Computer Physics Communications 282 (2023) 108552

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

TORCWA: GPU-accelerated Fourier modal method and gradient-based optimization for metasurface design ☆,☆☆

Changhyun Kim, Byoungho Lee *

Inter-University Semiconductor Research Center and School of Electrical and Computer Engineering, Seoul National University, Gwanak-Gu, Gwanak-ro 1, Seoul 08826, Republic of Korea

ARTICLE INFO

Article history:
Received 17 May 2022
Received in revised form 12 August 2022
Accepted 20 September 2022
Available online 27 September 2022

Keywords:
Rigorous coupled-wave analysis
Fourier modal method
Extended scattering matrix method
GPU-acceleration
Automatic differentiation
Optimization
Nanophotonics
Metasurfaces

ABSTRACT

TORCWA is an electromagnetic wave simulation and optimization tool utilizing rigorous coupled-wave analysis. One of the advantages of TORCWA is that it provides GPU-accelerated simulation. It shows a greatly accelerated simulation speed compared to when the same simulation is performed on a CPU-based. Although it has accelerated speed, the simulation results are almost identical to the commercialized electromagnetic wave simulations. The second advantage is that it provides GPU-accelerated gradient calculation for the simulation results with reverse-mode automatic differentiation of PyTorch version 1.10.1. In particular, the instability of gradient calculation of eigendecomposition is also improved. With this property, TORCWA can be utilized for the optimization of various nanophotonic devices. Here, we first introduce the formulation used in TORCWA, compare it with other commercial simulations, and show the computational performance in multiple environments. Then, the gradient calculation and optimization examples are shown. Thanks to accelerated computational performance and gradient calculation, TORCWA is a worthy program for designing and optimizing various nanophotonic devices.

Program summary
Program title: TORCWA
CPC Library link to program files: https://doi.org/10.17632/2dybvpk42g.1
Developer's repository link: https://github.com/kch3782/torcwa
Licensing provisions: LGPL v3
Programming language: Python 3.8
Nature of Problem: Time-harmonic electromagnetic fields.
Solution method: Fourier modal methods and extended scattering matrix method.
Additional comments including restrictions and unusual features: TORCWA utilizes PyTorch version 1.10.1 [1].

References

[1] PyTorch, https://pytorch.org/

1. Introduction

A metasurface is an array of artificially fabricated nanostructures smaller than the operation wavelength, which is a flat optical device with various functionalities [1,2]. The artificial nanostructures, called meta-atoms, can locally modulate the nature of light with subwavelength resolution, such as amplitude, phase, and polarization. Thanks to its excellent properties, metasurface can replace conventional optical systems or reduce their size. Therefore, many studies on metasurfaces and their applications have been introduced, such as diffrac-

☆ The review of this paper was arranged by Prof. N.S. Scott.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail address: byoungho@snu.ac.kr (B. Lee).

https://doi.org/10.1016/j.cpc.2022.108552
0010-4655/

tion grating [3–8], holography [9–13], and lenses [14–21]. There are two approaches for metasurface design: forward design and inverse design. In the forward design method, the light modulation characteristics must be determined at each position of the metasurface to perform specific optical functionality. At each position, meta-atoms are arranged to modulate the amplitude, phase, and polarization of light for sampling the required light modulation characteristics with subwavelength resolution. Therefore, first, finding meta-atoms with light modulation profiles for an optical device is needed. For the geometry of meta-atoms, simple structures such as rectangle nanopillar that humans can intuitively think of are commonly utilized. Then, their optical characteristics are obtained by performing electromagnetic simulations. From the meta-atoms library found through the simulation, the meta-atoms closest to the required modulation characteristics for each location are arranged to implement a device. This forward design method is mainly used when designing metasurface.

Another design method is inverse design [22–26]. It is challenging to design a metasurface with a complex functionality by only the forward design due to the limitation of the meta-atoms library. Even if such a metasurface is made by the forward design method, performance problems such as efficiency and noise occur significantly. However, the inverse design method can improve performance and achieve a high design degree of freedom. In the inverse design process, a figure of merit (FoM) or loss function to evaluate the performance of the metasurface is set first. Then, the metasurface device is parameterized, such as pixelating metasurface devices and defining the permittivity ratio between vacuum and material at each pixel. Next, the optimization technique is applied to FoM or loss function and update parameters.

