PIER C

(Received 12 September 2025, Accepted 24 November 2025, Scheduled 13 December 2025)

Matrix Square Root Based Differentiable RCWA Implementation for High-Performance Parallel Computing

Frank Van der Ceelen[1, *], Yifeng Shao[1], and Wim Coene[1, 2]

[1] *Optics Cluster, Imaging Physics Department, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, Netherlands*
[2] *ASML De Run 6501, 5504 DR Veldhoven, The Netherlands*

ABSTRACT: Rigorous Coupled-Wave Analysis (RCWA) is a semi-analytical method, used to determine the optical response of nanostructures, such as meta-materials. Recently, the ability to combine RCWA with automatic differentiation for optical response optimization has been demonstrated. We seek to build upon this use by attempting to address RCWA's poor performance on parallel computer architecture, stemming from the presence of an eigendecomposition. We do this by outlining an alteration of RCWA, which replaces the eigendecomposition with a matrix square root and matrix exponential evaluation. Furthermore, we demonstrate that these matrix functions can be evaluated using algorithms which are both differentiable and readily evaluated in parallel. Finally, we show that replacing the eigendecomposition with these matrix functions resolves the bottleneck and paves the way for higher-accuracy parameter retrieval using RCWA approaching real-time performance, without compromising stability.

1. INTRODUCTION

The Rigorous Coupled-Wave Analysis (RCWA), also known as the Fourier Modal Method (FMM) [1] is a semi-analytical computational method [2], initially created to determine the optical response of diffraction gratings [3]. Since then, RCWA has seen use in the design of solar cells [4] and metamaterials [5], as well as in optical metrology in the semiconductor industry [6, 7].

RCWA often serves as the model to simulate the optical response of a device as a function of the permittivity distribution of its structure. These models are then used to solve an inverse problem, in which an optimal permittivity distribution is determined to achieve a certain desired optical response. Such inverse problems appear regularly in fields such as the design of photonic devices, as well as in optical metrology.

One important inverse problem method is gradient descent optimization [8], which has seen extensive use in the field of machine learning, which has recently seen several innovations to facilitate the development, training, and running of these machine learning models. On the hardware side, there has been significant progress on the development of processing units, i.e., the "Graphical Processing Unit" (GPU), specifically optimized for massively-parallel computing of simple processes in machine learning models at industrial scale [9]. The fast computation of these simple processes, such as matrix additions, multiplications, and solves, is key to the performance of the respective applications.

On the software side, we have seen the development of machine learning platforms such as TensorFlow, PyTorch, and Jax. These platforms are integrated with NVIDIA's CUDA toolkit, providing GPU performance to high-level programming languages. They also provide an additional feature to facilitate the training of machine learning models: Automatic Differentiation (AD). As any algorithm can be abstracted to the composition of a function from more basal functions, AD can evaluate the global gradient of the composed function through the chain rule, provided that the platform handles the knowledge of the local gradients of the basal functions. These features have reduced the barrier of entry for machine learning and gradient descent optimization, leading to the proliferation of this technology to other fields, such as inverse design in nanophotonics using RCWA [5].

Here, we seek to apply gradient descent to a differentiable RCWA algorithm for profilometry purposes. Profilometry typically aims to reconstruct the 3D profile of structural features, e.g., lines, contact holes, and bricks, of a diffraction grating. In such cases, higher evaluation times can severely limit application. As a result, we seek to use higher computation power to evalute a single RCWA more quickly by exploiting parallelism.

RCWA, however, seems resistant to such speedup through parallelization. This mainly stems from the inclusion of an eigendecomposition in the method, being a serial algorithm with a potentially ill-posed gradient in the case of degenerate eigenvalues. While there has been success in resolving the gradient problem for the eigendecomposition in RCWA [5], efforts to meaningfully resolve the computational bottleneck for parallel computation have been less successful. These works have generally taken one of two approaches.

The first category attempts to use the GPU architecture to evaluate the eigendecomposition itself [10–12]. While definite improvements with respect to computation time can be recorded using this method, in our assessment, the maximum possible speedup using such approaches are limited, due to the

* Corresponding author: Frank Van der Ceelen (F.vanderCeelen@tudelft.nl).

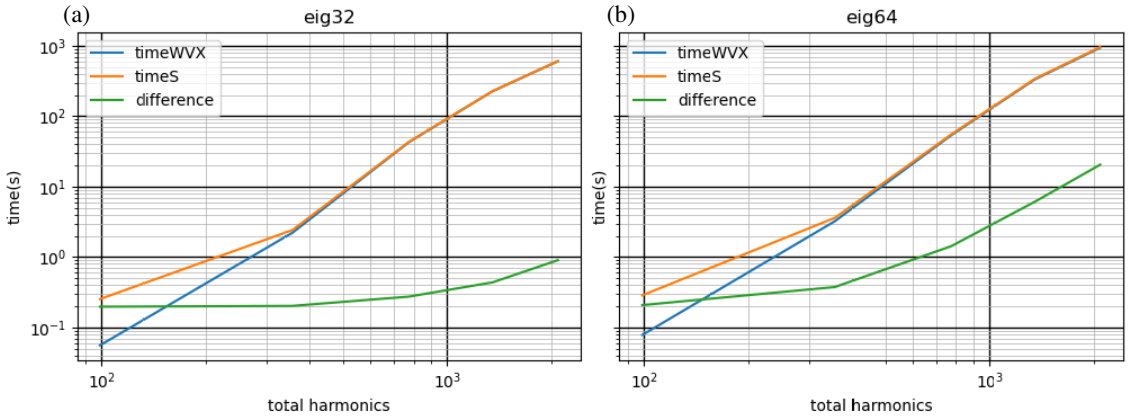

FIGURE 1. S-matrix computation times in (a) single and (b) double precisions. In each plot, we compare the time for computing the eigendecomposition through computing the **W**, **V**, **X** matrices (blue), the S-matrix (orange), and the difference between the two, which consists of mostly matrix solves and multiplications (green).

serial nature of the eigendecomposition. It should be noted that schemes exist to evaluate the eigendecomposition using parallel computing resources, but these have shown their own drawbacks with regard to accuracy and stability [13].

The second category attempts to completely avoid the evaluation of an eigendecomposition by evaluating a scattering transmission matrix (T-matrix), and evaluating the reflected/transmitted intensities from this information [14–16]. These methods are genuinely eigendecomposition-less, and have demonstrably strong performance on parallel architectures. The problem with such an approach is that T-matrices are numerically unstable, which limits their application to "sufficiently thin" slices, lest the method become dominated by rounding or truncation error. Furthermore, as the number of harmonics considered increases, the maximum thickness has to decrease accordingly, meaning that "sufficiently thin" becomes an increasingly prohibitive limitation [15].

This paper attempts to address both parallelization and differentiablity issues of RCWA stemming from the eigendecomposition. However, instead of doing so through altering the eigendecomposition algorithm, the eigendecomposition is removed outright. This was achieved through the use of an alternate formulation for computing the scattering matrix of each layer, which has identical numerical stability properties as that of the conventional formulation. The new formulation requires no eigendecomposition to be computed, instead requiring the evaluation of the matrix square root and matrix exponential. In this paper, we propose and validate an algorithm to compute the matrix square root, allowing the implementation of a differentiable RCWA that can be significantly accelerated using GPU.

2. ISSUE REGARDING THE EIGENDECOMPOSITION

To begin with, we wish to highlight the bottleneck observed in our algorithm, which we illustrate in Figure 1. A more elaborate explanation of the computational experiment can be found in Section 6.

Here, we effectively take the implementation of RCWA in the scattering matrix (S-matrix) formalism, as outlined in [17], and evaluated it for our base case. Here, we timed;

- The computation time of the eigendecomposition. More specifically, evaluating the **W**, **V**, **X** matrices, which are crucial intermediate steps of evaluating the S-matrices;
- The computation time of evaluating the S-matrix, which includes the eigendecomposition, but otherwise consists solely of matrix solves and matrix multiplications.

Through subtracting one from the other, we also determine the computation time of the "remainder" of the S-matrix evaluation algorithm, which effectively consists of evaluating Eqs. (19) and (20) in [17].

These were evaluated using CUDA on a GPU A-100 with floats in both single and double precisions. We shall note that the A-100 is designed for optimal performance in single precision arithmetic. By comparing computation time between these modes, we can get some semblance of information of how well certain parts of the algorithm parallelize.

These benchmarks were run with an increasing number of diffraction orders (or harmonics) considered, as outlined in Table 1. From this, we apply a log-log plot of the computation times as a function of number of harmonics considered. We implemented an RCWA algorithm, according to the formalism in [17], using TensorFlow. This formalism evaluates the S-matrices of individual layers, after which the overall S-matrix of the system is found through repeated use of the Redheffer star product.

