Fast Symmetric Eigenvalue Decomposition via WY Representation on Tensor Core

Shaoshuai Zhang, Ruchi Shah
{szhang36,rkshah5}@uh.edu
Department of Computer Science
University of Houston
Texas USA

Hiroyuki Ootomo, Rio Yokota
{ootomo.h@rio.,rioyokota@}gsic.titech.ac.jp
Global Scientific Information and Computing Center
Tokyo Institute of Technology
Tokyo Japan

Panruo Wu
pwu7@uh.edu
Department of Computer Science
University of Houston
Texas USA

Abstract

Symmetric eigenvalue decomposition (EVD) is a fundamental analytic and numerical tool used in many scientific areas. The state-of-the-art algorithm in terms of performance is typically the two-stage tridiagonalization method. The first stage in the two-stage tridiagonalization is called successive band reduction (SBR), which reduces a symmetric matrix to a band form, and its computational cost usually dominates. When Tensor Core (specialized matrix computational accelerator) is used to accelerate the expensive EVD, the conventional ZY-representation-based method results in suboptimal performance due to unfavorable shapes of the matrix computations. In this paper, we propose a new method that uses WY representation instead of ZY representation (see Section 3.2 for details), which can provide a better combination of locality and parallelism so as to perform better on Tensor Cores. Experimentally, the proposed method can bring up to 3.7x speedup in SBR and 2.3x in the entire EVD compared to state-of-the-art implementations.

CCS Concepts • **Mathematics of computing** → **Mathematical software performance**; • **Theory of computation** → **Parallel algorithms**; • **Computing methodologies** → **Parallel algorithms**.

Keywords Eigenvalue Decomposition, GPGPU, Numerical Linear Algebra, Tensor Core, HPC, Mixed-precision Computation, Matrix Computation, Singular Value Decomposition, Low Rank Approximation

PPoPP '23, February 25-March 1, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0015-6/23/02…$15.00
https://doi.org/10.1145/3572848.3577516

1 Introduction

With the advances in processor architecture, Tensor Core, a specialized unit of Nvidia Volta, Turing, and Ampere architecture, is designed to perform highly efficient half-precision (16-bit floating-point format) matrix multiplications accumulated in single precision. Recently, Tensor Core has been widely used to train large-scale deep neural networks by accelerating GEMMs (general matrix-matrix multiplications). Researchers also try to find the possibilities of deploying linear algebra algorithms on Tensor Cores. These algorithms include LU factorization [20], QR factorization [41] and some BLAS3 operations [42].

Eigenvalue decomposition is another important matrix computation in numerical linear algebra, as it is useful in incredibly diverse applications, ranging from quantum chemistry [33], quantum mechanics [15], and quantum physics [32], to data drive analysis and numerous machine learning and signal processing tasks. Some of these applications may require high-precision arithmetic, such as double precision, but increasingly single precision or even lower precision suffices in many emerging data-driven approaches. For example, principle component analysis [1], low-rank approximation [29], second-order optimization in deep learning [17] and the related applications [35, 43]. And Tensor Cores can typically be utilized to accelerate these algorithms and applications.

The symmetric eigenvalue problem can be defined by:

$$A = X \times \Lambda \times X^{-1}$$

Where A is a symmetric matrix, X is an orthogonal matrix containing eigenvectors, and Λ is a diagonal matrix that has eigenvalues on its diagonal. The most critical step in computing the eigenvalues of a symmetric dense matrix is tridiagonalization. The tridiagonalization step can be reliably computed by the celebrated Householder tridiagonalization [13]. Alternatively, the blocked variant from LAPACK involves a panel factorization and two-sided matrix trailing update [2], which improves locality substantially. The parallel implementation can be found in [18] and [24]. A more elaborate approach by Bischof and Sun [7] proposed the two-stage successive band reduction further to improve the performance of the hierarchical memory system. 1)The first

stage involves orthogonal similarity transformations to decompose the symmetric dense matrix to a band form:

$$A = X \times B \times X^{-1}$$

where B is a symmetric band matrix; 2)In the second stage, the bulge chasing transformation is performed to reduce the band form to a tridiagonal form. Some related investigation suggested that using the two-phase approach reduces the communication and computation bottleneck, yielding a better throughput [24].

However, there is no existing work regarding Tensor-Core-based eigenvalue decomposition or successive band reduction. There are two challenges. Firstly, the lower precision arithmetic in Tensor Core requires the careful design of algorithms. For example, the mixed-precision matrix factorization algorithms often follow an approximate-iterate scheme that first gets the approximate solution (preconditioner) from a low-precision factorization and then iteratively refines the solution to higher accuracy. This approximate-iterate approach is more difficult in EVD. According to the previous literature, only the SICE algorithm [38] can partially meet the requirement, as this algorithm can only work with a portion of the eigenvalues and eigenvectors requested. Secondly, the matrix shapes of conventional algorithms cannot be accelerated sufficiently by Tensor Cores whether the conventional tridiagonalization or the 2-stage tridiagonalization cannot avoid a large amount of tall and skinny GEMMs, which performs slowly on Tensor Cores. Particularly, using ZY representation in SBR can utilize the symmetric property in trailing matrix update (using `syr2k` routine), but Tensor Core does not support this kind of GEMM natively.

As a result, to improve the performance, we focus on SBR and demonstrate that using modified WY representation instead of conventional ZY representation can notably improve the performance by 'squeezing' the GEMMs in this paper. Furthermore, for the accuracy issue, if single precision is needed, we also introduce a new technique named error-correlation Tensor-Core-based GEMMs (EC-TCGEMM) [31] to bring the accuracy back to single precision from the beginning. Thus, we consider our contributions to be:

- We devise, implement and evaluate a fast and stable tall and skinny QR panel, which brings around 5x speedup compared to MAGMA and cuSOLVER panel factorization.
- We analyze and study the performance of using WY representation [5] and traditional ZY representation [12] in SBR, which addresses that the algorithm that generally would not be considered (WY-based) can perform better than the conventional algorithm (ZY-based) on new architecture (Tensor Core).
- We provide several implementations of SBR that can work in different situations. Compared to the state-of-the-art MAGMA relative routine, the speedup is 3.7x (half precision) and 1.8x (single precision) if eigenvectors are not needed.
- We check the feasibility of combining MAGMA's bulge-chasing process and divide and conquer and our SBR implementation, and the overall EVD performance improvement is up to 2.3x.

The rest of the paper is organized as follows: Section II addresses the related work, including band reduction, tridiagonalization, and Tensor Core techniques; section III gives some background knowledge regarding tridiagonalization, Householder transformation, and symmetric band reduction; section IV analyzes the reasons behind the higher performance of WY-based algorithm and shows some experimental results regarding the inside GEMMs; section V introduces the details of our implementation and optimization, and it also shows some related experimental results; section VI compares the performance and accuracy results with the state-of-the-art software and tools, and section VII draws the conclusion and depicts our future work.

2 Related Work

The investigation of the symmetric eigenvalue decomposition problem has a long history. Furthermore, the algorithms and methods in this area have been extensively studied.

