Performance Analysis and Optimisation of Two-Sided Factorization Algorithms for Heterogeneous Platform

Khairul Kabir[1], Azzam Haidar[1], Stanimire Tomov[1], and Jack Dongarra[1,2,3]

[1] University of Tennessee,
Knoxville, TN, USA
kkabir,haidar,tomov,dongarra@eecs.utk.edu
[2] Oak Ridge National Laboratory, Oak Ridge, TN, USA.
[3] University of Manchester, Manchester, U.K.

1 Introduction

Eigenvalue and singular value decomposition (SVD) problems are fundamental for many engineering and physics applications. For example, image processing, compression, facial recognition, vibrational analysis of mechanical structures, and computing energy levels of electrons in nanostructure materials can all be expressed as eigenvalue problems. Also, the SVD plays a very important role in statistics where it is directly related to the principal component analysis method in signal processing and pattern recognition as an essential filtering tool, and in analysis of control systems. It has applications in such areas as least squares problems, computing the pseudoinverse, and computing the Jordan canonical form. In addition, the SVD is used in solving integral equations, digital image processing, information retrieval, seismic reflection tomography, and optimization. The solution of these problems can be accelerated substantially by first reducing the matrix at hand to a condensed form matrix that has the same eigenvalues as the original one. The reductions are referred to as two-sided factorizations, as they are achieved by two-sided orthogonal transformations (see Section 3). The main ones, that are also the focus of this paper, are the bidiagonalization, tridiagonalization, and the upper Hessenberg factorizations. It is challenging to accelerate the two-sided factorizations on new architectures because they are rich in Level 2 BLAS operations, which are bandwidth limited and therefore do not scale on multicore architectures and run only at a fraction of the machine's peak performance. There are techniques that can replace Level 2 BLAS operations with Level 3 BLAS. For example, in factorizations like LU, QR, and Cholesky, the application of consecutive Level 2 BLAS operations that occur in the algorithms can be delayed and accumulated so that at a later moment the accumulated transformation can be applied at once as a Level 3 BLAS. This approach totally removes Level 2 BLAS from Cholesky, and reduces its amount to $O(n^2)$ in LU, and QR, thus making it asymptotically insignificant compared to the total $O(n^3)$ amount of operations for these factorizations. The same technique can be applied to the two-sided factorizations [6], but in contrast to the one-sided, a large fraction of the total number of floating point operations (flops) still remains Level 2 BLAS. For example, the block

Hessenberg reduction has about 20% of its flops in Level 2 BLAS, while both the bidiagonal and triadiagonal reductions have 50% of their flops in Level 2 BLAS. In practice, the flops in Level 2 BLAS do not scale well on current architectures and thus can significantly impact the total execution time. Moreover, we show that the asymptotic performance (for large matrix sizes) can be modeled very accurately based on the amount of Level 2 BLAS flops.

Besides the algorithmic and performance modeling aspects related to the importance of reducing the Level 2 BLAS flops, this work is also focused on the computational challenges of developing high-performance routines for new architectures. We describe a number of optimizations that lead to performance as high as 95% of the theoretical/model peak for multicore CPUs and 80% of the model peak for Intel Xeon Phi coprocessors. These numbers are indicative for a high level of optimization achieved – note that the use of accelerators is known to achieve smaller fraction of the peak compared to non-accelerated systems, e.g., for the Top500 HPL benchmark for GPU/MIC-based supercomputers this is about 60% of the peak, and for LU on single coprocessor is about 70% of the peak [4].

2 Related Work

The earliest standard method for computing the eigenvalues of a dense nonsymmetric matrix is based on the QR iteration algorithm [7]. This schema is prohibitively expensive compared to a two phases scheme that first reduces the matrix to Hessenberg form (using either elementary or orthogonal similarity transformations), and then uses a few QR iterations to compute the eigenvalues of the reduced matrix. This two phase approach using Householder reflectors [24] was implemented in the standard EISPACK software [5]. Blocking was introduced in LAPACK, where a product of Householder reflectors $H_i = I - \tau_i v_i v_i^T$, $i = 1, \ldots, nb$ were grouped together using the so called *compact WY transform* [2, 21]:

$$H_1 H_2 \ldots H_{nb} \equiv I - VTV^T,$$

where nb is the blocking size, $V = (v_1 | \ldots | v_{nb})$, and T is $nb \times nb$ upper triangular matrix.

