2022 IEEE/ACM Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems (ScalAH) | 978-1-6654-7570-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ScalAH56622.2022.00011

Mixed-Precision Algorithm for Finding Selected Eigenvalues and Eigenvectors of Symmetric and Hermitian Matrices[1]

Yaohung M. Tsai*, Piotr Luszczek*, and Jack Dongarra*
*Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA, {ytsai2,luszczek,dongarra}@icl.utk.edu

***Abstract*—The multi-precision methods commonly follow approximate-iterate scheme by first obtaining the approximate solution from a low-precision factorization and solve. Then, they iteratively refine the solution to the desired accuracy that is often as high as what is possible with traditional approaches. While targeting symmetric and Hermitian eigenvalue problems of the form $Ax = \lambda x$, we revisit the SICE algorithm proposed by Dongarra et al. By applying the Sherman-Morrison formula on the diagonally-shifted tridiagonal systems, we propose an updated SICE-SM algorithm. By incorporating the latest two-stage algorithms from the PLASMA and MAGMA software libraries for numerical linear algebra, we achieved up to $3.6\times$ speedup using the mixed-precision eigensolver with the blocked SICE-SM algorithm for iterative refinement when compared with full double complex precision solvers for the cases with a portion of eigenvalues and eigenvectors requested.**[1]

The symmetric eigenvalue problem is one of the most important problems in numerical linear algebra for analysis of invariant subspace. For real matrices, the objective is to find an eigenvalue λ and the corresponding eigenvector x such that

$$Ax = \lambda x \text{ where } A = A^\intercal,\ A \in \mathbb{C}^{n\times n} \tag{1}$$

The Hermitian eigenvalue problem is to find the eigenvalues and eigenvectors in complex domain.

I. Releated Work

a) Eigenvalue refinement: Symm and Wilkinson[1] proposed an algorithm to determine the error bounds of computed eigenvalues and eigenvectors, which can also be used to improve the accuracy of a given eigen-pair. Dongarra, Moler, and Wilkinson[2], [3], [4] later improved the algorithm with reduced computational cost and provided additional error analysis, including the comparison to Newton's method[5], [6], numerical results, and discussion of extending the algorithm for ill-conditioned problems with multiple close eigenvalues.

Other related work from Stewart[7] and Chatelin[8] answered the same question from the point of view of the invariant subspace problem. Demmel[9] later pointed out that these two methods and the one from Dongarra, Moler, and Wilkinson [2], [3], [4] can all be reduced to solving the same Riccati equation. He also extended the algorithm for the generalized eigenvalue problem of the form $Ax = \lambda Bx$.

Alefeld and Spreuer[10] followed the same approach but specifically targeted problems with doubly-repeated or numerically close eigenvalues. Tisseur[11] did the analysis of Newton's method under floating-point arithmetic for generalized eigenvalue problems. Prikopa and Gansterer[12] used the symmetry of the matrix and Householder tridiagonalization $A = QTQ^\intercal$ to reduce the computational cost.

[1]Support: NSF OAC 2004541 and the Exascale Comp. Proj.(17-SC-20-SC).

Ogita and Aishima[13] proposed a different iterative scheme, which heavily relies on matrix-matrix multiplication for those applications which require accuracy that is higher than the base IEEE-754 double precision. The algorithm is applied on the entire spectrum of eigenvalues but it is capable of improving at the same time the orthogonality and eigenvalue accuracy. However, it requires high-precision computation for the most parts of the algorithm, making it costly in practice. Later the authors extended the algorithm for clustered eigenvalues and singular value decomposition[14], [15].

b) Parallel Eigensolvers: The most recent work introduced a hybrid 2-stage algorithm[16], [17]. The first stage still consisted of blocked Householder transformations but it only reduced the matrix to a band form. Then, the left transformation will only be needed, as the right transformation will not be touching the first block of columns. It thus becomes an LQ factorization for the block of columns, which is much faster than applying the transformations from both sides (LQ and QR). The second stage uses the bulge-chasing algorithm from the successive band reductions.

c) The SICE Algorithm: In Algorithm 1, given the base eigenpair λ, x and its nearby eigenpair $\lambda+\mu$, $x+\tilde{y}$, then based on the original eigenproblem we have: [2], [3], [4]

$$A(x + \tilde{y}) = (\lambda + \mu)(x + \tilde{y}) \tag{2}$$

Assuming that x is normalized $|x|_\infty = 1 \equiv x_s$, we can remove one degree of freedom by requiring $\tilde{y}_s = 0$ we get:

$$(A - \lambda I)\tilde{y} - \mu x = \lambda x - Ax + \mu\tilde{y} \tag{3}$$

The last term is the second order term for the error in λ and x. By simplify the equation, we introduce vector y, defined as:

$$y^\intercal \stackrel{\triangle}{=} (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_{s-1}, \mu, \tilde{y}_{s+1}, \ldots, \tilde{y}_{n-1}, \tilde{y}_n) \tag{4}$$

So y encodes information from both $\tilde{y}$ and μ and thus Eq. (3) becomes:

$$By = r + y_s\tilde{y} = r + \mu\tilde{y} \tag{5}$$

where $r = \lambda x - Ax$ is the residual vector of λ and x and B is the matrix $A - \lambda I$ with column s replaced by $-x$.

We can also view it as the Newton's method. In particular, by setting $v = \binom{x}{\lambda}$ we formulate the eigenvalue problem as:

$$f(v) \equiv \begin{pmatrix} Ax - \lambda x \\ e_s^\intercal x - 1 \end{pmatrix} = 0 \tag{6}$$

where e_s is the s-th column of the identity matrix of size n. The Newton's method then solves the linear system of the Jacobian matrix:

978-1-6654-7570-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ScalAH56622.2022.00011

$$J\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix}=\begin{pmatrix}A-\lambda I & -x\\ e_s^\intercal & 0\end{pmatrix}\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix}=\begin{pmatrix}r\\ 0\end{pmatrix}=f(v) \quad (7)$$