2.1 Tridiagonalization and Tridiagonal Solver

The most widely used and well-known algorithm is the QR algorithm [39], which repeatedly calls the QR factorization and GEMM and finally converges to a diagonal matrix that contains eigenvalues on its diagonal. However, before the QR and GEMM iterations, a tridiagonalization step is usually processed as a 'preconditioner' to reduce the number of QR iterations. Typically, the tridiagonalization is performed by Householder transformation [14] and to improve the execution efficiency on modern high-performance architectures, the WY representation technique [5, 34] is generally applied to the transformation process. Another tridiagonalization method is called 2-stage tridiagonalization [19], which firstly reduces the matrix to a band form (1st stage) and then reduces the band form to a tridiagonal matrix (2nd stage). This method is proved pretty efficient on multi-core architectures [24, 25].

Another popular method is divide and conquer, and this method is implemented in most of the linear algebra packages, including LAPACK [2], MAGMA [37] and CuSOLVER[1]. Another flexible method is called bisection [10], which aims to find a subset of eigenvalues, such as the largest/smallest 100 or all eigenvalues within interval $[a, b]$. Furthermore, in 2004, the MRRR [11] method was proposed. It seeks to compute accurately orthogonal eigenvectors without expensive ($O(n^3)$ worst-case) reorthogonalization.

[1] https://docs.nvidia.com/cuda/cusolver/index.html

2.2 Other Eigen Decomposition Approaches

Polar decomposition [22] decomposes a matrix into a product of an orthonormal matrix and a positive definite symmetric matrix. It is connected to EVD and SVD, so recently, some new algorithms based on polar decomposition have been proposed. QDWH-eig (QR-based dynamically weighted Halley Eigenvalue decomposition) [30] uses QR factorization to compute the polar decomposition and then factorize the derived orthonormal matrix using the iterative subspace method. Later in 2016, a GPU implementation [36] of QDWH-eig and QDWH-SVD was proposed, but this work replaced the QDWH-eig with the 2-stage EVD. Another method to compute polar decomposition called scaled Newton [8] has lesser mathematical operations than QDWH. However, it highly relies on the backward stable inverse of a matrix.

Recently there has been growing interest in randomized Linear algebra [28], particularly randomized subspace iteration for computing a low-rank approximate eigenvalue/singular value decomposition. Two algorithms among them are randomized subspace iteration [16] and randomized block Lanczos [40]. These two algorithms are proven efficient in real-world applications, especially on modern high-performance architectures [35, 43]. However, these algorithms are typically based on multiplying a randomly generated matrix, which means they can only be applied to applications that are not sensitive to accuracy.

2.3 Tensor-Core-based Linear Algebra Algorithms

Nvidia introduced Tensor Core accelerator technology in 2017 in its Volta architecture [27]. Later in 2020, the A100 GPU [9] was released with more powerful Tensor Core Units whose half-precision GEMMs execution rate can boost to 300 TFLOPS. The Tensor Core unit is designed to accelerate GEMMs in deep neural networks. Some other applications seek the possibilities of utilizing Tensor Core to accelerate linear algebra algorithms, including the BLAS3 operations [42], the LU factorization [20, 23] and the QR factorization [41, 44].

3 Background

This section will introduce some background knowledge and related technologies and algorithms to band reduction.

3.1 Tridiagonalization

The tridiagonalization process is usually the pre-step to eigenvalue decomposition. The goal of tridiagonalization can be expressed as follows:

$$A = Q^{-1} \times T \times Q$$

Q is an orthogonal matrix, and T is a tridiagonal matrix. The conventional method of using Householder reflections eliminates the elements except for the tridiagonal part. However, only 50% of the computations can be fully blocked (BLAS3). In other words, 50% computations are BLAS2 operations that benefit less from modern hardware. Indeed, indicated by the MAGMA `ssyevd` routine experimental results, the unblocked computations take over 90% of the execution time of the tridiagonalization (`ssytrd` routine). An alternative way to increase the blocking percentage is called 2-stage tridiagonalization [19, 24, 25]. This method adds the successive band reduction before the tridiagonalization. Mathematically, the successive band reduction can be expressed by this equation: $A = Q^{-1} \times B \times Q$ where B is a band matrix with bandwidth b. Moreover, this is the first stage. The second stage reduces the band form to a tridiagonal form, called bulge chasing [6], and the performance of the 2nd stage depends on the bandwidth b. Figure 1 shows the general steps of the 2-stage tridiagonalization.

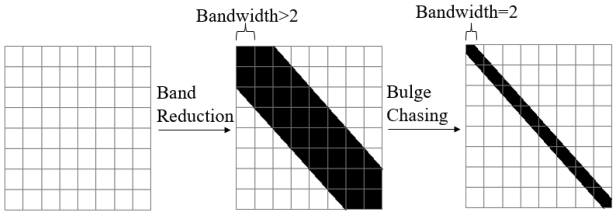

Figure 1. The 2-stage tridiagonalization

3.2 Householder Transformation

The Householder reflector is an orthogonal projection constructed from reflection against a hyperplane. It is useful to transform a given vector orthogonally to an axis (thereby eliminating all components but one). Specifically, given a vector x, the orthogonal matrix $H(v) = I - 2vv^T/(v^Tv)$ where $v = ||x||e_1 - x$ will map x to the first axis: $H(v)x = [||x||, 0, ..., 0]^T$. The above transformation is a rank-1 update, in other words, a BLAS2 operation. Fortunately, we can accumulate several Householder transformations by the WY representation [5] into a block to enrich BLAS3 operations. Suppose we have k Householder matrices $[H_1, H_2, ..., H_k]$, the WY representation will be:

$$H_k H_{k-1} ... H_2 H_1 = I - W_k Y_k^T$$

and if the $(k+1)$-th block is factorized, then we have:

$$H_{k+1} = I - w_{k+1} y_{k+1}^T$$
$$Y_{k+1} = [Y_k | y_{k+1}]$$
$$W_{k+1} = [W_k | w_{k+1} - W_k Y_K^T w_{k+1}]$$

3.3 Full to Band Algorithm

The symmetric matrix Householder update, also called 2-side update, is quite simple. See Figure 2. The first block includes A_1 and the grey region named Panel. Because we want the matrix to be reduced to band form, only the Panel needs to be factorized. After the first step, we have $QR(Panel) = (I - WY^T)R$, and the R matrix will override the upper triangular part of the Panel, and meanwhile, the lower triangular part

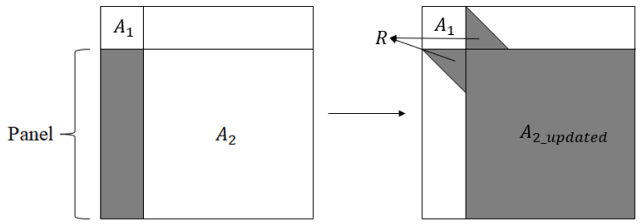

Figure 2. The single step of the 2-side Householder update upon a full matrix with block size k

of the Panel will be set to be 0s. And the further trailing matrix update is 2-side:

$$A_2 = (I - WY^T)^{-1}A_2(I - WY^T) \tag{1}$$

In practice, we usually use a mathematically equivalent ZY representation [12] to perform a rank-2k update upon the trailing matrix A_2:

$$Z = AW - \frac{1}{2}YW^TAW \tag{2}$$

$$A_2 = A_2 - YZ^T - ZY^T \tag{3}$$

The rest of the updating process is similar, as we can regard the updated A_2 as a new full matrix and factorize the matrix iteratively.