Gradient-based optimization [3,6,8,24,27–33], evolutionary algorithm [22,34–36], and deep learning [37–43] can be utilized for the optimization techniques. Inverse design based on evolutionary algorithms such as genetic or particle swarm optimization requires a huge set of simulations. Also, inverse design with deep learning requires a large dataset that pairs the metasurface device and the performance, such as spectrum. When such a large number of simulations are needed, it is necessary to shorten the simulation time. The gradient-based optimization requires a gradient calculation relative to the performance indicator. The gradient can be calculated in various methods [29, 44,45]. First, a numerical gradient can be calculated from the small deviation of the design parameters. However, this method repeatedly performs the simulation as the number of unknown variables. Second, the adjoint variable method gives an exact gradient by comparing two electromagnetic simulations: forward and adjoint simulations [26,44]. Therefore, only two simulations are required regardless of the number of unknown variables. However, additional simulations are needed as the performance evaluation becomes more complex. The automatic differentiation also gives an exact gradient by applying the chain rule for the simulation, considering the simulation as a collection of differentiable individual functions [29,30,32,45]. Gradient calculation for the performance evaluation is possible with a single simulation and chain-rule-based backpropagation. Only the gradient calculation time by chain-rule is additionally required.

For the electromagnetic simulations, the finite element method (FEM), finite-difference time-domain method (FDTD), and rigorous coupled-wave analysis (RCWA) are often used. Among these simulation methods, RCWA provides a semi-analytic solution [46–50], and studies on various algorithms for convergence [51–55] and scattering matrix computation [49,50,56–60] have been proposed. It computes the electromagnetic problem of a periodic structure in which a finite number of layers are stacked. In the Fourier domain, the characteristics of the entire device are calculated using the electromagnetic eigenmodes and boundary conditions between layers. For this reason, the accuracy of the RCWA depends on the size of this finite Fourier domain. Also, the operation speed is not affected by the height of the layers. Thanks to these characteristics and advantages, various RCWA software has been released, such as RETICOLO [61], S4 [62], GRCWA [30], and MAXIM [63] etc. [64–68]. The published RCWA software performs simulations based on a CPU. However, if GPU is used, the simulation speed can be further accelerated while retaining the characteristics and advantages of RCWA. Also, since GPU-accelerated automatic differentiation is not supported for almost RCWA software, lots of simulations should be performed for gradient-based optimization.

We introduce the RCWA software, TORCWA (PyTorch + RCWA), supporting two significant advantages. First, TORCWA is a GPU-accelerated RCWA simulation. It performs simulation significantly faster with almost similar accuracy than CPU operation. In addition, it can be used for gradient-based optimization by supporting GPU-accelerated reverse-mode automatic differentiation. For implementing these characteristics, PyTorch library with version 1.10.1 library is utilized, and in particular, the instability of gradient calculation came from eigendecomposition is solved in TORCWA [69–71]. In this article, we present the formulation, simulation, and optimization examples of the TORCWA. Section 2 shows the basic RCWA formulation. Section 3 introduces automatic differentiation and discusses some gradient calculation issues. Next, section 4 shows simulation verification, convergence test, and comparison with other numerical methods such as FDTD and FEM. Also, the performance comparison will show the accelerated simulation by GPU. Section 5 verifies the gradient calculation and demonstrates the gradient-based optimization.

2. RCWA formulations

This section will describe the summary of basic RCWA formulations [49,50]. The simulation flow chart and schematic illustration of RCWA are depicted in Fig. 1. RCWA first determines the eigenmodes of each layer by Fourier expansion of the material distribution. Then, the in-out relation and coupling coefficients of a single layer are calculated with the eigenmodes and the electromagnetic boundary condition. Next, in-out relation and coupling coefficients of the overall system are computed by the Redheffer star product. Electromagnetic fields can be recovered with input light source, global in-out-relation, and coupling coefficients. In each step, FoM is eventually connected via the complex multivariate functions of ε and μ.

2.1. Eigenmode expansion and S-matrix of single layer

The formulation is developed in the periodic condition where the periods in x- and y-directions are T_x and T_y, respectively. The plane wave incidents from the input layer with elevation angle θ_i and azimuthal angle θ_a and then propagates through the stacked layers. Each layer is homogenous in z-directions. The relative permittivity and permeability of each layer and the time-harmonic Maxwell's curl Equations in SI units can be expressed as Equations (1)-(4) [72].

$$\varepsilon(x, y) = \varepsilon\left(x + mT_x, y + nT_y\right), m, n \in Z \tag{1}$$

