Bidiagonalization with Parallel Tiled Algorithms

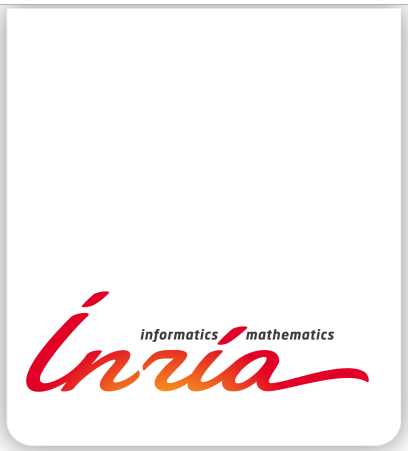
Mathieu Faverge, Julien Langou, Yves Robert, Jack Dongarra

Bidiagonalization with Parallel Tiled Algorithms

Mathieu Faverge, Julien Langou, Yves Robert, Jack Dongarra

RESEARCH REPORT
N° 8969
October 2016
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8969--FR+ENG



Bidiagonalization with Parallel Tiled Algorithms

Mathieu Faverge*†, Julien Langou‡, Yves Robert†§, Jack Dongarra†¶

Project-Team ROMA

Research Report n° 8969 — October 2016 — 38 pages

Abstract: We consider algorithms for going from a "full" matrix to a condensed "band bidiagonal" form using orthogonal transformations. We use the framework of "algorithms by tiles". Within this framework, we study: (i) the tiled bidiagonalization algorithm BiDiag, which is a tiled version of the standard scalar bidiagonalization algorithm; and (ii) the R-bidiagonalization algorithm R-BiDiag, which is a tiled version of the algorithm which consists in first performing the QR factorization of the initial matrix, then performing the band-bidiagonalization of the R-factor. For both bidiagonalization algorithms BiDiag and R-BiDiag, we use four main types of reduction trees, namely FlatTS, FlatTT, Greedy, and a newly introduced auto-adaptive tree, Auto. We provide a study of critical path lengths for these tiled algorithms, which shows that (i) R-BiDiag has a shorter critical path length than BiDiag for tall and skinny matrices, and (ii) Greedy based schemes are much better than earlier proposed variants with unbounded resources. We provide experiments on a single multicore node, and on a few multicore nodes of a parallel distributed shared-memory system, to show the superiority of the new algorithms on a variety of matrix sizes, matrix shapes and core counts.

Key-words: bidiagonalization, R-bidiagonalization, critical path, greedy algorithms, auto-adaptive reduction tree, tall and skinny matrix.

* Bordeaux INP, CNRS, Inria et Université de Bordeaux, France
† University of Tennessee, Knoxville TN, USA
‡ University of Colorado Denver, USA
§ Equipe-projet ROMA, Laboratoire LIP, École Normale Supérieure de Lyon & Inria, France
¶ University of Manchester, UK

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Algorithmes parallèles pour la bi-diagonalisation de matrices

Résumé : Dans ce rapport, nous concevons et évaluons des algorithmes parallèles pour la bidiagonalisation de matrices. Ces algorithmes travaillent avec des tuiles (sous-blocs de la matrice) plutôt qu'élément par éément. Nous étudions et comparons la bidiagonalisation et la R-bidiagonalisation. Dans cette dernière, on commence par une factorisation QR pour se ramener à une matrice carrée. Nous établissons plusieurs formules pour les chemins critiques de nos algorithmes. Nous rendons compte des résultats d'expériences menées sur un noeud multi-coeur (en mémoire partagée) et sur plusieurs noeuds multi-coeurs (en mémoire distribuée). Ces expériences démontrent la supériorité de nos algorithmes pour une large classe de tailles et formes de matrices, ainsi que de nombres de processeurs.

Mots-clés : bidiagonalisation, R-bidiagonalisation, chemin critique, algorithme glouton, arbre de réduction auto-adaptatif, matrice rectangulaire haute et étroite.

1 Introduction

This work is devoted to the design and comparison of tiled algorithms for the bidiagonalization of large matrices. Bidiagonalization is a widely used kernel that transforms a full matrix into bidiagonal form using orthogonal transformations. The importance of bidiagonalization stems from the fact that, in many algorithms, the bidiagonal form is a critical step to compute the singular value decomposition (SVD) of a matrix. The necessity of computing the SVD is present in many computational science and engineering areas. Based on the Eckart–Young theorem [18], we know that the singular vectors associated with the largest singular values represent the best way (in the 2-norm sense) to approximate the matrix. This approximation result leads to many applications, since it means that SVD can be used to extract the "most important" information of a matrix. We can use the SVD for compressing data or making sense of data. In this era of Big Data, we are interested in very large matrices. To reference one out of many application, SVD is needed for principal component analysis (PCA) in Statistics, a widely used method in applied multivariate data analysis.

We consider algorithms for going from a "full" matrix to a condensed "band bidiagonal" form using orthogonal transformations. We use the framework of "algorithms by tiles". Within this framework, we study: (i) the tiled bidiagonalization algorithm BiDiag, which is a tiled version of the standard scalar bidiagonalization algorithm; and (ii) the R-bidiagonalization algorithm R-BiDiag, which is a tiled version of the algorithm which consists in first performing the QR factorization of the initial matrix, then performing the band-bidiagonalization of the R-factor. For both bidiagonalization algorithms BiDiag and R-BiDiag, we use HQR-based reduction trees, where HQR stands for the Hierarchical QR factorization of a tiled matrix [14]. Considering various reduction trees gives us the flexibility to adapt to matrix shape and machine architecture. In this work, we consider many types of reduction trees. In shared memory, they are named FlatTS, FlatTT, Greedy, and a newly introduced auto-adaptive tree, Auto. In distributed memory, they are somewhat more complex and take into account the topology of the machine. The main contributions are the following:

- The design and comparison of the BiDiag and R-BiDiag tiled algorithms with many types of reduction trees. There is considerable novelty in this. Previous work [24,25,31,33] on tiled bidiagonalization has only considered one type of tree (FlatTS tree) with no R-BiDiag. Previous work [32] has considered Greedy trees only for the QR steps in BiDiag (so m of the $m + n$ steps), it considers FlatTS tree for the LQ steps (n of the $m + n$ steps), and it does not consider R-BiDiag. This paper is the first to study R-BiDiag for tiled bidiagonalization algorithm and to study Greedy trees for both steps (LQ and QR) of the tiled bidiagonalization algorithm.

- A detailed study of critical path lengths for FlatTS, FlatTT, Greedy

with BiDiag and R-BiDiag (so six different algorithms in total), which shows that:

  - The newly-introduced Greedy based schemes (BiDiag and R-BiDiag) are much better than earlier proposed variants with unbounded resources and no communication: for matrices of $p \times q$ tiles, $p \geq q$, their critical paths have a length $\Theta(q \log_2(p))$ instead of $\Theta(pq)$ for FlatTS and FlatTT
  - On the one hand, BiDiagGreedy has a shorter critical path length than R-BiDiagGreedy for square matrices; on the other hand, R-BiDiagGreedy has a shorter critical path length than BiDiagGreedy for "tall and skinny" matrices. For example, for a $p \times q$ tile matrix, when q go to infinity, and $p = \beta q^{1+\alpha}$, with $0 \leq \alpha < 1$, then the asymptotic ratio between BiDiagGreedy and R-BiDiagGreedy is $\frac{1}{1+\frac{\alpha}{2}}$.

- Implementation of our algorithms within the DPLASMA framework [5], which runs on top of the PaRSEC runtime system [6], and which enables parallel distributed experiments on multicore nodes. All previous tiled bidiagonalization study [24, 25, 31–33] were limited to shared memory implementation.

- A practical auto-adaptative tree (Auto) that self-tunes for increased performance. This tree is especially useful in many practical situations. For example, it is appropriate (1) when the critical path length is not a major consideration due to limited number of resources or (2) when the intra-node "communication" is expensive in which TS kernels are much faster than TT kernels or (3) both.

- Experiments on a single multicore node, and on a few multicore nodes of a parallel distributed shared-memory system, show the superiority of the new algorithms on a variety of matrix sizes, matrix shapes and core counts. The new Auto algorithm outperforms its competitors in almost every test case, hence standing as the best algorithmic choice for most users.

The rest of the paper is organized as follows. Section 2 provides a detailed overview of related work. Section 3 describes the BiDiag and R-BiDiag algorithms with the FlatTS, FlatTT and Greedy trees. Section 4 is devoted to the analysis of the critical paths of all variants. Section 5 outlines our implementation, and introduces the new Auto reduction tree. Experimental results are reported in Section 6. Finally, conclusion and hints for future work are given in Section 7.

2 Related Work

This section provides an overview of the various approaches to compute the singular values of a matrix, and positions our new algorithm with respect to existing numerical software kernels.

Computing the SVD. The SVD of a matrix is a fundamental matrix factorization and is a basic tool used in many applications. Computing the SVD of large matrices in an efficient and scalable way, is an important problem that has gathered much attention. The matrices considered here are rectangular m-by-n. Without loss of generality, we consider $m \geq n$. We call GE2VAL the problem of computing (only) the singular values of a matrix, and GESVD the problem of computing the singular values and the associated singular vectors.

From full to bidiagonal form. There are few algorithms to compute the singular value decomposition. A large class of these algorithms consists in first reducing the matrix to bidiagonal form with orthogonal transformations (GE2BD step), then processing the bidiagonal matrix to obtain the sought singular values (BD2VAL step). These two steps (GE2BD and BD2VAL) are very different in nature. GE2BD can be done in a known number of operations and has no numerical difficulties. On the other hand, BD2VAL requires the convergence of an iterative process and is prone to numerical difficulties. This paper mostly focuses on GE2BD: reduction from full to bidiagonal form. Clearly, GE2BD+BD2VAL solves GE2VAL: computing (only) the singular value of a matrix. If the singular vectors are desired (GESVD), one can also compute them by accumulating the "backward" transformations; in this example, this would consist in a VAL2BD step followed by a BD2GE step.

In 1965, Golub and Kahan [19] provides a singular value solver based on an initial reduction to bidiagonal form. In [19, Th. 1], the GE2BD step is done using a QR step on the first column, then an LQ step on the first row, then a QR step on the second column, etc. The steps are done one column at a time using Householder transformation. This algorithm is implemented as a Level-2 BLAS algorithm in LAPACK as `xGEBD2`. For an m-by-n matrix, the cost of this algorithm is (approximately) $4mn^2 - \frac{4}{3}n^3$.

Level 3 BLAS for GE2BD. In 1989, Dongarra, Sorensen and Hammarling [15] explains how to incorporate Level-3 BLAS in LAPACK `xGEBD2`. The idea is to compute few Householder transformations in advance, and then to accumulate and apply them in block using the WY transform [3]. This algorithm is available in LAPACK (using the compact WY transform [36]) as `xGEBRD`. Großer and Lang [21, Table 1] explain that this algorithm performs (approximately) 50% of flops in Level 2 BLAS (computing and accumulating Householder vectors) and 50% in Level 3 BLAS (applying Householder vectors). In 1995, Choi, Dongarra and Walker [12] presents the ScaLAPACK version, `PxGEBRD`, of the LAPACK `xGEBRD` algorithm of [15].

Multi-step approach. Further improvements for GE2BD (detailed thereafter) are possible. These improvements rely on combining multiple steps. These multi-step methods will perform in general much better for GE2VAL (when only singular values are sought) than for GESVD (when singular values and singular vectors are sought). When singular values and singular vectors are sought, all the "multi" steps have to be performed in "reverse" on the singular vectors adding a non-negligible overhead to the singular vector computation.