From this benchmark, we see the following:

- There is little difference in computation time for the eigendecomposition between single and double precision, confirming that the number of serial operations scales cubically with problem size.
- By contrast, we see improvement in the computation time of the "remainder" of the S-matrix algorithm, especially at large problem sizes. This suggests this part of the algorithm is readily parallelizable according to Gustafson's

law [18], where the possible attainable speedup increases with problem size.

With this, we seek to demonstrate that except the eigendecomposition, the remainder of the algorithm to evaluate the S-matrix is itself already readily in parallel, owing to the algorithm being rich in matrix multiplications and inverses, were it not for the eigendecomposition serving as a bottleneck. This logic extends to the remainder of the RCWA algorithm itself, where we observed all other parts of the RCWA algorithm being evaluated properly in parallel as well. Additionally, it has already been demonstrated that the wider RCWA algorithm is differentiable, once differentiability issues of the eigendecomposition are addressed [5], and the analytic jacobians of matrix multiplications and inverses are well known [13].

Our goal, then, is to find an RCWA formulation, which can be readily evaluated in parallel, through elimination of the eigendecomposition. In order to do so, we first need to understand what the eigendecomposition achieves in the wider RCWA algorithm, and then see how this can be replaced with a parallel and differentiable algorithm, potentially by making use of the basal matrix operations.

3. BOTTLENECK IN THE CURRENT FORMULATION

One can reformulate the Maxwell equations into two coupled spatial-frequency domain, first-order matrix differential equations with respect to z as follows:

$$\frac{\partial \mathbf{s}}{\partial z} = \mathbf{P}\mathbf{u}, \quad \frac{\partial \mathbf{u}}{\partial z} = \mathbf{Q}\mathbf{s}, \tag{1}$$

where $\mathbf{s}$ and $\mathbf{u}$ vectors denote the Fourier coefficients of the transverse components of the respective electric and magnetic fields at a certain z location, and $\mathbf{P}$ and $\mathbf{Q}$ matrices denote how the derivatives of the field vary with respect to z [17].

These matrices are dependent on the permittivity and permeability distributions in each layer, represented by μ_r and ϵ_r, as well as on the wave number, polar angle of incidence, and azimuthal angle of incidence, as per Floquet's theorem. These variables are represented by $k_0 = \frac{2\pi}{\lambda}, \theta$, and φ [19], respectively.

These two differential equations can be combined to form a second-order spatial derivative equation:

$$\frac{\partial^2}{\partial z^2}\mathbf{s} = \mathbf{P}\mathbf{Q}\mathbf{s} = \mathbf{W}\boldsymbol{\lambda}\mathbf{W}^{-1}\mathbf{s}, \tag{2}$$

where $\mathbf{W}$ and $\boldsymbol{\lambda}$ are the eigenvectors and eigenvalues of the second-order differential equation operator, respectively. From these, a relation between the fields at position z, and the field at a relative distance d at position $z+d$, is given as follows [17]:

$$\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z+d} = \begin{bmatrix}\mathbf{W} & \mathbf{W}\\ -\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1} & \mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\end{bmatrix}\begin{bmatrix}e^{-d\sqrt{\boldsymbol{\lambda}}} & \mathbf{0}\\ \mathbf{0} & e^{d\sqrt{\boldsymbol{\lambda}}}\end{bmatrix} * \begin{bmatrix}\mathbf{W} & \mathbf{W}\\ -\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1} & \mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z}, \tag{3}$$

where $\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}$ can be interpreted as the eigenvectors of the magnetic field, and $e^{-d\sqrt{\boldsymbol{\lambda}}}$ and $e^{d\sqrt{\boldsymbol{\lambda}}}$ represent the propagation of the forward and backward modes, respectively.

The main issue this formulation seeks to address is that, while the forward modes tend to shrink as they are propagated forward, the backward modes instead grow as they are propagated forward. This discrepancy of magnitudes will lead to catastrophic cancellation if one were to integrate Eq. (1) directly, leading to invalid reflectance and transmittance results.

Instead, most implementations make use of the fact that $e^{-d\sqrt{\boldsymbol{\lambda}}}$ is guaranteed to be numerically stable, and $e^{d\sqrt{\boldsymbol{\lambda}}}$, while numerically unstable, is its inverse. This allows us to handle the "unstable" part of the equation implicitly via its stable inverse, thereby resolving this source of instability. As a result, the optical response of each slice can be fully characterized using the $\mathbf{W}$, $\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}$, and $e^{-d\sqrt{\boldsymbol{\lambda}}}$ matrices, in the form of a scattering matrix. From this, the optical response of the wider system can be determined using the Redheffer star product [17].

Here, we wish to emphasize that Eq. (3), although identical mathematically to literature, differs from notation, so as to highlight the presence of two matrix functions; more specifically, the matrix square root $\sqrt{\boldsymbol{\lambda}}$ and matrix exponential $e^{-d\sqrt{\boldsymbol{\lambda}}}$. While evaluating matrix functions of diagonal matrices, such as $\sqrt{\boldsymbol{\lambda}}$, is trivial, and thus ignored in literature, the presence of these matrix functions is essential to removing the eigendecomposition.

4. ALTERNATIVE FORMULATION TO COMPUTE THE SCATTERING MATRIX

We start by altering the found solution to the following:

$$\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z+d} = \begin{bmatrix}\mathbf{I} & \mathbf{I}\\ -\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\mathbf{W}^{-1} & \mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\mathbf{W}^{-1}\end{bmatrix} * \begin{bmatrix}\mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{W}e^{d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\end{bmatrix} * \begin{bmatrix}\mathbf{I} & \mathbf{I}\\ -\mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\mathbf{W}^{-1} & \mathbf{Q}\mathbf{W}\sqrt{\boldsymbol{\lambda}}^{-1}\mathbf{W}^{-1}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z}. \tag{4}$$

It is important to note that this formulation shares the desired properties of Eq. (3), especially regarding what is needed to ensure stability: $\mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}$ and $\mathbf{W}e^{d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}$ are mutual inverses. Furthermore, they are similar to $e^{-d\sqrt{\boldsymbol{\lambda}}}$ and $e^{d\sqrt{\boldsymbol{\lambda}}}$, respectively, and inherit their stability properties as a result.

Secondly, since matrix functions are commutative with any similarity transform [20], $\mathbf{W}$ and $\mathbf{W}^{-1}$ can be placed inside the matrix functions. This in turn allows us to substitute all implicit eigendecompositions with $\mathbf{P}\mathbf{Q}$ as per Eq. (2), giving us the following alternate formulation [21, 22]:

$$\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z+d} = \begin{bmatrix}\mathbf{I} & \mathbf{I}\\ -\mathbf{Q}\sqrt{\mathbf{P}\mathbf{Q}}^{-1} & \mathbf{Q}\sqrt{\mathbf{P}\mathbf{Q}}^{-1}\end{bmatrix}\begin{bmatrix}e^{-d\sqrt{\mathbf{P}\mathbf{Q}}} & 0\\ 0 & e^{d\sqrt{\mathbf{P}\mathbf{Q}}}\end{bmatrix} * \begin{bmatrix}\mathbf{I} & \mathbf{I}\\ -\mathbf{Q}\sqrt{\mathbf{P}\mathbf{Q}}^{-1} & \mathbf{Q}\sqrt{\mathbf{P}\mathbf{Q}}^{-1}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{s}\\ \mathbf{u}\end{bmatrix}_{z}. \tag{5}$$

Because this formulation preserves the structure previously found in Eq. (3), the blocks found in this formulation can be used in lieu of the previous blocks, yielding a completely identical scattering matrix, when making use of gap matrices [17]. For more details, please refer to Appendix A.

4.1. Matrix Functions

While this new formulation no longer requires the evaluation of an eigendecomposition, it does introduce new difficulties in calculation elsewhere. More specifically, the evaluation of the matrix square root and a matrix exponential are no longer applied to diagonal matrices, which complicates their evaluation.

For such cases, other algorithms for the evaluation of the matrix square root [20, 23–26, 30] and the matrix exponential [20, 27–29] have been extensively studied in the literature. The need to evaluate these matrix functions tends to arise wherever linear algebra is used, e.g., in areas such as statistics [31] and machine learning [32]. In the case of the matrix square root, its evaluation can be used as an intermediate step for finding the matrix logarithm [33], the polar decomposition [34, 35], and by extension the singular value decomposition (SVD) [36]. In the case of the matrix exponential, it is used to find the solution for time-integration problems [37] and Markov chains [38].

Not all algorithms in literature to evaluate these matrix functions are readily evaluated in parallel and differentiable, however, so a careful choice must be made for both functions. Our choice of implementations of the matrix functions are composed of basal matrix operations only, from the principle that these operations ought to inherit the parallel evaluation and differentiability properties, as has been previously observed of other such functions. Full implementations are shown below.