$$\mu(x, y) = \mu\left(x + mT_x, y + nT_y\right), m, n \in Z \tag{2}$$

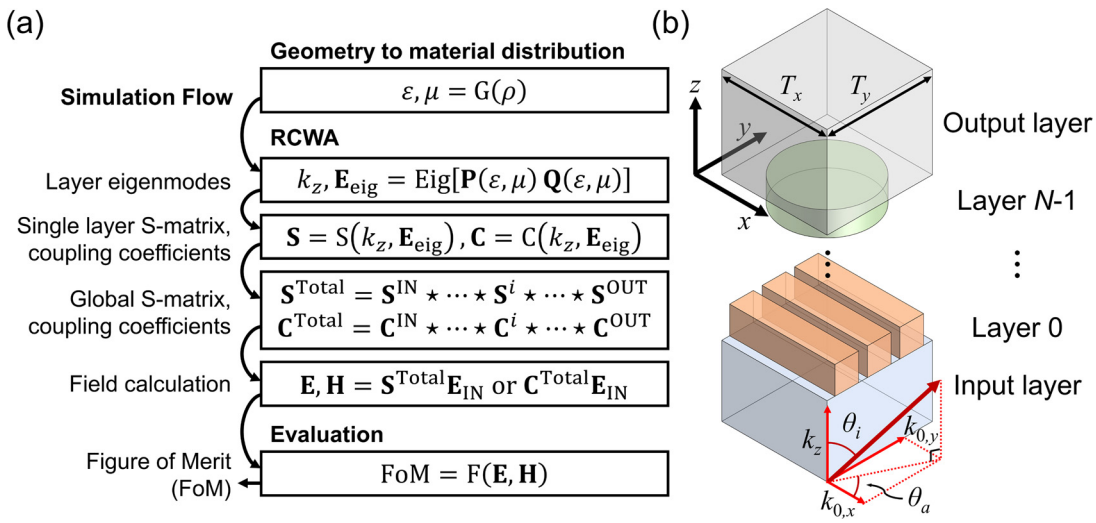

Fig. 1. (a) Simulation flow chart and (b) schematic illustration of RCWA simulation.

$$\nabla \times \mathbf{E} = j\omega\mu_0\mu\,(x, y)\,\mathbf{H} \tag{3}$$

$$\nabla \times \mathbf{H} = -j\omega\varepsilon_0\varepsilon\,(x, y)\,\mathbf{E} \tag{4}$$

Here, ω is the angular frequency of the light, μ_0 and ε_0 are the permeability and permittivity in free space, respectively. The time-harmonic notation of $exp\,(-j\omega t)$ is utilized. The relative permittivity and permeability of the layer can be expanded as the truncated Fourier series in x- and y-directions as below.

$$\varepsilon\,(x, y) = \sum_{m=-2M}^{2M} \sum_{n=-2N}^{2N} \varepsilon_{m,n} e^{j(mG_x x + nG_y y)} \tag{5}$$

$$\mu\,(x, y) = \sum_{m=-2M}^{2M} \sum_{n=-2N}^{2N} \mu_{m,n} e^{j(mG_x x + nG_y y)} \tag{6}$$

Where $G_x = 2\pi/T_x$ and $G_y = 2\pi/T_y$ are reciprocal lattice vectors in x- and y-directions, M and N are the truncated orders in Fourier harmonics. We transform the Equations (3) and (4) into Lorentz-Heaviside units into the Equations (7) and (8) for convenience and to minimize floating-point errors. In the Lorentz-Heaviside units, $\mathbf{E}$, $\mathbf{H}$, and ω are transformed as $\mathbf{E}_{\mathrm{SI}} \rightarrow \mathbf{E}_{\mathrm{LH}}$, $\sqrt{\mu_0/\varepsilon_0}\mathbf{H}_{\mathrm{SI}} \rightarrow \mathbf{H}_{\mathrm{LH}}$, $\sqrt{\mu_0\varepsilon_0}\omega_{\mathrm{SI}} \rightarrow \omega_{\mathrm{LH}}$.

$$\nabla \times \mathbf{E} = j\omega\mu\,(x, y)\,\mathbf{H} \tag{7}$$

$$\nabla \times \mathbf{H} = -j\omega\varepsilon\,(x, y)\,\mathbf{E} \tag{8}$$

The electromagnetic fields in a single layer can be expressed as truncated Fourier series.

$$\mathbf{E} = e^{j(k_{0,x}x + k_{0,y}y + k_z z)} \sum_{m=-M}^{M} \sum_{n=-N}^{N} \mathbf{E}_{m,n} e^{j(mG_x x + nG_y y)} \tag{9}$$

$$\mathbf{H} = e^{j(k_{0,x}x + k_{0,y}y + k_z z)} \sum_{m=-M}^{M} \sum_{n=-N}^{N} \mathbf{H}_{m,n} e^{j(mG_x x + nG_y y)} \tag{10}$$

$\mathbf{E}_{m,n}$ and $\mathbf{H}_{m,n}$ are the Fourier amplitudes of each diffraction channel. We can get matrix equations by substituting (9) and (10) into Maxwell's curl Equations and manipulating them into matrix form [49,50,63]. As in references [49,50,63], electromagnetic fields are divided into three cartesian components, and if the Fourier amplitudes are expressed as a column vector, six matrix equations are obtained. Here, $\mathbf{E}_z$ and $\mathbf{H}_z$ can be removed by substitution, and Equations (11) and (12) can be induced. These two equations describe the transformation between the tangential electric and magnetic fields.