Expanding it, we get Eq. (3) without the second-order term:

$$(A-\lambda I)\tilde{y}-\mu x=r \quad (8)$$

This is the basic idea of the SICE algorithm: by iteratively solving Eq. (5) we obtain both the correction to the eigenvalue and to the eigenvector. The original algorithm uses Schur decomposition and applies two steps of Givens rotation in order to solve Eq. (5). For any real matrix A, there exists an orthogonal matrix Q and an upper quasi-triangular matrix T, such that $A = QUQ^\intercal$ where U is upper quasi-triangular with some 2×2 diagonal blocks arising from complex conjugate eignevalue pairs. Here, we define $Z_\lambda \equiv Z-\lambda I$ and $z_{\lambda s}\equiv Z_\lambda e_s = (Z-\lambda I)e_s$. Using $c=-x-a_{\lambda s}$ in Eq. (5):

$$[A_\lambda-(x+a_{\lambda s})e_s^\intercal]y=(A_\lambda+ce_s^\intercal)y=r+y_s\tilde{y} \quad (9)$$

Then using the Schur decomposition $A=QUQ^\intercal$, we have:

$$Q(U_\lambda+Q^\intercal ce_s^\intercal Q)Q^\intercal y=r+y_s\tilde{y} \quad (10)$$

$$(U_\lambda+d\times f^\intercal)Q^\intercal y=Q^\intercal g \quad (11)$$

where $d=Q^\intercal c$, $f^\intercal=e_s^\intercal Q$ and $g=r+y_s\tilde{y}$. Matrix $d\times f^\intercal$ constitutes a rank-1 update. Then two steps of Givens rotation are introduced: the first one Q_1 is constructed so that

$$Q_1 d=(P_2P_3\dots P_n)d=\gamma e_1 \text{ where } \gamma=\|d\|_2 \quad (12)$$

and P_i is the rotation in $(i-1,i)$ plane that eliminates the i-th component in $P_{i+1}\dots P_n d$. We also have:

$$Q_1(U_\lambda+d\times f^\intercal)=Q_1U_\lambda+\gamma e_1 f^\intercal \quad (13)$$

The transformation Q_1 introduces one more nonzero element in the subdiagonal direction of U_λ. The new rank-one update $\gamma e_1\times f^\intercal$ has nonzero elements only in the first row, which preserves the original structure. The second step of Givens rotation Q_2 can be applied subsequently in order to obtain the upper triangular form $\bar{U}_\lambda = Q_2Q_1(U_\lambda+d\times f^\intercal)$ in $\bar{U_\lambda}Q^\intercal y=Q_2Q_1Q^\intercal g$.

II. Algorithm and Implementation

The original SICE algorithm is designed for a general real matrices and here we first focus on symmetric ones. The proposed algorithm utilizes the tridiagonalization as well as the Sherman–Morrison formula to solve the linear system for eigenvalue and eigenvector corrections.

a) SICE-SM Algorithm: For symmetric eigenvalue problems, the matrix A is first reduced to tridiagonal through unitary similarity transformations: $T=Q^\intercal AQ$ where $QQ^\intercal=I$ and T is a symmetric tridiagonal matrix. This corresponds to LAPACK routines `SSYTRD` and `DSYTRD` for single- and double-precision arithmetic, respectively. In the same fashion as SICE algorithm in Section I-c, we start with Eq. (9) and

Algorithm 1 SICE algorithm

Input: Matrix $A\in\mathbb{R}^{n\times n}$. An approximate eigenvalue λ_1 and the corresponding eigenvector x. $\text{iter}_{\max}$ denotes the maximum number of iterations.
Output: Refined eigenvalue λ_∞ and its eigenvector x.
function $[\lambda_\infty, x]\leftarrow$ **SICE**$(A,\lambda_1,x,iter)$
  $[Q,U]\leftarrow \text{schur}(A)$ ▷ obtain Schur decomposition $A=QUQ^\intercal$, $QQ^\intercal=I$.
  $[m,s]\leftarrow max(abs(x)); x\leftarrow x/m$ ▷ Normalizing x so that $\|x\|_\infty=s_x=1$.
  for i in $1:\text{iter}_{\max}$ **do**
    $r\leftarrow\lambda_i x-Ax$
    $c\leftarrow -x-a_{\lambda s}$
    $d\leftarrow Q^\intercal c$
    $f^\intercal\leftarrow Q(s,:)=e_s^\intercal Q$ ▷ s-th row of Q.
    $\bar{U}_\lambda\leftarrow Q_1(U-\lambda_i I); \bar{d}\leftarrow Q_1 d=\|d\|_2 e_1$ ▷ Givens rotations Q_1 from Eq. (12)
    $\bar{U}_\lambda\leftarrow\bar{U}_\lambda+\bar{d}(1)f^\intercal$
    $\bar{U}_\lambda\leftarrow Q_2\bar{U}_\lambda$ ▷ Givens rotations Q_2 to introduce upper triangular form.
    Solve the triangular system $\bar{U}_\lambda z=Q_2Q_1Q^\intercal r$
    $y\leftarrow Qz$
    $\lambda_{i+1}\leftarrow\lambda_i+y(s)$ ▷ Update eigenvalue.
    $y(s)\leftarrow 0$ ▷ Set $y(s)$ to 0.
    $x\leftarrow x+y$ ▷ Update eigenvector.
    if desired accuracy is reached **then**
      break
    end if
  end for
end function

apply the tridiagonal reduction to it. Eqs. (10) and (11) in this case become

$$Q\left(T_\lambda+Q^\intercal ce_s^\intercal Q\right)Q^\intercal y=r+y_s\tilde{y} \quad (14)$$

and

$$\left(T_\lambda+d\times f^\intercal\right)Q^\intercal y=Q^\intercal g \quad (15)$$

the same with $d=Q^\intercal c$, $f^\intercal=e_s^\intercal Q$ and $g=r+y_s\tilde{y}$. Dongarra [2] discussed the approach of using the Sherman–Morrison formula [18]

$$(A-uv^\intercal)^{-1}=A^{-1}-\frac{A^{-1}uv^\intercal A^{-1}}{1+v^\intercal A^{-1}u} \quad (16)$$

for solving the rank-one updated system. Eq. (15) does not apply since $T_\lambda=T-\lambda I$ is singular by construction. However, this may not be so in mixed-precision setting. Consider the scheme that first performs the tridiagonal reduction in single precision and then solves the tridiagonal eigenvalue problem in double precision. The initial λ_T will be the eigenvalue of T with double-precision accuracy, but it only approximates λ_A, the eigenvalue of A with single-precision accuracy. With suitably chosen offset δ of order of ϵ_{single}, $T-(\lambda+\delta)I$ will no longer be singular in double precision, and the Sherman–Morrison formula can be applied. The special case in which this would fail is when $\|\lambda_T-\lambda_A\|=O(\epsilon_{\text{double}})$: the initial eigenvalue is also an accurate eigenvalue of A in double precision. In such a case, we do not need to refine the eigenvalue and can simply apply the inverse iteration to find the eigenvector.

