Parallel Two-Sided Matrix Reduction to Band Bidiagonal Form on Multicore Architectures

Hatem Ltaief, *Member*, *IEEE*, Jakub Kurzak, *Member*, *IEEE*, and Jack Dongarra, *Fellow*, *IEEE*

Abstract—The objective of this paper is to extend, in the context of multicore architectures, the concepts of *tile algorithms* [Buttari et al., 2007] for Cholesky, LU, and QR factorizations to the family of two-sided factorizations. In particular, the bidiagonal reduction of a general, dense matrix is very often used as a preprocessing step for calculating the Singular Value Decomposition. Furthermore, in the Top500 list of June 2008, 98 percent of the fastest parallel systems in the world were based on multicores. This confronts the scientific software community with both a daunting challenge and a unique opportunity. The challenge arises from the disturbing mismatch between the design of systems based on this new chip architecture—hundreds of thousands of nodes, a million or more cores, reduced bandwidth and memory available to cores—and the components of the traditional software stack, such as numerical libraries, on which scientific applications have relied for their accuracy and performance. The many-core trend has even more exacerbated the problem, and it becomes critical to efficiently integrate existing or new numerical linear algebra algorithms suitable for such hardware. By exploiting the concept of *tile algorithms* in the multicore environment (i.e., high level of parallelism with fine granularity and high-performance data representation combined with a dynamic data-driven execution), the band bidiagonal reduction presented here achieves 94 Gflop/s on a $12{,}000 \times 12{,}000$ matrix with 16 Intel Tigerton 2.4 GHz processors. The main drawback of the *tile algorithms* approach for the bidiagonal reduction is that the full reduction cannot be obtained in one stage. Other methods have to be considered to further reduce the band matrix to the required form.

Index Terms—Bidiagonal reduction, singular value decomposition, tile algorithms, multicores.

1 Introduction

The objective of this paper is to extend, in the context of multicore architectures, the concepts of *tile algorithms* by Buttari et al. [7] for Cholesky, LU, and QR factorizations to the family of two-sided factorizations, i.e., Hessenberg reduction, Tridiagonalization, and Bidiagonalization. In particular, the Bidiagonal Reduction (BRD) of a general, dense matrix is very often used as a preprocessing step for calculating the Singular Value Decomposition (SVD) [14], [28]:

$$A = X\,\Sigma\,Y^T,$$
$$A \in \mathbb{R}^{m\times n}, \quad X \in \mathbb{R}^{m\times m}, \quad \Sigma \in \mathbb{R}^{m\times n}, \quad Y \in \mathbb{R}^{n\times n}.$$

The necessity of calculating SVDs emerges from various computational science disciplines, e.g., in statistics, where it is related to principal component analysis; in signal processing and pattern recognition; and also in numerical weather prediction [10]. The basic idea is to transform the dense matrix A to an upper bidiagonal form B by applying successive distinct transformations from the left (U) as well as from the right (V) as follows:

$$B = U^T\,A\,V,$$
$$B \in \mathbb{R}^{n\times n}, \quad U \in \mathbb{R}^{n\times n}, \quad A \in \mathbb{R}^{n\times n}, \quad V \in \mathbb{R}^{n\times n}.$$

The most commonly used algorithm to perform this two-sided reduction is the Golub-Kahan bidiagonalization [15]. Although this algorithm works for any matrix size, it adds extra floating-point operations for rectangular matrices, and thus, faster methods such as the Lawson-Hanson-Chan bidiagonalization are preferred [8]. Here, only square matrices are considered, and performance result comparisons of different bidiagonalization algorithms for rectangular matrices will appear in a companion paper.

Also, we only look at the first stage of BRD, which goes from the original dense matrix A to a band bidiagonal matrix B_b, with b being the number of upper diagonals. The second stage, which annihilates those additional b upper diagonals, has been studied especially by Lang [21] and is not examined in this paper. This *two-stage* transformation process is also explained by Grosser and Lang [16]. Although expensive, orthogonal transformations are accepted techniques and commonly used for this reduction because they guarantee stability, as opposed to Gaussian Elimination [28]. The two common transformations are based on Householder reflectors and Givens rotations. Previous work by the authors [22] demonstrates the effectiveness of Householder reflectors over Givens rotations. Therefore, the two-sided band BRD presented in this paper is achieved by using Householder reflectors.