4 Performance Analysis Between ZY-based and WY-based Algorithms

This section will mainly discuss the differences between the ZY-based and WY-based algorithms and explain why our new WY-based algorithm can perform better in the context of Tensor Core. We will also provide some experimental results to prove that the WY-based algorithm has better GEMM performance on Tensor Core.

4.1 General Performance of Tall and Skinny GEMM on Tensor Core

Table 1 shows the two types of GEMMs in SBR. The first one is a square matrix times a tall and skinny matrix, which appears in Equation 1 and 2 ($A \times W$); and the second one is an outer product that appears in Equation 1 and 3($W \times Y^T$, $Y \times Z^T$ and $Z \times Y^T$). Note that Tensor Core does not yet support the *syr2k* routine, so Equation 3 has to involve two outer products.

According to Table 1, the performance of the GEMMs is highly relevant to the k on Tensor Core, while the performance of SGEMM is much more stable as k increases. This is probably because the tall and skinny GEMM becomes memory bound rather than compute bound when the k is small, and the time cost of launching kernel in TCGEMMs is not trivial. Obviously, to improve the overall performance of SBR

			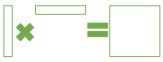	
k	TC-GEMM	SGEMM	TC-GEMM	SGEMM
32	6.28	9.36	20.02	9.31
64	11.69	9.65	33.30	9.85
128	24.44	10.22	49.83	10.02
256	42.65	10.33	97.41	10.23
512	66.57	10.36	122.89	10.33
1024	85.73	10.40	138.82	10.37
2048	112.08	12.91	121.55	13.13
4096	133.17	15.31	140.85	14.33

Table 1. TCGEMM and SGEMM performance on A100 GPU in TFLOPS as k changes from 32 to 4096 with fixed $m = 32768$. In columns 2-3, $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{m\times k}$. In columns 4-5, $A \in \mathbb{R}^{m\times k}$, $B \in \mathbb{R}^{k\times m}$.

on Tensor Core, we want the k as large as possible. However, unfortunately, in terms of the conventional WY and ZY representation algorithm, the k is fixed to be the bandwidth typically less than 256. As the computational complexity of bulge chasing is $O(nk^2)$, there is a cost to making the block size too large.

4.2 An Alternative Method to Change the GEMM's Shape

Inspired by the Tensor-Core-based QR factorization [41], the simple modification from the iterative method to the recursive method can change the GEMM shapes and leads to significant speedup. Nevertheless, the recursive strategy does not seem to work for SBR. Unlike QR factorization, the SBR is a two-side factorization; the trailing matrix cannot be simply divided into left half and right half.

4.2.1 An attempt of changing the GEMM shapes with ZY representation

Assume that the matrix Z_k and Y_k are the ZY representation in the k-th iteration of SBR. Then the entire Z and the Y matrix after the k-th iteration will be $Z = [Z_1|Z_2|...|Z_k]$ and $Y = [Y_1|Y_2|...|Y_k]$. To 'squeeze' the GEMMs, we can try to combine several Z and Y vectors together and update the trialing matrix one time. One solution is to figure out a recursive strategy similar to the Tensor-Core-based QR factorization [41]. However, unlike QR factorization, the trailing matrix update is two-side in SBR. When we are trying to update the next panel, we will need the previous trailing matrix to be fully updated. The key idea of converting tall and skinny GEMMs to square GEMMs in the recursive QR factorization algorithm [41] is updating the trailing matrices when we need to factorize them instead of updating the whole trailing matrix in each iteration. This method works well in the recursive QR factorization because it is a one-side

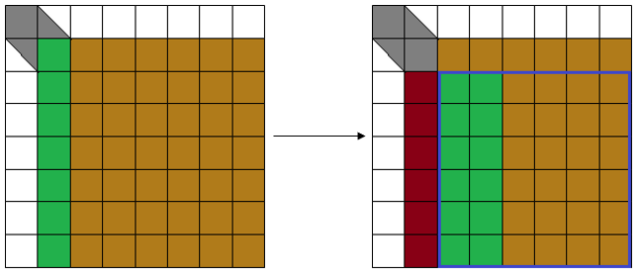

Figure 3. One hypothesis of SBR using ZY representation, the green region is the panel to be updated and the matrix with the blue outline is the trailing matrix

factorization. The right part of the matrix will not be touched until it is factorized.

Unfortunately, things become different in the two-side factorization. Figure 3 shows our hypothesis of using ZY representation in SBR. Suppose we have eight blocks with blocksize b and have already factorized the first block using ZY representation. Now we only want to update the green region (we can call it GA) in the matrix by $GA = GA - ZY(1:b,:)^T - YZ(1:b,:)^T$, while the brown region (BrA) is never touched. The next step (2nd picture in Figure 3) is factorizing the red region in the matrix and updating the green region (GA). The next step is updating the GA in the second picture, and if we have the $Y = [y_1|y_2]; Z = [z_1|z_2]$ then the GA can be updated easily by $GA = -ZY(1:2*b,:)^T - YZ(1:2*b,:)^T$. However, the problem is how to obtain Z_2. As we mentioned before, the matrix Z is related to the trailing matrix, actually if we already have the w_2 and y_2, and suppose the region with blue contour to be BA we can compute the z_2 by $z_2 = BA \times w_2 - \frac{1}{2} y_2 w_2^T \times BA \times w_2$. However, to deduce the correct GA, the submatrix BA needs to be updated by $BA = BA - z_1 y_1(b+1:n,:)^T - y_1 z_1(b+1:n,:)^T$ at first, which also includes the tall and skinny matrix multiplications. And we did not find a way to bypass the update of the entire trailing matrix with ZY representation.

4.2.2 Change the GEMM shapes with WY representation

An alternative strategy is using WY representation. The difference between the WY representation and the ZY representation in terms of trailing matrix update is that the WY representation updates the matrix by multiplications. In contrast, the ZY representation uses successive subtractions. Besides, updating the W matrix in WY representation does not require updating the entire trailing matrix. Instead, it only needs the updated panel. These properties of WY representation allow us to accumulate the Householder matrices and update the trailing matrix at once at the cost of more computations and memory usage because it has to update the W matrix in each iteration.