$$\mathbf{P}\begin{bmatrix} \mathbf{H}_x \\ \mathbf{H}_y \end{bmatrix} = \frac{k_z}{k_0}\begin{bmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{bmatrix} \tag{11}$$

$$\mathbf{Q}\begin{bmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{bmatrix} = \frac{k_z}{k_0}\begin{bmatrix} \mathbf{H}_x \\ \mathbf{H}_y \end{bmatrix} \tag{12}$$

$$\mathbf{PQ}\begin{bmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{bmatrix} = \left(\frac{k_z}{k_0}\right)^2 \begin{bmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{bmatrix} \tag{13}$$

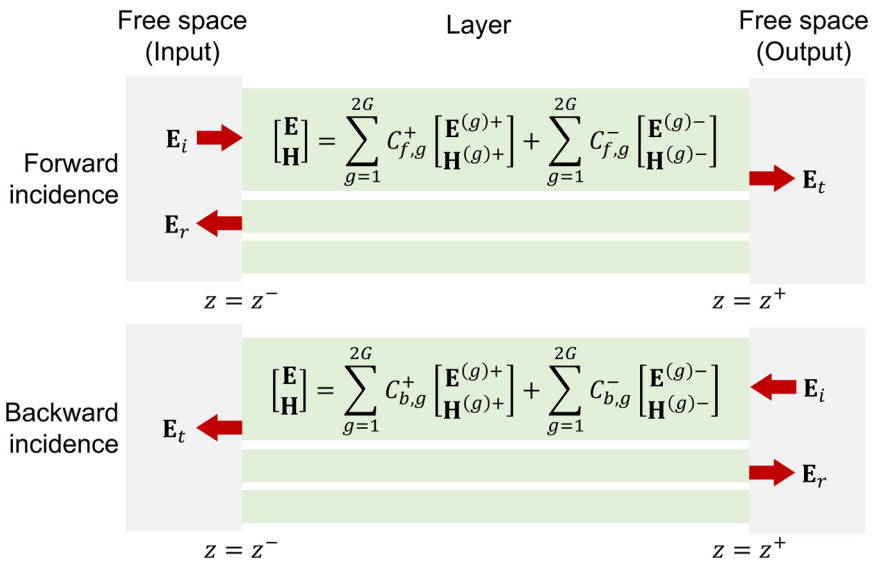

Fig. 2. Electromagnetic fields relations of single layer.

Then, by combining Equations (11) and (12), we can get the eigenvalue problem as Equation (13). Here, the tangential components of the electric field are eigenvectors, and the square of the normalized z-component of k-vectors are eigenvalues. For k_z, two different signs come out from the single eigenvalue. To ensure decay while propagation, a value with an imaginary part of 0 or larger is selected for the positive mode, and a negative imaginary part is selected for the negative mode. The diagonal matrix of the normalized k_z and electric field eigenmode matrix can be described as Equations (14) and (15).

$$\mathbf{K}_{z,e} = \frac{1}{k_0}\begin{bmatrix} k_z^{(1)} & 0 & \cdots & 0 \\ 0 & k_z^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_z^{(2G)} \end{bmatrix} \tag{14}$$

$$\mathbf{E}_e = \begin{bmatrix} \mathbf{E}_x^{(1)} & \cdots & \mathbf{E}_x^{(2G)} \\ \mathbf{E}_y^{(1)} & \cdots & \mathbf{E}_y^{(2G)} \end{bmatrix} \tag{15}$$

Equations (14) and (15) are described under the positive eigenmodes. For the negative eigenmodes, the sign of $\mathbf{K}_{z,e}$ is reversed, while $\mathbf{E}_e$ is retained. From the electric field eigenmode matrix, the magnetic field eigenmode matrix can be obtained as Equation (16).

$$\mathbf{H}_e = \mathbf{P}^{-1}\mathbf{E}_e\mathbf{K}_{z,e} = \mathbf{Q}\mathbf{E}_e\mathbf{K}_{z,e}^{-1} \tag{16}$$

$$\begin{bmatrix}\mathbf{E}\\ \mathbf{H}\end{bmatrix} = \sum_{g=1}^{2G} C_{f,g}^{+}\begin{bmatrix}\mathbf{E}^{(g)+}\\ \mathbf{H}^{(g)+}\end{bmatrix} + \sum_{g=1}^{2G} C_{f,g}^{+}\begin{bmatrix}\mathbf{E}^{(g)-}\\ \mathbf{H}^{(g)-}\end{bmatrix} \tag{17}$$

$$\begin{bmatrix}\mathbf{E}\\ \mathbf{H}\end{bmatrix} = \sum_{g=1}^{2G} C_{b,g}^{+}\begin{bmatrix}\mathbf{E}^{(g)+}\\ \mathbf{H}^{(g)+}\end{bmatrix} + \sum_{g=1}^{2G} C_{b,g}^{+}\begin{bmatrix}\mathbf{E}^{(g)-}\\ \mathbf{H}^{(g)-}\end{bmatrix} \tag{18}$$