Furthermore, in the Top500 list of June 2008 [1], 98 percent of the fastest parallel systems in the world were based on multicores. This confronts the scientific software community with both a daunting challenge and a unique opportunity. The challenge arises from the disturbing

• *The authors are with the Innovative Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Tennessee, Suite 413 Claxton, 1122 Volunteer Blvd, Knoxville, TN 37996-3450. E-mail: {ltaief, kurzak, dongarra}@eecs.utk.edu.*

Manuscript received 6 Oct. 2008; revised 8 Apr. 2009; accepted 5 May 2009; published online 11 May 2009.
Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-10-0404.
Digital Object Identifier no. 10.1109/TPDS.2009.79.

mismatch between the design of systems based on this new chip architecture—hundreds of thousands of nodes, a million or more cores, reduced bandwidth and memory available to cores—and the components of the traditional software stack, such as numerical libraries, on which scientific applications have relied for their accuracy and performance. The many-core trend has even more exacerbated the problem, and it becomes critical to efficiently integrate existing or new numerical linear algebra algorithms suitable for such hardware. As discussed in [7], a combination of several parameters is essential to match the architecture associated with the cores:

1. fine granularity to reach a high level of parallelism and to fit the cores' small caches;
2. asynchronicity to prevent any global barriers;
3. block data layout (BDL), a high-performance data representation to perform efficient memory access; and
4. dynamic data-driven scheduler to ensure that any enqueued task can immediately be processed as soon as all their data dependencies are resolved.

While points 1 and 3 represent important items for one-sided and two-sided transformations, points 2 and 4 are even more critical for two-sided transformations because of the tremendous amount of tasks generated by the right transformation. This imposes on the scheduler even more severe constraints due to the overlapping regions produced by the left and right transformations. Indeed, as a comparison, the algorithmic complexity for the QR factorization is $4/3\,n^3$, while it is $8/3\,n^3$ for the BRD algorithm. Besides, previous work done by Kurzak et al. [19], [20] shows how the characteristics of tiled algorithms perfectly match even the architectural features of modern multicore processors such as the Cell Broadband Engine processor.

However, the main drawback of the *tile algorithms* approach for the bidiagonal reduction is that the full reduction cannot be obtained in one stage. Other methods have to be considered to further reduce the band matrix to the required form. A section in this paper will address the origin of this issue.

The remainder of this paper is organized as follows: Section 2 recalls the standard BRD algorithm. Section 3 gives a detailed overview of previous projects in this area. Section 4 describes the implementation of the parallel tiled BRD algorithm. Section 5 outlines the pros and cons of static and dynamic scheduling. Section 6 presents the performance results. Comparison tests are run on shared-memory architectures against the state-of-the-art, high-performance dense linear algebra software libraries, LAPACK [3], and ScaLAPACK [9]. Finally, Section 7 summarizes the results of this paper and presents the ongoing work.

2 The Standard Bidiagonal Reduction

In this section, we review the original BRD algorithm of a general, dense matrix.

2.1 The Sequential Algorithm

The standard BRD algorithm of $A \in \mathbb{R}^{n \times n}$ based on Householder reflectors combines two factorization methods, i.e., QR (left reduction) and LQ (right reduction) decompositions. The two phases are written as follows:

Algorithm 1. Bidiagonal Reduction with Householder reflectors

1: **for** $j = 1$ to n **do**
2: $\quad x = A_{j:n,j}$
3: $\quad u_j = \text{sign}(x_1)\ \|x\|_2\ e_1\ +\ x$
4: $\quad u_j = u_j\ /\ \|u_j\|_2$
5: $\quad A_{j:n,j:n} = A_{j:n,j:n} - 2\ u_j(u_j^* A_{j:n,j:n})$
6: $\quad$ **if** $j < n$ **then**
7: $\quad\quad x = A_{j,j+1:n}$
8: $\quad\quad v_j = \text{sign}(x_1)\ \|x\|_2\ e_1\ +\ x$
9: $\quad\quad v_j = v_j\ /\ \|v_j\|_2$
10: $\quad\quad A_{j:n,j+1:n} = A_{j:n,j+1:n} - 2(A_{j:n,j+1:n} v_j)\ v_j^*$
11: $\quad$ **end if**
12 **end for**

Algorithm 1 takes as input a dense matrix A and gives as output the upper bidiagonal decomposition. The reflectors u_j and v_j can be stored in the lower and upper parts of A, respectively, to save memory space and used later if necessary. The bulk of the computation is located in line 5 and in line 10 in which the reflectors are applied to A from the left and then from the right, respectively. Four flops are needed to annihilate one element of the matrix, which makes the total number of operations for such algorithms $8/3\,n^3$ (the lower order terms are neglected). It is obvious that Algorithm 1 is not efficient as is, especially because it is based on matrix-vector Level-2 BLAS operations. Also, a single entire column/row is reduced at a time, which engenders a large stride access to memory. The main contribution described in this paper is to transform this algorithm to work on tiles instead to generate, as many as possible, matrix-matrix Level-3 BLAS operations. First introduced by Berry et al. in [5] for the reduction of a nonsymmetric matrix to block upper Hessenberg form and then revisited by Buttari et al. in [7], this idea considerably improves data locality and cache reuse.

The following section briefly comments on the previous work done in BRD algorithms.

3 Related Work

Yotov et al. [30] describes Cache-oblivious algorithms, which allow applications to take advantage of the memory hierarchy of modern microprocessors. These algorithms are based on the divide-and-conquer paradigm each division step creates subproblems of smaller size, and when the working set of a subproblem fits in some level of the memory hierarchy, the computations in that subproblem can be executed without suffering capacity misses at that level. In this way, divide-and-conquer algorithms adapt automatically to all levels of the memory hierarchy; in fact, for problems like matrix multiplication, matrix transpose, and FFT, these recursive algorithms are optimal to within constant factors for some theoretical models of the memory hierarchy.