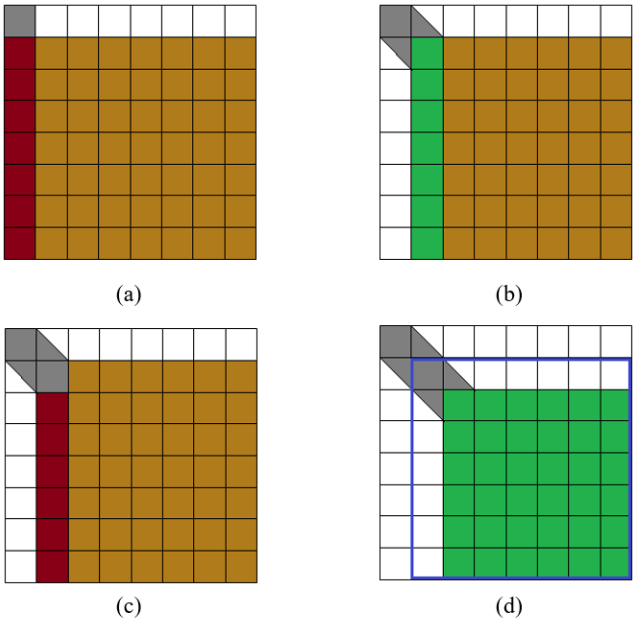

Figure 4. The WY-based SBR, the red region is the panel to be factorized, and the green region is the trailing matrix. The matrix inside the blue outline denotes the original matrix

Figure 4 depicts the several steps of the WY-based SBR before the first large iteration. Definitely the first step is QR factorize the red panel in (a) and get w_1 and y_1. In order to only update the green region (GA) in (b) we will let $y = y_1(1:b,:)$ and use y instead of the entire y_1 in the right multiplication. Then the $GA = (I - y_1 w_1^T) \times GA \times (I - w_1 y^T)$. The next step will be a panel factorization of the red region in (c) and form a new $Y = [y_1|y_2]$ and $W = [w_1|w_2 - W_1 y_1^T w_2]$. Before the recursion, we need to update the entire trailing matrix. Because we have never touched the green region in (d) before, we still have to use the submatrix with the blue outline (BA) of the original matrix. Then we replace the Y by $Y(2*b+1:n,:)$ and finally update the $GA = (I - YW^T) \times BA \times (I - WY^T)$.

Algorithm 1 gives the Matlab prototype of the WY-based SBR. A parameter nb should be given as a larger blocksize that inside the big block, only the panels will be updated; outside the big block, the whole trailing matrix will be updated.

4.3 GEMMs Performance Evaluation

4.3.1 Arithmetic Operations:

One of the differences between conventional ZY representation and Algorithm 1 is the number of arithmetic operations. The increment of operations in Algorithm 1 includes constructing the W matrix in the inner loop and the larger GEMM at lines 9, 13 in Algorithm 1 as we always have to use

Algorithm 1 Recursive WY-based Symmetric Band Reduction with blocksize **nb**

```
1  function [A] = sy2sb(A, oriA)
2    n = length(A);
3    OA=oriA(b+1:n, b+1:n);
4    for i=1:b:nb
5      [w,y]=PanelQR(A(i+b+1:n, i:i+b));
6      Y=[Y y];
7      W=[W w-W*Y'*w];
8      GA=A(i+b+1:n, i+b+1:i+2*b);
9      GA=(I-W*Y')'*OA*(I-W*Y(i:i+nb,:)');
10     A(i+b+1:n, i+b+1:i+2*b)=GA;
11   end
12   GA=A(nb+1:n, nb+1:n);
13   GA=(I-W*Y(nb+1:n,:)')'*OA*(I-W*Y(nb+1:n,:)');
14   A(nb+1:n,nb+1:n)=sy2sb(GA,GA);
15 end
```

	ZY	WY					
blocksize	128	128	256	512	1024	2048	4096
FLOPS	0.70	0.93	1.05	1.12	1.17	1.22	1.31

Table 2. The real number of arithmetic operations of ZY-based SBR (bandwidth 128) and WY-based SBR (with different block sizes from 128 to 4096); the exponent of the numbers is 10^{14}

the original matrix in the inner loop. And we will discuss them in sequence.

To show the increment of arithmetic operations intuitively, we compute the number of operations in ZY-based and WY-based algorithms, respectively, with matrix size 32768×32768 in Table 2. The number of operations increases considerably as the block size rises, indicating why the ZY-based algorithm is more popular on the previous hardware.

4.3.2 Experimental Results of GEMMs Performance

The WY-based SBR does convert some of the tall and skinny matrix multiplications to relatively square matrix multiplications. However, the problem is that it increases the total mathematical operations, which include forming the blocked W matrix and larger matrix sizes because the large submatrix (line 3 in Algo 1) of the original matrix is always reused inside the large block. Therefore, whether the WY-based algorithm can bring speedup remains unknown.

Let us start with the performance evaluation when the blocksize nb changes. The smaller nb leads to fewer computations but more tall and skinny GEMMs, while the larger nb brings more computations and square GEMMs. Our intuition is that there exists one nb that can give us the best performance, and Figure 5 supports our speculation. When nb is less than 1024, although the floating-point operations increase, the GEMMs rate increment from square GEMMs

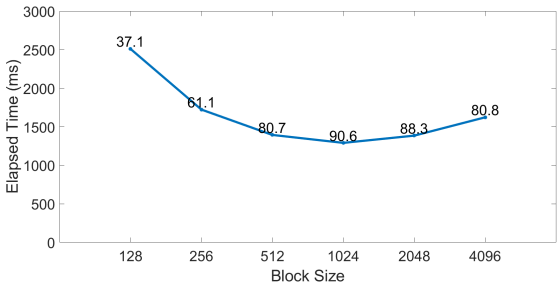

Figure 5. The total elapsed time of TCGEMMs in Algo 1 as the block size changes from 128 to 4096. The numbers upon the points are the TFLOPs of the TCGEMMs.

dominates the overall performance improvement. Nevertheless, when nb is larger than 1024, the rise of computations prevents the performance from improving.

Fix nb to be 1024, and Figure 6 compares the total execution time of the GEMMs between the two different strategies. When the matrix is as small as 4096, the WY-based algorithm does not have an advantage over the ZY-based algorithm. This is obvious because the GEMM execution rate in the two algorithms is similar when the matrix size is small. Therefore the change of GEMM shapes does not help. In contrast, the increment of the mathematical operations cannot be overlooked. Furthermore, that is why when the size is 4096 and 8192, the GEMM execution time in the WY-based algorithm is longer than in the ZY-based algorithm. However, when the size is quite large, the better shapes of the GEMMs improve the performance significantly. For instance, when the matrix size is 32768, the inside GEMMs in the WY-based algorithm can reach 240 TFLOPs, while the highest GEMM rate in the ZY-based algorithm is only around 50 TLFOPs. This means that even though we are doing more computations in the WY-based algorithm, the overall performance of GEMMs can still benefit a lot from the relatively square GEMMs (around 1.5x speedup in terms of GEMMs). We also perform another experiment on testing the SGEMMs and show the results in Figure 7. According to Table 1, the SGEMM cannot benefit from the square shapes. As a result, we can find in Figure 7 that if TCGEMMs are replaced with SGEMMs, the ZY-based algorithm can have better performance, which means the WY-based algorithm only brings speedup with Tensor Core support.