4.2. Matrix Square Root

The matrix square root is calculated with the following algorithm:

Algorithm 1 Calculating matrix square root

$\mathbf{Y} \leftarrow \mathbf{A}$
$\mathbf{Z} \leftarrow \mathbf{I}$
$\mathbf{X} \leftarrow \mathbf{ZY}$
while $|\mathbf{I} - \mathbf{X}| \geq \varepsilon$ **do**
  $\mathbf{P} \leftarrow \binom{2m+1}{1}\mathbf{I} + \binom{2m+1}{3}\mathbf{X}$
  $\mathbf{Q} \leftarrow \binom{2m+1}{0}\mathbf{I} + \binom{2m+1}{2}\mathbf{X}$
  $\mathbf{X}' \leftarrow \mathbf{X}$
  for $n \leftarrow 2$ to m **do**
    $\mathbf{X}' \leftarrow \mathbf{X}'\mathbf{X}$
    $\mathbf{P} \leftarrow \mathbf{P} + \binom{2m+1}{2n+1}\mathbf{X}'$
    $\mathbf{Q} \leftarrow \mathbf{Q} + \binom{2m+1}{2n}\mathbf{X}'$
  end for
  $\mathbf{F} \leftarrow \mathbf{Q}^{-1}\mathbf{P}$
  $\mathbf{Y} \leftarrow \mathbf{YF}$
  $\mathbf{Z} \leftarrow \mathbf{FZ}$
  $\mathbf{X} \leftarrow \mathbf{ZY}$
end while
$\sqrt{\mathbf{A}} \leftarrow \mathbf{Y}$

The derivation of this algorithm is based on an iterative method for a closely related function, the matrix sign, which in turn is based on iterative methods to find the scalar sign [23, 25]:

$$z_{i+1} = \frac{(1+z_i)^n - (1-z_i)^n}{(1+z_i)^n + (1-z_i)^n},$$
$$\lim_{i\to\infty} z_i = \operatorname{sgn}(z_0) = \begin{cases} -1, & \Re(z_0) < 0 \\ 1, & \Re(z_0) > 0 \end{cases}. \tag{6}$$

When n is taken as an integer, we can apply the binomial theorem to $(1+z_i)^n$ and $(1-z_i)^n$. In such a case, we see that the even power terms of z_i cancel out in the numerator, and the odd power terms of z_i cancel out in the denominator. When we refactor the sum to only make use of the non-zero terms, we get the following:

$$\begin{aligned} \frac{(1+z_i)^n - (1-z_i)^n}{(1+z_i)^n + (1-z_i)^n} &= \frac{\sum_{k=0}^{n}\binom{n}{k}(z_i^k - (-z_i)^k)}{\sum_{k=0}^{n}\binom{n}{k}(z_i^k + (-z_i)^k)} \\ &= \frac{\sum_{k'=0}^{\lfloor\frac{n-1}{2}\rfloor}\binom{n}{2k'+1}z_i^{2k'+1}}{\sum_{k'=0}^{\lfloor\frac{n}{2}\rfloor}\binom{n}{2k'}z_i^{2k'}}. \end{aligned} \tag{7}$$

Here, we force n to be odd by redefining it as $n = 2m+1$, where m is a positive integer. In such a case, we know that $\lfloor\frac{n}{2}\rfloor = \lfloor\frac{n-1}{2}\rfloor = m$, which gives us the final iteration we shall be using:

$$z_{i+1} = z_i \frac{\sum_{k'=0}^{m}\binom{2m+1}{2k'+1}(z_i^{2k'})}{\sum_{k'=0}^{m}\binom{2m+1}{2k'}(z_i^{2k'})}. \tag{8}$$

This iteration can also be applied to square matrices, which effectively determines the sign of each individual eigenvalue of the matrix:

$$\begin{aligned} \operatorname{msgn}(\mathbf{A}) &= \mathbf{W}[\operatorname{msgn}(\boldsymbol{\lambda})]\mathbf{W}^{-1} \\ &= \mathbf{W}\begin{bmatrix} \operatorname{sgn}(\lambda_1) & 0 & \dots & 0 \\ 0 & \operatorname{sgn}(\lambda_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \operatorname{sgn}(\lambda_n) \end{bmatrix}\mathbf{W}^{-1}. \end{aligned} \tag{9}$$

The iteration, as given by Eq. (8), when adapted to matrices, reads as follows:

$$\begin{aligned} \mathbf{A}_{i+1} &= \mathbf{A}_i f(\mathbf{A}_i^2), \\ f(\mathbf{A}^2) &= \left(\sum_{k'=0}^{m}\binom{2m+1}{2k'+1}\mathbf{A}^{2n}\right) \\ &\quad * \left(\sum_{k'=0}^{m}\binom{2m+1}{2k'}\mathbf{A}^{2n}\right)^{-1}. \end{aligned} \tag{10}$$

The way in which a matrix square root can be calculated using the matrix sign is by applying the matrix sign to the following 2-by-2 block-skew-diagonal matrix:

$$\operatorname{msgn}\left(\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{0} & \sqrt{\mathbf{A}} \\ \sqrt{\mathbf{A}}^{-1} & \mathbf{0} \end{bmatrix}. \tag{11}$$

In [34], it was proven that iterations with the form $\mathbf{A}_{i+1} = \mathbf{A}_i f(\mathbf{A}_i^2)$ can be efficiently applied to solve iterations with skew-diagonal form as follows:

$$\begin{aligned}\begin{bmatrix}\mathbf{0} & \mathbf{Y}\\ \mathbf{Z} & \mathbf{0}\end{bmatrix} f\left(\begin{bmatrix}\mathbf{0} & \mathbf{Y}\\ \mathbf{Z} & \mathbf{0}\end{bmatrix}^2\right) &= \begin{bmatrix}\mathbf{0} & \mathbf{Y}\\ \mathbf{Z} & \mathbf{0}\end{bmatrix}\begin{bmatrix}f(\mathbf{YZ}) & \mathbf{0}\\ \mathbf{0} & f(\mathbf{ZY})\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{0} & \mathbf{Y}f(\mathbf{ZY})\\ \mathbf{Z}f(\mathbf{YZ}) & \mathbf{0}\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{0} & \mathbf{Y}f(\mathbf{ZY})\\ f(\mathbf{ZY})\mathbf{Z} & \mathbf{0}\end{bmatrix}, \end{aligned} \tag{12}$$

where the last step is an application of the push-through identity [34], so as to avoid having to calculate $f(\mathbf{YZ})$ alongside $f(\mathbf{ZY})$. Because both the input and the output are skew-block matrices, we can instead formulate the iteration in terms of these blocks:

$$\begin{aligned}\mathbf{Y}_{i+1} &= \mathbf{Y}_i f(\mathbf{Z}_i\mathbf{Y}_i),\\ \mathbf{Z}_{i+1} &= f(\mathbf{Z}_i\mathbf{Y}_i)\mathbf{Z}_i,\end{aligned} \tag{13}$$

where

$$\begin{aligned}f(\mathbf{Z}_i\mathbf{Y}_i) &= \left(\sum_{k'=0}^{m}\binom{2m+1}{2k'+1}(\mathbf{Z}_i\mathbf{Y}_i)^{k'}\right)\\ &\quad * \left(\sum_{k'=0}^{m}\binom{2m+1}{2k'}(\mathbf{Z}_i\mathbf{Y}_i)^{k'}\right)^{-1}.\end{aligned} \tag{14}$$

Since $f(\mathbf{ZY})$ is shared in this coupled iteration, it only needs to be calculated once. Furthermore, since the numerator and the denominator of $f(\mathbf{ZY})$ are polynomials sharing the same terms, we only need m matrix multiplications and one matrix solve to determine $f(\mathbf{ZY})$, and two matrix multiplications to find $\mathbf{Y}_{i+1}$ and $\mathbf{Z}_{i+1}$. In aggregate, it requires $m+2$ matrix multiplications and 1 matrix solve per iteration.

4.3. Convergence --- Speed

This iteration has a convergence rate of $2m+1$ [25, 34]; that is, so long as $|\mathbf{I} - \mathbf{Z}_i\mathbf{Y}_i| < 1$, then

$$|\mathbf{I} - \mathbf{Z}_{i+1}\mathbf{Y}_{i+1}| \leq |\mathbf{I} - \mathbf{Z}_i\mathbf{Y}_i|^{2m+1}. \tag{15}$$

Convergence is determined by $\mathbf{Y}_i\mathbf{Z}_i = \mathbf{I}$; in such a case, we see that

$$\begin{aligned}f(\mathbf{I}) &= \left(\sum_{n=0}^{m}\binom{2m+1}{2n+1}\mathbf{I}\right)\left(\sum_{n=0}^{m}\binom{2m+1}{2n}\mathbf{I}\right)^{-1}\\ &= (2^{2m}\mathbf{I})(2^{2m}\mathbf{I})^{-1} = \mathbf{I},\end{aligned} \tag{16}$$

and $\mathbf{Y}_{i+1} = \mathbf{Y}_i$. Since $\mathbf{Z}_i\mathbf{Y}_i$ is evaluated every iteration, convergence can be tested for each iteration with minimal additional computational cost. As in [25], theorem 3.4, it is made clear that all these iterations share convergence behavior with one another, with the number of iterations required for convergence scaling inversely with $\log(2m+1)$. Based on this, we determined that the optimal performance was achieved when $m = 3$. For further details, please refer to Appendix B.