Grosser and Lang [16] describe an efficient parallel reduction to bidiagonal form by splitting the standard algorithm into two stages, i.e., *dense to banded* and *banded to bidiagonal*, in the context of distributed memory systems. The QR and LQ factorizations are done using a tree approach, where multiple column/row blocks can be reduced to triangular forms at the same time, which can ameliorate the overall parallel performance. However, those

triangular blocks are then reduced without taking into account their sparsity, which add some extra flops.

Ralha [25] proposed a new approach for the bidiagonal reduction called one-sided bidiagonalization. The main concept is to implicitly tridiagonalize the matrix A^TA by a one-sided orthogonal transformation of A, i.e., $F = AV$. As a first step, the right orthogonal transformation V is computed as a product of Householder reflectors. Then, the left orthogonal transformation U and the bidiagonal matrix B are computed using a Gram-Schmidt QR factorization of the matrix F. This procedure has numerical stability issues and the matrix U might loose its orthogonality properties.

Barlow et al. [4] and later, Bosner and Barlow [6], further improved the stability of the one-sided bidiagonalization technique by merging the two distinct steps to compute the bidiagonal matrix B. The computation process of the left and right orthogonal transformations is now interlaced. Within a single reduction step, their algorithms simultaneously perform a block Gram-Schmidt QR factorization (using a recurrence relation) and a postmultiplication of a block of Householder reflectors chosen under a special criterion.

4 The Parallel Reduction to Band Bidiagonal Form

In this section, we present the parallel implementation of the band BRD algorithm based on Householder reflectors.

4.1 Descriptions of the Fast Elementary Operations

There are eight overall kernels implemented for the two phases, four for each phase.

For phase 1 (left reduction), the first four kernels are identical to the ones used by Buttari et al. [7] for the QR factorization in which the reflectors are stored in column major form. DGEQRT is used to do a QR-blocked factorization using the WY technique for efficiently accumulating the Householder reflectors [26]. The DLARFB kernel comes from the LAPACK distribution and is used to apply a block of Householder reflectors. DTSQRT performs a block QR factorization of a matrix composed of two tiles, a triangular tile on top of a dense square tile. DSSRFB updates the matrix formed by coupling two square tiles and applying the resulting DTSQRT transformations. Buttari et al. give a detailed description of the different kernels [7].

For phase 2 (right reduction), the reflectors are now stored in rows. DGELQT is used to do an LQ-blocked factorization using the WY technique as well. DTSLQT performs a block LQ factorization of a matrix composed of two tiles, a triangular tile beside a dense square tile. However, minor modifications are needed for the DLARFB and DSSRFB kernels. These kernels now take into account the row storage of the reflectors.

Moreover, since the right orthogonal transformations do not destroy the zero structure and do not introduce fill-in elements, the computed left and right reflectors can be stored in the lower and upper annihilated parts of the original matrix, for later use. Although the algorithm works for rectangular matrices, for ease of presentation, only square matrices are considered. Let NBT be the number of tiles in each direction. Then, the tiled band BRD algorithm with Householder reflectors appears as in Algorithm 2. It basically performs a sequence of interleaved QR and LQ factorizations at each step of the reduction.

Algorithm 2. Tiled Band BRD Algorithm with Householder reflectors.

```
 1: for i = 1, 2 to NBT do
 2:    {QR Factorization}
 3:    DGEQRT(i, i, i)
 4:    for j = i+1 to NBT do
 5:       DLARFB(''L'', i, i ,j)
 6:    end for
 7:    for k = i+1 to NBT do
 8:       DTSQRT(i, k, i)
 9:       for j = i+1 to NBT do
10:          DSSRFB(''L'', i, k, j)
11:       end for
12:    end for
13:    if i < NBT then
14:       {LQ Factorization}
15:       DGELQT(i, i, i+1)
16:       for j = i+1 to NBT do
17:          DLARFB(''R'', i, j, i+1)
18:       end for
19:       for k = i+2 to NBT do
20:          DTSLQT(i, i, k)
21:          for j = i+1 to NBT do
22:             DSSRFB(''R'', i, j, k)
23:          end for
24:       end for
25:    end if
26: end for
```

The characters "L" and "R" stand for the Left and Right updates. In each kernel call, the triplets (i, ii, iii) specify the tile location in the original matrix, as in Fig. 1: i) corresponds to the reduction step in the general algorithm, ii gives the row index, and iii) represents the column index. For example, in Fig. 1a, the black tile is the input dependency at the current step, the white tiles are the zeroed tiles, the bright gray tiles are those which need to be processed, and finally, the dark gray tile corresponds to DTSQRT(1,4,1). In Fig. 1b, the striped tile represents the final data tile and the dark gray tile is DTSLQT(1,1,4). In Fig. 1c, the reduction is at step 3, where the dark gray tiles represent DSSRFB("L," 3,4,4). In Fig. 1d, the dark gray tiles represent DSSRFB("R," 3,4,5).