Algorithm 2 SICE-SM algorithm: SICE algorithm with Sherman–Morrison formula

1: **Input:** Matrix $A = A^\intercal \in \mathbb{R}^{n\times n}$. An approximate eigenvalue λ_1 and the corresponding eigenvector x. $\text{iter}_{\max}$ denotes the maximum number of iterations.
2: **Output:** Refined eigenvalue λ_∞ and eigenvector x.
3: **function** $[\lambda_\infty, x] \leftarrow$ **SICE_SM**$(A, \lambda_1, x, iter)$
4: $[Q, T] \leftarrow \text{tridiag}(A)$ ▷ Tridiagonalization $A = QTQ^\intercal$, $QQ^\intercal = I$.
5: $[m, s] \leftarrow max(abs(x)); x \leftarrow x/m$ ▷ Normalization of x so that $\|x\|_\infty = s_x = 1$.
6: **for** i in $1 : \text{iter}_{\max}$ **do**
7: $r \leftarrow \lambda_i x - Ax$
8: $c \leftarrow -x - a_{\lambda s}$
9: $d \leftarrow Q^\intercal c$
10: $f^\intercal \leftarrow Q(s,:) = e_s^\intercal Q$ ▷ s-th row of Q.
11: $rhs \leftarrow Q^\intercal r$
12: $u \leftarrow (T - \lambda_i I)^{-1} d$
13: $v \leftarrow (T - \lambda_i I)^{-1} rhs$
14: $y \leftarrow v - \frac{f^\intercal v}{1+f^\intercal u} u$ ▷ Sherman–Morrison formula
15: $y \leftarrow Qy$
16: $\lambda_{i+1} \leftarrow \lambda_i + y(s)$ ▷ Update eigenvalue.
17: **if** $i \neq 1$ **then**
18: $y(s) \leftarrow 0$ ▷ Set $y(s)$ to 0.
19: $x \leftarrow x + y$ ▷ Update eigenvector.
20: **end if**
21: **break if** accuracy reached
22: **end for**
23: **end function**

We outline the SICE algorithm with Sherman–Morrison formula in Algorithm 2. Applying Sherman–Morrison formula from Eq. (16) to Eq. (15) we get

$$Q^\intercal y = \left(T_\lambda^{-1} - \frac{T_\lambda^{-1} d \times f^\intercal T_\lambda^{-1}}{1 + f^\intercal T_\lambda^{-1} d} \right) Q^\intercal g \tag{17}$$

or

$$Q^\intercal y = T_\lambda^{-1} Q^\intercal g - \frac{f^\intercal (T_\lambda^{-1} Q^\intercal g)}{1 + f^\intercal (T_\lambda^{-1} d)} T_\lambda^{-1} d \tag{18}$$

These involve solving the tridiagonal system T_λ with two different right hand sides d and $Q^\intercal g$. It can be easily done with the Thomas algorithm which is a special case of Gaussian elimination. There are other parallel tridiagonal solvers available and we will discuss them in Section II-1.

The main difference between Algorithms 1 and 2 is the use of the Sherman–Morrison formula to solve the system from line 12 to 14 instead of using the Givens rotations for that purpose. It is applied to solving the same tridiagonal system T_λ with two different right hand sides d and $Q^\intercal g$. The two vector inner products are needed to obtain the scalar in order to form the solution. Note that in line 17, we only update the eigenvalue at the first iteration and leave the eigenvector unchanged because T_λ at the first iteration is nearly singular. Other approaches to this issue include manually applying a shift to the initial eigenvalue or using the Ritz value $x^\intercal Ax / x^\intercal x$ as the starting point. Apart from tridiagonalization, the computational cost for algorithm 2 is dominated by the matrix-vector multiplications.

TABLE I
PERFORMANCE OF $n \times n$ MATRIX TIMES $n \times m$ AGGREGATED VECTORS ON NVIDIA V100-SXM2-32GB GPU, DGEMM ROUTINE FROM CUBLAS V11.0.

Matrix size	Number of vectors	Time (ms)	Performance (GFLOP/s)
20000	1	3.76	212.65
20000	8	3.79	1688.17
20000	32	6.48	3949.32
20000	128	13.57	7544.43

Alternatively, as described in [12], one can also solve the Jacobian matrix with the special structure $J = \begin{pmatrix} T - \lambda I & y \\ z^\intercal & 0 \end{pmatrix}$, which is a tridiagonal system with an extra row and column at the bottom and right. However, it is hard to parallelize the corresponding solver for this special structure and it is even harder make it scalable. This is in stark contrast with the approach of solving the tridiagonal system which is well studied and admits several parallel implementations that target a variety of computing environments.

b) Blocked SICE-SM Algorithm: The computational cost of Algorithm 2 is dominated by matrix-vector multiplications especially inside the refinement iteration. In the matrix-vector multiplication, the whole matrix is read once and only a single multiplication and addition are performed per each of the fetched elements. This results in a low arithmetic intensity of 2, which results in very low inefficient on modern hardware including CPU, GPUs, and computational accelerators. To improve on this implementation aspect, we can aggregate several eigenpairs simultaneously and refine them at the same time while they are cached in higher levels of the memory hierarchy. This blocking strategy is common in numerical linear algebra since it was introduced in LAPACK[19] and relies on grouping computations so that Level 3 Basic Linear Algebra Subprograms (BLAS) may be utilized to perform operations that are rich in matrix-matrix multiplications. These operations perform more efficiently as they have higher arithmetic intensity resulting from higher data reuse in fast portions of the cache hierarchy. In our case, we assume that the matrix size is far greater than the number of eigenpairs to refine. Then the matrix-vector multiplication is dominated by the reading of the matrix elements. And with the blocked version, it the additional cost of refining extra eigenpairs is negligible. In Table I, we show examples of the performance rates and execution times for different numbers of vectors submitted to the DGEMM routine from cuBLAS on the NVIDIA V100 GPU. The times for 1 and 8 vectors are almost the same. And for 32 or 128 vectors the elapsed time increases $3.6\times$.