Alternatively to the Householder reflector approach, the use of stabilized elementary matrices for the Hessenberg reduction has been well known [18]. Later [8] proposed a new variant that reduce the general matrix further to tridiagonal form. The main motivation was that iterating with a tridiagonal form is attractive and extremely beneficial for non symmetric matrices. However, there are two major difficulties with this approach. First, the QR iteration does not maintain the tridiagonal form of a nonsymmetric matrix, and second, reducing the nonsymmetric matrix to tridiagonal by similarity transformations encounters stability and numerical issues. To overcome the first issue, [20] proposed the LR iteration algorithm which preserves the tridiagonal form. [8] proposed some recovery techniques in his paper and later [11,23] proposed another variant that reduce the nonsymmetric matrix to a similar banded form and [12] provided an error analysis of its BHESS algorithm. Blocking to the stabilized elementary reduction was introduced in [14], similar to the blocking for the LU, QR and Cholesky factorizations.

Hybrid Hessenberg, bidiagonal, and tridiagonal reduction that use both multicore CPUs and GPUs was introduced first through the MAGMA library [22]. The critical for the performance Level 2 BLAS were offloaded for execution to the high-bandwidth GPU and proper data mapping and task scheduling was applied to reduce CPU-to-GPU communications.

Recent algorithmic work on the two-sided factorizations has been concentrated on two- (or more) stage approaches. In contrast to the standard approach from LAPACK that uses a "single stage", the new ones first reduce the matrix to band form, and second, to the final form, e.g.,

tridiagonal for symmetric matrices. One of the first uses of a two-step reduction occurred in the context of out-of-core solvers for generalized symmetric eigenvalue problems [9], where a multi-stage method reduced a matrix to tridiagonal, bidiagonal, and Hessenberg forms [17]. With this approach, it was possible to recast the expensive memory-bound operations that occur during the panel factorization into a compute-bound procedure. Consequently, a framework called Successive Band Reductions was created [3]. A multi-stage approach has also been applied to the Hessenberg reduction [16] as well as the QZ algorithm [15] for the generalized non-symmetric eigenvalue problem. These approaches were also developed for hybrid GPU-CPU systems [10].

3 Background

The eigenvalue problem is to find an eigenvector x and eigenvalue λ that satisfy

$$Ax = \lambda x,$$

where A is a symmetric or nonsymmetric $n \times n$ matrix. When the entire eigenvalue decomposition is computed we have $A = X\Lambda X^{-1}$, where Λ is a diagonal matrix of eigenvalues and X is a matrix of eigenvectors. The SVD finds orthogonal matrices U, V, and a diagonal matrix Σ with nonnegative elements, such that $A = U\Sigma V^T$, where A is an $m \times n$ matrix. The diagonal elements of Σ are singular values of A, the columns of U are called its left singular vectors, and the columns of V are called its right singular vectors.

All of these problems are solved by a similar three-phase process:

1. **Reduction phase**: orthogonal matrices Q (Q and P for singular value decomposition) are applied on both the left and the right side of A to reduce it to a condensed form matrix – hence these are called "two-sided factorizations." Note that the use of two-sided orthogonal transformations guarantees that A has the same eigen/singular-values as the reduced matrix, and the eigen/singular-vectors of A can be easily derived from those of the reduced matrix (step 3);

2. **Solution phase**: an eigenvalue (respectively, singular value) solver further computes the eigenpairs Λ and Z (respectively, singular values Σ and the left and right vectors $\widetilde{U}$ and $\widetilde{V}^T$) of the condensed form matrix;

3. **Back transformation phase**: if required, the eigenvectors (respectively, left and right singular vectors) of A are computed by multiplying Z (respectively, $\widetilde{U}$ and $\widetilde{V}^T$) by the orthogonal matrices used in the reduction phase.

We performed a set of experiment in order to determine the percentage of time spent in each of these phases for the symmetric eigenvalue problem, the singular value decomposition problem, and the nonsymmetric eigenvalue problem. The results showed that the first phase is the most time consuming portion for the symmetric eigenvalue and the SVD problem. It consists of more than 80% or 90% of the total time when all eigenvectors/singular vectors or only eigenvalues/singular values are computed respectively. For the nonsymmetric eigenvalue problem, it consists about 25%. These observations illustrate the need to study and find a the possibility to improve the reduction phase. For that, we focus in this paper on the reduction phase and study its limitation. Furthermore, we propose and show how to accelerate it on Intel Xeon-Phi coprocessor architectures.