Preprocessing the bidiagonalization with a QR factorization (preQR step). In 1982, Chan [11] explains that, for tall-and-skinny matrices, in order to perform less flops, one can pre-process the bidiagonalization step (GE2BD) with a QR factorization. In other words, Chan propose to do preQR(m,n)+GE2BD(n,n) instead of GE2BD(m,n). A curiosity of this algorithm is that it introduces nonzeros where zeros were previously introduced; yet, there is a gain in term of flops. Chan proves that the crossover points when preQR(m,n)+GE2BD(n,n) performs less flops than GE2BD(m,n) is when m is greater than $\frac{5}{3}n$. Chan also proved that, asymptotically, preQR(m,n)+GE2BD(n,n) will perform half the flops than GE2BD(m,n) for a fixed n and m going to infinity. If the singular vectors are sought, preQR has more overhead: (1) the crossover point is moved to more tall-and-skinny matrices, and there is less gain; also (2) there is some complication as far as storage goes.

The preQR step was somewhat known by the community and an earlier reference is for example the book of Lawson and Hanson 1974 [30]. For this reason, the fact of pre-processing the bidiagonalization by a QR factorization (preQR) is referred to as the LHC (Lawson, Hanson and Chan) algorithm in [37].

Three-step approach: partialGE2BD+preQR+GE2BD. In 1997, Trefethen and Bau [37] present a "three-step" approach. The idea is to first start a bidiagonalization; then whenever the ratio $\frac{m}{n}$ passes the $\frac{5}{3}$ mark, switch to a QR factorization; and then finish up with GE2BD.

Two-step approach: GE2BND+BND2BD. In 1999, Großer and Lang [21] studied a two-step approach for GE2BD: (1) go from full to band (GE2BND), (2) then go from band to bidiagonal (BND2BD). In this scenario, GE2BND has most of the flops and performs using Level-3 BLAS kernels; BND2BD is not using Level-3 BLAS but it executes much less flops and operates on a smaller data footprint that might fit better in cache. There is a trade-off for the bandwidth to be chosen. If the bandwidth is too small, then the first step (GE2BND) will have the same issues as GE2BD. If the bandwidth is too large, then the second step BND2BD will have many flops and dominates the run time.

As mentioned earlier, when the singular vectors are sought (GESVD), the overhead of the method in term of flops is quite large. In 2013, Haidar, Kurzak, and Luszczek [24] reported that despite the extra flops a two-step approach leads to speedup even when singular vectors are sought (GESVD).

Tiled Algorithms for the SVD. In the context of massive parallelism, and of reducing data movement, many dense linear algebra algorithms have been moved to so-called "tiled algorithms". In the tiled algorithm framework, algorithms operates on *tiles* of the matrix, and tasks are scheduled thanks to a runtime. In the context of the SVD, tiled algorithms naturally leads to band bidiagonal form. In 2010, Ltaief, Kurzak and Dongarra [31] present a tiled algorithm for GE2BND (to go from full to band bidiagonal form). In 2013 (technical report in 2011), Ltaief, Luszczek, Dongarra [33] add the second step (BND2BD) and present a tiled algorithm for GE2VAL using GE2BND+BND2BD+BD2VAL. In 2012, Ltaief, Luszczek, and Dongarra [32] improve the algorithm for tall and skinny matrices by using "any" tree instead of flat trees in the QR steps. In 2012, Haidar, Ltaief, Luszczek and Dongarra [25] improve the BND2BD step of [33]. Finally, in 2013, Haidar, Kurzak, and Luszczek [24] consider the problem of computing singular vectors (GESVD) by performing

GE2BND+BND2BD+BD2VAL+VAL2BD+BD2BND+BND2GE.

They show that the two-step approach (from full to band, then band to bidiagonal) can be successfully used not only for computing singular values, but also for computing singular vectors.

BND2BD step. The algorithm in LAPACK for BND2BD is xGBBRD. In 1996, Lang [28] improved the sequential version of the algorithm and developed a parallel distributed algorithm. Recently, PLASMA released an efficient multi-threaded implementation [25, 33]. We also note that Rajamanickam [35] recently worked on this step.

BD2VAL step. Much research has been done and is done on this kernel. Much software exists. In LAPACK, to compute the singular values and optionally the singular vectors of a bidiagonal matrix, the routine `xBDSQR` uses the Golub-Kahan QR algorithm [19]; the routine `xBDSDC` uses the divide-and-conquer algorithm [22]; and the routine `xBDSVX` uses bisection and inverse iteration algorithm. Recent research was trying to apply the MRRR (Multiple Relatively Robust Representations) method [39] to the problem.

BND2BD+BD2VAL steps in this paper. This paper does not focus neither on BND2BD nor BD2VAL. As far as we are concerned, we can use any of the methods mentioned above. The faster these two steps are, the better for us. For this study, during the experimental section, for BND2BD, we use the PLASMA multi-threaded implementation [25, 33] and, for BD2VAL, we use LAPACK `xBDSQR`.

Intel MKL. We note that, since 2014 and Intel MKL 11.2, the Intel MKL library is much faster to compute GESVD [38]. Much of the literature on tiled bidiagonalization algorithm was before 2014, and so it was comparing to much slower version of MKL that is now available. The reported speedup in this manuscript for the tiled algorithms over MKL, are therefore much less impressive than previously reported.

We also note that, while there was a great performance improvement for computing GESVD [38] from Intel MKL 11.1 to Intel MKL 11.2, there was no performance improvement for GEBRD. The reason is that the interface for GEBRD prevents a two-step approach, while the interface of GESVD allows for a two-step approach. Similarly, our two-step algorithm does not have the same interface as LAPACK GEBRD. Consequently, it would not be fair to compare our algorithm with LAPACK GEBRD or Intel MKL GEBRD because our algorithm does not have a native LAPACK GEBRD interface. (I.e., it will not produce the Householder vector column by column as requested by the LAPACK interface.) For this reason, we will always compare with GESVD, and not GEBRD.

Approach to compute the SVD without going by bidiagonal form. We note that going to bidiagonal form is not a necessary step to compute the singular value decomposition of a matrix. There are approaches to compute the SVD of a matrix which avoid the reduction to bidiagonal form. In the "direct method" category, we can quote Jacobi SVD [16, 17], QDWHSVD [34]; one can also think to think the singular value decomposition directly from the block bidiagonal form although not much work has been done in this direction so far. Also, we can also refer to all "iterative methods" for computing the singular values. The goal of these iterative methods is in general to compute a few singular triplets. As far as software packages, we can quote SVDPACK [2], PROPACK [29], and PRIMME SVD [40]. As far methods, we can quote [26, 27, 41].

Connection with Symmetric Eigenvalue Problem. Many ideas (but not all) can be applied (and have been applied) to symmetric tridiagonalization. This paper only focuses on bidiagonalization algorithm but we believe that some of the tricks we show (but not all) could be used for creating an efficient symmetric tridiagonalization.

3 Tiled Bidiagonalization Algorithms

In this section, we provide background on tiled algorithms for QR factorization, and then explain how to use them for the BiDiag and R-BiDiag algorithms.

3.1 Tiled algorithms

Tiled algorithms are expressed in terms of tile operations rather than elementary operations. Each tile is of size $n_b \times n_b$, where n_b is a parameter tuned to squeeze the most out of arithmetic units and memory hierarchy. Typically, n_b ranges from 80 to 200 on state-of-the-art machines [1].

Consider a rectangular tiled matrix A of size $p \times q$. The actual size of A is thus $(pn_b) \times (qn_b)$, where n_b is the tile size. We number rows from 1 to p, columns from 1 to q, and factorization steps from $k = 1$ to $k = q$. Algorithm 1 outlines a tiled QR algorithm, where loop indices represent tiles:

Algorithm 1: $QR(p, q)$ algorithm for a tiled matrix of size (p, q).

```
for k = 1 to min(p, q) do
    Step k, denoted as QR(k):
    for i = k + 1 to p do
        elim(i, piv(i, k), k)
```

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Table 1: Kernels for tiled QR. The unit of time is $\frac{n_b^3}{3}$, where n_b is the blocksize.

In Algorithm 1, k is the step, and also the panel index, and $elim(i, piv(i, k), k)$ is an orthogonal transformation that combines rows i and $piv(i, k)$ to zero out the tile in position (i, k). We explain below how such orthogonal transformations can be implemented.

3.2 Kernels

To implement a given orthogonal transformation $elim(i, piv(i, k), k)$, one can use six different kernels, whose costs are given in Table 1. In this table, the unit of time is the time to perform $\frac{n_b^3}{3}$ floating-point operations.

There are two main possibilities to implement an orthogonal transformation $elim(i, piv(i, k), k)$: The first version eliminates tile (i, k) with the *TS (Triangle on top of square)* kernels, as shown in Algorithm 2:

Algorithm 2: Elimination $elim(i, piv(i, k), k)$ via *TS (Triangle on top of square)* kernels.

```
GEQRT(piv(i, k), k)
TSQRT(i, piv(i, k), k)
for j = k + 1 to q do
    UNMQR(piv(i, k), k, j)
    TSMQR(i, piv(i, k), k, j)
```

Here the tile panel $(piv(i, k), k)$ is factored into a triangle (with *GEQRT*). The transformation is applied to subsequent tiles $(piv(i, k), j)$, $j > k$, in row $piv(i, k)$ (with *UNMQR*). Tile (i, k) is zeroed out (with *TSQRT*), and subsequent tiles (i, j), $j > k$, in row i are updated (with *TSMQR*). The flop count is $4 + 6 + (6 + 12)(q - k) = 10 + 18(q - k)$ (expressed in same time unit as in

Table 1). Dependencies are the following:

$$
\begin{array}{ll}
GEQRT(piv(i,k),k) \prec TSQRT(i,piv(i,k),k) & \\
GEQRT(piv(i,k),k) \prec UNMQR(piv(i,k),k,j) & \text{for } j > k \\
UNMQR(piv(i,k),k,j) \prec TSMQR(i,piv(i,k),k,j) & \text{for } j > k \\
TSQRT(i,piv(i,k),k) \prec TSMQR(i,piv(i,k),k,j) & \text{for } j > k
\end{array}
$$

$TSQRT(i, piv(i,k), k)$ and $UNMQR(piv(i,k), k, j)$ can be executed in parallel, as well as $UNMQR$ operations on different columns $j, j' > k$. With an unbounded number of processors, the parallel time is thus $4 + 6 + 12 = 22$ time-units.

Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via *TT (Triangle on top of triangle)* kernels.

```
GEQRT(piv(i,k),k)
GEQRT(i,k)
for j = k+1 to q do
    UNMQR(piv(i,k),k,j)
    UNMQR(i,k,j)
TTQRT(i,piv(i,k),k)
for j = k+1 to q do
    TTMQR(i,piv(i,k),k,j)
```

The second approach to implement the orthogonal transformation $elim(i, piv(i,k), k)$ is with the *TT (Triangle on top of triangle)* kernels, as shown in Algorithm 3. Here both tiles $(piv(i,k), k)$ and (i,k) are factored into a triangle (with $GEQRT$). The corresponding transformations are applied to subsequent tiles $(piv(i,k), j)$ and (i,j), $j > k$, in both rows $piv(i,k)$ and i (with $UNMQR$). Tile (i,k) is zeroed out (with $TTQRT$), and subsequent tiles (i,j), $j > k$, in row i are updated (with $TTMQR$). The flop count is $2(4+6(q-k))+2+6(q-k) = 10+18(q-k)$, just as before. Dependencies are the following:

$$
\begin{array}{ll}
GEQRT(piv(i,k),k) \prec UNMQR(piv(i,k),k,j) & \text{for } j > k \\
GEQRT(i,k) \prec UNMQR(i,k,j) & \text{for } j > k \\
GEQRT(piv(i,k),k) \prec TTQRT(i,piv(i,k),k) & \\
GEQRT(i,k) \prec TTQRT(i,piv(i,k),k) & \\
TTQRT(i,piv(i,k),k) \prec TTMQR(i,piv(i,k),k,j) & \text{for } j > k \\
UNMQR(piv(i,k),k,j) \prec TTMQR(i,piv(i,k),k,j) & \text{for } j > k \\
UNMQR(i,k,j) \prec TTMQR(i,piv(i,k),k,j) & \text{for } j > k
\end{array}
$$

Now the factor operations in row $piv(i,k)$ and i can be executed in parallel. Moreover, the $UNMQR$ updates can be run in parallel with the $TTQRT$ factorization. Thus, with an unbounded number of processors, the parallel time is $4 + 6 + 6 = 16$ time-units.