4.4 Forming Eigenvectors

The GEMMs are even more efficient when the eigenvectors are needed with the WY-based algorithm. Because forming the W matrix (back transformation) in the inner product is not wasted at all. As the W matrix is fully obtained in each inner loop, so the final matrix W will be of this form: $W = [W_1|W_2|...|W_k]$ where $k = n/nb$. Then forming the entire W will be quite easy, and it can be formed in a recursive way to

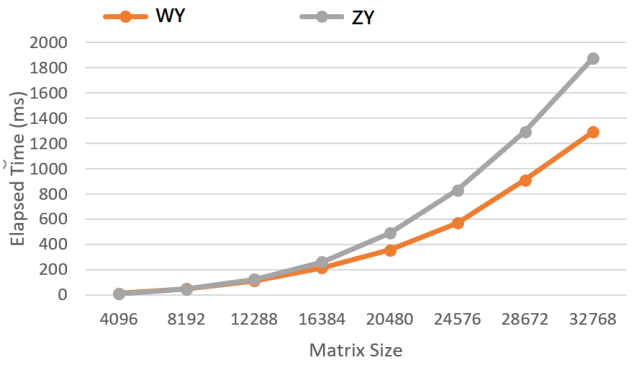

Figure 6. The total elapsed time comparison of the TCGEMMs in WY-based algorithm and the ZY-based algorithm, the block size in the WY-based algorithm is fixed to be 1024

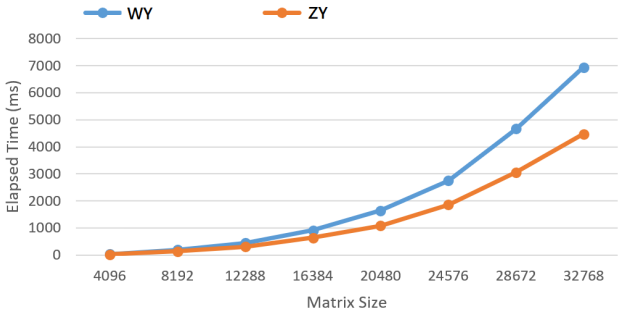

Figure 7. The total elapsed time comparison of the SGEMMs in the WY-based algorithm and the ZY-based algorithm, the block size in the WY-based algorithm is also fixed to be 1024

Algorithm 2 Recursive W construction with blocksize **nb**

```
function [W] = FormW(W,Y)
  n=length(W);
  if n<=2*nb
    W1=W(:,1:nb);
    W2=W(:,nb+1:2*nb);
    Y1=Y(:,1:nb);
    W=[W1|W2-W1*Y1'*W2];
  end
  %left recurse
  W1=FormW(W(:,1:n/2), Y(:,1:n/2));
  %right recurse
  W2=FormW(W(:,n/2+1:n), Y(:,n/2+1:n));
  W=[W1|W2-W1*Y1'*W2];
end
```

'squeeze' the GEMMs. See Algo 2 for details. And based on our experiments, given a matrix whose size is 32768×32768,

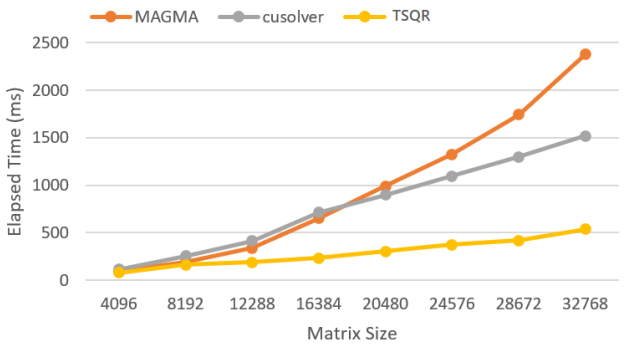

Figure 8. The total elapsed time of panel QR factorization comparison between MAGMA, cuSOLVER and our TSQR implementation,

the back transformation takes 320ms vs. 420ms in the WY-based and ZY-based algorithms, respectively, which gives around another 10% acceleration.

5 Implementation and Optimization

In this section, we will address our GPU implementation and the optimization strategy to improve performance and accuracy.

5.1 Tall and Skinny Panel QR Factorization

Figure 2 shows that the panel is tall and skinny when the bandwidth is small. Because of low data locality and parallelism, factorizing such a matrix shape is much slower than factorizing a square matrix, especially on a GPU. Communication-Avoiding QR (CAQR) [3] factorization algorithm efficiently performs QR factorization. Tall-Skinny QR (TSQR) is a particular case of CAQR that only deals with tall and skinny matrices. And the general idea and implementation can be found in the Tensor-Core-based QR paper [41]. But there are still some modifications: 1)we apply Householder transformation rather than modified Gram-Schmidt process on each small block to maintain the stability; 2)each warp is in charge of one column instead of one row to get better performance.

We evaluate the panel factorization in band reduction as matrix size changes from 4096 to 32768, by comparing the panel factorization execution time in MAGMA (`ssytrd_sy2sb`[2] routine), cuSOLVER (`sgeqrf` and `sorgqr` routine) and our TSQR; see Figure 8. Note that the elapsed time in Figure 8 is not the pure QR factorization; it also includes reconstructing the W and Y matrices, which will be discussed in the next section, in cuSOLVER and the TSQR implementation.

[2]Symmetric matrix to symmetric band form

5.2 Reconstruct Householder Vectors

Although TSQR can accelerate the panel compared to cuSOLVER `SGEQRF()`, there is still a problem that, unlike cuSOLVER `SGEQRF()`, what we obtain from TSQR is the explicit Q. In contrast, cuSOLVER `SGEQRF()` provides Householder vectors. Moreover, with the explicit Q instead of Householder vectors, the further trailing matrix updates will lead to unstable results. Therefore, it is necessary to develop an algorithm that can output Householder vectors from the explicit Q formed by TSQR. One solution is to reconstruct Householder vectors by the explicit Q [4]. The idea is as follows. Given an orthogonal matrix, Q can be represented as $Q = I - Y \times T \times Y^T$ (memory-efficient WY representation), and this equation can also be modified to $I - Q = Y \times T \times Y^T$. Y is a lower triangular matrix, and T is an upper triangular matrix. Hence, it can be regarded as a LU factorization with $I - Q = L \times U = (Y) \times (T \times Y^T)$. The paper[4] also reports that the LU factorization provides unique L and U, and partial pivoting is unnecessary. In our algorithm, we factorize the upper $n \times n$ part of the matrix $I - Q$, and then we do a triangular solve (STRSM) to get the whole L matrix. After obtaining the Householder vectors L from LU factorization, we do another triangular solve to construct W; see Algorithm 3.

Algorithm 3 Reconstruct WY representation by explicit Q generated from TSQR

```
function [W,Y] = ReconstructWY(Q)
  [m,n] = size(Q);
  I = eye(m,n);
  A = I-Q;
  [L1,U] = non_pivoting_LU(A(1:n,:));
  L2 = A(n+1:m,:)/U;
  Y=[L1;L2];
  W=A/Y';
end
```

Ballard et al. perform a LU factorization on $Q - S$ [4], where S is a diagonal sign matrix corresponding to the sign choices made inside the Householder QR algorithm. Note that this step is also essential to avoid rank deficiency in the LU factorization. And combining this algorithm with our TSQR implementation, we can have the panel factorization that generates W and Y.