4 Performance Bound Analysis

In order to evaluate the performance behavior of the two-sided factorizations and to analyse if there are opportunities for improvments, we conduct a computational analysis of the reduction to condensed form for the three two-sided reductions (TRD, BRD, and HRD). Similar to the one-sided factorizations (LU, Cholesky, QR), the two-sided factorizations are split into a *panel factorization* and a *trailing matrix update*. Unlike the one-sided factorizations, the panel factorization requires computing Level 2 BLAS matrix-vector products involving the entire trailing matrix. This requires loading the entire trailing matrix into memory, incurring a significant amount of memory bound operations. The application of two-sided transformations creates data dependencies and produces artificial synchronization points between the panel factorization and the trailing submatrix update that prevent the use of standard techniques to increase the computational intensity of the computation, such as the look-ahead technique used extensively in the one-sided LU, QR, and Cholesky factorizations. As a result, the algorithms follow an expensive fork-and-join parallel computing model.

4.1 Flops count and its distribution in Level 2 and 3 BLAS

Performance of an algorithm can be modeled by the performances of its basic kernels. In our case, the algorithms of interest are expressed as sequence of BLAS routines, and therefore is important to determine their flops count, and more importanltly, the flops distribution in correspondingly the Level 2 and Level 3 BLAS. The blocked implementations of the reductions proceed by steps of size n_b. We give the detailed cost of step i as a cost for the *panel* and a cost for the *update*:

- The panel is of size n_b columns. The factorization of every column is primarily dominated by either one (Tridiagonal, and Hessenberg reduction) or two (Bidiagonal reduction) matrix-vector products with the trailing matrix. Thus the cost of a panel is $2\ n_b\ l^2 + \Theta(n)$ for the tridiagonal and the Hessenberg reductions, and $4\ n_b\ l^2 + \Theta(n)$ for the Bidiagonal reduction, where l is the size of the trailing matrix. For simplicity, we omit $\Theta(n)$ and roundup the cost of the panel by the cost of the matrix-vector product.

- The update of the trailing matrix consists of applying the Householder reflectors generated during the panel factorization to the trailing matrix from both the left and the right side using Level 3 BLAS routines.

  For Tridiagonal, it is updated as:
  $A_{i+n_b:n,i+n_b:n} \leftarrow A_{i+n_b:n,i+n_b:n} - V \times W^\mathsf{T} - W \times V^\mathsf{T}$ where V and W have been computed during the panel phase. This Level 3 BLAS operation is computed by the syr2k routine and its cost is $2\ n_b\ k^2$, where $k = n - i\ n_b$ is the size of the trailing matrix at step i.

  For Bidiagonal, similarly it is computed:
  $A_{i+n_b:n,i+n_b:n} \leftarrow A_{i+n_b:n,i+n_b:n} - V \times Y^\mathsf{T} - X \times U^\mathsf{T}$ where V and Y are the Householder reflectors computed during the panel phase, X and Y are two rectangular matrices needed for the update and also computed during the panel phase. This update phase can be performed by two matrix-matrix products using the gemm routine and its cost is $2 \times 2\ n_b\ k^2$ where k is the size of the trailing matrix at step i.

  For Hessenberg, the update follows three steps, first and second are the application from the right and third is the application from left:

1) $A_{1:n,i+n_b:n} \leftarrow A_{1:n,i+n_b:n} - Y \times V^{\mathsf{T}}$ using gemm,
2) $A_{1:i,i+1:i+n_b} \leftarrow A_{1:i,i+1:i+n_b} - Y_{1:i} \times V^{\mathsf{T}}$ using trmm,
3) $A_{i:n,i+n_b:n} \leftarrow A_{i:n,i+n_b:n}(I - V \times T^{\mathsf{T}} V^{\mathsf{T}})$ using larfb.
Its cost is $2\ n_b\ k\ n + n_b\ i^2 + 4\ n_b\ k\ (k + n_b)$ where k is the size of the trailing matrix at step i. Note that V, T, and Y are generated by the panel phase.

For all steps (n/n_b), the trailing matrix size varies from n to n_b by steps of size n_b, where l varies from n to n_b and k varies from $(n - n_b)$ to $2\ n_b$. Thus the total cost for the n/n_b steps is:

For Tridiagonal:

$$\begin{aligned} &\approx 2n_b \sum_{n_b}^{n/n_b} l^2 + 2n_b \sum_{2n_b}^{\frac{n-n_b}{n_b}} k^2 \\ &\approx \tfrac{2}{3}n^3_{\text{symv}} + \tfrac{2}{3}n^3_{\text{Level 3}} \\ &\approx \tfrac{4}{3}n^3. \end{aligned}$$

