turns out the eigenvector matrix X computation to be very challenging. The authors believe this is beyond the scope of the paper and prefer to dedicate a companion paper to address this critical computational step. Last but not least, although the third and fourth stage have been already addressed in Haidar et al. [2], the comprehensive integration into a single framework containing four heterogeneous stages engenders non trivial challenges. For instance, the dynamic asynchronous execution generates an out-of-order scheduling of computational tasks, which may potentially engender overlapping between computational task from the different stages. It becomes then crucial to ensure the data dependencies are not violated.

The next Section describes the runtime environment system QUARK, which simultaneously tracks task data dependencies from various stages.

3. The Runtime Environment System QUARK

Restructuring linear algebra algorithms as a sequence of tasks that operate on tiles of data can remove the fork-join bottlenecks seen in block algorithms. This is accomplished by enabling out-of-order execution of tasks, which can hide the work done by the sequential tasks. Dynamic task scheduling environment called QUARK (QUeuing And Runtime for Kernels) [8], is included with the PLASMA library. A similar framework, called SuperMatrix [14], is also available inside FLAME. In order for a scheduler to be able to determine dependencies between the tasks, it needs to know how each task is using its arguments. Kernel arguments can be declared as `VALUE`, whose copies are forwarded to the task, or they may be declared as `INPUT`, `OUTPUT`, or `INOUT`. The last three designation define the dependencies between the tasks to form an implicit DAG, which is never explicitly formed in its entirety. The tasks are inserted into the scheduler's queue, which stores them to be executed when all their dependencies are satisfied. The execution of ready tasks is handled by worker threads that simply wait for tasks to become ready and execute them using a combination of default tasks assignments and work stealing. The number of tasks in the reduction is $\theta(n^3)$ and it grows very rapidly with the number of `TILES` of data. To avoid forming the entire DAG of tasks QUARK maintains a configurable window of tasks that holds the current portion of the DAG. The usage of a window of tasks has implications in how the loops of an application are unfolded and how much look ahead is available to the scheduler. Since we are working with tiles of data that should fit in the local caches on each core, QUARK provides the algorithm designer the ability to hint the cache locality behavior – a parameter in a task call can be declared with the `LOCALITY` flag. After a core executes such a task, the scheduler will assign any future tasks with the same input data to the same core. Work stealing may disrupt such cache-friendly assignment of tasks to cores.

4. Experimental Results

All experiments have been performed in real double precision and conducted on an 4 sockets, 12 core AMD Opteron(tm) Processor 6180 SE (48 cores total @ 2.5GHz) with 256 Gb of main memory. Each core has a theoretical peak of 10 Gflop/s and the whole machine 480 Gflop/s. The cache size consists of 512 kB per core. The machine is a NUMA architecture and it provides Intel Compilers 11.1 and the Intel MKL 10.2 math libraries. Figures 3-6 represent successively the DAGs of each reduction stage in the situation where synchronous execution is performed. Figure 2 highlights the asynchronous out-of-order task execution, made possible thanks to the dynamic scheduler QUARK. The synchronous and asynchronous behaviors can also be clearly identified, when looking at traces from Figures 7 and 8, respectively. Figure 9 shows timing results comparing our tile four-stage GSEVP algorithm against the equivalent functions from the state-of-the-art open-source and commercial numerical libraries i.e., LAPACK 3.2 linked with optimized MKL BLAS and Intel MKL ver. 10.2, respectively. The usual LAPACK function name given to the routine solving the GSEVP is DSYGV. DSYGV can then be decomposed into DPOTRF (first stage), DSYGST (second stage), DSYTRD (third and fourth stage) and DTSEQR (eigenvalue computations). We compare our four-stage implementation followed by DTSEQR against four different combinations: (1) LAPACK DSYGV, (2) MKL DSYGV using the Netlib implementation of the successive band reduction (SBR) toolbox [15] for the tridiagonal reduction, (3) MKL DSYGV using the one-stage DSYTRD for the tridiagonal reduction and (4) MKL DSYGV using the MKL SBR implementation for the tridiagonal reduction. Our implementation achieves up to 21-fold speed up against the open-source LAPACK software and up to 4-fold speed up against the commercial Intel MKL library.

5. Summary

This paper describes an efficient implementation of the generalized symmetric eigenvalue problem on multicore architecture using tile algorithms. The tile four-stage approach significantly outperforms the state-of-the-art numerical libraries (up to 21-fold speed up against multithreaded LAPACK with optimized multithreaded MKL BLAS and up to 4-fold speed up against the corresponding routine from the commercial numerical software Intel MKL) on four sockets twelve cores AMD system with a 24000×24000 matrix size. The authors are currently looking at exploiting hardware accelerators (e.g., GPUs) to further improve the overall performance of the tile GSEVP.

Acknowledgements

The authors would like to thank Mathieu Faverge for generating the execution traces of our algorithms.

References

[1] P. Luszczek, H. Ltaief, and J. J. Dongarra. Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures. In *International Parallel and Distributed Pro-*

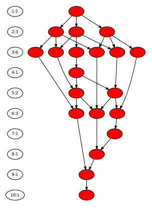
Figure 3. First Stage: Cholesky factorization.

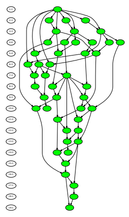
Figure 4. Second stage: Triangular solve.