The sign + and – denote the positive and negative modes, and g denotes the eigenmode number. Also, f and b denote the forward incidence and the backward incidence case, respectively. By the summation with mode coupling coefficient, C^+ and C^-, the electromagnetic fields in the layer can be expressed as Equations (17) and (18). Equation (17) is for forward incidence, and Equation (18) is for backward incidence. Here, $2G$ is the total eigenmode number, where $G = (2M+1)(2N+1)$.

The next step is getting the S-matrix and coupling coefficient of a single layer [49,50,63]. In this case, the S-matrix represents the transmission and reflection relation between each diffraction channel. First, we consider forward incidence as depicted in Fig. 2. The incident light $\mathbf{E}_i$ is the known value. The reflected light $\mathbf{E}_r$, transmitted light $\mathbf{E}_t$, and the coupling coefficients are unknown values that we must get. From the boundary condition between free space and the layer, the tangential components of electromagnetic fields must be continuous. If the boundary condition is applied to electromagnetic field relation, the coupling coefficient of Equations (17) and (18) and the S-matrix of Equation (19) can be calculated [49,50,63].

$$\mathbf{S} = \begin{bmatrix}\mathbf{S}_{11} & \mathbf{S}_{12}\\ \mathbf{S}_{21} & \mathbf{S}_{22}\end{bmatrix} = \begin{bmatrix}\mathbf{T}_f & \mathbf{R}_b\\ \mathbf{R}_f & \mathbf{T}_b\end{bmatrix} \tag{19}$$

2.2. Global S-matrix

In the next step, the extended S-matrix method is exploited. Here, the scattering matrix of the overall system can be obtained by connecting the scattering matrices of the single layer [49,50,60,63]. As shown in Fig. 3, the combined scattering matrix can be calculated

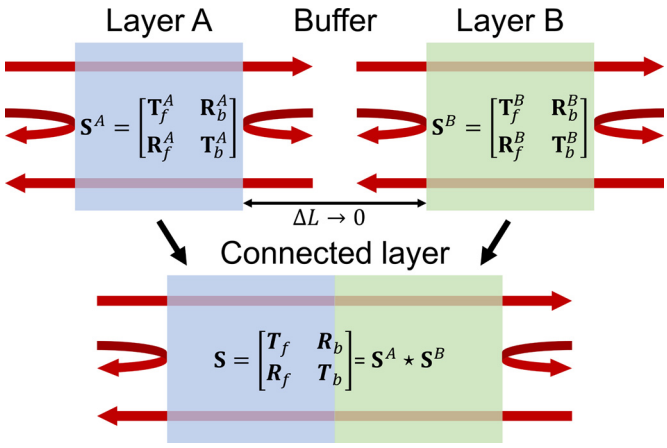

Fig. 3. Layer connection and global S-matrix.

according to the thickness of the buffer placed between the two systems. Then, if we take the limit of buffer thickness as 0, we can calculate the S-matrix of the combined system. In this way, the combined S-matrix of the layer system is calculated through an operation called the extended Redheffer star product denoted as $\star$. The S-matrix and coupling coefficients are updated as Equations (20)-(23) [49,50,60,63].

$$\mathbf{S}^{A+B} = \mathbf{S}^A \star \mathbf{S}^B = \begin{bmatrix} \mathbf{T}_f^B \left(\mathbf{I} - \mathbf{R}_b^A \mathbf{R}_f^B\right)^{-1} \mathbf{T}_f^A & \mathbf{R}_b^B + \mathbf{T}_f^B \left(\mathbf{I} - \mathbf{R}_b^A \mathbf{R}_f^B\right)^{-1} \mathbf{R}_b^A \mathbf{T}_b^B \\ \mathbf{R}_f^A + \mathbf{T}_b^A \left(\mathbf{I} - \mathbf{R}_f^B \mathbf{R}_b^A\right)^{-1} \mathbf{R}_f^B \mathbf{T}_f^A & \mathbf{T}_b^A \left(\mathbf{I} - \mathbf{R}_f^B \mathbf{R}_b^A\right)^{-1} \mathbf{T}_b^B \end{bmatrix} \tag{20}$$