In Algorithms 2 and 3, it is understood that if a tile is already in triangle form, then the associated $GEQRT$ and update kernels do not need to be applied.

3.3 QR factorization

Consider a rectangular tiled matrix A of size $p \times q$, with $p \geq q$. There are many algorithms to compute the QR factorization of A, and we refer to [8] for a survey. We use the three following variants:

FlatTS This simple algorithm is the reference algorithm used in [9, 10]. At step k, the pivot row is always row k, and we perform the eliminations $elim(i, k, k)$ in sequence, for $i = k + 1$, $i = k + 2$ down to $i = p$. In this algorithm, TS kernels are used.

FlatTT This algorithm is the counterpart of the FlatTS algorithm with TT kernels. It uses exactly the same elimination operations, but with different kernels.

Greedy This algorithm is asymptotically optimal, and turns out to be the most efficient on a variety of platforms [7, 14]. An optimal algorithm is Grasap [7, 14]. The difference between Grasap and Greedy is a small constant term. We work with Greedy in this paper. The main idea is to eliminate many tiles in parallel at each step, using a reduction tree (see [8] for a detailed description).

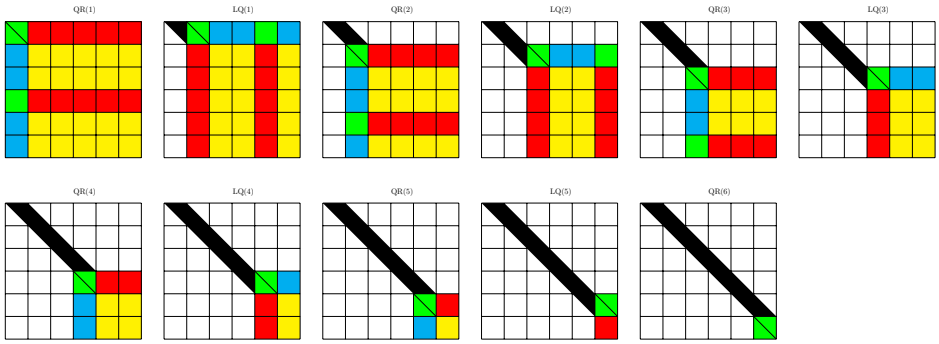


Figure 1: Snapshots of the bidiagonalization algorithm BiDiag.

3.4 Bidiagonalization

Consider a rectangular tiled matrix A of size $p \times q$, with $p \geq q$. The bidiagonalization algorithm BiDiag proceeds as the QR factorization, but interleaves one step of LQ factorization between two steps of QR factorization (see Figure 1). More precisely, BiDiag executes the sequence

$$QR(1); LQ(1); QR(2); \ldots; QR(q-1); LQ(q-1); QR(q)$$

Algorithm 4: Step $LQ(k)$ for a tiled matrix of size $p \times q$.

Step k, denoted as $LQ(k)$:
for j $= k+1$ *to* q **do**
  col-elim$(j, piv(j,k), k)$

where $QR(k)$ is the step k of the QR algorithm (see Algorithm 1), and $LQ(k)$ is the step k of the LQ algorithm. The latter is a right factorization step that executes the column-oriented eliminations shown in Algorithm 4.

In Algorithm 4, *col-elim*$(j, piv(k,j), k)$ is an orthogonal transformation that combines columns j and $piv(k,j)$ to zero out the tile in position (k,j). It is the exact counterpart to the row-oriented eliminations *elim*$(i, piv(i,k), k)$ and be implemented with the very same kernels, either TS or TT.

3.5 R-Bidiagonalization

When p is much larger than q, R-bidiagonalization should be preferred, if minimizing the operation count is the objective. This R-BiDiag algorithm does a QR factorization of A, followed by a bidiagonalization of the upper square $q \times q$ matrix. In other words, given a rectangular tiled matrix A of size $p \times q$, with $p \geq q$, R-BiDiag executes the sequence

$$QR(p,q); LQ(1); QR(2); \ldots; QR(q-1); LQ(q-1); QR(q)$$

Let $m = pn_b$ and $n = qn_b$ be the actual size of A (element wise). The number of arithmetic operations is $4n^2(m - \frac{n}{3})$ for BiDiag and $2n^2(m+n)$ for R-BiDiag [20, p.284]. These numbers show that R-BiDiag is less costly than BiDiag whenever $m \geq \frac{5n}{3}$, or equivalently, whenever $p \geq \frac{5q}{3}$. One major contribution of this paper is to provide a comparison of BiDiag and R-BiDiag in terms of parallel execution time, instead of operation count.

3.6 Comments

Case $m \leq n$. The paper focuses on the $m \geq n$ case but everything holds in the $m \leq n$ case. One simply can transpose the initial matrix and change switch LQ steps for QR steps and vice versa.

Real/Complex arithmetic. There is no restriction of our work on whether the matrices are real or complex. Actually, our codes have been generated to allow for both real and complex arithmetics, single and double precisions.

Memory usage of our algorithm. In the standard LAPACK case, GEBRD takes as input a matrix and overrides the matrix with the Householder reflectors and the bidiagonal matrix. The algorithm is therefore down in place and there is no extra storage needed. In the case R-BiDiag, it is important to note that

as in the case of the LHC algorithm [11], an extra storage of size $n^2/2$ is needed if one wants to recompute the singular vectors. (If one wants the singular value, all that matters is the bidiagonal form and any Householder vectors generated can safely be discarded.)

Any tree is possible. We note that whether we choose to perform BiDiag or R-BiDiag, any tree is possible at each step of the algorithm. For example a restriction of our study thereafter is to fix the trees in the QR step and the BiDiag step of R-BiDiag to be the same. Clearly there is no such need, and one can consider for R-BiDiag (for example) a combination like (QRFlatTS +BiDiagGreedy). We do not study such combinations. This also simplifies notations. Since we consider the same tree for the QR step and the BiDiag step of R-BiDiag, when we write R-BiDiagFlatTT (for example), we mean (QRFlatTT +BiDiagFlatTT).

BiDiagGreedy and BiDiagBinomial are the same. Another name for BiDiagGreedy would be BiDiagBinomial. Since in the BiDiag algorithm, we repeat the same tree (in the Greedy case, a Binomial tree) over and over again. So BiDiagGreedy and BiDiagBinomial are the exact same algorithm. (Whereas QRGreedy is not the same as QRBinomial.)

4 Critical paths

In this section, we compute exact or estimated values of the critical paths of the BiDiag and R-BiDiag algorithms with the FlatTS, FlatTT, and Greedy trees.

4.1 Bidiagonalization

Given a rectangular tiled matrix A of size $p \times q$, with $p \geq q$, the bidiagonalization algorithm BiDiag executes the sequence

$$QR(1); LQ(1); QR(2); \ldots; QR(q-1); LQ(q-1); QR(q)$$

To compute the critical path, we first observe that there is no overlap between two consecutive steps $QR(k)$ and $LQ(k)$. To see why, consider w.l.o.g. the first two steps $QR(1)$ and $LQ(1)$ on Figure 1. Tile $(1, 2)$ is used at the end of the $QR(1)$ step to update the last row of the trailing matrix (whichever it is). In passing, note that all columns in this last row are updated in parallel, because we assume unlimited resources when computing critical paths. But tile $(1, 2)$ it is the first tile modified by the $LQ(1)$ step, hence there is no possible overlap. Similarly, there is no overlap between two consecutive steps $LQ(k)$ and $QR(k+1)$. Consider steps $LQ(1)$ and $QR(2)$ on Figure 1. Tile $(2, 2)$ is used at the end of the $LQ(1)$ step to update the last column of the trailing matrix (whichever it is), and it is the first tile modified by the $QR(1)$ step.

As a consequence, the critical path length of BiDiag is the sum of the critical path lengths of each QR and LQ step. And so an optimal BiDiag algorithm is an algorithm which is optimal at each QR and LQ step independently. We know that an optimal QR step is obtained by a binomial tree. We know that an optimal LQ step is obtained by a binomial tree. Therefore the scheme alternating binomial QR and LQ trees, that is BiDiagGreedy, is optimal. This is repeated in the following theorem.

Theorem 1. BiDiagGreedy *is optimal, in the sense that it has the shortest critical path length, over all* BiDiag *algorithms, which are tiled algorithms that alternate a QR step and an LQ step to obtain a band bidiagonal matrix with a sequence of any QR trees and any LQ trees.*

We now compute critical path lengths. From [7, 8, 14] we have the following values for the critical path of one QR step applied to a tiled matrix of size (u, v):

FlatTS

$$QR - FTS_{1step}(u,v) = \begin{cases} 4 + 6(u-1) & \text{if } v = 1, \\ 4 + 6 + 12(u-1) & \text{otherwise.} \end{cases}$$

FlatTT

$$QR - FTT_{1step}(u,v) = \begin{cases} 4 + 2(u-1) & \text{if } v = 1, \\ 4 + 6 + 6(u-1) & \text{otherwise.} \end{cases}$$

Greedy

$$QR - GRE_{1step}(u,v) = \begin{cases} 4 + 2\lceil \log_2(u) \rceil & \text{if } v = 1, \\ 4 + 6 + 6\lceil \log_2(u) \rceil & \text{otherwise.} \end{cases}$$

The critical path of one LQ step applied to a tiled matrix of size (u, v) is $LQ_{1step}(u, v) = QR_{1step}(v, u)$. Finally, in the BiDiag algorithm, the size of the matrix for step $QR(k)$ is $(p-k+1, q-k+1)$ and the size of the matrix for step $LQ(k)$ is $(p-k+1, q-k)$. Altogether, we derive the following values:

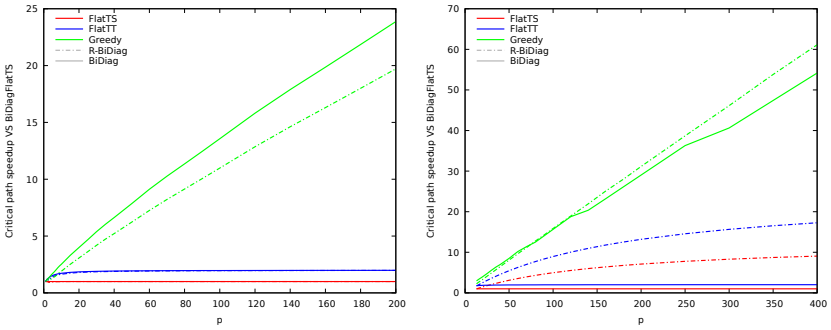
FlatTS

$$\begin{aligned} \text{BiDiagFlatTS}(p,q) &= \sum_{k=1}^{q-1}(10 + 12(p-k)) + \sum_{k=1}^{q-1}(10 + 12(q-k-1)) + (4 + 6(p-q)) \\ &= 12pq - 6p + 2q - 4 \end{aligned}$$

FlatTT

$$\begin{aligned} \text{BiDiagFlatTT}(p,q) &= \sum_{k=1}^{q-1}(10 + 6(p-k)) + \sum_{k=1}^{q-1}(10 + 6(q-k-1)) + (4 + 2(p-q)) \\ &= 6pq - 4p + 12q - 10 \end{aligned}$$

We devote the whole next subsection to the study of the critical path of BiDiagGreedy(p, q), because this is more complicated.



(a) Square ($p = q$)

(b) Tall and skinny ($q = 13$)

(c) Tall and skinny ($q = 63$)

Figure 2: Critical paths: ratio of the different methods over BiDiag with FlatTS, on three configurations. Solid lines are for BiDiag and dashed lines for R-BiDiag.