4.4. Convergence --- Branch Cut

Because $\mathbf{Y}_0$ and $\mathbf{Z}_0$ commute with one another, all iterations commute with one another, as well as $\mathbf{A}$, save for rounding error. As a result, the convergence behavior of this iterative method depends solely on the spectrum of $\mathbf{PQ}$. More specifically, the iteration will have converged when all the eigenvalues of $\mathbf{Y}_i$ have converged to those of $\sqrt{\mathbf{PQ}}$, so the overall convergence is dominated by the worst-converging eigenvalue.

In order to test this, we applied this algorithm to the scalar equivalent of algorithm 1. In the case where $m = 1$, this simplifies to the following iteration:

$$y_{i+1} = y_i\frac{3+y_iz_i}{3y_iz_i+1}, \quad z_{i+1} = z_i\frac{3+y_iz_i}{3y_iz_i+1}, \tag{17}$$

for various complex values of y_0, and $z_0 = 1$. Convergence was determined as the number of iterations needed to achieve $|y_iz_i - 1| < 10^{-12}$. The results of this are plotted in Figure 2.

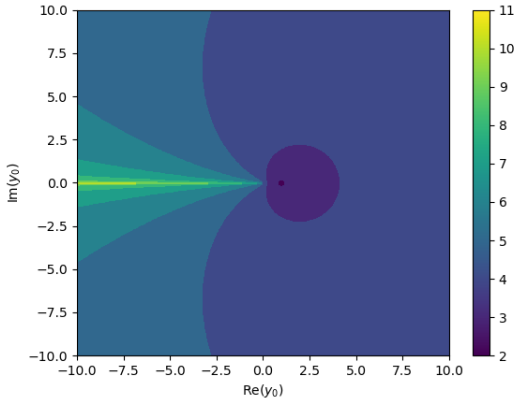

FIGURE 2. Iterations needed for Eq. (17) to converge, for various values of y_0 and $z_0 = 1$.

We determined that overall convergence is poor for eigenvalues which are very large, very small, and those which are placed very close to the branch cut, with the method failing to converge if there are eigenvalues on the branch cut. Unfortunately, the conventional branch cut is placed on the negative real axis, which is populated with propagating modes in the case of lossless media. As such, a new branch cut must be chosen, analogous to enforcing sign convention.

Here, the choice was made to place the branch cut in the third quadrant of the complex plane. From our experience, we could not find any case where eigenvalues were found in the third quadrant of the complex plane. This was not true for any other quadrant [39]. While we still lack understanding as to why the third quadrant is devoid of eigenvalues, it does mean that the branch cut can be safely rotated into the third quadrant of the complex plane without altering the found square root, thereby maintaining the stability granted by Eq. (5). This is done by redefining the square root as follows:

$$\sqrt{\mathbf{PQ}} := e^{-i\phi/2}\sqrt{e^{i\phi}\mathbf{PQ}}. \tag{18}$$

Here, we chose $\phi = -\frac{\pi}{4}$, so that the branch cut extends diagonally through the third quadrant. This way, the argument of

any eigenvalue is guaranteed to differ from the argument of the branch cut by at least $\pm\frac{\pi}{4}$. This, combined with a normalization step, means that the number of iterations required until convergence scales logarithmically with the condition number of **PQ** only.

As per the equivalence principle, this convergence behavior also applies to different choices of m [25]; more details can be found in Appendix B.

5. MATRIX EXPONENTIAL

In a similar vein, we also had to inspect how to calculate the matrix exponential. This method has similarly received a lot of attention for how to calculate it in a timely manner, while still being differentiable.

Here, two methods of calculating the matrix exponential stood out. The first is the scaling-and-squaring method, which uses the following identity:

$$e^{\mathbf{A}} = \left(e^{\mathbf{A}/2n}\right)^{2n}. \tag{19}$$

With this identity, one can instead calculate $e^{\mathbf{A}/2n}$, which is typically easier to calculate and better posed, and apply repeated self-multiplication to arrive at the desired answer.

The second method is calculating $e^{\mathbf{A}}$ through some kind of polynomial, or polynomial fraction, of **A**. These include cases such as the Padé approximant [40]:

$$e^{\mathbf{A}} \approx \left(\sum_{i=0}^{N} c_n \mathbf{A}^n\right)\left(\sum_{i=0}^{N} c_n (-\mathbf{A})^n\right)^{-1}. \tag{20}$$

Since the default matrix exponential function of TensorFlow makes use of these two algorithms, this implementation was sufficient regarding computational performance and differentiability.

6. VALIDATION OF RCWA IMPLEMENTATIONS

We benchmarked this new approach against the previous implementation. This was done by having two implementations of how the S-matrix is determined from the terms found in Eq. (1). Here, we outline the pseudocode which outlines how the scattering matrix is derived from the depth and the permittivity and permeability distributions of each layer:

Algorithm 2 scattering matrix using eigendecomposition

1: $\mathbf{W}, \boldsymbol{\lambda} \leftarrow \text{eig}(\mathbf{P} * \mathbf{Q})$
2: $\sqrt{\boldsymbol{\lambda}} \leftarrow \text{sqrt}(\boldsymbol{\lambda})$
3: $\mathbf{V} \leftarrow \mathbf{Q} * \mathbf{W} * \text{inv}(\sqrt{\boldsymbol{\lambda}})$
4: $\mathbf{X} \leftarrow \exp(-d * \sqrt{\boldsymbol{\lambda}})$
5: $\mathbf{S} \leftarrow \text{get\_s}(\mathbf{W}, \mathbf{V}, \mathbf{X})$

Algorithm 3 scattering matrix using matrix functions

1: $\mathbf{W} \leftarrow \mathbf{I}$
2: $\sqrt{\mathbf{PQ}} \leftarrow \text{sqrtm}(\mathbf{P} * \mathbf{Q})$
3: $\mathbf{V} \leftarrow \mathbf{Q} * \text{inv}(\sqrt{\mathbf{PQ}})$
4: $\mathbf{X} \leftarrow \text{expm}(-d * \sqrt{\mathbf{PQ}})$
5: $\mathbf{S} \leftarrow \text{get\_s}(\mathbf{W}, \mathbf{V}, \mathbf{X})$

Again, we wish to emphasize here that the intermediate states of **W**, **V**, **X** differ between implementations, but yield the same analytic result, except for rounding errors. Furthermore, step 5 does not differ between algorithms.

These two different algorithms are embedded in the same overall RCWA algorithm. Additionally, the whole model can also be configured to evaluate all operations with IEEE single precision and double precision floating point format. As such, a total of four configurations of RCWA are tested. This algorithm was evaluated on an NVIDIA A-100 TPU tensor-core GPU. Tensor cores are GPU architectures, which achieve a much higher number of floating point operations in exchange for numerical accuracy. As a result, the A-100 boasts a larger number of FLOPS when evaluating the algorithm with single-precision operations when compared to double-precision operations. This serves as our gauge for parallelizability. Based on configurations of the GPU, the difference in computational power between these configurations can be a factor of 8, 16, or 32, depending on the internal configuration of TensorFlow.

For the purpose of benchmarking, steps 1–4 of algorithms 1 and 2 were timed. Additionally, steps one through five of algorithms 1 and 2 were also timed. From this, the computation time of step 5 can also be derived. **P**, **Q** were determined from the relative permittivity and permeability using the inverse rule [41]. The S-matrix was evaluated using gap matrices, with the optical properties of vacuum [17]. The global S-matrix is obtained by concatenating the S-matrix of the layer with the S-matrix of both the substrate and superstrate using the Redheffer star product. The reflection and transmission of the fields can be read from the entries of this S-matrix, which then can be used to evaluate the reflected and transmitted diffraction intensities.

The calculated diffraction intensities are compared to reference diffraction intensities. We know that this data was produced also by an eigendecomposition RCWA implementation, with various configurations of diffraction orders considered, computed with double floating point precision, and that **P**, **Q** were determined from the relative permittivity and permeability using the inverse rule [41]. The discrepancy between the two predicted and reference diffraction intensities is quantified using a sum-square loss function:

$$\mathcal{L} = \sum_i |R_{pred,i} - R_{ref,i}|^2 + \sum_i |T_{pred,i} - T_{ref,i}|^2, \tag{21}$$

where R and T are the far-field intensities of each reflected and transmitted diffraction order, respectively.

Since the loss function compares the predictions of two mathematical models (one of which is used as the reference), what the loss value found can wholly be attributed to differences in implementations.