For Bidiagonal:

$$\begin{aligned} &\approx 4n_b \sum_{n_b}^{n/n_b} l^2 + 4n_b \sum_{2n_b}^{\frac{n-n_b}{n_b}} k^2 \\ &\approx \tfrac{4}{3}n^3_{\text{gemv}} + \tfrac{4}{3}n^3_{\text{Level 3}} \\ &\approx \tfrac{8}{3}n^3. \end{aligned}$$

For Hessenberg:

$$\begin{aligned} &\approx 2n_b \sum_{n_b}^{n/n_b} l^2 + 2n_b n \sum_{2n_b}^{\frac{n-n_b}{n_b}} k + \\ &\qquad n_b \sum_{n_b}^{n/n_b} i^2 + 4n_b n \sum_{2n_b}^{\frac{n-n_b}{n_b}} k(k+n_b) \\ &\approx \tfrac{2}{3}n^3_{\text{gemv}} + n^3_{\text{Level 3}} + \\ &\qquad \tfrac{1}{3}n^3_{\text{Level 3}} + \tfrac{4}{3}n^3_{\text{Level 3}} \approx \tfrac{10}{3}n^3. \end{aligned}$$

4.2 Performance bounds derivation

According to the equations above we derive below the maximum performance P_{max} that can be reached by any of these reduction algorithms. In particular, for large matrix sizes n, $P_{max} = \frac{number\ of\ operations}{minimum\ time\ t_{min}}$ and thus P_{max} is expressed as:

For Tridiagonal:

$$\frac{\frac{4}{3}n^3}{\frac{2}{3}n^3 * \frac{1}{P_{symv}} + \frac{2}{3}n^3 * \frac{1}{P_{Level3}}}$$

$$\frac{2 * P_{Level3} * P_{symv}}{P_{Level3} + P_{symv}}$$

$$\approx 2P_{symv} \quad when \quad P_{Level3} \gg P_{symv}. \tag{1}$$

For Bidiagonal:

$$\frac{\frac{8}{3}n^3}{\frac{4}{3}n^3 * \frac{1}{P_{gemv}} + \frac{4}{3}n^3 * \frac{1}{P_{Level3}}}$$

$$\frac{2 * P_{Level3} * P_{gemv}}{P_{Level3} + P_{gemv}}$$

$$\approx 2P_{gemv} \quad when \quad P_{Level3} \gg P_{gemv}. \tag{2}$$

For Hessenberg:

$$\frac{\frac{10}{3}n^3}{\frac{2}{3}n^3 * \frac{1}{P_{gemv}} + \frac{8}{3}n^3 * \frac{1}{P_{Level3}}}$$

$$\frac{5 * P_{Level3} * P_{gemv}}{P_{Level3} + 4 * P_{gemv}}$$

$$\approx 5P_{gemv} \quad when \quad P_{Level3} \gg P_{gemv}. \tag{3}$$

The performance of the Level 2 BLAS routines such as the matrix-vector multiplication (symv or gemv) is memory bound and very low compared to the Level 3 BLAS routines which can achieve close to the machine's peak performance. For example, on a multicore CPU system with 16 Sandy Bridge cores the performance of dgemv is about 14 Gflop/s, while for dgemm it is 323 Gflop/s. Thus, one can expect from Equations (1,2,3) that the performance of the reduction algorithms is bound by the performance of the Level 2 BLAS operations. This explains the well known low performance behavior observed for the three algorithm.

5 Performance results, analysis, and optimizations

We benchmark and study our implementations on an Intel multicore system with two 8-core Intel Xeon E5-2670 (Sandy Bridge) CPUs, running at 2.6 GHz. Each CPU has a 24 MB shared L3 cache, and each core has a private 256 KB L2 and 64 KB L1 caches. The system is equipped

with 52 GB of memory. The theoretical peak in double precision is 20.8 Gflop/s per core, giving 332 Gflop/s in total. For the accelerators experiments, we used an Intel Xeon-Phi KNC 7120 coprocessor. It has 15.1 GB, runs at 1.23 GHz, and yields a theoretical double precision peak of 1,208 Gflop/s. We used the MPSS 2.1.5889-16 software stack, the icc compiler that comes with the composer_xe_2013_sp1.2.144 suite, and BLAS implementation from MKL (Math Kernel Library) 11.01.02 [13].