4.2 Study of the critical path of BiDiagGreedy(p, q)

4.2.1 A formula with a summation

By following the reasoning of the previous subsection, we get

Greedy

$$\begin{aligned}
&\textsc{BiDiagGreedy}(p,q)\\
&= \textstyle\sum_{k=1}^{q-1}(10+6\lceil\log_2(p+1-k)\rceil)+\sum_{k=1}^{q-1}(10+6\lceil\log_2(q-k)\rceil)+(4+2\lceil\log_2(p+1-q)\rceil\\
&= 6\textstyle\sum_{k=p-q+2}^{p}\lceil\log_2(k)\rceil+6\sum_{k=1}^{q-1}\lceil\log_2(k)\rceil+20q+2\lceil\log_2(p+1-q)\rceil-16\\
&= 6\textstyle\sum_{k=1}^{p}\lceil\log_2(k)\rceil-6\sum_{k=1}^{p-q+1}\lceil\log_2(k)\rceil+6\sum_{k=1}^{q-1}\lceil\log_2(k)\rceil+20q+2\lceil\log_2(p+1-q)\rceil-16
\end{aligned} \tag{1}$$

The latter formula is exact and is easy enough to compute with a small computer code. However it does not provide us with much insight. To get a better grasp on $\textsc{BiDiagGreedy}(p, q)$, we develop some more Equation (1).

4.2.2 An exact formula when p and q are powers of two

First we note that:

Let r be a power of 2 :

$$\begin{aligned}
\sum_{k=1}^{r}\lceil\log_2(k)\rceil &= 1+2+2+3+3+3+3+4+4+4+4+4+4+4+4+5\ldots\\
&= r\log_2(r)-r+1
\end{aligned} \tag{2}$$

One way to understand the previous relation is to remember that "the" antiderative of $\log(x)$ is $x\log(x)-x$. The previous relation reminds us of this result. The general case (r is not a power of 2) of Equation (2) is

$$\text{Let } r \text{ be an integer,}\quad \sum_{k=1}^{r}\lceil\log_2(k)\rceil = (\lfloor\log_2(r)\rfloor-1)\,2^{\lfloor\log_2(r)\rfloor}+1+\left(r-2^{\lfloor\log_2(r)\rfloor}\right)\lceil\log_2(r)\rceil \tag{3}$$

From Equations (1) and (2), we derive that if q is a power of two:

$$\text{Let } q \text{ be a power of 2,}\quad \textsc{BiDiagGreedy}(q,q) = 12q\log_2(q)+8q-6\log_2(q)-4$$

From Equations (1), (2) and (3), we derive that, if both p and q are powers of two, with $p > q$:

Let p and q be powers of 2 :

$$\text{with } p>q,\quad \textsc{BiDiagGreedy}(p,q) = 6q\log_2(p)+6q\log_2(q)+14q-4\log_2(p)-6\log_2(q)-10.$$

Theorem 2. *Let p and q be powers of 2,*

$$\begin{aligned}
&\textit{If } p=q, && \textsc{BiDiagGreedy}(q,q) = 12q\log_2(q)+8q-6\log_2(q)-4,\\
&\textit{Else if } p>q, && \textsc{BiDiagGreedy}(p,q) = 6q\log_2(p)+6q\log_2(q)+14q-4\log_2(p)-6\log_2(q)-10.
\end{aligned}$$

4.2.3 Using Stirling's formula

We can also consider Equation (1) again and obtain simpler bounds by rounding down and up the ceiling function in the logarithms and then using asymptotic analysis. We consider the bounds:

$$\text{Let } k \text{ be an integer}, \quad \log_2(k) \leq \lceil \log_2(k) \rceil \leq \log_2(k) + 1.$$

Applied to Equation (1), this leads to

$$\begin{gathered} 6\log_2(p!) - 6\log_2((p-q+1)!) + 6\log_2((q-1)!) + 20q + 2\log_2(p+1-q) - 16 \\ \leq \textsc{BiDiagGreedy}(p,q) \leq \\ 6\log_2(p!) - 6\log_2((p-q+1)!) + 6\log_2((q-1)!) + 32q + 2\log_2(p+1-q) - 26. \end{gathered}$$

We note that the difference between the lower bound (left side) and the upper bound (right side) is $12q - 10$.

We now consider "large" p and q, and we use Stirling's formula as follows:

$$\log(p!) = p\log(p) - p + \mathcal{O}(\log p).$$

And with one more term we get:

$$\log(p!) = p\log(p) - p + \frac{1}{2}\log(2\pi p) + \mathcal{O}\left(\frac{1}{p}\right).$$

In base 2,

$$\log_2(p!) = p\log_2(p) - \log_2(e)p + \frac{1}{2}\log_2(p) + \log_2(\sqrt{2\pi}) + \frac{\log_2(e)}{12p} + \mathcal{O}\left(\frac{1}{p^3}\right).$$

We obtain that

$$\begin{array}{ll} 6p\log_2(p) - 6(p-q)\log_2(p-q+1) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\ \quad +(20 - 12\log_2(e))(q) & [\text{linear terms}] \\ \quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\ \quad +6\log_2(\sqrt{2\pi}) + 12\log_2(e) - 16 & [\text{constant terms}] \\ \quad +\frac{1}{2}\log_2(e)\frac{1}{p} + \frac{1}{2}\log_2(e)\frac{1}{q-1} - \frac{1}{2}\log_2(e)\frac{1}{p-q+1} & \\ \quad +\mathcal{O}\left(\max(\frac{1}{(p-q)^3}, \frac{1}{q^3})\right) & \\ \qquad \leq \textsc{BiDiagGreedy}(p,q) \leq & \\ 6p\log_2(p) - 6(p-q)\log_2(p-q+1) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\ \quad +(32 - 12\log_2(e))(q) & [\text{linear terms}] \\ \quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\ \quad +6\log_2(\sqrt{2\pi}) + 12\log_2(e) - 26 & [\text{constant terms}] \\ \quad +\frac{1}{2}\log_2(e)\frac{1}{p} + \frac{1}{2}\log_2(e)\frac{1}{q-1} - \frac{1}{2}\log_2(e)\frac{1}{p-q+1} & \\ \quad +\mathcal{O}\left(\max(\frac{1}{(p-q)^3}, \frac{1}{q^3})\right). & \end{array}$$

We stop the expansion a little earlier.

$$
\begin{array}{ll}
6p\log_2(p) - 6(p-q)\log_2(p-q+1) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\
\quad +(20 - 12\log_2(e))(q) & [\text{linear terms}] \\
\quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\
\quad +\mathcal{O}(1) & \\
\multicolumn{2}{c}{\leq \textsc{BiDiagGreedy}(p,q) \leq} \\
6p\log_2(p) - 6(p-q)\log_2(p-q+1) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\
\quad +(32 - 12\log_2(e))(q) & [\text{linear terms}] \\
\quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\
\quad +\mathcal{O}(1)\,. &
\end{array}
$$

We now study the term: $p\log_2(p) - (p-q)\log_2(p-q+1)$, this gives:

$$
p\log_2(p) - (p-q)\log_2(p-q+1) \quad = \quad q\log_2(p) - (p-q)\log_2(1 - \frac{q-1}{p}).
$$

We now study the term: $(p-q)\log_2(1-\frac{q-1}{p})$. We can set $p = q + \alpha q$ with $\alpha \geq 0$. We get

$$
(p-q)\log_2(1 - \frac{q-1}{p}) = q\left(\alpha\log_2(\frac{\alpha}{1+\alpha} + \frac{1}{p})\right).
$$

We have that

$$
\text{for } \alpha \geq 0, \quad \alpha\log_2(\frac{\alpha}{1+\alpha}) \leq \log_2(\frac{\alpha}{1+\alpha} + \frac{1}{p}) \leq 0.
$$

We are therefore drawn to study for $\alpha \geq 0$, $\alpha\log_2(\frac{\alpha}{1+\alpha})$. We see that this is a decreasing function of α. We are interested in the limit value when α goes to ∞. For large values of α, we have

$$
\begin{aligned}
\alpha\log_2(\frac{\alpha}{1+\alpha}) &= \alpha\log_2(e)\log\left(1 - \frac{1}{1+\alpha}\right), \\
&= \alpha\log_2(e)\log\left(-\frac{1}{1+\alpha} + \mathcal{O}(\frac{1}{\alpha^2})\right), \\
&= \log_2(e)\log\left(-\frac{\alpha}{1+\alpha} + \mathcal{O}(\frac{1}{\alpha})\right).
\end{aligned}
$$

So that

$$
\lim_{\alpha\to+\infty}\left(\alpha\log_2(\frac{\alpha}{1+\alpha})\right) = -\log_2(e).
$$

So all in all, we have that

$$
\text{for } \alpha \geq 0, \quad -\log_2(e) \leq \alpha\log_2(\frac{\alpha}{1+\alpha}).
$$

And so

$$
-\log_2(e)q \leq (p-q)\log_2(1 - \frac{q-1}{p}) \leq 0.
$$

Thus

$$q\log_2(p) \leq p\log_2(p) - (p-q)\log_2(p-q+1) \leq q\log_2(p) + \log_2(e)q.$$

So we get

$$\begin{array}{ll}
6q\log_2(p) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\
\quad +(20 - 12\log_2(e))(q) & [\text{linear terms}] \\
\quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\
\quad +\mathcal{O}(1) & \\
\multicolumn{2}{c}{\leq \textsc{BiDiagGreedy}(p,q) \leq} \\
6q\log_2(p) + 6q\log_2(q-1) & [x.\log_2 x \text{ terms}] \\
\quad +(32 - 6\log_2(e))(q) & [\text{linear terms}] \\
\quad +3\log_2(p) - 3\log_2(q-1) - 7\log_2(p-q+1) & [\log_2 \text{ linear terms}] \\
\quad +\mathcal{O}(1). &
\end{array} \quad (4)$$

We note that the difference between the lower bound (left side) and the upper bound (right side) is $(12 + 6\log_2(e))q - 10$.

We now get the following theorem

Theorem 3.

$$\textsc{BiDiagGreedy}(p,q) = 6q\log_2(p) + 6q\log_2(q) + \mathcal{O}(\max(\log_2(p), q)).$$

This theorem matches the power of 2 case. Also we understand that we perform q QR steps that each are done with a binomial tree of length $\log_2(p)$. This explains the $q\log_2(p)$ term; we perform q LQ steps that each are done with a binomial tree of length $\log_2(q)$. This explains the $q\log_2(q)$ term. The coefficient 6 in front is due to the weights use for the kernels. This theorem holds for any p and q.

In the $p = q$ case, we get that

$$\textsc{BiDiagGreedy}(q,q) = 12q\log_2(q) + \mathcal{O}(q).$$

In the $p = \beta q^{1+\alpha}$ case, we get that

$$\textsc{BiDiagGreedy}(\beta q^{1+\alpha}, q) = (12 + 6\alpha)q\log_2(q) + \mathcal{O}(q).$$

We use Equation (4) and get

for q constant, and p going to infinity,

$$\textsc{BiDiagGreedy}(p,q) = (6q-4)\log_2(p) + \mathcal{O}(1)$$

We note that this formula agrees with the case when p and q are powers of 2. Also note that the $\mathcal{O}(1)$ includes $q\log_2(q)$ terms, q terms, etc since q is a constant.

4.2.4 An exact formula for all p and q

We conclude the subsection on the study of BiDiagGreedy(p,q) by giving an exact formula. The first-order term in the exact formula is the same as in Subsubsection 4.2.3 with asymptotic analysis. However one lower-order term is harder to analyze. So we prefer to present this formula last. This is nevertheless an exact formula for our problem so it has some merit in itself.