These kernels are very rich in matrix-matrix operations. By working on small tiles with BDL, the elements are stored contiguous in memory, and thus, the access pattern to memory is more regular, which makes these kernels high performing. It appears necessary then to efficiently schedule the kernels to get high performance in parallel.

The next section describes the number of operations needed to apply this reduction.

4.2 Algorithmic Complexity

The algorithmic complexity for the band BRD is split into two phases: QR factorization and a band LQ factorization. The total number of flops is then $8/3n^3 + 2n^2 - 4n^2b$ (the lower order terms are ignored) with b being the tile size (equivalent to the bandwidth of the matrix). Compared to the full BRD reduction complexity, i.e., $8/3n^3 + 2n^2$, the band BRD

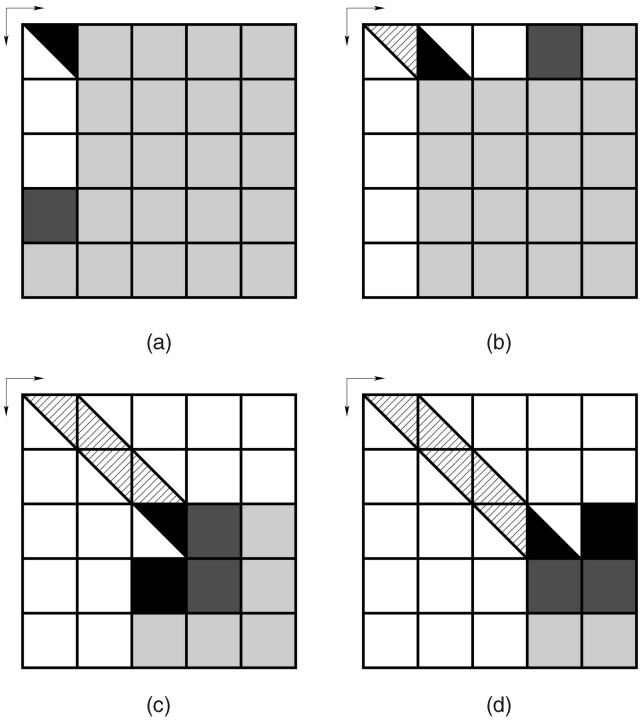


Fig. 1. BRD algorithm applied on a tiled matrix with $\mathrm{NBT} = 5$. (a) BRD: left reduction step 1. (b) BRD: right reduction step 1. (c) BRD: left reduction step 3. (d) BRD: right reduction step 3.

algorithm is doing $O(n^2 b)$ less flops, which is a negligible expense of the overall BRD algorithm cost provided $n \gg b$.

Furthermore, by using updating factorization techniques as suggested in [14], [27], the kernels for both implementations can be applied to tiles of the original matrix. Using updating techniques to tile, the algorithms were first proposed by Yip [29] for LU to improve the efficiency of out-of-core solvers, and were recently reintroduced in [17], [23] for LU and QR, once more in the out-of-core context. The cost of these updating techniques is an increase in the operation count for the whole BRD reduction. However, as suggested in [11], [12], [13], by setting up inner blocking within the tiles during the panel factorizations and the trailing submatrix update, DGEQRT-DGELQT-DTSQRT-DTSLQT kernels and DLARFB-DSSRFB kernels, respectively, those extra flops become negligible provided $s \ll b$, with s being the inner blocking size (see Buttari et al. [7] for further information). This blocking approach has been also described in [17], [24].

However, it is noteworthy to mention the high cost of reducing the band bidiagonal matrix to the full bidiagonal matrix. Indeed, using technics such as bulge chasing to reduce the band matrix is very expensive and may dramatically slow down the overall algorithms. Another approach would be to apply the Divide-and-Conquer (SVD) on the band matrix but this strategy is still under investigation.

The next section explains the limitation origins of the *tile algorithms* concept for two-sided transformations, i.e., the reduction achieved only to band form.

4.3 Limitations of Tile Algorithms Approach for Two-Sided Transformations

The concept of *tile algorithms* is very suitable for one-sided methods (i.e., Cholesky, LU, QR, and LQ). Indeed, the transformations are only applied to the matrix from one side. With the two-sided methods, the right transformation needs to preserve the reduction achieved by the left transformation. In other words, the right transformation should not destroy the zeroed structure by creating fill-in elements. That is why, the only way to keep intact the obtained structure is to perform a shift of a tile in the adequate direction. For the BRD, we decided to shift one tile right from the top-left corner of the matrix. The final matrix shape is an upper block bidiagonal structure. We could have also performed the shift one tile bottom from the top-left corner of the matrix and the final structure of the matrix would be lower block bidiagonal.