5.1 Kernel optimization for CPU and Xeon Phi

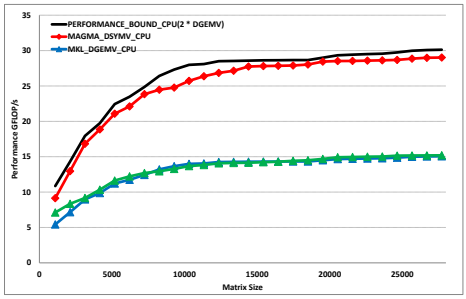
The goal of the performance analysis is to make our algorithm achieve the best performance on a specific architecture. This requires analysis of the performance obtained from the critical kernels they use (e.g., symv for TRD and gemv for BRD and HRD). We briefly describe the optimizations for the symv and gemv that we performed in order to make their performance match our expectation.

The symv kernel: Our first version of the tridiagonal reduction on CPU achieved asymptotically around 28 GFlop/s (see the brown curve for LAPACK in Figure 1b). The expected performance of symv is twice the one of gemv according to the upper bound formula developed in Equation (1). Therefore, the tridiagonal reduction should reach around 60 GFlop/s. Thus, the analysis shows that symv is not performing as expected. Instead of delivering twice the performance of gemv, it is attaining one in the same range. For that we developed an optimized implementation (magma_dsymv) that matched the expected theoretical bound. We illustrate the performance obtained from our optimized magma_dsymv and compare it to the one from the Intel MKL library in Figure 1a. We also studied the symv routine performance on the Phi coprocessor. We observed the same behavior as the CPU and developed our own optimized implementation. We depict in Figure 4a the performance obtained from MKL as well as the one achieved by our implementation.

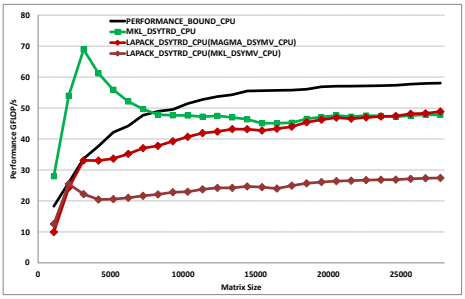
The gemv kernel: The gemv performance on the Phi is as expected – achieving around 40 GFlop/s in double precision, which translate to a bandwidth of about 160GB/s. However, the result obtained from the bidiagonal reduction (BRD) is not satisfying its upper limit defined by equation (2). According to equation (2) the performance upper bound is about twice the gemv, while our experiment shows that the BRD attains less than 40 GFlop/s. A detailed analysis of the gemv kernel showed that its performance highly depends on the location of the data in the memory, and in particular on the memory alignment. We benchmarked gemv with decreasing consecutive matrices sizes similarly to the way that the BRD reduction calls it and found out that its performance fluctuates as shown in Figure 2 (the blue curves) according to the offset from which the matrix is accessed. Thus, we proposed a fix to this problem, and developed a version that always accesses the matrix from its aligned data, performing a very small amount of extra work but keeping its performance stable at its best. The red curves of Figure 2 shows our improvment.

5.2 Tridiagonal reduction on multicore CPUs

Figure 1b shows the performance for the tridiagonal reduction on multicore. The critical for performance kernel is the dsymv. As shown, MKL's dsymv runs at about 15 GFlop/s, which translates to an achieved bandwidth of 30 GB/s. Although this kernel runs at the performance of dgemv, we expect it to be twice faster due to the symmetry. This warents an inprovement and we developed a kernel that uses the symmetry (denoted by MAGMA_DSYMV_CPU). The new kernel performs as expected (around 60 GB/s). Its effect is shown on the LAPACK implementation (LAPACK_DSYTRD_CPU) by comparing versions that use the MKL's dsymv and the new one,

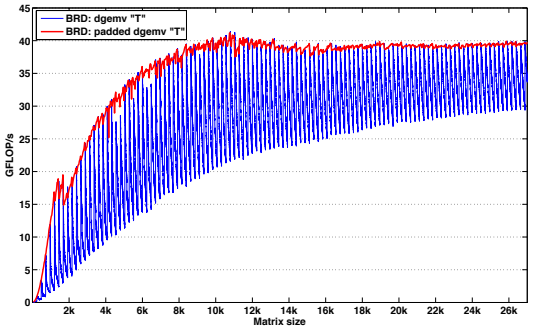


(a) Perf. of magma_dsymv on Xeon E5-2670.

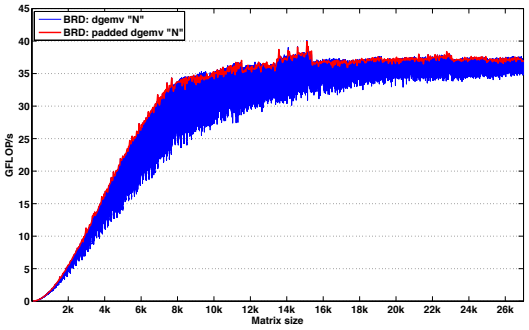


(b) Performance of dsytrd on Xeon E5-2670.