We consider a two-dimensional periodic grating consisting of rectangular pillars, aligned with the orthogonal grid. The grating is with a duty cycle of 50% in both directions and the pillar in each cycle is with pitches of 600 nm and 500 nm along the x and y directions, respectively.

The periodic grating consists of a single layer of photo-resist with a permittivity of $2.25+0i$ relative to vacuum, and the depth of this layer (the height of the pillars) is 100 nm. This grating

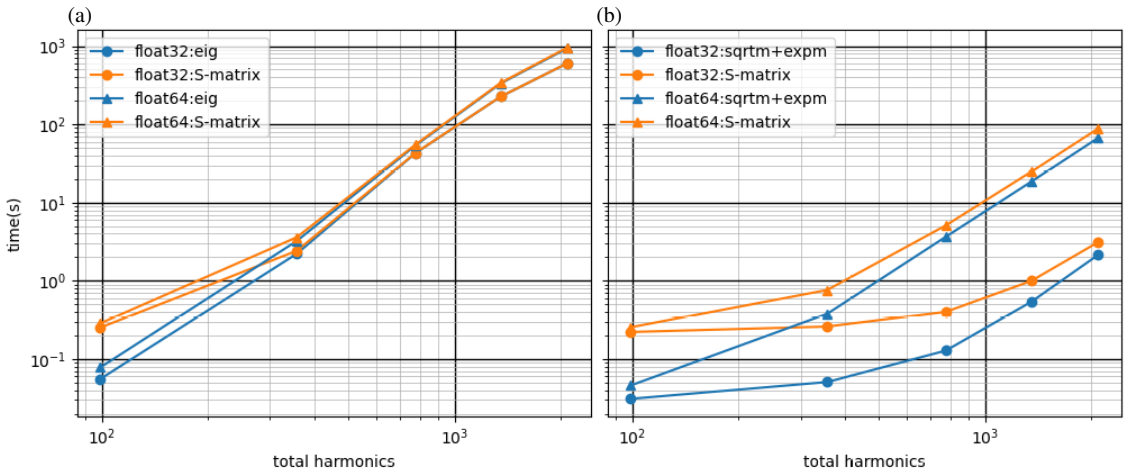

FIGURE 3. Comparison between computation times for computing the S-matrix (orange) using (a) the eigendecomposition and (b) the matrix square root and matrix exponential, for various numbers of harmonics considered and two precision formats. Also included is the time spent on essential intermediate steps (blue) specifically alone on (a) the eigendecomposition and (b) the matrix square root and matrix exponential. Both axes are scaled logarithmically.

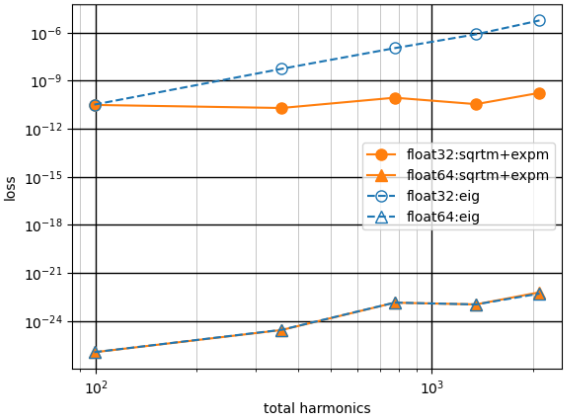

FIGURE 4. Plot highlighting found sum-square error between reflectance and transmittance values predicted by the model and the reference data for various numbers of harmonics considered. The predictions were obtained through computing the eigendecomposition (blue) and matrix functions (orange) with single (circle) and double (triangle) precision floating point format, respectively. Both axes are scaled logarithmically. Loss shown is the sum of the loss value evaluated for normal and conical incidence.

is surrounded by a uniform superstrate consisting of vacuum, as well as a uniform substrate of silicon with a permittivity of $16 + 0i$. Throughout the entire system, we assumed a unit permeability.

The system is illuminated from the superstrate side with p-polarized light at a wavelength of 425 nm. Two illumination configurations are used for diversity in the loss function: one with polar incidence ($\theta = 0°, \varphi = 0°$) and the other with conical incidence ($\theta = 30°, \varphi = 30°$).

We benchmarked the speedup and accuracy of our method against eigendecomposition method for various number of diffraction orders, listed in detail in Table 1.

For these various permutations, we time the step for computing the S-matrix and the essential intermediate steps and evaluate the loss value between the predicted and reference diffraction orders using the loss function as defined in Eq. (21). The speedup and accuracy benchmark results are illustrated in Figure 3 and Figure 4, respectively.

TABLE 1. Table outlining the number of diffraction orders (or harmonics) for different data entries.

Entry	Diffraction orders x	Diffraction orders y	Total harmonics
1	$-5, \ldots, +5$	$-4, \ldots, +4$	99
2	$-10, \ldots, +10$	$-8, \ldots, +8$	357
3	$-15, \ldots, +15$	$-12, \ldots, +12$	775
4	$-20, \ldots, +20$	$-16, \ldots, +16$	1353
5	$-25, \ldots, +25$	$-20, \ldots, +20$	2091

7. NUMERICAL RESULTS

7.1. Speedup

By comparing the computation time of the S-matrix using the matrix square root and exponential, we observe a significant speedup when switching from double precision to single precision, related to the aforementioned performance difference due to GPU architecture. At the highest number of harmonics, we observed a speedup factor of approximately 31.3 for the matrix square root and matrix exponential, and a speedup of 28.2 for the whole S-matrix. As the maximum possible speedup achievable is a factor 32, this suggests this method is able to benefit from further acceleration due to parallelization.

Additionally, we observe that the speedup achieved is minimal with a small number of harmonics, and increases with number of harmonics considered. This behavior is consistent with Gustafson's law.

With this difference in parallel performance, we see a speedup relative to the eigendecomposition method which widens both with problem size and parallel computing resources available. We observe a difference in time to compute

the S-matrix of 1 order of magnitude when using double-precision arithmetic, and of 2 orders of magnitude using single-precision arithmetic. We expect this performance gap to widen going into the future, as cutting-edge computing achieve further speedup through ever-increasing parallelization.

The slope on this logarithmic graph indicates that all methods have an approximately cubic computational complexity, which is in accordance with the computational complexity of linear algebra operations. While methods exist to reduce this computational complexity, these are typically not used in CUDA architecture, as this results in higher numerical error and poorer parallel execution.

7.2. Accuracy

Here, we record the loss between the diffraction pattern intensity between various methods, and the result provided externally. This loss value is the sum for the loss value with normal incidence, and the loss value with conical incidence. Since for this case the structure parameters are unperturbed, and the discretizations are identical leading to identical truncation error, the loss can be wholly attributed to relative rounding errors arising from differences in implementation, and thus serves to highlight the amount of numerical error inherent in each variation.

The loss values found for the square root method and eigendecomposition method for double precision are effectively indistinguishable, and are both smaller than the relative error which can be represented using double precision floating point arithmetic. As a result, we conclude that the new method has effectively negligible impact on numerical accuracy when evaluated using double precision.

When we instead evaluate the loss function by comparing the single-precision models to the double-precision validation data, the found loss values are much larger. This can mainly be attributed to the fact that single-precision floating point numbers have larger relative rounding error by design. The rounding error does seem to diverge between the square root method and the eigendecomposition method for higher number of harmonics, which we mainly attribute this to poor numerical accuracy of diagonal matrices when used with tensor cores [42].

7.3. Parameter Retrieval

To demonstrate the ability of the RCWA method to perform parameter retrieval, the material parameters of the permittivity distribution were perturbed. The perturbed variables are outlined in Table 2. All other parameters, which were not mentioned, were not perturbed.

With these perturbed parameters, we evaluated the model with the number of diffraction orders of entry 2 in Table 1 for normal and conical incidence, and evaluated the loss function with respect to entry 2 in the dataset with normal and conical incidence, respectively. By evaluating the gradient of the loss function with respect to the perturbed parameters, we retrieved the "ground truth" parameters using the Adam optimizer [43], with a training rate of 0.05. The parameter retrieval was done by replicating the configuration of, and comparing with the dataset of, entry 2 of the validation dataset, as given by Table 1. The diffraction pattern found with normal incidence, and the diffraction pattern with conical incidence, together form a dataset with two entries. The perturbed parameters are retrieved using this dataset with minibatching and no shuffling. This training was done with all four previously described configurations of RCWA. The loss function, and the errors in the retrieved parameters, can be seen per epoch in Figure 5.

TABLE 2. Table outlining the "ground truth" parameters and the perturbed values.

Variable	Ground truth	Perturbed value
Re (ε)	2.25	2.15
Im (ε)	0.00	0.1
h	100 nm	125 nm
x	300 nm	325 nm
y	250 nm	225 nm

For all of these graphs, the curve for eigendecomposition with double precision was omitted, as these curves were indistinguishable from the ones for the matrix square root and exponential with double precision.