There are a few issues we need to solve while formulating a blocked variant of the algorithm. First, in SICE, the eigenvector is first normalized in infinity norm. The index s is also picked so that $\|x\|_\infty = s_x = 1$. If we allow different s for each of the eigenpairs, then we will have to access different columns in A to construct vector c, and also different rows of Q for vector $f^\intercal$. The row access required for the latter is performed in column major layout and results in non-coalescing memory accesses which are extremely slow

and should be avoided as much as possible due to their low utilization of the GPU's memory bandwidth. To show that it is fine to choose s arbitrarily, we need to take a closer look at the matrix in Eq. (14) and expand it without canceling any terms we get

$$(QT_\lambda Q^\intercal + QQ^\intercal v e_s^\intercal QQ^\intercal)y = r + y_s\tilde{y} \tag{19}$$

Again, for our mixed-precision scheme, we would like to perform the tridiagonalization in single precision. Hence $QT_\lambda Q^\intercal$ is only an approximation of A with precision ϵ_{single}, i.e. $\|A_\lambda - QT_\lambda Q^\intercal\| \sim O(\epsilon_{\text{single}})$. The same applies to $QQ^\intercal$ which is only an approximation of I with $\|QQ^\intercal - I\| \sim O(\epsilon_{\text{single}})$. So no matter which index s we pick, we will always get an error of order ϵ_{single} in the correction of eigenvalue y_s coming from the other elements in the solution vector y. There could be a potential problem if the eigenvalue itself is small and the error is preventing the eigenvalue to be refined to desire accuracy. This can be remedied by pre-scaling the matrix so that the eigenvalues are not too small.

The other issue is that by treating the eigenpairs independently they might lose their orthogonality. In the worst case, they might all converge to the same eigenpair. However, it is easy to reorthogonalize with

$$X' = X + \frac{1}{2}X(I - X^\intercal X) \tag{20}$$

In practice, we found that it is sufficient to reorthogonalize after the refinement is done. Doing so in each iteration would not speed up the convergence. The computation of $I - X^\intercal X$ also lets us detect if they converged to the same eigenvector. By combining these considerations, we arrive at Algorithm 3.

Because a Hermitian matrix can also be tridiagonalized into real matrix, algorithm 3 can easily be extended to be applied on Hermitian matrices. The transformation matrix Q now becomes complex, as well as the intermediate vectors. However, the coefficients in $T - \lambda I$ are all real so it can be optimized to avoid doing all the operations in complex space.

c) Implementation Details: The one-stage eigensolver has the following components with its corresponding LAPACK routine names: DSYTRD (Tridiagonalization via Householder transformations), DSTEDC (Tridiagonal symmetric eigensolver based on divide-and-conquer), DORMTR (back transformation for eigenvectors).

First the system is transformed to the tridiagonal form via Householder transformations. Then the tridiagonal eigensolver is called. We will not discuss the details of eigensolvers here, as it is not the focus of this work. After the eigenvalues and eigenvectors of the tridiagonal system are computed, the back transformation is applied, which is the inverse of the Householder transformations from tridiagonalization stage. Because the transformation is orthogonal, the inverse is simply a transpose. If only a portion of the eigenvectors are requested, the transform would not be explicitly formed for performance reasons. The transform in the form of elementary reflectors is directly applied on eigenvectors of the tridiagonal system to obtain the eigenvectors for the original matrix.

Algorithm 3 Blocked SICE-SM algorithm

Input: $A = A^T \in \mathbb{R}^{n\times n}$, initial eigenvectors $X = [x_1|x_2|...|x_\ell] \in \mathbb{R}^{n\times\ell}$ and the corresponding initial eigenvalues $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)^T \in \mathbb{R}^l$. $\text{iter}_{\max}$ denotes the maximum number of iterations.
Output: Refined eigenvectors X and refined eigenvalues Λ.
function $[X, \Lambda] \leftarrow$ **SICE_SM_BLK**$(A, X, \Lambda, iter)$
  $[Q, T] \leftarrow \text{tridiag}(A)$ ▷ Tridiagonalization $A = QTQ^\intercal$, $QQ^\intercal = I$.
  for i in $1 : \text{iter}_{\max}$ **do**
    $s \leftarrow i$
    $R \leftarrow X \times \text{diag\_matrix}(\Lambda) - A \times X$ ▷ Residual vectors need higher precision.
    for j in $1 : \ell$ **do**
      $c_j \leftarrow -x_j - A(:, s)$
    end for
    Compose matrix $C = [c_1|c_2|...|c_\ell]$ from column vectors c_j
    $C(s, :) \leftarrow C(s, :) + \Lambda^T$
    $D = [d_1|d_2|...|d_\ell] \leftarrow Q^T \times C$ ▷ Can be in lower precision.
    $RHS = [rhs_1|rhs_2|...|rhs_\ell] \leftarrow Q^T \times R$ ▷ Can be in lower precision.
    $f \leftarrow Q(s, :)$ ▷ s-th row of Q.
    for j in $1 : \ell$ **do**
      $u_i \leftarrow (T - \lambda I)^{-1} d_i$
      $v_i \leftarrow (T - \lambda I)^{-1} rhs_i$
      $y_i \leftarrow v_i - \frac{f^\intercal v_i}{1 + f^\intercal u_i} u_i$ ▷ Sherman–Morrison
    end for
    Compose matrix $Y = [y_1|y_2|...|y_\ell]$ from correction vectors y_j
    $Y \leftarrow Q \times Y$
    $\Lambda \leftarrow \Lambda + Y(s, :)^T$ ▷ Update eigenvalues.
    if $i \neq 1$ **then**
      $Y(s, :) \leftarrow 0$ ▷ Set $y_i(s)$ to 0.
      $X \leftarrow X + Y$ ▷ Update eigenvectors.
      Normalize eigenvectors x_i in X.
    end if
    if desired accuracy reached **then**
      break
    end if
  end for
  $X \leftarrow X + \frac{1}{2}X(I - X^\intercal X)$ ▷ Orthogonalization.
end function

Algorithm 4 Mixed precision one stage symmetric eigensolver with iterative refinement

1: SSYTRD: Tridiagonalization via Householder transformations in single precision.
2: DSTEDC: Tridiagonal symmetric eigensolver (divide and conquer) in double precision.
3: SORGTR: Generate the transformation matrix Q from elementary reflectors in single precision.
4: Blocked SICE-SM (algorithm 3) for iterative refinement.