Figure 1



(a) Performance of dgemv_T at each BRD iteration w/o the proposed virtual padding.



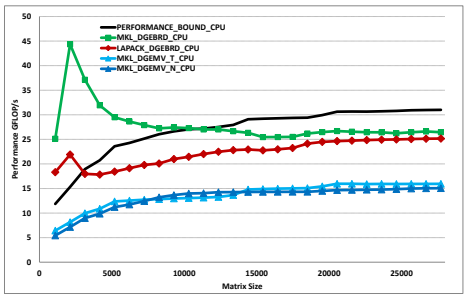
(b) Performance of dgemv_N at each BRD iteration w/o the proposed virtual padding.

Figure 2

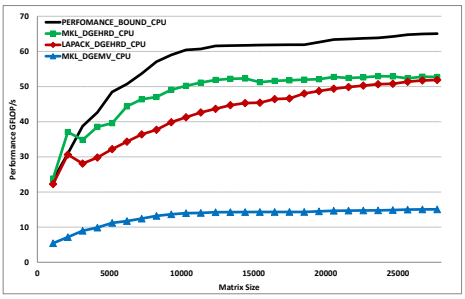
respectively. The results also show that MKL's dsytrd internally uses a well optimized dsymv. Both LAPACK and MKL reach asymptotically about 84% of the peak. Note that, the jump of performance observed for matrices of sizes between 3,000 and 4,000 is due to the fact that a large part of the matrix data at these sizes can be loaded in the L2 cache, and thus the memory bandwidth is higher, letting the Level 2 BLAS routine reach higher performance. But for larger matrix sizes no single cache level is large enough to hold the matrix data and the performance of the Level 2 BLAS is stubbornly bound by the bandwidth which limit the performance of the reduction. For example the TRD reduction is limited by 50 Gflop/s on 16 Intel Xeon E5-2670 cores, which is about twice the performance observed for our optimized magma_dsymv routine. We also would mention that the performances obtained on 8 and 16 cores are very close. Despite the fact that half of the floating point operations are done by the dsyr2k routine which benefits every additional core devoted to the computation, the performance ceiling remains low. This cannot be changed by using additional cores because the performance is bound by what can be achieved with dsymv, which is limited by the available memory bandwidth.

5.3 Bidiagonal reduction on multicore CPUs

Figure 3a shows the performance for the bidiagonal reduction on multicore. The critical for performance kernel is a combination of two gemvs – a transposed and a non-transposed. Both kernels perform as expected and the performance of the dgebrd is within 80% of the the theoretical peak. As in the other two factorizations, MKL is better for small and mid-size problems,

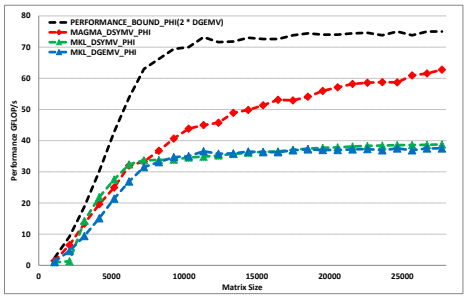


(a) Performance of dgebrd on Xeon E5-2670.

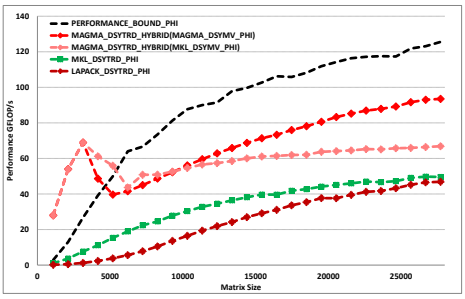


(b) Performance of dgehrd on Xeon E5-2670.

Figure 3



(a) Performance of magma_dsymv on Xeon Phi.



(b) Performance of magma_dsytrd on Xeon Phi.