We observe effectively mutual agreement on the training behavior of all configurations, up until certain loss values, at which point the single-precision methods stagnate. This is approximately 10^{-8} for the eigendecomposition method, and 10^{-11} for the square root method. Considering these values are the same as the rounding error for said configurations in Figure 4, it suggests that the loss (and thus the gradient vector) becomes dominated by rounding errors, inhibiting further gradient descent.

The double-precision variations remain in agreement until the end of the training period, and instead both cease convergence once the relative error becomes approximately 10^{-16}, which is the smaller than the relative difference which can be represented with double precision arithmetic.

8. DISCUSSION

In this study, a comparison is made between two implementations of RCWA on GPU. Because of this, any questions of the performance of RCWA, such as its convergence behavior or accuracy relative to other methods, are intentionally left unaddressed. Similarly, no work has been done to inspect the impact of other sources of error on the retrieval of parameters such as noise and non-uniqueness of the solutions to the inverse scattering problems.

There has been much literature on the concern that the gradient of an eigendecomposition is increasingly ill-posed when two eigenvalues are nearly degenerate [5, 13, 44]. However, we did not observe this to be the case in our work. We do expect degenerate eigenvalues due to symmetries in the simulated setup. However, since these symmetries cannot be broken, this did not lead to nearly-degenerate eigenmodes, thereby potentially circumventing this issue. It is also possible that TensorFlow has updated the eigendecomposition method for improved back-

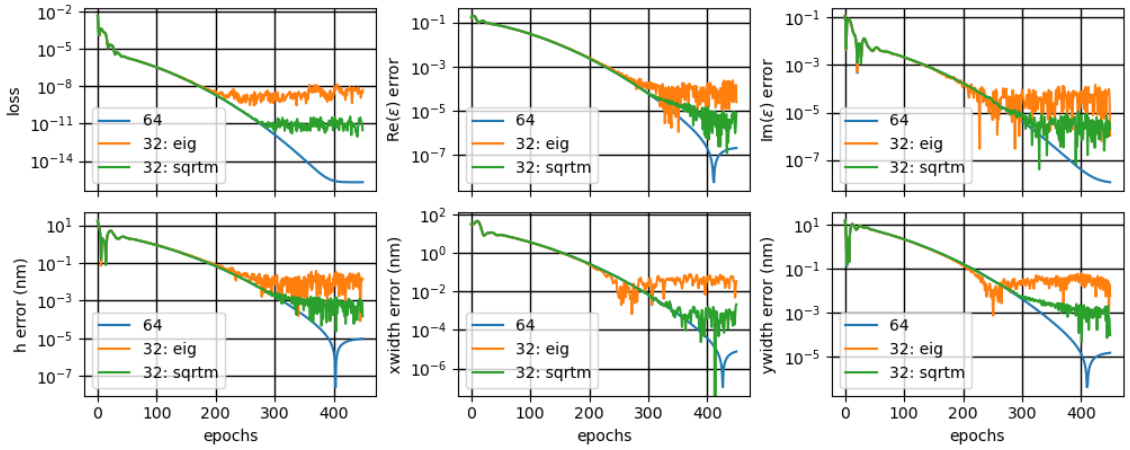

FIGURE 5. Comparison between parameter retrieval results obtained using eigendecomposition (orange) and matrix functions in both single (green) and double (blue) precision. Top row from left to right: L2 error of the prediction with respect to the reference (the loss), and L1 error of the real and imaginary parts of the permittivity. Bottom from left to right: L1 error of the perturbed shape parameters.

ward stability, or otherwise does not use the explicit formulation for the eigendecomposition gradient. As such, we were not able to demonstrate that the new algorithm is more resistant to degenerate eigenmodes than the eigendecomposition.

Special attention must be paid to parameter retrieval as the imaginary part of relative permittivities can turn negative during the training, thereby introducing gain to the layer and making the shift of the branch cut potentially invalid. Failure to determine the matrix square root in such a case has been observed in our work. However, such failures can easily be prevented by forcing the permittivies to have non-negative imaginary parts.

It is also theoretically possible that the matrix square root may fail due to one eigenvalue existing at the origin in the complex plane, rendering the matrix **PQ** singular. In such a case, however, RCWA implementations using the eigendecomposition would have failed as well, since this would entail a divide-by-zero error when finding the term **V** in Eq. (3).

Our method is implemented on TensorFlow which uses CUDA for GPU computation. However, the computation was not solely executed on GPU. More precisely, instructions are first evaluated on CPU, which then sends commands (as well as necessary data) to the GPU. Once the processing is finished, the information is then sent back from GPU to CPU, in preparation for future instructions. For completeness, this process can potentially lead to a new bottleneck, but we did not observe this to be the case.

In the optimization, the units of the parameters to be retrieved must be carefully chosen, as the optimization is not scale invariant. Doubling all the spatial sizes, for example, will cause the gradient of these variables with respect to the loss function to be halved, resulting in these variables converging at only a quarter of the rate. As such, an arbitrary choice of units can have significant impact on the convergence behavior. A normalization strategy will be needed in the future when optimizing parameters with various orders of magnitudes and physical origins. This effect will likewise need to be accounted for if the refractive index n and extinction coefficients k are to be retrieved, rather than the complex permittivity.

In our computation time benchmark, we also observed a computation time decrease for the eigendecomposition, when switching from double precision to single precision. We cannot explain this increase in computational performance. However, we do not think this to be a significant pointer toward worse performance.

In the benchmarking calculations, it is assumed that the increase in computational power from the GPU switching from double-precision to single-precision can solely be attributed to an increase in effective parallel process count. However, this does not need to be the case, as an increase in the floating point accuracy might also lead to an increase in effective clock speed. This might explain why the eigendecomposition time also decreased by lowering the numerical precision. However, such logic is circular.

We wish to make clear that the timing used only concerns the evaluation of the S-matrix. As a result, the computation times (and derived speedup) are only representative for determining the S-matrix, rather than the computation time of RCWA as a whole. The remainder of the RCWA is dominated by the Redheffer star product, which in turn can be evaluated using matrix additions, multiplications, and solves [17, 45]. As a result, we did not observe this to be a new bottleneck in the parameter retrieval process.

Further research is needed on whether this scheme can be implemented with other analytical slicing methods, such as the enhanced transmittance matrix method [3]. We do expect so, considering this method also requires evaluating a similarity transform of structure as in Eq. (5), and then evaluates the reflectance and transmittance from there using only matrix multiplications and inverses.

There are three approaches worth mentioning, which were ultimately not used, to evaluate the matrix square root. Firstly, since matrix functions can be defined with the use of the eigendecomposition, evaluating the matrix square root can be done trivially once the eigendecomposition is evaluated. This algorithm would defeat the purpose of not evaluating an eigendecomposition, however, so it was not chosen.

Another approach worth mentioning is the blocked Schur algorithm for the matrix square root [46]. However, this approach suffers from poor performance for dense non-triangular matrices, which we know is the case for the **PQ** matrix.

A third line of approach is alternate iterations which can be used to find the matrix square root, such as the Denman-Beavers Iteration [34]. This iterative method is overall easier to implement, and otherwise shares all desireable properties as the actual iterative method used. However, computational cost tends to be slightly higher, as this method has a convergence rate of 2, and requires evaluating two inverses per iteration instead of just one.

Finally, it is also possible to analytically find the derivative of the matrix square root, which ought to be more memory efficient than the current implementation; derivation of such a potential implementation can be found in [32].

9. CONCLUSION

In this paper, we derived and implemented a new RCWA algorithm which was able to be accelerated properly on GPU. Our method replaces the eigendecomposition with two matrix functions, i.e., matrix exponential and matrix square-root. These matrix functions can be evaluated using conventional linear algebra arithmetic, which can be readily accelerated on GPU.

We observed a significant speedup over two orders of magnitude in the most extreme case in evaluating the S-matrix by switching from eigendecomposition to this new implementation, suggesting that the previous bottleneck has been sufficiently addressed. We emphasize that this speedup is limited by the finite computation power of GPU in our setup, rather than the fundamental formulation. We expect that a more powerful GPU can lead to an even further speedup and a larger discrepancy in performance between eigendecomposition and this new implementation.

Further comparison between S-matrix evaluation using single precision and double precision shows a 28.6 times speedup, which indicates that this new implementation can fully exploit the potential of parallelization on GPU. This is because a GPU can only compute up to 32 times more floating point operations using single precision than double precision when evaluating arithmetic.

For double precision, we found negligible differences in numerical error between our method and the eigendecomposition method, while for single precision, we found significant differences in numerical error when comparing to double-precision methods, favouring our method. As a result, our method is numerically more stable than the eigendecomposition method in single precision. This feature is particularly useful when one wants to maximize the speedup while maintaining an acceptable level of numerical error.