In the following part, we present a comparison of two approaches for tile scheduling, i.e., a static and a dynamic data-driven execution scheduler that ensures that the small kernels (or tasks) generated by Algorithm 2 are processed as soon as their respective dependencies are satisfied.

5 Static Scheduling versus Dynamic Scheduling

Two types of schedulers were used, a dynamic one, where scheduling decisions are made at runtime, and a static one, where the schedule is predetermined.

The dynamic scheduling scheme similar to [7] has been extended for the two-sided orthogonal transformations. A Directed Acyclic Graph (DAG) is used to represent the data flow between the nodes/kernels. While the DAG is quite easy to draw for a small number of tiles, it becomes very complex when the number of tiles increases and it is even more difficult to process than the one created by the one-sided orthogonal transformations. Indeed, the right updates impose severe constraints on the scheduler by filling up the DAG with multiple additional edges. The dynamic scheduler maintains a central progress table, which is accessed in the critical section of the code and protected with mutual exclusion primitives (POSIX mutexes in this case). Each thread scans the table to fetch one task at a time for execution. As long as there are tasks with all dependencies satisfied, the scheduler will provide them to the requesting threads and will allow an out-of-order execution. The scheduler does not attempt to exploit data reuse between tasks. The centralized nature of the scheduler is inherently nonscalable with the number of threads. Also, the need for scanning potentially large table window, in order to find work, is inherently nonscalable with the problem size. However, this organization does not cause performance problems for the numbers of threads, problem sizes, and task granularities investigated in this paper.

The static scheduler used here is a derivative of the scheduler used successfully in the past to schedule Cholesky and QR factorizations on the Cell processor [18], [20]. The static scheduler imposes a linear order on all the tasks in the factorization. Each thread traverses the tasks space in this order picking a predetermined subset of tasks for execution. In the phase of applying transformations from the right, each thread processes one block-column of the matrix; In the phase of applying transformations from the left, each thread processes one block-row of the matrix (Fig. 2). A dependency check is performed before executing each task. If dependencies are not satisfied, the thread stalls until they are (implemented by busy waiting). Dependencies are tracked by a progress table, which contains global progress information and is replicated on all threads. Each thread calculates the

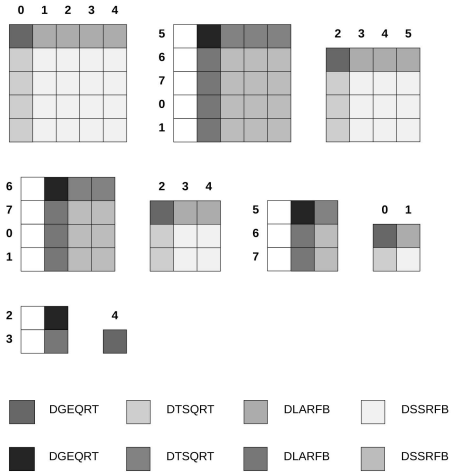


Fig. 2. Task Partitioning with eight cores on a 5×5 tile matrix.

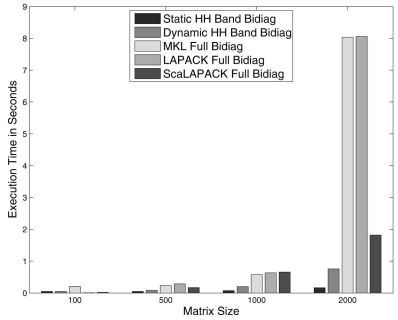
task traversal locally and checks dependencies by polling the local copy of the progress table. Due to its decentralized nature, the mechanism is much more scalable and of virtually no overhead. Also, processing of tiles along columns and rows provides for greater data reuse between tasks to which the authors attribute the main performance advantage of the static scheduler. Since the dynamic scheduler is more aggressive in fetching of tasks, it completes each step of the factorization faster. The static scheduler, on the other hand, takes longer to complete a given step of the factorization, but successfully overlaps consecutive steps achieving the pipelining effect, what leads to very good overall performance (Fig. 3).

In the next section, we present the experimental results comparing our band BRD implementations with the two schedulers against the state-of-the-art libraries, i.e., LAPACK [3], ScaLAPACK [9], and MKL version 10 [2].

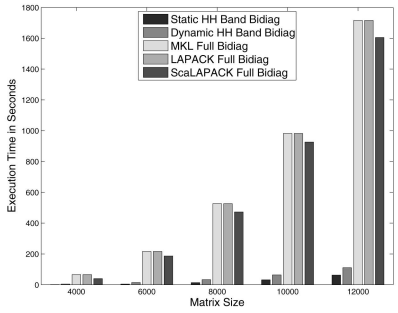
6 Experimental Results

The experiments have been achieved on two different platforms: a quad-socket dual-core Intel Itanium 2 1.6 GHz (8 total cores) with 16 GB of memory, and a quad-socket quad-core Intel Tigerton 2.4 GHz (16 total cores) with 32 GB of memory. Hand tuning based on empirical data has been performed for large problems to determine the optimal tile size $b = 200$ and inner blocking size $s = 40$ for the tiled band BRD algorithm. The block sizes for LAPACK and ScaLAPACK have also been hand tuned to get a fair comparison,

Fig. 3. Scheduler tracing with six Intel Tigerton 2.4 GHz cores: top dynamic—bottom static.