For the mixed-precision eigensolver in Alg. 4, we first perform tridiagonalization in single precision as it is computationally intensive requiring $O(n^3)$ operations. After the system is transformed to tridiagonal form, the eigensolver is applied. The eigensolver operates in double precision as we need to be able to distinguish nearby eigenvalues that

are closer than ϵ_{single} but not closer than ϵ_{double}. If single precision is used for this case, the eigenvalues are very likely to be considered as repeated, and the returned eigenvectors could be an arbitrary orthogonal basis of the eigenspace. For the back transformation, the matrix Q needs to be explicitly formed in order for us to solve Eq. (15). Then the Blocked SICE-SM (Algorithm 3) is used to iteratively refine the eigenpairs to the desired accuracy. Most of the operations in the refinement process are matrix-matrix operations, which have been developed internally. The batched tridiagonal solver in line 16 will be discussed in section II-1.

For two-stage algorithms, the structure is similar to the one-stage method but both the forward- and back-transformations are split into two reduction steps: first stage symmetric to band via Householder transformations, second stage band to tridiagonal via bulge chasing, tridiagonal symmetric eigensolver (divide and conquer), back-transformation for second stage on eigenvectors, back-transformation for first stage on eigenvectors.

Mixed precision for a two-stage eigensolver is actually more problematic performance-wise. The main reason is that accumulation of the back transformations from the second stage of bulge chasing is costly: it has a lot of small transformations and is expensive to apply on a square transform matrix Q compared to the case of only computing the eigenvectors. However, we need to explicitly form Q for the later refinement. Here, we exploit the fact that the back transformation is not applied on the eigenvectors; it can actually start as soon as the first stage is finished. So we are reversing the order of back transformations to start it first. Similarly, the back transformation of the second stage can start when both the second stage and the back transformation of the first stage are completed. This is shown in Algorithm 5. For the case of MAGMA implementation, this would enable more parallelism. The back transformation of the first stage can be done on the GPU while the second stage of bulge chasing is done on the CPU. The eigensolver, which is mainly done on the CPU, can be overlapped with the back-transformation of the second stage on the GPU.

Algorithm 5 Mixed precision two stages symmetric eigensolver with iterative refinement

1: First stage symmetric to band via Householder transformations in single precision.
2: Second stage band to tridiagonal via bulge chasing in single precision.
3: Tridiagonal symmetric eigensolver (divide and conquer) in double precision.
4: Generate the transformation matrix Q from first stage in single precision. This can start as soon as 1. finishes.
5: Apply the back transformation for second stage onto Q in single precision. This can start as soon as both 2. and 4. finish.
6: Blocked SICE-SM (algorithm 3) for iterative refinement.

1) Batched Tridiagonal Solver: Line 16 in Algorithm 3 iterates over all the eigenvalues and solves the shifted tridiagonal system for each of them as in Batched BLAS[20], [21]. On multicore CPUs, the straightforward and efficient approach is to assign one system to each thread at a time which is likely bound to a single CPU core. Each thread can use the Thomas algorithm. But on the GPU, we need more parallelism to saturate the computational potential of the hardware. There are previous studies[22], [23], [24] that investigated the solving of one big tridiagonal system on GPUs. One of the techniques is based on the cyclic reduction (CR). Consider a tridiagonal system with 8 unknowns:

$$\mathbb{R}^{8\times 8} \ni A \times \vec{x} = \vec{y} \in \mathbb{R}^8 \tag{21}$$

By combing all the even-indexed equations with odd-indexed equation, we are able to have an updated system with half of the size:

$$\begin{bmatrix} b'_1 & c'_1 & & \\ a'_3 & b'_3 & c'_3 & \\ & a'_5 & b'_5 & c'_5 \\ & & a'_7 & b'_7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix} = \begin{bmatrix} y'_1 \\ y'_3 \\ y'_5 \\ y'_7 \end{bmatrix} \tag{22}$$

The coefficients of the updated system can be computed with the following formulas:

$$\begin{aligned} k_1 = \frac{a_i}{b_{i-1}},\ k_2 = \frac{c_i}{b_{i+1}} \\ a'_i = -a_{i-1}k_1,\ b'_i = b_i - c_{i-1}k_1 - a_{i+1}k_2 \\ c'_i = -c_{i+1}k_2,\ y'_i = y_i - y_{i-1}k_1 - y_{i+1}k_2 \end{aligned} \tag{23}$$

By recursively reducing the size of the system by half, it is possible to bring the size down to a single unknown with a trivial solution. Then, the back-substitutions follows the same path in reverse order and thus the solution of the full system is obtained. Alternatively, while reducing the size of systems, we can produce two independent systems, one with odd-indexed unknowns and the other with the even-indexed unknowns. Both systems can be solved independently with only its own coefficients. By repeating the process, we will arrive at trivial systems with a single unknown $b''_i x_i = y''_i$ for all of the unknowns x_i. The back substitutions wold not be needed for this approach, which is called *parallel cyclic reduction* (PCR). The PCR method exposes more parallelism towards the end but with requires more computation which represents a design trade-off. For our GPU implementation, we used PCR to solve one tridiagonal system by each of the thread blocks.

III. Numerical and Performance Results

The numerical experiments in this section will be divided into two parts: convergence and performance.

a) Numerical Convergence: The numerical experiments in this section were performed in MATLAB version R2020a with implementations of Algorithm 3 (blocked SICE-SM). The expression `A = gallery('randsvd',n,-cond)` was used to generate symmetric test matrices with a prescribed condition number from random eigenvectors and geometrically

distributed eigenvalues in range $(1, \frac{1}{\text{cond}})$. The input matrix is first converted to `single` precision and subsequently tridiagonalized using `[Q,T] = hess(A)` function in single precision. Then converted back to double precision for finding the eigenvalues and eigenvectors using expression `[V,D] = eig(A)`. The eigenvectors in `D` and column eigenvectors in `QV` will be used as the starting point of our refinement algorithms.