5.3 Error Correction Tensor-Core-based GEMMs

The Refined TCGEMM [31] is a Tensor Core-based GEMM that provides the same accuracy as CUBLAS SGEMM using FP32 but exceeds the theoretical peak performance of FP32, yielding a much higher throughput. They are an enhanced version of the error correction methods proposed by Markidis [26] to perform matrix-matrix multiplication on Tensor Core. Implementing error-correcting methods would not provide the accurate results of matrix-matrix multiplication on FP32 SIMT cores. The key to achieving this high accuracy is dealing with rounding off inside Tensor Cores. Refined TCGEMM saves the mantissa loss and uses it to correct the accuracy of matrix-matrix multiplication. This high-accuracy, high-performance, and low-power consuming implementation is developed in NVIDIA's CUTLASS library and achieves 51TFLOPS/s for limited exponent range and 33TFLOPS/s for full exponent range of FP32 using TF32 Tensor Cores on NVIDIA A100 GPU. This implementation outperforms the theoretical FP32 SIMT Core peak performance of 19.5TFLOPS/s.

To be more specific, consider a GEMM $C = A \times B$. To compute this GEMM on Tensor Core, the truncation of A and B from FP32 to FP16 should be performed in advance. Then we have $A = \tilde{A} + \Delta A$ and $B = \tilde{B} + \Delta B$, where the $\tilde{A}$ and $\tilde{B}$ is stored in FP16 precision. The TCGEMM with FP32 accumulation is actually doing $C = \tilde{A} \times \tilde{B}$ and the accurate result should be $C = (\tilde{A}+\Delta A)\times(\tilde{B}+\Delta B) = \tilde{A}\times\tilde{B}+\tilde{A}\times\Delta B+\Delta A\times\tilde{B}+\Delta A\times\Delta B$. In practice, the $\Delta A \times \Delta B$ is so trivial that it can be typically overlooked. Therefore, performing two extra TCGEMMs seems to recover the accuracy loss. But the rounding errors inside Tensor Core and the underflow problems still prevent the naive recovering solution from getting accurate results. As a result, the authors also apply two other methods to solve the problem: 1)try to avoid the rounding for accumulation inside Tensor Cores to recover the FP32 accuracy; and 2) try to scale the matrix to reduce underflow. And finally, the refined TCGEMMs can outperform the SGEMMs and meanwhile bring the accuracy back to single precision. And we will also show the performance impact of using the error correlation TCGEMMs in our SBR in Figure 10.

6 Experimental Evaluation

We use a 5.4.0-99-generic Linux operating system with NVIDIA A100 GPU (A100-PCIE-40GB) for all the experiments. The CUDA version is 11.2, which contains a C++ compiler and libraries cuBLAS and cuSOLVER. For the EVD case study experiments, we use the random matrix generation routine (`magma_generate`) from MAGMA 2.6.1 to generate a random matrix with a specific condition number and singular value distribution.

6.1 The Effect of TSQR Panel and Tensor Core Regarding Band Reduction

The panel factorization and the Tensor Core play essential roles in the entire band reduction. Figure 9 illustrates the impact of using the TSQR panel and Tensor Core on the entire SBR execution time. When the matrix size is small, the TSQR panel is more critical; in contrast, when the matrix is large, Tensor Core contributes more acceleration. For example, without Tensor Core, the performance of the WY-based

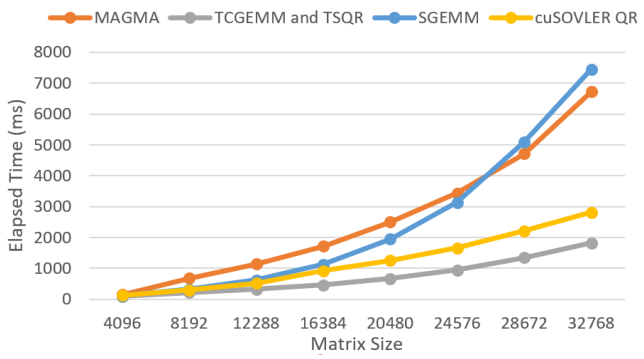

Figure 9. The total elapsed time comparison of the band reduction in WY-based algorithm with comparison to MAGMA band reduction baseline (orange line). Three settings of the implementation include Tensor Core on and TSQR on (grey line), Tensor Core off and TSQR on (blue line), and Tensor Core on and TSQR off (yellow line, using cuSOLVER QR instead of TSQR)

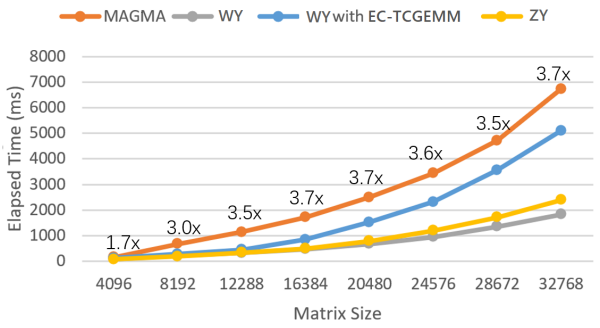

Figure 10. The total elapsed time comparison of the band reduction between the WY-based algorithm, WY-based algorithm with EC-TCGEMMs, ZY-based algorithm, and MAGMA baseline

algorithm is even worse than MAGMA when the matrix size is large.

6.2 Overall Performance of Band Reduction

The overall performance improvement compared to the MAGMA implementation is also shown in Figure 10, and the numbers over the orange line denote the speedup between the WY-based algorithm and the MAGMA `sy2sb` routine. Our WY-based implementation is much faster than the MAGMA relative routine. When the matrix size is huge (larger than 20,000), the WY-based algorithm also brings about a 1.3x speedup than the conventional ZY-based algorithm. If the TCGEMMs in the WY-based algorithm are replaced with EC-TCGEMMs (blue line), the overall band reduction performance is still slightly better than the MAGMA baseline (around 1.3x).

Matrix type	Backward error	Orthogonality
Normal	9.45e-04	5.27e-04
Uniform	4.73e-04	5.45e-04
SVD_Cluster0 1e5	9.34e-04	4.17e-04
SVD_Cluster1 1e5	9.45e-04	6.89e-04
SVD_Arith 1e1	9.45e-04	4.89e-04
SVD_Arith 1e3	9.45e-04	7.09e-04
SVD_Arith 1e5	9.45e-04	4.39e-04
SVD_Geo 1e1	9.45e-04	7.39e-04
SVD_Geo 1e3	9.46e-04	4.21e-04
SVD_Geo 1e5	9.45e-04	3.68e-04

Table 3. The backward error (E_b, 2nd column) and the orthogonality (E_o, 3rd column) of our Tensor Core-based SBR with different matrix types

6.3 Accuracy

As we will compute the eigenvalues in the following experiments (section 6.4.2), the accuracy of the eigenvalues of the band matrix is not that necessary. Thus, in this section, we will only consider the backward stability [21] and the orthogonality of our implementation.

The backward error (orthogonal transformation error) of the band reduction is defined as:

$$E_b = \frac{||A - Q \times B \times Q^{-1}||_F}{N \times ||A||_F}$$

where A is the matrix that needs to be factorized, B is the band matrix, and Q is the orthogonal matrix that can be used to form the eigenvectors. And the orthogonality:

$$E_o = \frac{||I - Q^{-1} \times Q||_F}{N}$$

denotes if the transformation matrices Q are orthogonal.