$$\begin{aligned} \mathbf{C}^{A+B} &= \begin{bmatrix} \mathbf{C}_f^{A,\{A,B\}} & \mathbf{C}_f^{B,\{A,B\}} \\ \mathbf{C}_b^{A,\{A,B\}} & \mathbf{C}_b^{B,\{A,B\}} \end{bmatrix} = \mathbf{C}^A \star \mathbf{C}^B = \begin{bmatrix} \mathbf{C}_f^A \\ \mathbf{C}_b^A \end{bmatrix} \star \begin{bmatrix} \mathbf{C}_f^B \\ \mathbf{C}_b^B \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{C}_f^A + \mathbf{C}_b^A \left(\mathbf{I} - \mathbf{R}_f^B \mathbf{R}_b^A\right)^{-1} \mathbf{R}_f^B \mathbf{T}_f^A & \mathbf{C}_f^B \left(\mathbf{I} - \mathbf{R}_b^A \mathbf{R}_f^B\right)^{-1} \mathbf{T}_f^A \\ \mathbf{C}_b^A \left(\mathbf{I} - \mathbf{R}_f^B \mathbf{R}_b^A\right)^{-1} \mathbf{T}_b^B & \mathbf{C}_b^B + \mathbf{C}_f^B \left(\mathbf{I} - \mathbf{R}_b^A \mathbf{R}_f^B\right)^{-1} \mathbf{R}_b^A \mathbf{T}_b^B \end{bmatrix} \end{aligned} \tag{21}$$

$$\mathbf{S}^{\text{Layer}} = \mathbf{S}^0 \star \mathbf{S}^1 \star \cdots \star \mathbf{S}^{N-1} \tag{22}$$

$$\mathbf{C}^{\text{Layer}} = \mathbf{C}^0 \star \mathbf{C}^1 \star \cdots \star \mathbf{C}^{N-1} \tag{23}$$

Here, the total S-matrix and coupling coefficients obtained are enclosed in free space. However, since a substrate or a coated material surrounds an actual device, the total S-matrix and coupling coefficients must be updated accordingly. This can be solved by considering half-infinite and homogenous input and output layers and getting the S-matrix of two layers $\mathbf{S}^{\text{IN}}$ and $\mathbf{S}^{\text{OUT}}$ [49,50,60,63]. The overall scattering matrix can be obtained as Equation (24). Also, the coupling coefficients can be updated in the same manner.

$$\mathbf{S}^{\text{Total}} = \mathbf{S}^{\text{IN}} \star \mathbf{S}^{\text{Layer}} \star \mathbf{S}^{\text{OUT}} \tag{24}$$

The reflected and transmitted fields can be recovered as Equations (25) and (26) for forward incidence. Also. The coupling coefficients in k-th layer can be obtained by Equation (27), and then fields can be recovered by combining eigenmodes as Equations (17) and (18).

$$\begin{bmatrix} \mathbf{E}_{r,x} \\ \mathbf{E}_{r,y} \end{bmatrix} = \mathbf{R}_f^{\text{Total}} \begin{bmatrix} \mathbf{E}_{i,x} \\ \mathbf{E}_{i,y} \end{bmatrix} \tag{25}$$

$$\begin{bmatrix} \mathbf{E}_{t,x} \\ \mathbf{E}_{t,y} \end{bmatrix} = \mathbf{T}_f^{\text{Total}} \begin{bmatrix} \mathbf{E}_{i,x} \\ \mathbf{E}_{i,y} \end{bmatrix} \tag{26}$$

$$\begin{bmatrix} \mathbf{C}_f^{k,+} \\ \mathbf{C}_f^{k,-} \end{bmatrix} = \mathbf{C}_f^{k,\text{Total}} \begin{bmatrix} \mathbf{E}_{i,x} \\ \mathbf{E}_{i,y} \end{bmatrix} \tag{27}$$

3. Gradient calculation with reverse-mode automatic differentiation

3.1. Automatic differentiation

For optimization or analyzing a photonic device, it is necessary to calculate the gradient of the evaluation function with respect to the geometry parameters. Numerical gradients and adjoint variable methods are utilized to calculate the gradient, as introduced in section 1. However, thanks to open-source software development, libraries that support automatic differentiation that calculate gradients for the entire function system using chain rules have been released, such as TensorFlow [73], PyTorch [74], and JAX [75]. In particular, packages that support forward calculation and automatic differentiation on GPU are actively used to optimize a system expressed by synthesizing a vast amount of functions, such as deep learning.