ACKNOWLEDGEMENT

The research in this publication is part of the project 3D Nanoscale Imaging (3DNI) with project number P21-30 within the Perspective Research Program as financed by the Dutch Research Council (NWO).

We would like to acknowledge Mark van Kraaij for providing the reference data, which we used for validation and quantification of numerical error.

DISCLOSURES

The authors declare no conflicts of interest.

DATA AVAILABILITY

Data underlying the results presented in this paper are not publicly available at this time but may be obtained from the authors upon reasonable request.

APPENDIX A. S-MATRIX EQUIVALENCE

The S-matrix with gap layers is usually determined from $\mathbf{W}$, $\mathbf{V}$, $e^{-d\sqrt{\boldsymbol{\lambda}}}$ through the following equations:

$$\begin{aligned}
\mathbf{A} &= \mathbf{W}^{-1}\mathbf{W_0} + \mathbf{V}^{-1}\mathbf{V_0}, \\
\mathbf{B} &= \mathbf{W}^{-1}\mathbf{W_0} - \mathbf{V}^{-1}\mathbf{V_0}, \\
\mathbf{S}_{11} &= \mathbf{S}_{22} = (\mathbf{A} - e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B}\mathbf{A}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B})^{-1} \\
&\quad * (e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B}\mathbf{A}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{A} - \mathbf{B}), \\
\mathbf{S}_{12} &= \mathbf{S}_{21} = (\mathbf{A} - e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B}\mathbf{A}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B})^{-1} \\
&\quad * e^{-d\sqrt{\boldsymbol{\lambda}}}(\mathbf{A} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}),
\end{aligned} \tag{A1}$$

where $\mathbf{W_0}$ and $\mathbf{V_0}$ characterize the electric and magnetic fields, respectively, of the eigen-modes of the gap medium. While it is convention that the gap medium is taken to be free space [17], the following proof applies generally, so $\mathbf{W_0}$ and $\mathbf{V_0}$ are left undefined. If we make the substitutions:

$$\begin{aligned}
\mathbf{W} &\to \mathbf{W}\mathbf{W}^{-1}, \\
\mathbf{V} &\to \mathbf{V}\mathbf{W}^{-1}, \\
e^{-d\sqrt{\boldsymbol{\lambda}}} &\to \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1},
\end{aligned} \tag{A2}$$

then the resulting S-matrix terms solve to the following:

$$\begin{aligned}
\mathbf{A}' &= \mathbf{W}\mathbf{W}^{-1}\mathbf{W_0} + \mathbf{W}\mathbf{V}^{-1}\mathbf{V_0} = \mathbf{W}\mathbf{A}, \\
\mathbf{B}' &= \mathbf{W}\mathbf{W}^{-1}\mathbf{W_0} - \mathbf{W}\mathbf{V}^{-1}\mathbf{V_0} = \mathbf{W}\mathbf{B}, \\
\mathbf{S}'_{11} &= \mathbf{S}'_{22} = (\mathbf{W}\mathbf{A} - \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{W}\mathbf{B}\mathbf{A}^{-1} \\
&\quad * \mathbf{W}^{-1}\mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{W}\mathbf{B})^{-1} \\
&\quad * (\mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{W}\mathbf{B}\mathbf{A}^{-1}\mathbf{W}^{-1} \\
&\quad * \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{W}\mathbf{A} - \mathbf{W}\mathbf{B}),
\end{aligned} \tag{A3}$$

$$\begin{aligned}\mathbf{S}'_{12} = \mathbf{S}'_{21} &= (\mathbf{WA} - \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{WBA}^{-1} \\ &\quad * \mathbf{W}^{-1}\mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}\mathbf{WB})^{-1} \\ &\quad * \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}(\mathbf{WA} - \mathbf{WBA}^{-1}\mathbf{W}^{-1}\mathbf{WB}).\end{aligned}$$

After cancelling some terms, we see that the S-submatrices found solve to the following:

$$\begin{aligned}\mathbf{S}'_{11} = \mathbf{S}'_{22} &= (\mathbf{A} - e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{BA}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B})^{-1} \\ &\quad * (e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{BA}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{A} - \mathbf{B}), \\ \mathbf{S}'_{12} = \mathbf{S}'_{21} &= (\mathbf{A} - e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{BA}^{-1}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{B})^{-1} \\ &\quad * e^{-d\sqrt{\boldsymbol{\lambda}}}(\mathbf{A} - \mathbf{BA}^{-1}\mathbf{B}).\end{aligned} \tag{A4}$$

Considering these terms are identical as given by Eq. (A1), we prove that we can use these alternate terms as substitutes for $\mathbf{W}$, $\mathbf{V}$, and $e^{-d\sqrt{\boldsymbol{\lambda}}}$ to calculate the same S-matrix, save for negligible differences arising from rounding error:

$$\mathbf{S}(\mathbf{WW}^{-1}, \mathbf{VW}^{-1}, \mathbf{W}e^{-d\sqrt{\boldsymbol{\lambda}}}\mathbf{W}^{-1}) = \mathbf{S}(\mathbf{W}, \mathbf{V}, e^{-d\sqrt{\boldsymbol{\lambda}}}). \tag{A5}$$

APPENDIX B. CONVERGENCE PRINCIPLE

In [25], it was demonstrated that the iterations for the matrix sign can be used interchangeably. More specifically, two different sequences of steps, where each step has convergence rate of $k_i + m_i + 1$, are equivalent when:

$$\prod^{r}(m_r + k_r + 1) = \prod^{\tilde{r}}(m_{\tilde{r}} + k_{\tilde{r}} + 1) = \rho. \tag{B1}$$

Here, we use this concept of "equivalence" and extend it to real numbers, so as to compare convergence rates. Since the operations become idempotent when convergence is achieved, we also assert that any geometric multiplicity larger than ρ is also converged.

With this knowledge, we assert that there exists a finite ρ, for which any method will have converged. Based on this, any multiplicity of $\alpha\rho$, where $\alpha > 1$, will also achieve said convergence. Based on this, any one method with a convergence rate of $2m + 1$ will reach total multiplicity of at least ρ in

$$\left\lceil \frac{\log(\rho)}{\log(2m+1)} \right\rceil$$

iterations. Based on this, we loosely assert the following relation between the amount of iterations needed to convergence, and m, as follows:

$$\begin{aligned} n_{convergence} &= \left\lceil \frac{\log(\rho)}{\log(2m+1)} \right\rceil \\ &\approx \frac{\log(\rho)}{\log(2m+1)} \propto \frac{1}{\log(2m+1)}. \end{aligned} \tag{B2}$$

If we are interested in the relative computation time between choices of m, then ρ no longer influences the choice with this type of modeling. From this, we can calculate the relative cost by multiplying the relative number of iterations needed by the iteration cost, which is:

$$\text{cost} = 2 + m + \alpha. \tag{B3}$$

where α denotes the cost of the matrix inverse, relative to the cost of the matrix multiplication. Based on this, we formulate the following figure of merit, to describe the computational performance, as a function of α and m:

$$F(m, \alpha) = \frac{\log(2m+1)}{2 + m + \alpha}. \tag{B4}$$

Figure B1 denotes the relative performance (i.e., inverse of total computational cost) for various values of m as a function of α is given below.

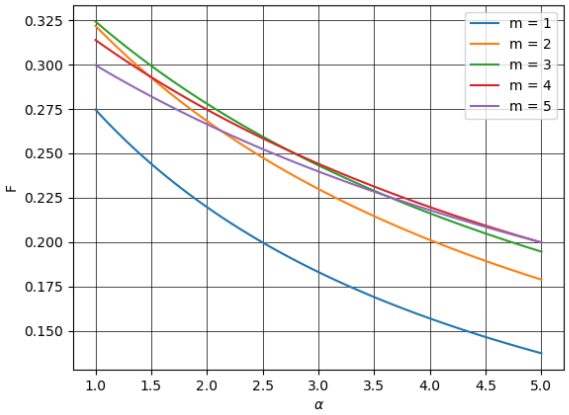

FIGURE B1. Plot showing the relative performance, for different choices of m, as a function of α.

Based on these results, a choice of $m = 3$ comes out as optimal if the cost of a matrix inverse is approximately twice the cost of a matrix-matrix product, which is what we observed.

REFERENCES

[1] Li, L., "Fourier modal method," *Gratings: Theory and Numeric Applications, Second Revisited Edition*, 13.1–13.40, E. Popov (ed.), Aix Marseille Université, CNRS, Centrale Marseille, Institut Fresnel, 2014.

[2] Li, L., "4. Mathematical reflections on the fourier modal method in grating theory," *Mathematical Modeling in Optical Science*, 111–139, Frontiers in Applied Mathematics, SIAM, 2001.

[3] Moharam, M. G., D. A. Pommet, E. B. Grann, and T. K. Gaylord, "Stable implementation of the rigorous coupled-wave analysis for surface-relief gratings: Enhanced transmittance matrix approach," *Journal of the Optical Society of America A*, Vol. 12, No. 5, 1077–1086, 1995.