(a)



(b)

Fig. 4. Elapsed time in seconds for the band bidiagonal reduction on a dual-socket quad-core Intel Itanium2 1.6 GHz with MKL BLAS V10.0.1. (a) Small data size. (b) Large data size.

$b = 32$ and $b = 64$, respectively. The authors understand that it may not be a fair comparison to do against those latter libraries, since the reduction is completely achieved in that case. The purpose of showing such performance curves is only to give a rough idea in term of elapsed time and performance, of the whole reduction process.

Fig. 4 shows the elapsed time in seconds for small and large matrix sizes on the Itanium system with eight cores. The band BRD algorithms based on Householder reflectors with static scheduling are slightly better than with dynamic scheduling. However, both implementations by far outperform the others. Fig. 5a presents the parallel performance in Gflop/s of the band BRD algorithm on the Itanium system. The algorithm with dynamic scheduling runs at 82 percent of the machine theoretical peak of the system and at 92 percent of the DGEMM peak. Fig. 5b zooms in on the three other implementations, and the parallel performance of the full BRD with ScaLAPACK is significantly higher than the full BRD of LAPACK and MKL for small matrix sizes. Also, the performances are almost the same for larger matrix sizes.

The same experiments have been conducted on the Xeon system with 16 cores. Fig. 6 shows the execution time in seconds for small and large matrix sizes. Again, both band BRD algorithms almost behave in the same manner and outperform the other libraries. Fig. 7a presents the parallel performance in Gflop/s of the band BRD algorithm. It scales quite well while the matrix size increases, reaching 94 Gflop/s. It runs at 61 percent of the system theoretical peak and 72 percent of the DGEMM peak. The zoom-in seen in Fig. 7b highlights the weakness of the full BRD algorithm of MKL, LAPACK, and ScaLAPACK. Note that the full BRD of ScaLAPACK is twice as fast as than the full BRD of MKL and LAPACK most likely due to the two-dimensional block cyclic distribution.

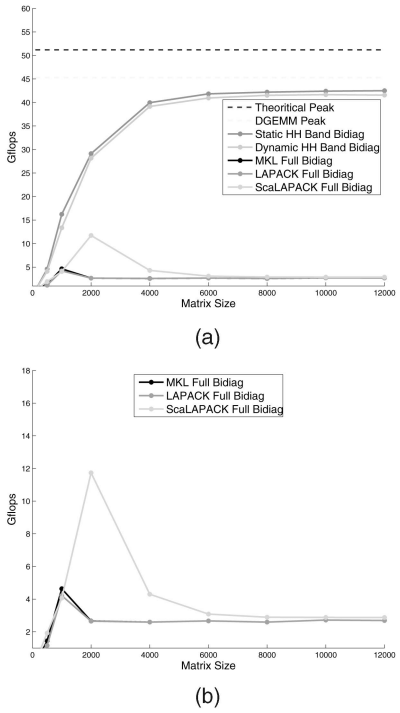


Fig. 5. Parallel performance of the band bidiagonal reduction on a dual-socket quad-core Intel Itanium2 1.6 GHz processor with MKL BLAS V10.0.1. (a) Performance comparisons. (b) Zoom-in.

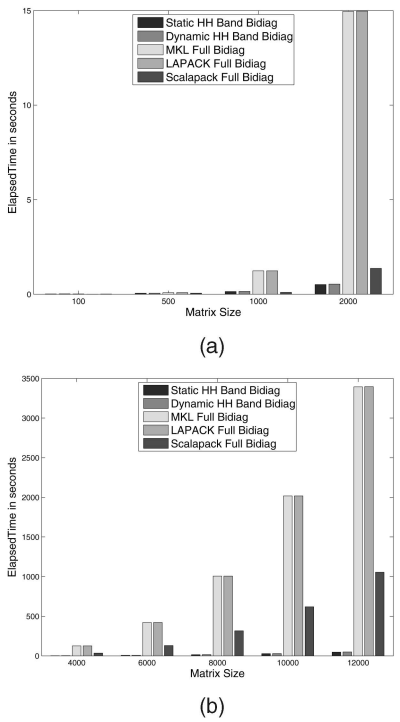


Fig. 6. Elapsed time in seconds for the band bidiagonal reduction on a quad-socket quad-core Intel Xeon 2.4 GHz processor with MKL BLAS V10.0.1. (a) Small data size. (b) Large data size.