Figure 1 shows the convergence of Algorithm 3: the blocked SICE-SM. The input symmetric input matrix had size 100 with geometrically distributed eigenvalues from 1 to 10^{-7}. The convergence in terms of residual $\|Ax-\lambda x\|_\infty$ of each eigenvalues are plotted in different colors from blue as largest eigenvalue 1 to red as the smallest eigenvalue 10^{-7}. For the first iteration, we only updated the eigenvalues so there was no initial improvement. For large eigenvalues, the method converges quickly in two iterations. However, for small eigenvalues, that are much closer to each other due to the geometrical distribution and thus we observer the resulting slowdown of convergence.

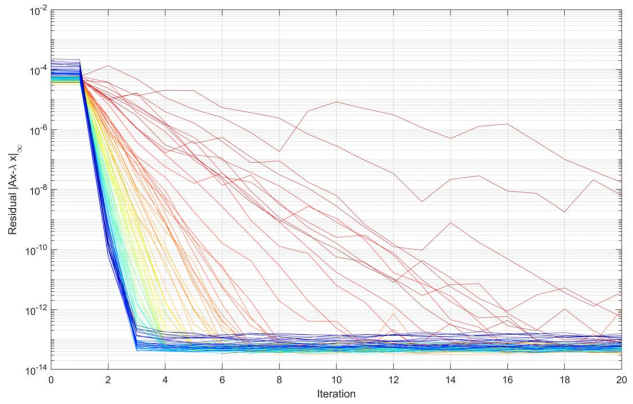

Fig. 1. Blocked SICE-SM convergence of a 100×100 matrix with geometrically distributed eigenvalues from 1 (blue) to 10^{-7} (red).

b) Performance Results: The system we are using has two sockets of Intel(R) Xeon(R) CPU E5-2650 v3 CPUs. But only one is being used for more stable results. The system is accelerated by a Tesla V100 GPU. The theoretical peak performance of a V100 is 7.8 TFLOP/s in double precision and 15.6 TFLOP/s in single precision. The software stacks was composed of Intel Parallel Studio Cluster 2020. (for C and Fortran compilers and BLAS rouintes from MKL library), NVIDIA CUDA v11.0.2, and MAGMA version 2.5.4. The input symmetric matrix $A \equiv [a_{ij}]$ was generated with random elements from a uniform distribution in range $(0, 1)$: $a_{ij} \sim \mathcal{U}(0, 1)$ and $a_{ij} = a_{ji}$. The Hermitian matrix is also generated in the same fashion for it's imaginary part. The largest eigenvalues in the spectrum were requested. The blocked SICE-SM algorithm was implemented in both PLASMA and MAGMA.

In Figure 2, PLASMA was used in a CPU-only mode and no GPUs were used in the system. The symmetric input matrix had size $n = 10000$. The three stacked bars represent the breakdown of time from mixed-precision with refinement, single precision, and double precision from the two-stage algorithm, respectively. The time for single precision is about half of that of double precision and each of the components take proportionally the same time for both precisions.

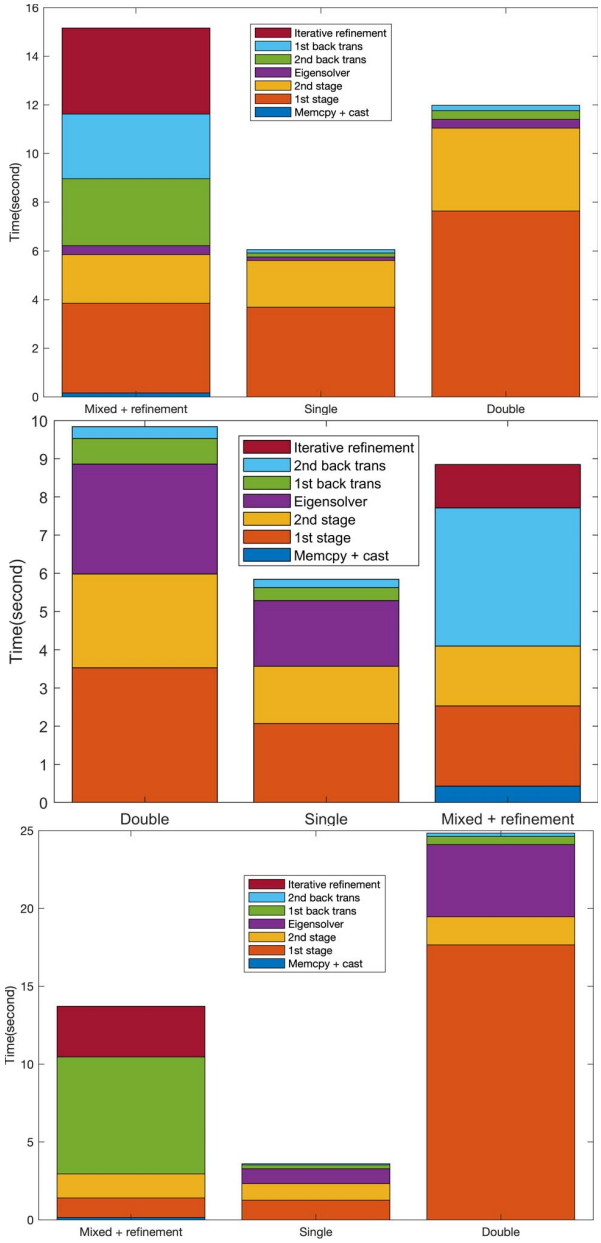

Fig. 2. Breakdown of timings of two-stage eigensolvers with 32 largest eigenparis requested in PLASMA (top), MAGMA on NVIDIA Volta V100 (middle), and MAGMA on NVIDIA GTX1060 (bottom). The problem sizes are 10 000, 20 000, and 12 000, respectively.

Figure 3 shows the performance results from the MAGMA. First the solid lines are the one-stage algorithm in double, single, and mixed precision (with iterative refinement). The input matrix sizes range from 1000 to 20000, and the largest 32 eigenpairs are requested. Single precision is about $1.7\times$ faster than double precision and the mixed precision is about $1.3\times$ faster. The dashed lines represent the two-stage algorithm. They are at least $2\times$ faster than their corresponding single

stage algorithm in general. The performance improvement over double precision is about $1.2\times$. Left of Figure 4 shows the performance results of complex Hermitian solvers. Complex operations has higher arithmetic intensity so the performance gap between single and double would also be larger. Mixed precision algorithm can also have greater chance to benefit it. On the system wit NVIDIA V100, we are observing complex single is $2.44\times$ faster than complex double and mixed precision solver is $1.45\times$.