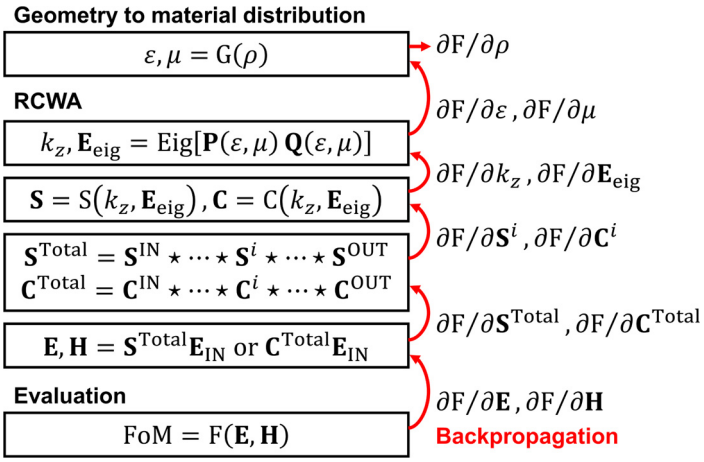

Fig. 4. Backpropagation flow chart for gradient calculation. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Fig. 4 shows the flow chart of the RCWA simulation as described in Section 2. RCWA can be regarded as a composition of various functions, such as exponentiation, matrix multiplication, inverse matrix, eigendecomposition, and many other operations that have Wirtinger derivatives. Suppose converting the geometry to the spatial permittivity and permeability distribution and the Figure of Merit (FoM) from the results are differentiable. In that case, the overall system from geometric parameters to evaluation is differentiable. As shown in the red arrow in Fig. 4, the chain-rule is applied by automatic differentiation.

$$\frac{\partial F}{\partial \rho} = \frac{\partial F}{\partial \varepsilon}\left(\frac{\partial \varepsilon}{\partial \rho}\right) + \frac{\partial F}{\partial \mu}\left(\frac{\partial \mu}{\partial \rho}\right) \tag{28}$$

$$\frac{\partial F}{\partial \rho} = \left(\frac{\partial F}{\partial \mathbf{E}}\right)\frac{\partial \mathbf{E}}{\partial \rho} + \left(\frac{\partial F}{\partial \mathbf{H}}\right)\frac{\partial \mathbf{H}}{\partial \rho} \tag{29}$$

There are forward-mode and reverse-mode accumulation methods for applying the chain rule in automatic differentiation [29]. Forward-mode is a method of accumulating the gradient starting from the function of the input variable, as in the parentheses of Equation (28). On the other hand, the reverse-mode or backpropagation is a method of accumulating the gradient starting from the function of the output variable, as in the parentheses of Equation (29). The larger the output dimension than the input dimension in the forward-mode, and the larger the input dimension than the output dimension in the reverse-mode, the more efficient the operation. In most optimizations and deep learning, reverse-mode is efficient and is used by most libraries because there is single evaluation output.

3.2. Gradient of eigendecomposition

$$\mathbf{A}\mathbf{X} = \mathbf{X}\mathbf{\Lambda} \tag{30}$$

$$\frac{\partial L}{\partial \overline{\mathbf{A}}} = \mathbf{X}^{-\dagger}\left[\frac{\partial L}{\partial \overline{\mathbf{\Lambda}}} + \overline{\mathbf{F}} \circ \left(\mathbf{X}^{\dagger}\frac{\partial L}{\partial \overline{\mathbf{X}}}\right)\right]\mathbf{X}^{\dagger} \tag{31}$$

$$\mathbf{F}_{ij} = \begin{cases} 0, & i = j \\ \frac{1}{\lambda_j - \lambda_i}, & i \neq j \end{cases} \tag{32}$$

One of the essential computational processes in RCWA is eigendecomposition and calculating the gradient of this is also an important issue. In Equation (30), $\mathbf{A}$ is the matrix for which eigendecomposition is performed, and $\mathbf{X}$ and $\mathbf{\Lambda}$ are the eigenvector and diagonal eigenvalue matrices, respectively. The reverse-mode accumulation of eigendecomposition can be calculated as Equations (31) and (32), where L is the evaluation function [69–71]. The dagger and overscore denote Hermitian and complex conjugate, respectively. Various libraries support automatic differentiation for eigendecomposition, and PyTorch above 1.9.0 or later supports such operation on GPU and complex domain.

However, the reverse-mode accumulation of eigendecomposition is unstable when the degenerated modes exist (same eigenvalue), as shown in Equation (32). Lorentzian broadening is utilized to solve this instability, which avoids division by zero by adding a small number to the denominator [30,32,71]. Then, there is a trade-off between stability and precision for the gradient of eigendecomposition. Here, we utilize Lorentzian broadening form like references [30,32,71], and the division by zero in the complex domain is prevented, as shown in Equation (33). In TORCWA, we write a new eigendecomposition class operating the same forward calculation but deriving a stable gradient.

$$\mathbf{F}_{ij} = \begin{cases} 0, & i = j \\ \frac{\overline{\lambda}_j - \overline{\lambda}_i}{|\lambda_j - \lambda_i|^2 + \varepsilon}, & i \neq j \end{cases} \tag{33}$$

The minimum positive value that the data type can represent is utilized as broadening parameter ε to bring both stability and accuracy. Specifically, the minimum positive subnormal value is about 1.4×10^{-45} for the single-precision floating-point format (torch.float32 and torch.complex64 in the PyTorch). Also, the minimum positive subnormal value is about 4.9×10^{-324} for the double-precision floating-point format (torch.float64 and torch.complex128 in the PyTorch). If the custom broadening parameter is not determined, it is automatically set in the TORCWA depending on which data type is utilized.