Figure 4

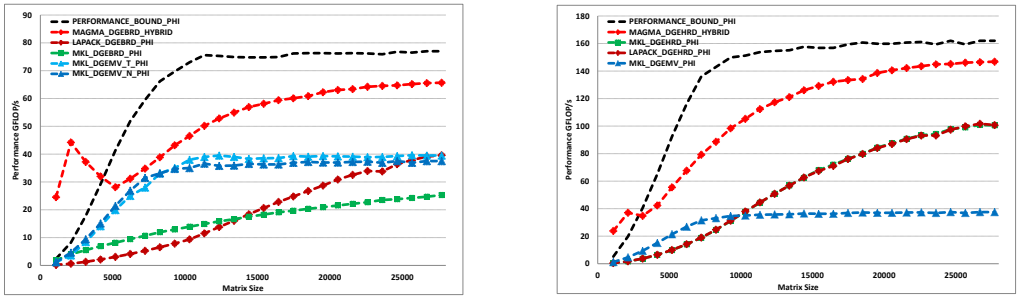
and about the same performance as LAPACK asymptoticly, for large problems.

5.4 Hessenberg reduction on multicore CPUs

Figure 3b shows the performance for the Hessenberg reduction on multicore. Compared is LAPACK (in red) *vs.* the Intel MKL implementation (in green). The result shows that MKL has optimizations for small to medium size problems. Both have about the same performance asymptotically, for large matrix sizes, reaching to about 80% of the peak. The performance of dgemv, about 16 GFlop/s, translates into an achieved bandwidth of about 64 GB/s. Still, these results are state of the art, do not show major deficiency, and therefore we consider them sufficiently high as not to warrant further optimizations in the scope of this work. The bumps in performance for small sizes are due to cache effects – matrices fitting into the L3 cache, resulting into higher bandwidth and hence improved gemv and overall gehrd performances.

5.5 Tridiagonal reductions for Xeon Phi

Figure 4b shows the performance for the tridiagonal reduction using the Xeon Phi. The native MKL and LAPACK implementations here are much slower than MAGMA. In particular native reaches about 64% of the peak, while the MAGMA is within 86%. Again, the bound is derived from equation (1) according to the BLAS performance on the MIC, and since MAGMA fallback to CPUs for small problems we observe the peak for small problems due to the CPU cache reuse effect that exceeds the theoretical bound for the MIC. We also illustrate the effect of our



(a) Performance of magma_dgebrd on Xeon Phi. (b) Performance of magma_dgehrd on Xeon Phi.

Figure 5

optimized version of symv where we can see an improvement of about 20 GFlop/s for large size and where the curve shows asymptotic behavior compared to the bound.

5.6 Bidiagonal reductions for Xeon Phi

Figure 5a shows the performance for the bidiagonal reduction on the Xeon Phi. Similarly to the tridiagonal factorization the MKL and LAPACK implementations are much slower – 51% of the peak *vs.* 86% for MAGMA. For the same reason explained above, the MAGMA performance on small size is related to cache effect of the CPU since it use exclusively the CPU for small sizes. The optimized transposed and non-transposed matrix-vector dgemv performances are as expected – reaching about 40 GFlop/s, or in terms of bandwidth, about 160 GB/s. This is consistent, even slightly exceeding, the bandwidth achieved by the STREAM benchmark [19].

5.7 Hybrid and native Hessenberg reductions for Xeon Phi

Figure 5b shows the performance for the Hessenberg reduction on the Xeon Phi. The performance bound is derived from the MIC BLAS results, i.e., it is linked to the gemv's performance on the MIC. Note that because of the small caches associated with the MIC's cores, in contrast to the multicore case, there is no bump in the performance for small problems on the MIC. The native implementations for MKL and LAPACK have the same performance, reaching up to about 62% of the performance bound peak. On the other hand, the hybrid MAGMA implementation reaches up to 91% of the peak. This may seem counterintuitive since both implementations use the same performance-critical gemv kernel, and moreover the hybrid code has the expensive CPU-to-MIC data transfers. It turns out that the fork-and-join parallelism, employed in the hybrid codes, has very high overhead on the highly-parallel MIC architectures when different BLAS kernels are called in a sequence. Indeed, benchmarks with consecutive executions of the same kernel, e.g., a sequence of gemvs, do not show overheads, while combining a gemv with other the BLAS kernels in the panel factorizations significantly slows down all kernels (including the gemv). This mixing of kernels does not exist in the hybrid code, and as a result, in spite of the extra CPU-to-MIC communications, performance is much higher.

Note that the hybrid uses the multicore CPUs for small problems, explaining the bump in performance for those sizes. The bound is exceeded since the bound is derived, as mentioned above, for the MIC-native implementation.