Theoretically, the backward error and the orthogonality should be bounded by the machine ϵ because the TSQR, as well as the band reduction algorithms, are proven to be backward stable. In the context of Tensor Core computations, the machine ϵ is $1e-4$. Table 3 gives the errors of the band reduction with different matrix types and condition numbers, which MGAMA generates. The SVD_'NAME' means the singular values of the generated matrix obey the 'NAME' distribution, and the number next to the matrix type is the condition number. For example, SVD_Arith 1e5 denotes the singular values of the matrix are arithmetically distributed with $S_{max}/S_{min} = 1e5$. And it is obvious in Table 3 that both backward errors and orthogonality of the band reduction are bounded by the machine ϵ of Tensor Core. In addition, the bulge chasing process and 2nd stage in MAGMA are stable, which means once our SBR is stable, the entire EVD will also be stable.

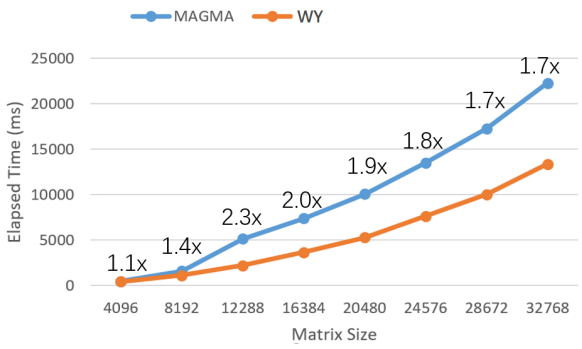

Figure 11. The total elapsed time comparison of the 2-stage EVD between our WY-based SBR and MAGMA baseline

6.4 A Case Study of eigenvalue Decomposition

As we have a faster band reduction implementation, it is worthwhile to use this module to solve the eigen problems. Still, we will consider the performance and the accuracy.

6.4.1 Performance

The implementation of the eigensolver is quite simple. We start everything and compute the band reduction on GPU, transfer the band matrix to CPU, and then use MAGMA to perform the bulge chasing and divide and conquer algorithm. Note that the execution time of our implementation includes the data movement from device to host (the rate is around 12GB/s). See Figure 11. To get the best performance of the eigenvalue in MAGMA, we set the number of MKL threads to be 1, 1, 4, 8, 8, 12, 16, 16 for matrix from 4096 to 32768, respectively. The results indicate that with our Tensor Core-based band reduction, the eigenvalue decomposition (no eigenvectors) can be accelerated around 2x.

6.4.2 Accuracy

We did not perform the EVD with eigenvectors generated, so the accuracy measurement will only be the eigenvalues. And the error can be expressed as:

$$E_s = \frac{||D_s - D_{tc}||_2}{N \times ||D_s||_2}$$

where D_s is the real eigenvalues computed by LAPACK and the D_{tc} is the eigenvalues from the band matrix computed by Tensor Core. Similar to the accuracy experiments of band reduction, we also test the accuracy with different matrix types. See Table 4. Still, the numbers next to the matrix type denote the condition number. As we can observe in the table, the accuracy level of our implementation is pretty good. Indeed, for the SVD geometric distribution with condition numbers 1e3 and 1e5, the checking function in MAGMA even gives a pass symbol. The results indicate that our Tensor Core implementation is able to compute the eigenvalues with the machine ϵ tolerance. If we do not need accurate results or want to use the EVD as a pre-step in low-rank approximation [40], the Tensor Core-based implementation will meet our requirement. Otherwise, we can replace the TCGEMMs with EC-TCGEMMs to get more accurate results.

Matrix type	Tensor Core	MAGMA
Normal	7.21E-05	4.59E-06
Uniform	1.38E-04	5.19E-07
SVD_Cluster0 1e5	3.59E-05	1.64E-07
SVD_Cluster1 1e5	8.80E-05	1.37E-06
SVD_Arith 1e1	7.58E-05	4.51E-06
SVD_Arith 1e3	8.46E-05	1.39E-05
SVD_Arith 1e5	6.81E-05	1.67E-05
SVD_Geo 1e1	5.77E-05	2.05E-06
SVD_Geo 1e3	5.11E-05	4.43E-06
SVD_Geo 1e5	5.20E-05	3.68E-06

Table 4. The accuracy comparison of the eigenvalues (E_s) between our Tensor Core implementation (2nd column) and MAGMA ssyevdx() routine (3rd column) with different matrix types

7 Conclusion and Future Work

In this paper, we propose a Tensor Core GPU implementation of the symmetric band reduction which can bring up to 3.7x speedup compared to the state-of-the-art SBR software. We have evaluated the TSQR factorization and replaced the general Householder QR factorization, which is much faster than cuSOLVER and MAGMA. To ensure the QR results can be utilized in further computations, we use non-pivoting LU factorization to reconstruct the WY representation from the explicit Q. We abandon the traditional ZY-based method in terms of the trailing matrix update. Instead, we use the WY representation to update the trailing matrix to convert the tall and skinny GEMMs to relatively square GEMMs. Although the WY-based algorithm brings more mathematical operations, the performance of the new sizes of GEMMs proves the attempt, which shows a 1.5x speedup regarding GEMMs, to be worthy. In addition, to recover the accuracy loss generated by Tensor Core, we use EC-TCGEMMs, which have the same accuracy as SGEMM but are faster. We also did a case study based on the MAGMA's implementation of the 2nd stage of the tridiagonalization and the divide and conquer method and obtained up to 2.3x speedup.

However, there are also some limitations of the Tensor-Core-based SBR. First, the target precision is single precision, which means this algorithm cannot work when a double-precision result is needed. Secondly, the WY-based method only performs better on Tensor Core because, without Tensor Core, the speedup which benefits from GEMMs' shapes cannot counteract the performance decrease from the increment of mathematical operations. Thirdly, the proposed algorithm requires more device memory to store the original matrix and the WY representation.

The research areas on eigenproblems are so broad that band reduction is only a tiny part of the eigenproblems. Moreover, even in the symmetric band reduction algorithm, we can still do furthermore works to optimize it. For example, we can try to implement the Tensor-Core-based symmetric rank 2k update (`syr2k`). Indeed, in our current program, this kind of GEMM is regarded as a normal GEMM that does 2x more computations because it does not utilize the symmetric property of the matrix. Additionally, in terms of the entire EVD, we are only implementing the 1st stage of tridiagonalization, the 2nd stage, and the divide and conquer algorithm called the MAGMA routines. So the data movement between GPU and CPU still costs some time. As a result, we try to use GPU in the 2nd stage. Iterative methods on GPU will also be considered in the future. Additionally, the error analysis of the Tensor-Core-based eigen problems also needs more attention. Based on our previous investigation on Tensor-Core-based computations, the error is typically bounded by the machine ϵ. For Tensor Core, it is 1e-4. However, according to our experiments on computing the eigenvalues, the accuracy is better than our expectation, nearly 1e-5. Unfortunately, we only provide the experimental accuracy results. This is because analyzing the errors will be out of scope and can be a separate paper, as the theoretical error analysis is too complicated. Hopefully, we can theoretically analyze the mathematical reasons behind the experimental results and explain them soon. Besides, we can also try to recover the precision with the method proposed in 2021 [38] in the future.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant 2146509. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We are grateful for the support of the Research Computing Data Core at the University of Houston for assistance with the calculations carried out in this work. We would also like to thank the YOKOTA Laboratory of Tokyo Insititute of Technology for the Hinadori Cluster, which is supported by JSPS KAKENHI Grant Number JP20K20624, and JP18H03248.