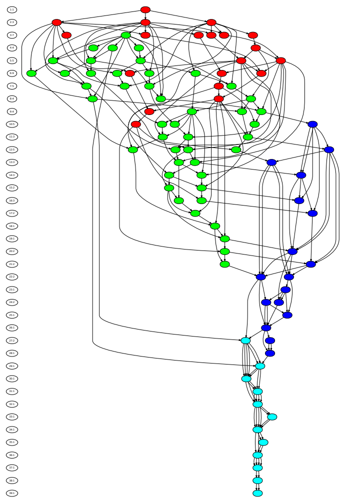
Figure 2. Merged DAG with a 4×4 tile matrix

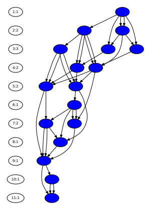
Figure 5. Third stage: Reduction to symmetric tridiagonal band form.

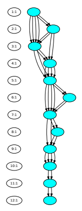
Figure 6. Fourth stage: Reduction to tridiagonal form.

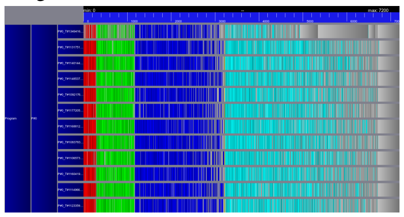
Figure 7. Synchronous execution trace on 12 cores with N= 4000.

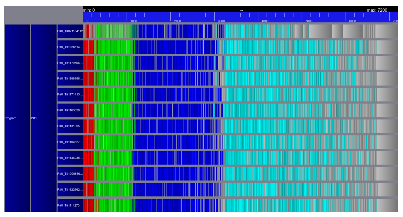
Figure 8. Asynchronous out-of-order execution trace on 12 cores with N= 4000.

cessing Symposium, Anchorage, AK, May 2011. IEEE. Innovative Computing Laboratory, University of Tennessee.

[2] A. Haidar, H. Ltaief, and J. J. Dongarra. Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems using Fine-Grained and Memory-Aware Kernels. *Accepted for publication at SC'11, University of Tennessee Computer Science Technical Report, UT-CS-11-677 (also LAPACK Working Note 254)*, 2011.

[3] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

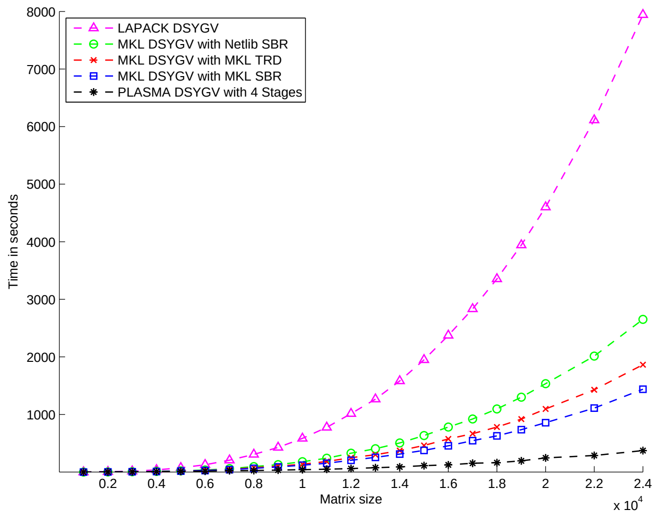

Figure 9. Elapsed Time in Seconds of the tile four-stage GSEVP on AMD Opteron 2.5 GHz with MKL BLAS 10.2

[4] M. S. Hybertsen and S. G. Louie. Electronic correlation in semiconductors and insulators: Band gaps and quasiparticle energies. *Phys. Rev. B*, 34(5390), 1986.

[5] H. N. Rojas, R. W. Godby, and R. J. Needs. Space-time method for ab-initio calculations of self-energies and dielectric response functions of solids. *Phys. Rev. Lett.*, 74(1827), 1995.

[6] W. Aulbur. *Parallel implementation of quasiparticle calculations of semiconductors and insulators*. PhD thesis, The Ohio State University, October 1996.

[7] M. P. Sears, K. Stanley, and G. Henry. Application of a high performance parallel eigensolver to electronic structure calculations. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 1998.

[8] Asim YarKhan, Jakub Kurzak, and Jack Dongarra. QUARK Users' Guide: QUeueing And Runtime for Kernels. *University of Tennessee Innovative Computing Laboratory Technical Report ICL-UT-11-02*, 2011.

[9] University of Tennessee. *PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.4.1*, June 2011.

[10] E. Anderson, Z. Bai, C. Bischof, Suzan L. Blackford, James W. Demmel, Jack J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and Danny C. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 3rd edition, 1999.

[11] A. Buttari, J. Langou, J. Kurzak, and J. J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parellel Comput. Syst. Appl.*, 35:38–53, 2009. `http://dx.doi.org/10.1016/j.parco.2008.10.002` DOI: 10.1016/j.parco.2008.10.002.

[12] E. S. Quintana-Ortí and R. A. van de Geijn. Updating an LU factorization with pivoting. *ACM Trans. Math. Softw.*, 35(2):11, 2008. `http://doi.acm.org/10.1145/1377612.1377615` DOI: 10.1145/1377612.1377615.

[13] Field G. Van Zee, Ernie Chan, Robert A. van de Geijn, Enrique S. Quintana-Orti, and Gregorio Quintana-Orti. The `libflame` library for dense matrix computations. *Computing in Science and Engineering*, 11(6):56–63, November/December 2009.

[14] Ernie Chan, Enrique S. Quintana-Orti, Gregorio Quintana-Orti, and Robert 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.

[15] Christian H. Bischof, Bruno Lang, and Xiaobai Sun. Algorithm 807: The sbr toolbox—software for successive band reduction. *ACM Trans. Math. Softw.*, 26(4):602–616, 2000.