7 Conclusion and Future Work

By exploiting the concepts of *tile algorithms* in the multicore environment, i.e., high level of parallelism with fine granularity and high-performance data representation combined with a dynamic data-driven execution, the BRD algorithm with Householder reflectors achieves 94 Gflop/s on a $12{,}000 \times 12{,}000$ matrix size with 16 Intel Tigerton 2.4 GHz processors. This algorithm performs most of the operations in Level-3 BLAS. Although the algorithm considerably surpasses in performance of the BRD algorithm of MKL, LAPACK, and ScaLAPACK, its main inefficiency comes from the implementation of the kernel operations. The most performance critical, DSSRFB, kernel only achieves roughly 61 percent of peak for the tile size used ($b = 200$) in the experiments. For comparison, a simple call to the DGEMM routine easily crosses 85 percent of the peak. Unlike DGEMM, however, DSSRFB is not a single call to BLAS, but is composed of multiple calls to BLAS in a loop (due to the inner blocking), hence, the inefficiency. DSSRFB could easily achieve similar performance if implemented as a monolithic code and heavily optimized. Finally, this work can be extended to the BRD of any matrix sizes (m, n) by using the appropriate method depending on the ratio between both dimensions.

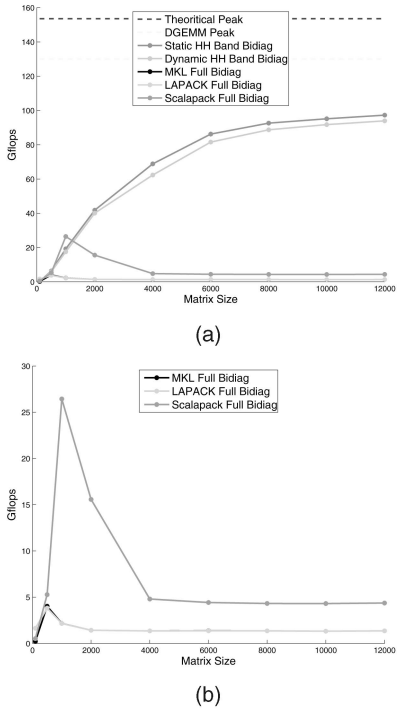


Fig. 7. Parallel performance of the band bidiagonal reduction on a quad-socket quad-core Intel Xeon 2.4 GHz processor with MKL BLAS V10.0.1. (a) Performance comparisons. (b) Zoom-in.

Acknowledgments

The authors thank Alfredo Buttari for his insightful comments, which greatly helped to improve the quality of this paper. This work was supported in part by grants from the US National Science Foundation (NSF) and the US Department of Energy (DoE).

References

[1] http://www.top500.org, 2009.
[2] http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm, 2009.

[3] 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,* third ed. Soc. Industrial and Applied Math., 1999.

[4] J.L. Barlow, N. Bosner, and Z. Drmač, "A New Stable Bidiagonal Reduction Algorithm," *Linear Algebra and Its Applications,* vol. 397, no. 1, pp. 35-84, Mar. 2005.

[5] M.W. Berry, J.J. Dongarra, and Y. Kim, "LAPACK Working Note 68: A Highly Parallel Algorithm for the Reduction of a Nonsymmetric Matrix to Block Upper-Hessenberg Form," Technical Report UT-CS-94-221, Dept. of Computer Science, Univ. of Tennessee, Feb. 1994.

[6] N. Bosner and J.L. Barlow, "Block and Parallel Versions of One-Sided Bidiagonalization," *SIAM J. Matrix Analysis and Applications,* vol. 29, no. 3, pp. 927-953, 2007.

[7] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "Parallel Tiled QR Factorization for Multicore Architectures," *Concurrency and Computation,* vol. 20, no. 13, pp. 1573-1590, 2008.

[8] T.F. Chan, "An Improved Algorithm for Computing the Singular Value Decomposition," *ACM Trans. Math. Software,* vol. 8, no. 1, pp. 72-83, Mar. 1982.

[9] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, "ScaLAPACK, a Portable Linear Algebra Library for Distributed Memory Computers-Design Issues and Performance," *Computer Physics Comm.,* vol. 97, nos. 1/2, pp. 1-15, 1996.

[10] D.M. Christopher, K. Eugenia, and M. Takemasa, "Estimating and Correcting Global Weather Model Error," *Monthly Weather Rev.,* vol. 135, no. 2, pp. 281-299, 2007.

[11] E. Elmroth and F.G. Gustavson, "New Serial and Parallel Recursive QR Factorization Algorithms for SMP Systems," *Proc. Fourth Int'l Workshop Applied Parallel Computing, Large Scale Scientific and Industrial Problems (PARA '98),* pp. 120-128, June 1998.

[12] E. Elmroth and F.G. Gustavson, "Applying Recursion to Serial and Parallel QR Factorization Leads to Better Performance," *IBM J. Research and Development,* vol. 44, no. 4, pp. 605-624, 2000.

[13] E. Elmroth and F.G. Gustavson, "High-Performance Library Software for QR Factorization," *Proc. Fifth Int'l Workshop, Applied Parallel Computing, New Paradigms for HPC in Industry and Academia (PARA '00),* pp. 53-63. June 2000, http://dx.doi.org/10.1007/3-540-70734-4_9.