6 Conclusions and future work directions

We presented for a first time results on the main two-sided factorizations for Xeon Phi architectures. We used the same programming methodology for all three factorizations. We used and compared native and hybrid programming models, and developed implementations and optimizations guided by performance analysis. We achieved up to about 80% on average of the optimal performance bounds. Moreover, we showed that the hybrid implementations are 50% faster than the MIC-native implementations (in both MKL and our own LAPACK versions optimized for the Xeon Phi). We showed that only the symv kernel has to be further accelerated on MIC. We are currently exploring acceleration strategies, similarly to our improvement for the symv on multicore CPUs, which will be described in a future paper targeting low level MIC kernels. Future work is also concentrated on the development of two-stage approaches for MIC that first reduce the matrices to band-forms, followed by bulge chasing procedures to finish the reductions to the final Hessenberg, tridiagonal or bidiagonal forms.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. ACI-1339822, the Department of Energy, and Intel. The results were obtained in part with the financial support of the Russian Scientific Fund, Agreement N14-11-00190.

References

[1] 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, Third edition, 1999.

[2] C. Bischof and C. van Loan. The WY representation for products of Householder matrices. *J. Sci. Stat. Comput.*, 8:2–13, 1987.

[3] Christian H. Bischof, Bruno Lang, and Xiaobai Sun. Algorithm 807: The SBR Toolbox—software for successive band reduction. *ACM Transactions on Mathematical Software*, 26(4):602–616, 2000.

[4] J. Dongarra, M. Gates, A. Haidar, K. Kabir, P. Luszczek, S. Tomov, and I. Yamazaki. MAGMA MIC 1.3 Release: Optimizing Linear Algebra for Applications on Intel Xeon Phi Coprocessors. `http://icl.cs.utk.edu/projectsfiles/magma/pubs/MAGMA_MIC_SC14.pdf`, November 2014.

[5] Jack J. Dongarra and Cleve B. Moler. EISPACK: A package for solving matrix eigenvalue problems. chapter 4, pages 68–87. 1984.

[6] Jack J. Dongarra, Danny C. Sorensen, and Sven J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27(1-2):215 – 227, 1989. Special Issue on Parallel Algorithms for Numerical Linear Algebra.

[7] F. Francis. The QR transformation, part 2. *Computer Journal*, 4:332–345, 1961.

[8] George Geist. Reduction of a general matrix to tridiagonal form. *SIAM J. Mat. Anal. Appl*, 12:362–373, 1991.

[9] Roger G. Grimes and Horst D. Simon. Solution of large, dense symmetric generalized eigenvalue problems using secondary storage. *ACM Transactions on Mathematical Software*, 14:241–256, September 1988.

[10] A. Haidar, S. Tomov, J. Dongarra, R. Solca, and T. Schulthess. A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks. *International Journal of High Performance Computing Applications*, 28(2):196–209, May 2014.

[11] Gary W. Howell and Nadia Diaa. Algorithm 841: Bhess: Gaussian reduction to a similar banded hessenberg form. *ACM Trans. Math. Softw.*, 31(1):166–185, March 2005.

[12] Geist George Howell, Gary W. and T Rowan. Error analysis of reduction to banded hessenberg form. *Tech. Rep. ORNL/TM-13344*.

[13] Intel. Math kernel library. `https://software.intel.com/en-us/en-us/intel-mkl/`.

[14] Khairul Kabir, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Performance analysis and design of a Hessenberg reduction using stabilized blocked elementary transformations on new architectures. Computer science dept. technical report, University of Tennessee, November 2014.

[15] Bo Kågström, Daniel Kressner, Enrique Quintana-Orti, and Gregorio Quintana-Orti. Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited. *BIT Numerical Mathematics*, 48:563–584, 2008.

[16] Larss Karlsson and Bo 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.

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

[18] RogerS. Martin and J.H. Wilkinson. Similarity reduction of a general matrix to Hessenberg form. *Numerische Mathematik*, 12(5):349–368, 1968.

[19] John McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. (`http://www.cs.virginia.edu/stream/`).

[20] Heinz Rutishauser. Solution of eigenvalue problems with the LR transformation. *Nat. Bur. Standards Appl. Math. Ser.*, 49:47–81, 1958.

[21] R. Schreiber and C. van Loan. A storage-efficient WY representation for products of Householder transformations. *J. Sci. Stat. Comput.*, 10:53–57, 1991.

[22] Stanimire Tomov, Rajib Nath, and Jack Dongarra. Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Comput.*, 36(12):645–654, December 2010.

[23] E. L. Wachspress. Similarity matrix reduction to banded form. *manuscript*, 1995.

[24] J. H. Wilkinson. Householder's method for the solution of the algebraic eigenproblem. *The Computer Journal*, 3(1):23–27, 1960.