For the general case, (either p or q is not a power of 2,) using Equations (1), (2) and (3), the expression gets more complicated: letting

$$\begin{aligned}
\alpha_p &= \lceil \log_2(p) \rceil - \log_2(p) \\
\alpha_k &= \lceil \log_2(p-q+1) \rceil - \log_2(p-q+1) \\
\alpha_q &= \lceil \log_2(q-1) \rceil - \log_2(q-1); \\
\beta_p &= \alpha_p - 2^{\alpha_p} - \alpha_k + 2^{\alpha_k}; \\
\beta_q &= \alpha_q - 2^{\alpha_q} + \alpha_k - 2^{\alpha_k};
\end{aligned}$$

we derive that

$$\begin{aligned}
&\text{BiDiagGreedy}(p,q) \\
&= 6p\log_2(p) - 6(p-q+1)\log_2(p-q+1) + 6(q-1)\log_2(q-1) \\
&+ 20q + 6\beta_p p + 6\beta_q(q-1) + 2\lceil \log_2(p+1-q) \rceil - 10.
\end{aligned}$$

We see that we have a term in p. Namely $6\beta_p p$. This is counter-intuitive since we expect an expansion with the term $p\log_2(p) - (p-q+1)\log_2(p-q+1) + (q-1)\log_2(q-1)$ as we already have seen in Subsubsection 4.2.3, and then, for the next terms after, we expect a term in q and no term in p. The next term in p should at most be a $\log_2(p)$. The reason is that the term $6\beta_p p$ behaves has a linear function of q. (Although this is not trivial to understand.)

If further analysis is sought from this formula, it is important to note that β_p and β_q, while being complicated functions of p and q, are bounded. Indeed, since α_p, α_k, and α_q are between 0 and 1, we have that $\alpha - 2^\alpha$ is between $-(1+\log\log 2))/\log 2$ and -1, (that is between: -0.9139 and -1,) and so

$$\begin{aligned}
-(1-(1+\log(\log(2)))/\log(2)) &\leq \beta_p \leq +(1-(1+\log(\log(2)))/\log(2)) \\
-0.0861 &\leq \beta_p \leq +0.0861
\end{aligned}$$

and

$$\begin{aligned}
-2 &\leq \beta_q \leq -2\left((1+\log(\log(2)))/\log(2)\right) \\
-2 &\leq \beta_q \leq -1.8278
\end{aligned}$$

4.3 R-Bidiagonalization

4.3.1 Formula for tiled QR factorization using various trees

For a tiled matrix of size (p,q), R-BiDiag executes the sequence

$$QR(p,q); LQ(1); QR(2); \ldots; QR(q-1); LQ(q-1); QR(q)$$

From [7, 8, 14], we have the following critical paths for the QR factorization of a tiled (p, q) matrix, which can be computed using the three variants described in Section 3:

FlatTS

$$QR - FTS(p,q) = \begin{cases} 6p-2 & \text{if } q = 1, \\ 30q-34 & \text{if } q = p, \\ 12p+18q-32 & \text{otherwise.} \end{cases}$$

FlatTT

$$QR - FTT(p,q) = \begin{cases} 2p+2 & \text{if } q = 1, \\ 22q-24 & \text{if } q = p, \\ 6p+16q-22 & \text{otherwise.} \end{cases}$$

For $QR-GRE$ we have no closed-form formula. By combining [7, Theorem 3.5] with [13, Theorem 3] we derive that

$$QR - GRE(p,q) = 22q + o(q)$$

whenever $p = o(q^2)$, which includes the case where $p = \beta q^{1+\alpha}$, with $0 \leq \alpha < 1$.

4.3.2 Critical path length of R-BiDiag with various trees

Computing the critical path of R-BiDiag is more difficult than for BiDiag, because kernels partly overlap. For example, there is no need to wait for the end of the (left) QR factorization to start the first (right) factorization step $LQ(1)$. In fact, this step can start as soon as the first step $QR(1)$ is over because the first row of the matrix is no longer used throughout the whole QR factorization at this point. However, the interleaving of the following kernels gets quite intricate. For example we note that R-BiDiagGreedy$(65, 5)$ = 262 while R-BiDiagGreedy$(66, 5) = 260$. So we factorize a larger matrix in a shorter critical path. The reason is that, even though the critical path of $QR-GRE$(66,5) is longer than the critical path of $QR-GRE$(65,5), the DAG of $QR - GRE$(66,5) interleaves with the DAG of BiDiag (5,5) more than the DAG of $QR - GRE$(65,5) interleaves with the DAG of BiDiag (5,5). There are many such examples. For example R-BiDiagGreedy$(134, 10) = 680$ while R-BiDiagGreedy$(133, 10) = 682$; R-BiDiagGreedy$(535, 50) = 4946$ while R-BiDiagGreedy$(534, 50) = 4948$; etc.

Taking into account the interleaving, or not, does not change the higher-order terms, in the following we simply sum up the values obtained without overlap, adding the cost of the QR factorization of size (p, q) to that of the bidiagonalization of the top square (q, q) matrix, and subtracting step $QR(1)$ as discussed above. In other words

$$\text{R-BiDiag}(p,q) \leq QR(p,q) + \text{BiDiag}(q,q) - QR_{1step}(q).$$

This leads us to the following upper bounds:

FlatTS

R-BiDiagFlatTS$(p,q) \leq (12p+18q-32)+(12q^2-4q-4)-(12q-14) = 12q^2+12p+2q-22$

FlatTT

R-BiDiagFlatTT$(p,q) \leq (6p+16q-22)+(6q^2+8q-10)-(6q-2) = 6q^2+6p+18q-30$

Greedy

R-BiDiagGreedy$(p,q) \leq (22q+o(q))+(12q\log_2(q)+(20-12\log_2(e))q+o(q))-o(q) = 12q\log_2(q)+(42-12\log_2(e))q+o(q)$ whenever $p=o(q^2)$.

For the sake of completeness, here are the exact values (obtained using the PaRSEC programming tool, see Section 6) for the critical paths of R-BiDiag with the FlatTS and FlatTT variants:

$$\text{R-BiDiagFlatTS}(p,q) = \begin{cases} 6p-2 & \text{if } q=1, \\ 36 & \text{if } q=2 \text{ and } q=p, \\ 12p+4 & \text{if } q=2 \text{ and } q\neq p, \\ 12q^2-16q+12p+4 & \text{if } q=3 \text{ and } q=p, \\ 12q^2-16q+12p+6 & \text{otherwise.} \end{cases}$$

$$\text{R-BiDiagFlatTT}(p,q) = \begin{cases} 2p+2 & \text{if } q=1, \\ 30 & \text{if } q=2 \text{ and } q=p, \\ 6p+20 & \text{if } q=2 \text{ and } q\neq p, \\ 6q^2+18q-36 & \text{if } q=3 \text{ and } q=p, \\ 6q^2+12q+6p-34 & \text{if } q=3 \text{ and } q\neq p, \\ 6q^2+6q+6p-16 & \text{otherwise.} \end{cases}$$

In both cases, we check that the exact and approximate values differ by a factor $O(q)$ and have same higher-order term $O(q^2)$.

4.3.3 Comparison between BiDiagGreedy and R-BiDiagGreedy

Again, we are interested in the asymptotic analysis of R-BiDiagGreedy, and in the comparison with BiDiagGreedy. In fact, when $p=o(q^2)$, say $p=\beta q^{1+\alpha}$, with $0 \leq \alpha < 1$, the cost of the QR factorization $QR(p,q)$ is negligible in front of the cost of the bidiagonalization BiDiagGreedy(q,q), so that R-BiDiagGreedy(p,q) is asymptotically equivalent to BiDiagGreedy(q,q), and we derive that:

$$\text{for } 0 \leq \alpha < 1, \quad \lim_{q\to\infty} \frac{\text{BiDiagGreedy}(\beta q^{1+\alpha}, q)}{\text{R-BiDiagGreedy}(\beta q^{1+\alpha}, q)} = 1 + \frac{\alpha}{2} \tag{5}$$

Asymptotically, BiDiagGreedy is at least as costly (with equality is p and q are proportional) and at most 1.5 times as costly as R-BiDiagGreedy (the maximum ratio being reached when $\alpha = 1-\varepsilon$ for small values of ε.

Just as before, R-BiDiagGreedy is asymptotically optimal among all possible reduction trees, and we have proven the following result, where for notation convenience we let BiDiag(p,q) and R-BiDiag(p,q) denote the optimal critical path lengths of the algorithms:

Theorem 4. *For $p = \beta q^{1+\alpha}$, with $0 \leq \alpha < 1$:*

$$\lim_{q\to\infty} \frac{\text{BiDiag}(p,q)}{(12+6\alpha)q\log_2(q)} = 1$$

$$\lim_{q\to\infty} \frac{\text{BiDiag}(p,q)}{\text{R-BiDiag}(p,q)} = 1 + \frac{\alpha}{2}$$

When p and q are proportional ($\alpha = 0$, $\beta \geq 1$), both algorithms have same asymptotic cost $12qlog_2(q)$. On the contrary, for very elongated matrices with fixed $q \geq 2$, the ratio of the critical path lengths of BiDiag and R-BiDiag gets high asymptotically: the cost of the QR factorization is equivalent to $6\log_2(p)$ and that of BiDiag(p,q) to $6q\log_2(p)$. Since the cost of BiDiag(q,q) is a constant for fixed q, we get a ratio of q. Finally, to give a more practical insight, Figure 2 reports the ratio of the critical paths of all schemes over that of BiDiag with FlatTS. We observe the superiority of R-BiDiag for tall and skinny matrices.

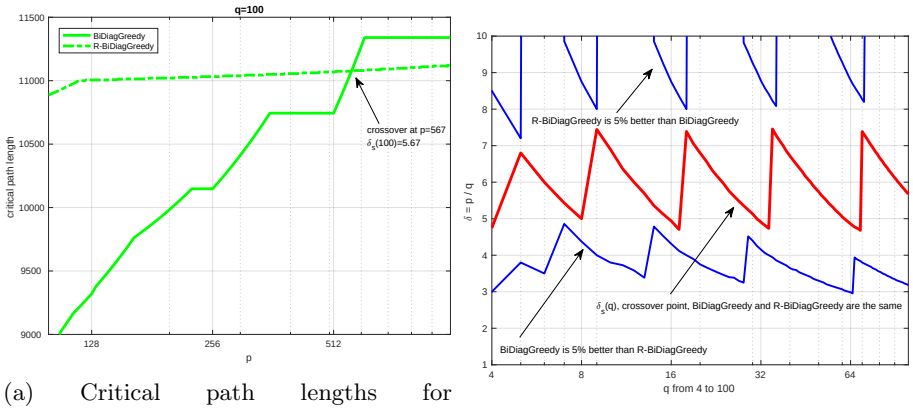
4.3.4 Switching from BiDiag and R-BiDiag

For square matrices, BiDiag is better than R-BiDiag. For tall and skinny matrices, this is the opposite. For a given q, what is the ratio $\delta = p/q$ for which we should switch between BiDiag and R-BiDiag? Let δ_s denote this crossover ratio. The question was answered by Chan [11] when considering the operation count. The question has multiple facets. The optimal switching point is not the same whether one wants singular values only, right singular vectors, left, or both (left and right). Chan [11] answers all these facets. For example, Chan [11] shows that the optimal switching point between BiDiag and R-BiDiag when singular values only are sought is $\delta = \frac{5}{3}$. Here, we consider a similar question but when critical path length (instead of number of flops) is the objective function. Since BiDiagGreedy is optimal and R-BiDiagGreedy is asymptotically optimal, we only focus on these two algorithms. It turns out that δ_s is a complicated function of q. The function of $\delta_s(q)$ oscillates between 5 and 8 (see Figure 3b) To find δ_s, we coded snippets that explicitly compute the critical path lengths for given p and q, and find the intersection for a given q. See Figure 3 for an example with $q = 100$. We see that, in this case, $\delta_s(100) = 5.67$.

4.4 Critical path length study

In Figures 2a, 2b and 2c, we present the critical path length speedup taking for base BiDiagFlatTS. The higher the better. We see that the Greedy base BiDiag and R-BiDiag algorithms (green curves) are much better any other trees as expected by the theory. In the square case, we do not present R-BiDiag (since there is no interest in R-BiDiag in the square case).

We set the dimension of the problem p and q so as to reproduce the dimension in our experimental section. In the experimental section, we will use a tile size of 160. Figure 2a studies the square case, it is the same condition as Figure 5a.