[4] Semenikhin, I., M. Zanuccoli, M. Benzi, V. Vyurkov, E. Sangiorgi, and C. Fiegna, "Computational efficient RCWA method for simulation of thin film solar cells," *Optical and Quantum Electronics*, Vol. 44, No. 3, 149–154, 2012.

[5] Colburn, S. and A. Majumdar, "Inverse design and flexible parameterization of meta-optics using algorithmic differentiation," *Communications Physics*, Vol. 4, No. 1, 65, 2021.

[6] Sherwin, S., I. Cordova, R. Miyakawa, L. Waller, A. Neureuther, and P. Naulleau, "Quantitative phase retrieval for EUV photomasks," in *Extreme Ultraviolet (EUV) Lithography XI*, Vol. 11323, 281–290, 2020.

[7] Den Boef, A. J., "Optical wafer metrology sensors for process-robust CD and overlay control in semiconductor device manufacturing," *Surface Topography: Metrology and Properties*, Vol. 4, No. 2, 023001, 2016.

[8] Amari, S.-i., "Backpropagation and stochastic gradient descent method," *Neurocomputing*, Vol. 5, No. 4–5, 185–196, 1993.

[9] Krizhevsky, A., I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, Vol. 60, No. 6, 84–90, 2017.

[10] Evanschitzky, P., T. V. Nguyen, C. Schwemmer, and A. Erdmann, "Highly parallelized RCWA with optimized eigenvalue problem for efficient simulation of curvilinear mask structures," in *40th European Mask and Lithography Conference (EMLC 2025)*, Vol. 13787, 108–116, Dresden, Germany, 2025.

[11] Wei, X. and S. Chen, "Parallel computing for application in rigorous coupled-wave analysis," in *2012 Fifth International Symposium on Computational Intelligence and Design*, Vol. 2, 186–189, Hangzhou, China, 2012.

[12] Tong, J. and S. Chen, "Computation improvement for the rigorous coupled-wave analysis with GPU," in *2012 Fourth International Conference on Computational and Information Sciences*, 123–126, Chongqing, China, 2012.

[13] Giles, M., "An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation," Report No. 08/01, Oxford University Computing Laboratory, Oxford, England, 2008.

[14] Li, J., L. Shi, Y. Ma, Y. Ran, Y. Liu, and J. Wang, "Efficient and stable implementation of RCWA for ultrathin multilayer gratings: T-matrix approach without solving eigenvalues," *IEEE Antennas and Wireless Propagation Letters*, Vol. 20, No. 1, 83–87, 2021.

[15] Xu, J. and M. D. B. Charlton, "Highly efficient rigorous coupled-wave analysis implementation without eigensystem calculation," *IEEE Access*, Vol. 12, 127 966–127 975, 2024.

[16] Li, J., J.-B. Wang, Z. Sun, L.-H. Shi, Y. Ma, Q. Zhang, S.-C. Fu, Y.-C. Liu, and Y.-Z. Ran, "Efficient rigorous coupled-wave analysis without solving eigenvalues for analyzing one-dimensional ultrathin periodic structures," *IEEE Access*, Vol. 8, 198 131–198 138, 2020.

[17] Rumpf, R. C., "Improved formulation of scattering matrices for semi-analytical methods that is consistent with convention," *Progress In Electromagnetics Research B*, Vol. 35, 241–261, 2011.

[18] Gustafson, J. L., "Reevaluating Amdahl's law," *Communications of the ACM*, Vol. 31, No. 5, 532–533, 1988.

[19] Barkeshli, K. and S. Khorasani, "Periodic structures," in *Advanced Electromagnetics and Scattering Theory*, 329–335, K. Barkeshli and S. Khorasani (eds.), Springer, 2015.

[20] Higham, N. J., *Functions of Matrices: Theory and Computation*, SIAM, 2008.

[21] Li, J., L. Shi, Y. Ma, Z. Sun, Q. Zhang, S. Fu, Y. Liu, Y. Ran, and J. Wang, "Efficient implementation of rigorous coupled-wave analysis for analyzing binary gratings," *IEEE Antennas and Wireless Propagation Letters*, Vol. 19, No. 12, 2132–2135, 2020.

[22] Li, J., L. Shi, D. Ji, E. L. Tan, Q. Lei, Y. Ma, Y. Ran, Y. Liu, and J. Wang, "Efficient implementation of fourier modal method for 2-D periodic structures," *IEEE Microwave and Wireless Components Letters*, Vol. 32, No. 5, 375–378, 2022.

[23] Kenney, C. S. and A. J. Laub, "A hyperbolic tangent identity and the geometry of Padé sign function iterations," *Numerical Algorithms*, Vol. 7, No. 2, 111–128, 1994.

[24] Kenney, C. S. and A. J. Laub, "The matrix sign function," *IEEE Transactions on Automatic Control*, Vol. 40, No. 8, 1330–1348, 1995.

[25] Kenney, C. and A. J. Laub, "Rational iterative methods for the matrix sign function," *SIAM Journal on Matrix Analysis and Applications*, Vol. 12, No. 2, 273–291, 1991.

[26] Denman, E. D. and A. N. Beavers, Jr., "The matrix sign function and computations in systems," *Applied Mathematics and Computation*, Vol. 2, No. 1, 63–94, 1976.

[27] Moler, C. and C. V. Loan, "Nineteen dubious ways to compute the exponential of a matrix," *SIAM Review*, Vol. 20, No. 4, 801–836, 1978.

[28] Moler, C. and C. V. Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM Review*, Vol. 45, No. 1, 3–49, 2003.

[29] Higham, N. J., "The scaling and squaring method for the matrix exponential revisited," *SIAM Journal on Matrix Analysis and Applications*, Vol. 26, No. 4, 1179–1193, 2005.

[30] Higham, N. J., "Newton's method for the matrix square root," *Mathematics of Computation*, Vol. 46, No. 174, 537–549, 1986.

[31] Schott, J. R., *Matrix Analysis for Statistics*, John Wiley & Sons, 2016.

[32] Lin, T.-Y. and S. Maji, "Improved bilinear pooling with CNNs," *ArXiv Preprint ArXiv:1707.06772*, 2017.

[33] Al-Mohy, A. H. and N. J. Higham, "Improved inverse scaling and squaring algorithms for the matrix logarithm," *SIAM Journal on Scientific Computing*, Vol. 34, No. 4, C153–C169, 2012.

[34] Higham, N. J., D. S. Mackey, N. Mackey, and F. Tisseur, "Functions preserving matrix groups and iterations for the matrix square root," *SIAM Journal on Matrix Analysis and Applications*, Vol. 26, No. 3, 849–877, 2005.

[35] Higham, N. J. and P. Papadimitriou, "A parallel algorithm for computing the polar decomposition," *Parallel Computing*, Vol. 20, No. 8, 1161–1173, 1994.

[36] Nakatsukasa, Y. and N. J. Higham, "Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD," *SIAM Journal on Scientific Computing*, Vol. 35, No. 3, A1325–A1349, 2013.

[37] Hochbruck, M. and C. Lubich, "On krylov subspace approximations to the matrix exponential operator," *SIAM Journal on Numerical Analysis*, Vol. 34, No. 5, 1911–1925, 1997.

[38] Sidje, R. B. and W. J. Stewart, "A numerical study of large sparse matrix exponentials arising in markov chains," *Computational Statistics & Data Analysis*, Vol. 29, No. 3, 345–368, 1999.

[39] Hench, J. J. and Z. Strakoš, "The RCWA method — A case study with open questions and perspectives of algebraic computations," *Electronic Transactions on Numerical Analysis*, Vol. 31, 331–357, 2008.

[40] Al-Mohy, A. H. and N. J. Higham, "A new scaling and squaring algorithm for the matrix exponential," *SIAM Journal on Matrix Analysis and Applications*, Vol. 31, No. 3, 970–989, 2010.

[41] Li, L., "Use of Fourier series in the analysis of discontinuous periodic structures," *Journal of the Optical Society of America A*, Vol. 13, No. 9, 1870–1876, 1996.

[42] Markidis, S., S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & preci-

sion," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 522–531, Vancouver, BC, Canada, 2018.

[43] Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization," *ArXiv Preprint ArXiv:1412.6980*, 2014.

[44] Zhu, Z. and C. Zheng, "Differentiable scattering matrix for optimization of photonic structures," *Optics Express*, Vol. 28, No. 25, 37 773–37 787, 2020.

[45] Redheffer, R., "Difference and functional equations in transmission line theory," *Modern Mathematics for the Engineer Second*, Vol. 30, No. 3, 282–337, 1959.

[46] Deadman, E., N. J. Higham, and R. Ralha, "Blocked Schur algorithms for computing the matrix square root," in *International Workshop on Applied Parallel Computing*, 171–182, Springer, Berlin, Heidelberg, 2012.