[14] G.H. Golub and C.F. Van Loan, *Matrix Computation,* John Hopkins Studies in the Math. Sciences, third ed. Johns Hopkins Univ. Press, 1996.

[15] G.H. Golub and W. Kahan, "Calculating the Singular Values and the Pseudo Inverse of a Matrix," *SIAM J. Numerical Analysis,* vol. 2, pp. 205-224, 1965.

[16] B. Grosser and B. Lang, "Efficient Parallel Reduction to Bidiagonal Form," *Parallel Computing,* vol. 25, no. 8, pp. 969-986, 1999.

[17] B.C. Gunter and R.A. van de Geijn, "Parallel Out-of-Core Computation and Updating of the QR Factorization," *ACM Trans. Math. Software,* vol. 31, no. 1, pp. 60-78, Mar. 2005.

[18] J. Kurzak, A. Buttari, and J.J. Dongarra, "Solving Systems of Linear Equation on the CELL Processor Using Cholesky Factorization," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 9, pp. 1175-1186, Sept. 2008.

[19] J. Kurzak, A. Buttari, and J.J. Dongarra, "Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 9, pp. 1-11, Sept. 2008.

[20] J. Kurzak and J.J. Dongarra, "QR Factorization for the CELL Processor," *J. Scientific Programming,* special issue on high performance computing on CELL B.E. processors, pp. 1-12, 2008.

[21] B. Lang, "Parallel Reduction of Banded Matrices to Bidiagonal Form," *Parallel Computing,* vol. 22, no. 1, pp. 1-18, 1996.

[22] H. Ltaief, J. Kurzak, and J. Dongarra, "LAPACK Working Note 208: Parallel Block Hessenberg Reduction Using Algorithms-by-Tiles for Multicore Architectures Revisited," Technical Report UT-CS-08-624, Dept. of Computer Science, Univ. of Tennessee, Aug. 2008.

[23] E.S. Quintana-Ortí and R.A. van de Geijn, "Updating an LU Factorization with Pivoting," *ACM Trans. Math. Software,* vol. 35, no. 2, July 2008.

[24] G. Quintana-Ortí, E.S. Quintana-Ortí, E. Chan, R.A. van de Geijn, and F.G. Van Zee, "Scheduling of QR Factorization Algorithms on SMP and Multi-Core Architectures," *Proc. Int'l Conf. Parallel, Distributed and Network-Based Processing (PDP),* pp. 301-310, 2008.

[25] R. Ralha, "One-Sided Reduction to Bidiagonal Form," *Linear Algebra and Its Applications,* vol. 358, pp. 219-238, Jan. 2003.

[26] R. Schreiber and C. Van Loan, "A Storage Efficient WY Representation for Products of Householder Transformations," *SIAM J. Scientific and Statistical Computing,* vol. 10, pp. 53-57, 1989.

[27] G.W. Stewart, *Matrix Algorithms Volume I: Matrix Decompositions.* SIAM, 1998.

[28] L.N. Trefethen and D. Bau, *Numerical Linear Algebra.* SIAM, 1997.

[29] E.L. Yip, "Fortran Subroutines for Out-of-Core Solutions of Large Complex Linear Systems," Technical Report CR-159142, NASA, Nov. 1979.

[30] K. Yotov, T. Roeder, K. Pingali, J. Gunnels, and F. Gustavson, "An Experimental Comparison of Cache-Oblivious and Cache-Conscious Programs," *Proc. 19th Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '07),* pp. 93-104, 2007.

Hatem Ltaief received the first MSc degree from the School of Engineering at the University of Claude Bernard Lyon I, France, and the second MSc degree in applied mathematics and the PhD degree in computer science from the University of Houston. He is a research associate in the Innovative Computing Laboratory in the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. His research interests include parallel algorithms, specifically in the area of numerical linear algebra, and also parallel programming models and performance optimization for parallel architectures spanning distributed and shared memory systems, as well as next generation multicore and many-core processors. He is a member of the IEEE.

Jakub Kurzak received the MSc degree in electrical and computer engineering from Wroclaw University of Technology, Poland, and the PhD degree in computer science from the University of Houston. He is a research associate in the Innovative Computing Laboratory in the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. His research interests include parallel algorithms, specifically in the area of numerical linear algebra, and also parallel programming models and performance optimization for parallel architectures spanning distributed and shared memory systems, as well as next generation multicore and many-core processors. He is a member of the IEEE.

Jack Dongarra received the bachelor of science degree in mathematics from Chicago State University in 1972, the master of science degree in computer science from the Illinois Institute of Technology in 1973, and the PhD degree in applied mathematics from the University of New Mexico in 1980. He worked at the Argonne National Laboratory until 1989, becoming a senior scientist. He now holds an appointment as the university distinguished professor of computer science in the Department of Electrical Engineering and Computer Science at the University of Tennessee, has the position of a distinguished research staff member in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL), turing fellow in the Computer Science and Mathematics Schools at the University of Manchester, and an adjunct professor in the Computer Science Department at Rice University. He is a fellow of the IEEE.