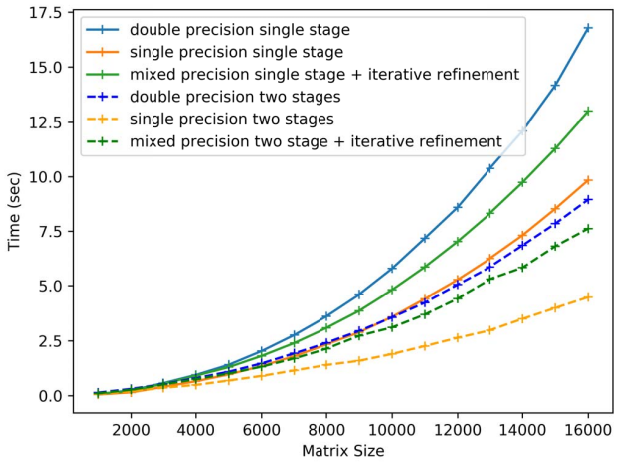

Fig. 3. Performance comparison of single, double, and mixed precision solvers for real symmetric matrix on MAGMA for both single stage and two-stage algorithms on NVIDIA V100 GPU with varying sizes of matrices and fixed number of requested eigenpairs.

The left of Figure 5 shows the performance when requesting different numbers of eigenpairs with the input matrix size fixed at $n = 20000$. Mixed precision is noticeably faster than double precision if 64 or fewer eigenpairs are requested. For larger eigenpair count, the time in iterative refinement grows linearly with the number of requested eigenpairs and it eventually looses its performance advantage.

The middle of Figure 2 shows the detailed profile for matrix size $n = 20000$ and 32 eigenvalues/eigenvectors requested. The details of computational components were explained in Section II-0c. The single precision routine took 60% of time compared to double, and the ratios between components across precisions were about the same. For mixed precision, there is a 0.5 second overhead at the beginning to convert the whole matrix from double to single precision.

We tested another machine with a drastically different setup by using a consumer-grade gaming GPU. It has the same CPUs as the V100 system. The GPU is NVIDIA GTX1060 6GB GPU. The theoretical peak performance of GTX1060 is 136.7 GFLOP/s in double and 4.375 TFLOP/s in single precision. This is a notable different as the gaming maintains 1:32 double-single ratio compared to server-grade NVIDIA V100 with the ratio being 1:2. The right of Figure 4 shows the performance with different matrix sizes on GTX1060 when requesting the largest 32 eigenpairs. The performance of single precision is about $8\times$ better than that of double precision and the mixed precision with refinement is about $2\times$ better than

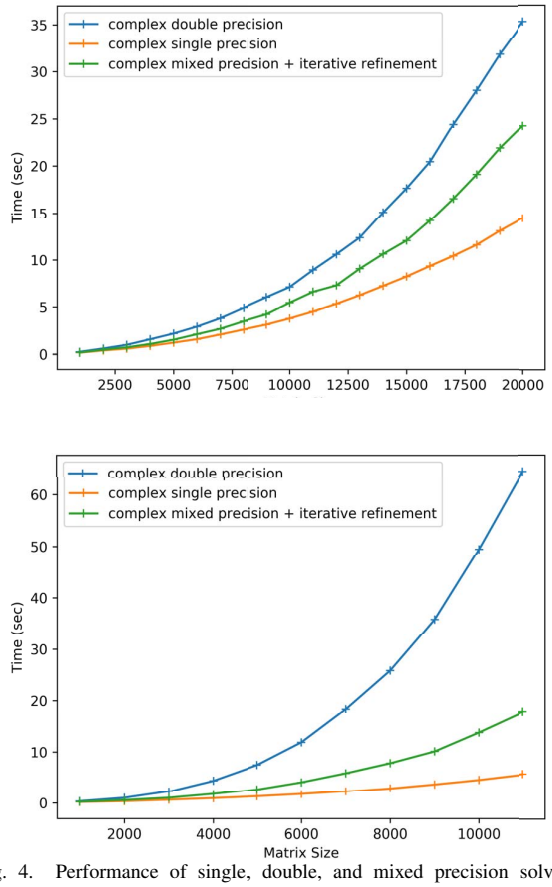

Fig. 4. Performance of single, double, and mixed precision solvers for complex Hermitian matrix based on MAGMA two-stage algorithm with varying sizes of matrices and fixed number of requested eigenpairs on NVIDIA V100 GPU (top) and NVIDIA GTX1060 GPU (bottom).

double precision. The left of Figure 4 shows the complex Hermitian solver and the the speed up over complex double is $3.6\times$. In the right of Figure 5, we show performance results when the matrix size was fixed at $n = 12000$ but with varied number of requested eigenpairs. The mixed precision solver is still faster than double precision when 128 eigenpairs are requested, but the time in iterative refinement became significant if more eigenvalues and eigenvectors were requested.

The right of Figure 2 shows the profiling results with timing breakdown for matrix size $n = 12000$ and the 32 largest eigenpairs requested. In double precision, almost 80% of time was spent at the first stage to reduce the matrix from symmetric to band-symmetric form. The operation is compute-bound and relies on GPU's matrix-matrix multiplication efficiency. But the consumer-grade GPU does not have hardware to support high-efficiency processing for the double floating-point units and consequently extra clock cycles are used to emulate higher precision with single precision instructions. The mixed-precision algorithm does the first-stage reduction in single precision and does not suffer from the same penalty. The back-transformation of second stage is still costly but it is done with single precision on the GPU. Overall, the performance of mixed precision with the iterative refinement algorithm is $2\times$ faster over purely double two-stage algorithm.

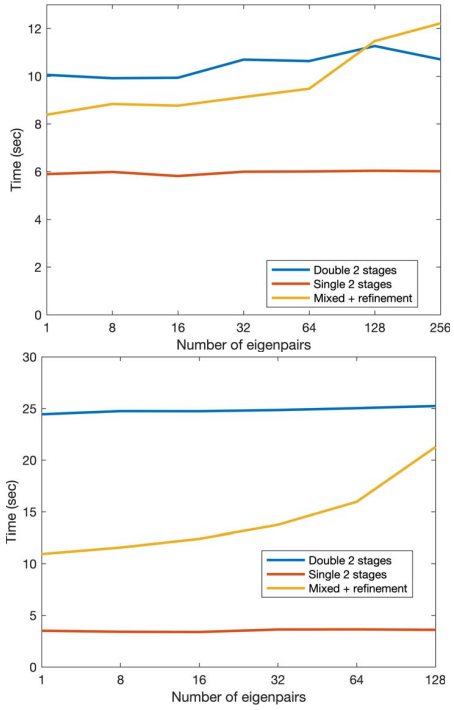

Fig. 5. Performance comparison of single, double, and mixed precision solvers on top of MAGMA with varying number of requested eigenpairs and fixed matrix size on NVIDIA V100 GPU (top) and NVIDIA GTX1060 GPU (bottom). The problem sizes are 12 000, and 20 000, respectively.