(a) Critical path lengths for BiDiagGreedy and R-BiDiagGreedy for $q = 100$ as a function of p. We see that $\delta_s(100) = 5.67$.

(b) Values of δ_s where to switch as a function of q.

Figure 3: Switching between BiDiag and R-BiDiag.

Figure 2c studies the rectangular case when $n = 2,000$, so when $q = 13$, it is the same condition as Figure 5b. Figure 2c studies the rectangular case when $n = 10,000$, so when $q = 63$, it is the same condition as Figure 5c.

From Figure 2a, we see that the critical path of BiDiagGreedy for a square 40x40 tile matrix is more than 6 times shorter than for BiDiagFlatTS.

From Figure 2b, we see that the critical path of BiDiagGreedy and R-BiDiagGreedy for a square 400x13 tile matrix is more than 60 times shorter than for BiDiagFlatTS. We also the crossover point between BiDiagGreedy and R-BiDiagGreedy. For "very tall and skinny" matrix ($p \gg q$), R-BiDiagGreedy is better than BiDiagGreedy. For "not-too tall and skinny" matrix ($p \approx q$), BiDiagGreedy is better than R-BiDiagGreedy.

4.5 Comments

When singular vectors are requested. It is important to understand that the optimality in Theorem 1 assumes that we do not request singular vectors. The whole analysis is significantly changed if one requests singular vectors. In this paper, we explain that, since there is no overlap between QR and LQ steps, then the shortest trees at each step makes up an optimal algorithm. When we apply the trees in reverse to obtain the singular vectors, then we apply all the LQ steps in sequence (without the QR steps) to obtain W, and we apply all the QR steps in sequence (without the LQ steps) to obtain U. In this case, we can clearly pipeline the trees as in a regular QR factorization. It is therefore interesting to use a "full" greedy tree or even a standard FlatTT tree where a pipeline happens. The BiDiagGreedy algorithm becomes less interesting. This paper focuses only on the bidiagonal phase. This means in some sense

that we are interested in the singular value problem where we only request the singular values, we do not request the singular vectors. Another study should be done when the singular vectors are requested. However, we believe that, despite the absence of pipeline in the construction of the singular vectors phase, BiDiagGreedy is still an algorithm of choice since the overhead in the construction is of order $\log_2 p$ while the overhead of the other algorithms in the bidiagonalization is of order p.

Three-step algorithm: partialGE2BD+preQR+GE2BD. In 1997, Trefethen and Bau [37] present a "three-step" approach. The idea is that having a cut-off when to switch from BiDiag to R-BiDiag algorithm seems unnatural and leads to a continuous but not smooth function. Smooth is better and seems more natural. So instead of doing either R-BiDiag or BiDiag, we consider the following algorithm. We start with BiDiag until the matrix is tall and skinny enough. (We note that BiDiag makes the matrix more and more tall and skinny.) Once the matrix is tall and skinny enough, we switch to R-BiDiag. Trefethen and Bau [37] studied the "three-step" approach where they want to minimize the number of flops. Here we experimentally study their "three-step" approach when we want to minimize the critical path length. We do not have an exact formula to know when to stop the first bidiagonalization. Therefore, for a first study, we consider the brute force approach and consider all possible stops for the first bidiagonalization, and then we take the one stop that gives the shortest critical path length. In fine, we obtain for example a figure like Figure 4.

We fix $q = 30$ and consider p varying from 30 to 400. We see that three-step algorithm (black curve) is always the best. (This is expected because we consider all possibilities including BiDiag and R-BiDiag, and take the minimum over all possibilities. So clearly, three-step will always be better than BiDiag and R-BiDiag.) Initially three-step is the same as BiDiag. Then, in the middle region $p = 60$ to 260, we see that three-step is better than both BiDiag and R-BiDiag. This is when indeed we do not perform BiDiag, we do not perform R-BiDiag, we perform a "true" three-step algorithm where we start a bidiagonalization algorithm, then stop the bidiagonalization, switch to a QR factorization of the remaining submatrix, and finish with a bidiagonalization. Finally after $p = 260$, three-step is same as R-BiDiag.

Once more, the idea behind three-step is that: (1) R-BiDiag is better for tall-and-skinny matrices; (2) the BiDiag process makes the trailing submatrix more and more tall-and-skinny. So the idea is to start BiDiag until the matrix is tall-and-skinny enough and switch to R-BiDiag. If we consider the minimum of BiDiag (blue curve) and R-BiDiag (red curve) then we have a non-smooth function. This is not natural. When we consider the three-step algorithm (black curve), we have a nice smooth function. This is more natural.

We see that three-step is better than BiDiag and R-BiDiag. However when we look at the gain (y-axis), we do not have staggering gain. So, while three-step is an interesting and elegant idea, we did not push its study further.

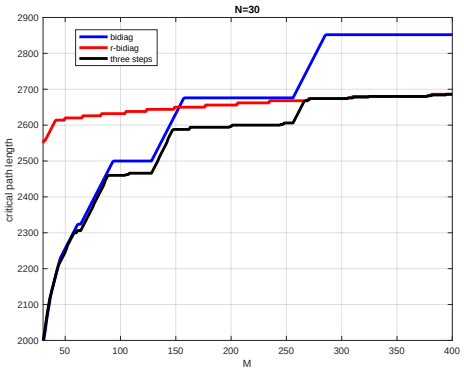


Figure 4: BiDiag, R-BiDiag and three-step algorithm for $q = 30$ and p varying from 30 to 400.

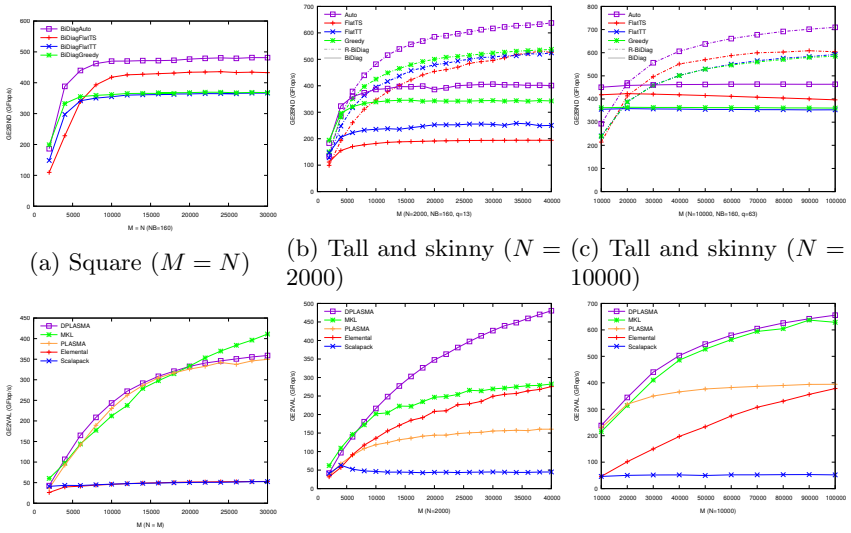
5 Implementation

To evaluate experimentally the impact of the different reduction trees on the performance of the GE2BND and GE2VAL algorithms, we have implemented both the BiDiag and R-BiDiag algorithms in the DPLASMA library [5], which runs on top of the PaRSEC runtime system [6]. PaRSEC is a high-performance fully-distributed scheduling environment for generic data-flow algorithms. It takes as input a problem-size-independent, symbolic representation of a Direct Acyclic Graph in which each node represents a tasks, and each edge a dependency, or data movement, from one task to another. PaRSEC schedules those tasks on distributed parallel machine of multi-cores, potentially heterogeneous, while complying with the dependencies expressed by the programmer. At runtime, tasks executions trigger data movements, and create new ready tasks, following the dependencies defined by the DAG representation. The runtime engine is responsible for actually moving the data from one machine (node) to another, if necessary, using an underlying communication mechanism, like MPI. Tasks that are ready to compute are scheduled according to a data-reuse heuristic: each core will try to execute close successors of the last task it ran, under the assumption that these tasks require data that was just touched by the terminated one. This policy is tuned by the user through a priority function: among the tasks of a given core, the choice is done following this function. To balance load between the cores, tasks of a same cluster in the algorithm (reside on a same shared memory machine) are shared between the computing cores, and a NUMA-aware job stealing policy is implemented. The user is then responsible only to provide the algorithm, the initial data distribution, and potentially the task distribution. The last one is usually correlated to the data distribution when the (default) owner-compute rule is applied. In our case, we use a $2D$ block-cyclic data distribution as used in the ScaLAPACK library, and we map the computation together with the data. A full description of PaRSEC can be

found in [6].

The implementation of the BiDiag and R-BiDiag algorithms have then been designed as an extension of our previous work on HQR factorization [14] within the DPLASMA library. The HQR algorithm proposes to perform the tiled QR factorization of a $(p \times q)$-tile matrix, with $p \geq q$, by using a variety of trees that are optimized for both the target architecture and the matrix size. It relies on multi-level reduction trees. The highest level is a tree of size R, where R is the number of rows in the $R \times C$ two-dimensional grid distribution of the matrix, and it is configured by default to be a flat tree if $p \geq 2q$, and a Fibonacci tree otherwise. The second level, the domino level, is an optional intermediate level that enhances the pipeline of the lowest levels when they are connected together by the highest distributed tree. It is by default disabled when $p \geq 2q$, and enabled otherwise. Finally, the last two levels of trees are use to create parallelism within a node and work only on local tiles. They correspond to a composition of one or multiple FlatTS trees that are connected together with an arbitrary tree of TT kernels. The bottom FlatTS tree enables highly efficient kernels while the TT tree on top of it generates more parallelism to feed all the computing resources from the architecture. The default is to have FlatTS trees of size 4 that are connected by a Greedy tree in all cases. This design is for QR trees, a similar design exists for LQ trees. Using these building blocks, we have crafted an implementation of BiDiag and R-BiDiag within the abridged representation used by PaRSEC to represent algorithms. This implementation is independent of the type of trees selected for the computation, thereby allowing the user to test a large spectrum of configuration without the harassment of rewriting all the algorithm variants.

One important contribution is the introduction of two new tree structures dedicated to the BiDiag algorithm. The first tree, Greedy, is a binomial tree which reduces a panel in the minimum amount of steps. The second tree, Auto, is an adaptive tree which automatically adapts to the size of the local panel and number of computing resources. We developed the auto-adaptive tree to take advantage of (i) the higher efficiency of the TS kernels with respect to the TT kernels, (ii) the highest degree of parallelism of the Greedy tree with respect to any other tree, and (iii) the complete independence of each step of the BiDiag algorithm, which precludes any possibility of pipelining. Thus, we propose to combine in this configuration a set of FlatTS trees connected by a Greedy tree, and to automatically adapt the number of FlatTS trees, and by construction their sizes, a, to provide enough parallelism to the available computing resources. Given a matrix of size $p \times q$, at each step k, we need to apply a QR factorization on a matrix of size $(p-k-1) \times (q-k-1)$, the number of parallel tasks available at the step beginning of the step is given by $\lceil (p-k-1)/a \rceil * (q-k-1)$. Note that we consider the panel as being computed in parallel of the update, which is the case when a is greater than 1, with an offset of one time unit. Based on this formula, we compute a at each step of the factorization such that the degree of parallelism is greater than a quantity $\gamma \times nb_{cores}$, where γ is a parameter and nb_{cores} is the number of cores. For the experiments, we set $\gamma = 2$. Finally, we point out that Auto is defined for



(a) Square ($M = N$)

(b) Tall and skinny ($N = 2000$)

(c) Tall and skinny ($N = 10000$)

(d) Square ($M = N$)

(e) Tall and skinny ($N = 2000$)

(f) Tall and skinny ($N = 10000$)

Figure 5: Shared memory performance of the multiple variants for the GE2BND algorithm on the first row, and for the GE2VAL algorithm on the second row, using a single 24 core node of the `miriel` cluster.

a resourced-limited platform, hence computing its critical path would have no meaning, which explains a posteriori that it was not studied in Section 4.