References

[1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[2] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. 1999. *LAPACK Users' guide*. SIAM.

[3] Michael Anderson, Grey Ballard, James Demmel, and Kurt Keutzer. 2011. Communication-avoiding QR decomposition for GPUs. In *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 48–58.

[4] Grey Ballard, James Demmel, Laura Grigori, Mathias Jacquelin, Hong Diep Nguyen, and Edgar Solomonik. 2014. Reconstructing Householder vectors from tall-skinny QR. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 1159–1170.

[5] Christian Bischof and Charles Van Loan. 1987. The WY representation for products of Householder matrices. *SIAM J. Sci. Statist. Comput.* 8, 1 (1987), s2–s13.

[6] Christian H Bischof, Bruno Lang, and Xiaobai Sun. 2000. A framework for symmetric band reduction. *ACM Transactions on Mathematical Software (TOMS)* 26, 4 (2000), 581–601.

[7] Christian H. Bischof, Xiaobai Sun, and Bruno Lang. 1994. Parallel tridiagonalization through two-step band reduction. *Proceedings of IEEE Scalable High Performance Computing Conference* (1994), 23–27.

[8] Ralph Byers and Hongguo Xu. 2008. A new scaling for Newton's iteration for the polar decomposition and its backward stability. *SIAM J. Matrix Anal. Appl.* 30, 2 (2008), 822–843.

[9] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[10] RZ Dautov, AD Lyashko, and SI Solov'ev. 1994. The bisection method for symmetrie eigenvalue problems with a parameter entering nonlinearly. (1994).

[11] Inderjit S Dhillon and Beresford N Parlett. 2004. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra Appl.* 387 (2004), 1–28.

[12] Jack J Dongarra, Danny C Sorensen, and Sven J Hammarling. 1989. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.* 27, 1-2 (1989), 215–227.

[13] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations* (third ed.). The Johns Hopkins University Press.

[14] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations*. JHU press.

[15] Roger Grimes, Henry Krakauer, John Lewis, Horst Simon, and Su-Hai Wei. 1987. The solution of large dense generalized eigenvalue problems on the Cray X-MP/24 with SSD. *J. Comput. Phys.* 69, 2 (1987), 471–481.

[16] Ming Gu. 2015. Subspace iteration randomization and singular value problems. *SIAM Journal on Scientific Computing* 37, 3 (2015), A1139–A1173.

[17] Vineet Gupta, Tomer Koren, and Yoram Singer. 2018. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*. PMLR, 1842–1850.

[18] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. 2011. Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems Using Aggregated Fine-Grained and Memory-Aware Kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. Association for Computing Machinery, New York, NY, USA, Article 8, 11 pages. https://doi.org/10.1145/2063384.2063394

[19] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. 2011. Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[20] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J Higham. 2018. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 603–613.

[21] Desmond J Higham and Nicholas J Higham. 1998. Structured backward error and condition of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.* 20, 2 (1998), 493–512.

[22] Nicholas J Higham. 1986. Computing the polar decomposition—with applications. *SIAM J. Sci. Statist. Comput.* 7, 4 (1986), 1160–1174.

[23] Florent Lopez and Théo Mary. 2020. Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint. (2020).

[24] Hatem Ltaief, Piotr Luszczek, Azzam Haidar, and Jack Dongarra. 2012. Solving the generalized symmetric eigenvalue problem using tile algorithms on multicore architectures. In *Applications, Tools and Techniques on the Road to Exascale Computing*. IOS Press, 397–404.

[25] Piotr Luszczek, Hatem Ltaief, and Jack Dongarra. 2011. Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures. In *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 944–955.

[26] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. 2018. NVIDIA Tensor Core Programmability, Performance & Precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. https://doi.org/10.1109/ipdpsw.2018.00091

[27] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. 2018. Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 522–531.

[28] PG Martinsson and JA Tropp. [n.d.]. Randomized numerical linear algebra: foundations & algorithms (2020). *arXiv preprint arXiv:2002.01387* ([n. d.]).

[29] Per-Gunnar Martinsson and Sergey Voronin. 2016. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. *SIAM Journal on Scientific Computing* 38, 5 (2016), S485–S507.

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

[31] Hiroyuki Ootomo and Rio Yokota. 2022. Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance. *arXiv preprint arXiv:2203.03341* (2022).

[32] Matt Probert. 2011. Electronic Structure: Basic Theory and Practical Methods, by Richard M. Martin: Scope: graduate level textbook. Level: theoretical materials scientists/condensed matter physicists/computational chemists.

[33] Notker Rösch, Sven Krüger, Vladimir A Nasluzov, and Alexei V Matveev. 2005. ParaGauss: The density functional program paragauss for complex systems in chemistry. In *High Performance Computing in Science and Engineering, Garching 2004*. Springer, 285–296.

[34] Robert Schreiber and Charles Van Loan. 1989. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Statist. Comput.* 10, 1 (1989), 53–57.

[35] Ruchi Shah, Shaoshuai Zhang, Ying Lin, and Panruo Wu. 2019. xSVM: Scalable distributed kernel support vector machine training. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 155–164.

[36] Dalal Sukkari, Hatem Ltaief, and David Keyes. 2016. A high performance QDWH-SVD solver using hardware accelerators. *ACM Transactions on Mathematical Software (TOMS)* 43, 1 (2016), 1–25.

[37] Stanimire Tomov, Rajib Nath, Peng Du, and Jack Dongarra. 2011. MAGMA Users' Guide. *ICL, UTK (November 2009)* (2011).

[38] YAOHUNG M Tsai, PIOTR Luszczek, and JACK Dongarra. 2021. *Mixed-precision algorithm for finding selected eigenvalues and eigenvectors of symmetric and Hermitian matrices.* Technical Report. Technical report ICL-UT-21-05, Innovative Computing Laboratory, The ....

[39] David S Watkins. 1982. Understanding the QR algorithm. *SIAM review* 24, 4 (1982), 427–440.

[40] Qiaochu Yuan, Ming Gu, and Bo Li. 2018. Superlinear convergence of randomized block lanczos algorithm. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1404–1409.

[41] Shaoshuai Zhang, Elaheh Baharlouei, and Panruo Wu. 2020. High accuracy matrix computations on neural engines: A study of qr factorization and its applications. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. 17–28.

[42] Shaoshuai Zhang, Vivek Karihaloo, and Panruo Wu. 2020. Basic Linear Algebra Operations on TensorCore GPU. In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. IEEE, 44–52.

[43] Shaoshuai Zhang, Ruchi Shah, and Panruo Wu. 2020. TensorSVM: accelerating kernel machines with tensor engine. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–11.

[44] Shaoshuai Zhang and Panruo Wu. 2021. Recursion Brings Speedup to Out-of-Core TensorCore-based Linear Algebra Algorithms: A Case Study of Classic Gram-Schmidt QR Factorization. In *50th International Conference on Parallel Processing*. 1–11.