IV. Conclusions

We developed an iterative refinement algorithm for symmetric and Hermitian eigenvalue problems based on the initial work from the SICE algorithm. By utilizing the Sherman–Morrison formula, our new solver has more opportunity to be parallelized compared to the serial Givens rotations in the SICE algorithm. The blocked version of the algorithm was also proposed in order to refine multiple pairs of eigenvalues and eigenvectors simultaneously for higher utilization of the computational resources with lower demand for memory bandwidth. The implementation of the mixed-precision algorithm is based on the two-stage eigensolver in either the PLASMA and MAGMA software libraries for numerical linear algebra, which gives our implementation the advantage of both portability and performance. The computational components inside the mixed-precision algorithm have been reordered to create more parallelism at runtime and allow additional overlap to computational stages more efficiently. Compared to the double-precision solver, the performance benefit has been shown for the cases in which only a portion of eigenvalues and corresponding eigenvectors are requested. This remains true across hardware with a varying ratio of performance of single and double precision units.

References

[1] H. Symm and J. H. Wilkinson, "Realistic error bounds for a simple eigenvalue and its associated eigenvector," *Numerische Mathematik*, vol. 35, no. 2, pp. 113–126, 1980.

[2] J. J. Dongarra, "Improving the accuracy of computed matrix eigenvalues," Argonne National Lab., Chicago, IL, USA, Tech. Rep., 1980.

[3] ——, "Algorithm 589: SICEDR: a FORTRAN subroutine for improving the accuracy of computed matrix eigenvalues," *ACM Transactions on Mathematical Software (TOMS)*, vol. 8, no. 4, pp. 371–375, 1982.

[4] J. J. Dongarra, C. B. Moler, and J. H. Wilkinson, "Improving the accuracy of computed eigenvalues and eigenvectors," *SIAM Journal on Numerical Analysis*, vol. 20, no. 1, pp. 23–45, 1983.

[5] G. Peters and J. H. Wilkinson, "Inverse iteration, ill-conditioned equations and Newton's method," *SIAM review*, vol. 21, no. 3, pp. 339–360, 1979.

[6] T. Yamamoto, "Error bounds for computed eigenvalues and eigenvectors," *Numerische Mathematik*, vol. 34, no. 2, pp. 189–199, 1980.

[7] G. W. Stewart, "Error and perturbation bounds for subspaces associated with certain eigenvalue problems," *SIAM review*, vol. 15, no. 4, pp. 727–764, 1973.

[8] F. Chatelin, "Simultaneous Newton's iteration for the eigenproblem," in *Defect correction methods*. Springer, 1984, pp. 67–74.

[9] J. W. Demmel, "Three methods for refining estimates of invariant subspaces," *Computing*, vol. 38, no. 1, pp. 43–57, 1987.

[10] G. Alefeld and H. Spreuer, "Iterative improvement of componentwise errorbounds for invariant subspaces belonging to a double or nearly double eigenvalue," *Computing*, vol. 36, no. 4, pp. 321–334, 1986.

[11] F. Tisseur, "Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 4, pp. 1038–1057, 2001.

[12] K. E. Prikopa and W. N. Gansterer, "On mixed precision iterative refinement for eigenvalue problems," *Procedia Computer Science*, vol. 18, pp. 2647–2650, 2013.

[13] T. Ogita and K. Aishima, "Iterative refinement for symmetric eigenvalue decomposition," *Japan Journal of Industrial and Applied Mathematics*, vol. 35, no. 3, pp. 1007–1035, 2018.

[14] ——, "Iterative refinement for symmetric eigenvalue decomposition II: clustered eigenvalues," *Japan Journal of Industrial and Applied Mathematics*, vol. 36, no. 2, pp. 435–459, 2019.

[15] ——, "Iterative refinement for singular value decomposition based on matrix multiplication," *Journal of Computational and Applied Mathematics*, vol. 369, p. 112512, 2020.

[16] A. Haidar, H. Ltaief, and J. Dongarra, "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*, 2011, pp. 1–11.

[17] A. Haidar, S. Tomov, J. Dongarra, R. Solcà, and T. Schulthess, "A novel hybrid CPU–GPU generalized eigensolver for electronic structure calculations based on fine-grained memory aware tasks," *The International journal of high performance computing applications*, vol. 28, no. 2, pp. 196–209, 2014.

[18] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[19] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney *et al.*, *LAPACK Users' guide*. SIAM, 1999.

[20] J. Dongarra, S. Hammarling, N. J. Higham, S. D. Relton, P. Valero-Lara, and M. Zounon, "The design and performance of batched BLAS on modern high-performance computing systems," *Procedia Computer Science*, vol. 108, pp. 495–504, 2017.

[21] A. Abdelfattah, T. Costa, J. Dongarra, M. Gates, A. Haidar, S. Hammarling, N. J. Higham, J. Kurzak, P. Luszczek, S. Tomov, and M. Zounon, "A set of batched basic linear algebra subprograms and LAPACK routines," *ACM TOMS*, vol. 47, no. 3, p. 1–23, 2020, dOI: 10.1145/3431921.

[22] Y. Zhang, J. Cohen, and J. D. Owens, "Fast tridiagonal solvers on the GPU," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 127–136, 2010.

[23] A. Davidson, Y. Zhang, and J. D. Owens, "An auto-tuned method for solving large tridiagonal systems on the GPU," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 956–965.

[24] L.-W. Chang, J. A. Stratton, H.-S. Kim, and W.-M. W. Hwu, "A scalable, numerically stable, high-performance tridiagonal solver using GPUs," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

This work was supported in part by NSF OAC 2004541 and the Exascale Computing Project (17-SC-20-SC).