6 Experiments

In this section, we evaluate the performance of the proposed algorithms for the GE2BND kernel against existing competitors.

6.1 Architecture

Experiments are carried out using the PLAFRIM experimental testbed[1]. We used up to 25 nodes of the `miriel` cluster, each equipped with 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 and 128GB of memory. The nodes are interconnected with an Infiniband QDR TrueScale network with provides a bandwidth of 40Gb/s. All the software are compiled with gcc 4.9.2, and linked against the sequential BLAS implementation of the Intel MKL 11.2 library. For the distributed runs, the MPI library used is OpenMPI 2.0.0. The practical GEMM performance is of 37 GFlop/s on one core, and 642 GFlop/s when the 24 cores are used. For each experiment, we generated a matrix with prescribed singular values using LAPACK `LATMS` matrix generator and checked that the computed singular values were satisfactory up to machine precision.

6.2 Competitors

This paper presents new parallel distributed algorithms and implementations for GE2BND using DPLASMA. To compare against competitors on GE2VAL, we follow up our DPLASMA GE2BND implementation with the PLASMA multi-threaded BND2BD algorithm, and then use the Intel MKL multi-threaded BD2VAL implementation. We thus obtain GEVAL by doing GE2BND+BND2BD+BD2VAL.

It is important to note that we do not use parallel distributed implementations neither for BND2BD nor for BD2VAL. We only use shared memory implementations for these two last steps. Thus, for our distributed memory runs, after the GE2BND step in parallel distributed using DPLASMA, the band is gathered on a single node, and BND2BD+BD2VAL is performed by this node while all all other nodes are left idle. We will show that, despite this current limitation for parallel distributed, our implementation outperforms its competitors.

On the square test cases, only 23 cores of a 24-core node were used for computation, and the 24^{th} core was left free to handle MPI communications progress.

The implementation of the algorithm is available in a public fork of the DPLASMA library at `https://bitbucket.org/mfaverge/parsec`.

[1]Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS, see `https://www.plafrim.fr/`.

PLASMA is the closest alternative to our proposed solution but it is only using FLATTS as its reduction tree, and is limited to single-node platform, and is supported by a different runtime. For both our code, and PLASMA, the tile size parameter is critical to get good performance: a large tile size will get an higher kernel efficiency and a faster computation of the band, but it will increase the number of flops of the BND2BD step which is heavily memory bound. On the contrary, a small tile size will speed up the BND2BD step by fitting the band into cache memory, but decreases the efficiency of the kernels used in the GE2BND step. We tuned the n_b (tile size) and i_b (internal blocking in TS and TT kernels) parameters to get the better performance on the square case $m = n = 20000$, and $m = n = 30000$ on the PLASMA code. The selected values are $n_b = 160$, and $i_b = 32$. We used the same parameters in the DPLASMA implementation for both the shared memory runs ant the distributed ones. The PLASMA 2.8.0 library was used.

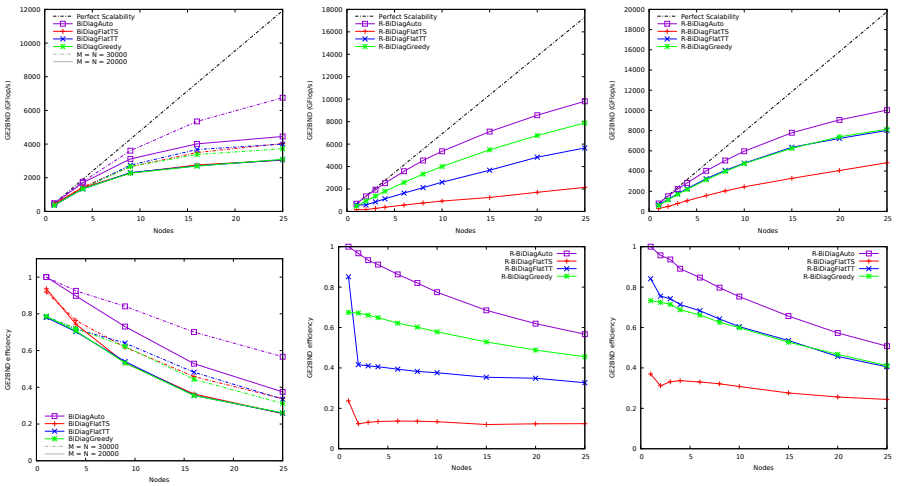
Intel MKL proposes an multi-threaded implementation of the GE2VAL algorithm which gained an important speedup while switching from version 11.1 to 11.2 [38]. While it is unclear which algorithm is used beneath, the speedup reflects the move to a multi-stage algorithm. **Intel MKL** is limited to single-node platforms.

ScaLAPACK implements the parallel distributed version of the LAPACK GEBRD algorithm which interleaves phases of memory bound BLAS2 calls with computational bound BLAS3 calls. It can be used either with one process per core and a sequential BLAS implementation, or with a process per node and a multi-threaded BLAS implementation. The latter being less efficient, we used the former for the experiments. The blocking size n_b is critical to get performances since it impacts the phase interleaving. We tuned the n_b parameter to get the better performance on a single node with the same test cases as for PLASMA, and $n_b = 48$ was selected.

Elemental implements an algorithm similar to SCALAPACK, but it automatically switches to Chan's algorithm [11] when $m \geq 1.2n$. As for SCALAPACK, it is possible to use it as a single MPI implementation, or an hybrid MPI-thread implementation. The first one being recommended, we used this solution. Tuning of the n_b parameter similarly to previous libraries gave us the value $nb = 96$. A better algorithm developed on top of the LibFLAME [23] is provided by Elemental, but this one is used only when singular vectors are sought.

In the following, we compare all these implementation on the `miriel` cluster with 3 main configurations: (i) square matrices; (ii) tall and skinny matrices with $n = 2,000$; this choice restricts the level of parallelism induced by the number of panels to half the cores; and (iii) tall and skinny matrices with $n = 10,000$: this choice enables for more parallelism. For all performance comparisons, we use the same operation count as in [4, p. 123] for the GE2BND and GE2VAL algorithms. The BD2VAL step has a negligible cost $O(n^2)$. For R-BIDIAG, we use the same number of flops as for BIDIAG; in other words, we do not assess the absolute performance of R-BIDIAG, instead we provide a direct comparison with BIDIAG.

6.3 Shared Memory



Square ($m = n$) Tall & skinny ($m = 2,000,000, n = 2,000$) Tall and skinny ($m = 1,000,000, n = 10,000$)

Figure 6: Distributed memory performance (first row) and efficiency (second row) of the multiple variants for the GE2BND algorithm on the `miriel` cluster. Grid data distributions are $\sqrt{nb_{nodes}} \times \sqrt{nb_{nodes}}$ for square matrices, and $nb_{nodes} \times 1$ for tall and skinny matrices. For the square case, solid lines are for $m = n = 20,000$ and dashed lines for $m = n = 30,000$.

The top row of Figure 5 presents the performance of the three configurations selected for our study of GE2BND. On the top left, the square case perfectly illustrates the strengths and weaknesses of each configuration. On small matrices, FlatTT in blue and Greedy in green illustrate the importance of creating algorithmically more parallelism to feed all resources. However, on large size problems, the performance is limited by the lower efficiency of the TT kernels. The FlatTS tree behaves at the opposite: it provides better asymptotic performance thanks to the TS kernels, but lacks parallelism when the problem is too small to feed all cores. Auto is able to benefit from the advantages of both Greedy and FlatTS trees to provide a significant improvement on small matrices, and a 10% speedup on the larger matrices.

For the tall and skinny matrices, we observe that the R-BiDiag algorithm (dashed lines) quickly outperforms the BiDiag algorithm, and is up to 1.8 faster. On the small case ($n = 2,000$), the crossover point is immediate, and both FlatTT and Greedy, exposing more parallelism, are able to get better performances than FlatTS. On the larger case ($n = 10,000$), the parallelism from the larger matrix size allows FlatTS to perform better, and to postpone the crossover point due to the ratio in the number of flops. In both cases, Auto provides the better performance with an extra 100 GFlop/s.

On the bottom row of Figure 5, we compare our best solutions, namely AUTO tree with BIDIAG for square cases and with R-BIDIAG on tall and skinny cases, to the competitors on the GE2VAL algorithm. The difference between our solution and PLASMA, which is using the FLATTS tree, is not as impressive due to the additional BND2BD and BD2VAL steps which have limited parallel efficiency. Furthermore, in our implementation, due to the change of runtime, we cannot pipeline the GE2BND and BND2BD steps to partially overlap the second step. However these two solutions still provide a good improvement over MKL which is slower on the small cases but overtakes at larger sizes. For such sizes, Elemental and SCALAPACK are note able to scale and reach up a maximum of 50 Gflop/s due to their highly memory bound algorithm.

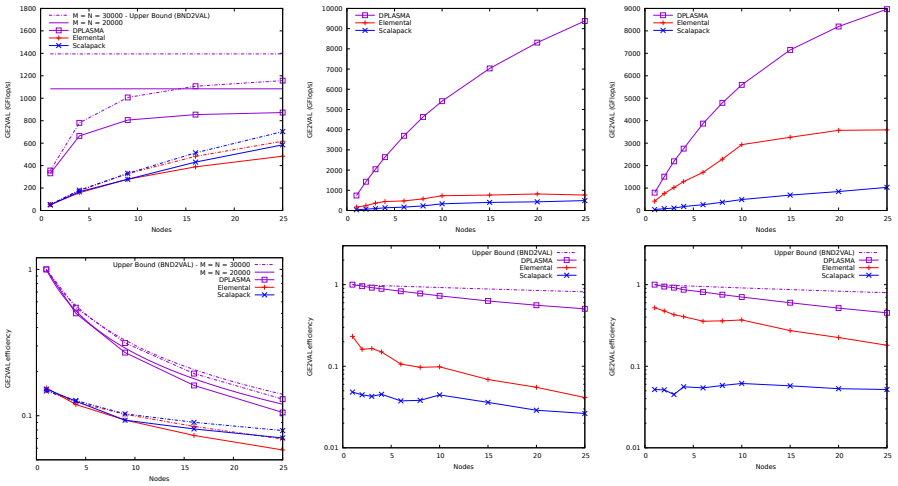
On the tall and skinny cases, differences are more emphasized. We see the limitation of using only the BIDIAG algorithm on MKL, PLASMA and SCALAPACK, while our solution and elemental keep scaling up with matrix size. We also observe that MKL behaves correctly on the second test case, while it quickly saturates in the first one where the parallelism is less important. In that case, we are able to reach twice the MKL performance.

6.4 Distributed Memory

Strong Scaling Figure 6 presents a scalability study of the three variants on 4 cases: two square matrices with BIDIAG, and two tall and skinny matrices with R-BIDIAG. For all of them, we couple high-level distributed trees, and low-level shared memory trees. FLATTS and FLATTT configuration are coupled with a high level flat tree, while GREEDY and AUTO are coupled with a high level GREEDY tree. The configuration of the preQR step is setup similarly, except for AUTO which is using the automatic configuration described previously.

On all cases, performances are as expected. FLATTS, which is able to provide higher efficient kernels, hardly behaves better on the large square case; GREEDY, which provides better parallelism, is the best solution out of the three on the first tall and skinny case. We also observe the impact of the high level tree: GREEDY doubles the number of communications on square cases [14], which impacts its performance and gives an advantage to the flat tree which performs half the communication volume. Overall, AUTO keeps taking benefit from its flexibility, and scales well despite the fact that local matrices are less than 38×38 tiles, so less than 2 columns per core.

When considering the full GE2VAL algorithm on Figure 7, we observe a huge drop in the overall performance. This is due to the integration of the shared memory BND2BD and BD2VAL steps which do not scale when adding more nodes. For the the square case, we added the upper bound that we cannot beat due to those two steps. However, despite this limitation, our solution brings an important speedup to algorithms looking for the singular values, with respect to the competitors presented here. Elemental again benefits from the automatic switch to the R-BIDIAG algorithm, which allows a better scaling on tall and skinny matrices. However, it surprisingly reaches a plateau after 10 nodes where the performance stops increasing significantly. Our solution auto-



Square ($m = n$) | Tall and skinny ($m = 2,000,000, n = 2,000$) | Tall and skinny ($m = 1,000,000, n = 10,000$)

Figure 7: Distributed memory performance (first row) and efficiency (second row) of the GE2VAL algorithm on the `miriel` cluster. Grid data distributions are $\sqrt{nb_{nodes}} \times \sqrt{nb_{nodes}}$ for square matrices, and $nb_{nodes} \times 1$ for tall and skinny matrices.

matically adapts to create more or fewer parallelism, and reduces the amount of communications, which allows it to sustain a good speedup up to 25 nodes (600 cores).

Weak Scaling Figure 8 presents a weak scalability study with tall and skinny matrices of width $n = 2,000$ on the first row, and $n = 10,000$ on the second row[2]. As previously, FlatTS quickly saturates due to its lack of parallelism. FlatTT is able to compete with, and even to outperform, Greedy on the larger case due to its lower communication volume. Auto offers a better scaling and is able to reach 10 TFlop/s which represents 400 to 475 GFlop/s per node. When comparing to Elemental and ScaLAPACK on the GE2VAL algorithm, the proposed solution offers a much better scalability. Both Elemental and ScaLAPACK suffer from their memory bound BiDiag algorithm. With the switch to a R-BiDiag algorithm, Elemental is able to provide better performance than ScaLAPACK, but the lack of scalability of the Elemental QR factorization compared to the HQR implementation quickly limits the overall performance of the GE2VAL implementation.

[2]Experiments for the $n = 10,000$ case stop at 20 nodes due to the 32 bit integer default interface for all libraries

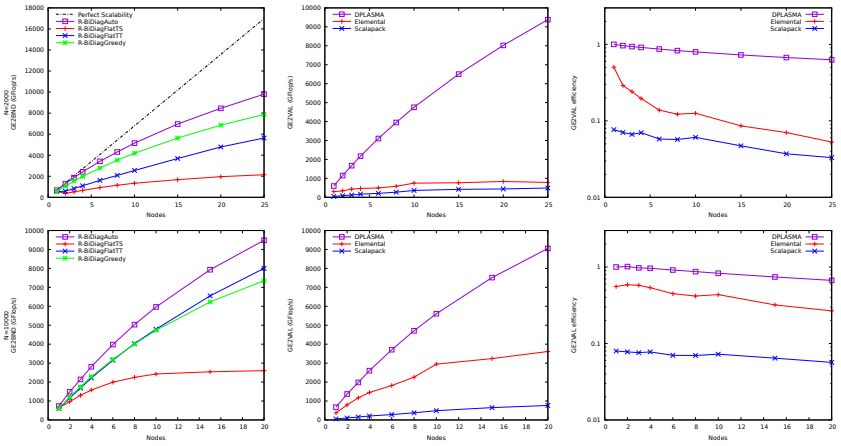


Figure 8: Study of the distributed weak scalability on tall and skinny matrices of size $(80,000\ nb_{nodes}) \times 2,000$ on the first row, and $(100,000\ nb_{nodes}) \times 10,000$ on the second row. First column presents the GE2BND performance, second column the GE2VAL performance, and third column the GE2VAL scaling efficiency.

7 Conclusion

In this paper, we have presented the use of many reduction trees for tiled bidiagonalization algorithms. We proved that, during the bidiagonalization process, the alternating QR and LQ reduction trees cannot overlap. Therefore, minimizing the time of each individual tree will minimize the overall time. Consequently, if one considers an unbounded number of cores and no communication, one will want to use a succession of greedy trees. We show that such an approach (BiDiagGreedy) is asymptotically much better than previously presented approach (FlatTS). In practice, in order to have an effective solution, one have to take into account load balancing and communication, hence we propose trees that adapt to the parallel distributed topology (highest level tree) and enable more sequential but faster kernels on a node (Auto).

We have also studied R-bidiagonalization in the context of tiled algorithms. While R-bidiagonalization is not new, it had never been used in the context of tiled algorithms. Previous work was comparing bidiagonalization and R-bidiagonalization in term of flops, while our comparison is conducted in term of critical path lengths. We show that bidiagonalization has a shorter critical path than R-bidiagonalization, that this is the opposite for tall and skinny matrices, and provide an asymptotic analysis. Along all this work, we give detailed critical path lengths for many of the algorithms under study.

Our implementation is the first parallel distributed tiled algorithm implementation for bidiagonalization. We show the benefit of our approach (DPLASMA)

on a multicore node against existing software (PLASMA, Intel MKL, Elemental and ScaLAPACK) for various matrix sizes, for computing the singular values of a matrix. We also show the benefit of our approach (DPLASMA) on a few multicore nodes against existing software (Elemental and ScaLAPACK) for various matrix sizes, for computing the singular values of a matrix.

Future work will be devoted to gain access to a large distributed platform with a high count of multicore nodes, and to assess the efficiency and scalability of our parallel distributed BiDiag and R-BiDiag algorithms. Other research directions are the following: (i) investigate the trade-off of our approach when singular vectors are requested; a previous study [24] in shared memory was conclusive for FlatTS and no R-BiDiag (square matrices only); the question is to study the problem on parallel distributed platforms, with or without R-BiDiag, for various shapes of matrices and various trees; and (ii) develop a scalable parallel distributed BND2BD step; for now, for parallel distributed experiments on many nodes, we are limited in scalability by the BND2BD step, since it is performed using the shared memory library PLASMA on a single node.

Acknowledgments

Work by J. Langou was partially supported by NSF award 1054864 and NSF award 1645514. Yves Robert is with Institut Universitaire de France.

References

[1] E. Agullo, B. Hadri, H. Ltaief, and J. Dongarra. Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, pages 1–12. IEEE Computer Society Press, 2009.

[2] M. W. Berry. Large-scale sparse singular value computations. *Int. J. Supercomputer. Appl.*, 6(1):13–49, 1992.

[3] C. Bischof and C. V. Loan. The WY representation for products of Householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987.

[4] S. Blackford and J. J. Dongarra. Installation guide for LAPACK. Technical Report 41, LAPACK Working Note, June 1999. originally released March 1992.

[5] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra. Flexible development of dense linear algebra algorithms

on massively parallel architectures with DPLASMA. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1432–1441, May 2011.

[6] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra. DAGuE: A generic distributed DAG engine for High Performance Computing. *Parallel Computing*, 38(1–2):37–51, 2012. Extensions for Next-Generation Parallel Programming Models.

[7] H. Bouwmeester. *Tiled Algorithms for Matrix Computations on Multicore Architectures*. PhD thesis, Colorado University DEnver, 2013. Available at `http://arxiv.org/abs/1303.3182`.

[8] H. Bouwmeester, M. Jacquelin, J. Langou, and Y. Robert. Tiled QR factorization algorithms. In *SC'2011, the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis*. ACM Press, 2011.

[9] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency Computat.: Pract. Exper.*, 20(13):1573–1590, 2008.

[10] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, 2009.

[11] T. F. Chan. An improved algorithm for computing the singular value decomposition. *ACM Trans. Math. Softw.*, 8(1):72–83, Mar. 1982.

[12] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numerical Algorithms*, 10(2):379–399, 1995.

[13] M. Cosnard, J.-M. Muller, and Y. Robert. Parallel QR decomposition of a rectangular matrix. *Numerische Mathematik*, 48:239–249, 1986.

[14] J. Dongarra, M. Faverge, T. Herault, M. Jacquelin, J. Langou, and Y. Robert. Hierarchical QR factorization algorithms for multi-core cluster systems. *Parallel Computing*, 39(4-5):212–232, 2013.

[15] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27(1):215 – 227, 1989.

[16] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm. I. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1322–1342, 2008.

[17] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm. II. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1343–1362, 2008.

[18] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[19] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, 2(2):205–224, 1965.

[20] G. H. Golub and C. F. V. Loan. *Matrix computations*. Johns Hopkins, 1989.

[21] B. Großer and B. Lang. Efficient parallel reduction to bidiagonal form. *Parallel Comput.*, 25(8):969–986, Aug. 1999.

[22] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM Journal on Matrix Analysis and Applications*, 16(1):79–92, 1995.

[23] J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn. Flame: Formal linear algebra methods environment. *ACM Trans. Math. Softw.*, 27(4):422–455, Dec. 2001.

[24] A. Haidar, J. Kurzak, and P. Luszczek. An improved parallel singular value algorithm and its implementation for multicore hardware. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 90:1–90:12, New York, NY, USA, 2013. ACM.

[25] A. Haidar, H. Ltaief, P. Luszczek, and J. Dongarra. A comprehensive study of task coalescing for selecting parallelism granularity in a two-stage bidiagonal reduction. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 25–35, May 2012.

[26] M. E. Hochstenbach. A jacobi–davidson type svd method. *SIAM Journal on Scientific Computing*, 23(2):606–628, 2001.

[27] Z. Jia and D. Niu. An implicitly restarted refined bidiagonalization lanczos method for computing a partial singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 25(1):246–265, 2003.

[28] B. Lang. Parallel reduction of banded matrices to bidiagonal form. *Parallel Computing*, 22(1):1 – 18, 1996.

[29] R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 27(537), 1998.

[30] C. Lawson and R. Hanson. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, 1995. reprinting with corrections and a new appendix of a 1974 Prentice Hall text.

[31] H. Ltaief, J. Kurzak, and J. Dongarra. Parallel two-sided matrix reduction to band bidiagonal form on multicore architectures. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):417–423, Apr. 2010.

[32] H. Ltaief, P. Luszczek, and J. Dongarra. Enhancing parallelism of tile bidiagonal transformation on multicore architectures using tree reduction. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics: 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part I*, pages 661–670. Springer Berlin Heidelberg, 2012.

[33] H. Ltaief, P. Luszczek, and J. Dongarra. High-performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures. *ACM Trans. Math. Softw.*, 39(3):16:1–16:22, May 2013.

[34] Y. Nakatsukasa and N. J. Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD. *SIAM Journal on Scientific Computing*, 35(3):A1325–A1349, 2013.

[35] S. Rajamanickam. *Efficient algorithms for sparse singular value decomposition*. PhD thesis, University of Florida, 2009.

[36] R. Schreiber and C. V. Loan. A storage-efficient WY representation for products of householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989.

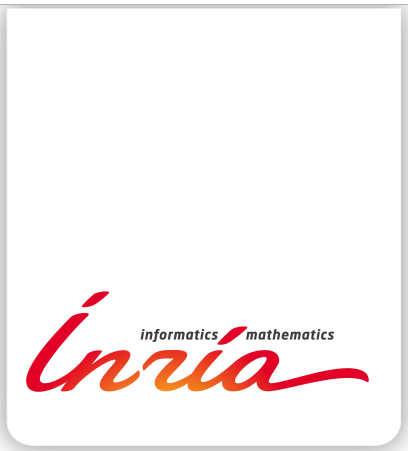
[37] L. N. Trefethen and D. Bau. *Numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[38] K. E. K. Vipin. Significant performance improvement of symmetric eigensolvers and SVD in Intel MKL 11.2, 2014. https://software.intel.com/en-us/articles/significant-performance-improvment-of-symmetric-eigensolvers-and-svd-in-intel-mkl-112.

[39] P. R. Willems, B. Lang, and C. Vömel. Computing the bidiagonal svd using multiple relatively robust representations. *SIAM Journal on Matrix Analysis and Applications*, 28(4):907–926, 2006.

[40] L. Wu, E. Romero, and A. Stathopoulos. PRIMME_SVDS: A high-performance preconditioned SVD solver for accurate large-scale computations. Technical Report 1607.01404, arXiv, 2016.

[41] L. Wu and A. Stathopoulos. A preconditioned hybrid svd method for accurately computing singular triplets of large matrices. *SIAM Journal on Scientific Computing*, 37(5):S365–S388, 2015.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399