Table 1
Simulation program sequence with Python code.

```
import torch, torcwa

# (1) Set simulation environment
sim = torcwa.rcwa(freq=1/532, order=[15,15], L=[300,300], ...
      dtype=torch.complex64, device=torch.device('cuda')) # Except for incident angle

# (2) Add input/output layer(s)
sim.add_input_layer(eps=2, mu=1) # Regarded as free space if not added
sim.add_output_layer(eps=2, mu=1) # Step (2) can be skipped if both layers are free space

# (3) Set incident angle
sim.set_incident_angle(inc_ang=15*pi/180, azi_ang=45*pi/180, angle_layer='input')

# (4) Add internal layer(s)
# Add in the order closest to the input layer
sim.add_layer(thickness=200, eps=layer0_eps_grid, mu=1)
sim.add_layer(thickness=400, eps=layer1_eps_grid, mu=1)

# (5) Solve global S-matrix and get S-parameters
sim.solve_global_smatrix()
# S-parameter can be acquired by
S = sim.S_parameters(orders=[0,0], direction='forward', ...
      port='transmission', polarization='xx', ref_order=[0,0])
# End at here if not need electromagnetic fields

# (6) Set light source and get electromagnetic fields
sim.source_planewave(amplitude=[1,0], direction='forward')
# or by Fourier expanded source
sim.source_fourier(amplitude=[[1,0],[0.7,0.2], orders=[[0,0],[1,0]], direction='forward')
# Electromagnetic fields can be acquired by
E, H = sim.field_xz(x_axis=xa, z_axis=za, y=150) # or
E, H = sim.field_yz(y_axis=ya, z_axis=za, x=150) # or
E, H = sim.field_xy(layer_num=0, x_axis=xa, y_axis=ya, z_prop=dz)
```

3.3. Gradient calculation for layers with high order eigenmodes

If the layer has a high order eigenmode, the normalized k_z of the mode is small. If the low precision data type is utilized, such as the single-precision floating-point, the matrix $\mathbf{P}$ or $\mathbf{Q}$ that transforms between $\mathbf{E}$ and $\mathbf{H}$ in Equations (11)-(13) and (16) may numerically approach a nearly singular matrix. A large error may occur in the inverse matrix operation and its gradient operation for a nearly singular matrix, particularly like the operation in Equation (16).

$$P_u = \text{maximum}\left\{\max\left|\mathbf{P}\mathbf{P}^{-1} - \mathbf{I}\right|, \max\left|\mathbf{P}^{-1}\mathbf{P} - \mathbf{I}\right|\right\} \tag{34}$$

$$Q_u = \text{maximum}\left\{\max\left|\mathbf{Q}\mathbf{Q}^{-1} - \mathbf{I}\right|, \max\left|\mathbf{Q}^{-1}\mathbf{Q} - \mathbf{I}\right|\right\} \tag{35}$$

Therefore, by default, TORCWA first calculates the numerical value of how unstable the inverse matrix of $\mathbf{P}$ and $\mathbf{Q}$ is, as in Equations (34) and (35). Here, maximum$\{x, y\}$ returns larger of x and y, $\max|\cdot|$ returns the maximum value of matrix elements, $|\cdot|$ returns elementwise absolute values. Then, if the value P_u does not exceed a specific value (default is 0.005), $\mathbf{H}_e$ is calculated using the inverse matrix of $\mathbf{P}$ in Equation (16). If it exceeds, $\mathbf{H}_e$ is computed using the $\mathbf{Q}$ of Equation (16) to calculate the transverse magnetic field eigenvector. However, if the inverse matrix of both $\mathbf{P}$ and $\mathbf{Q}$ has a high unstable numerical value, the high precision of the data type for the entire simulation should be used, which is a more fundamental method. Therefore, TORCWA provides the P_u and Q_u to give such judgment criteria for manually avoiding unstable inverse matrix.

4. Simulation program sequence and performance

In this section, the TORCWA simulation tool will be verified. First, a convergence test will be performed with respect to the number of Fourier orders. Then, the simulation will be compared to the FDTD and FEM simulations. Next, a performance comparison such as simulation time and error between CPU and GPU for some selected devices will be shown.

4.1. Simulation program sequence

Python version 3.8 and PyTorch version 1.10.1 are utilized to write TORCWA. PyTorch is a library used in deep learning that supports various mathematical functions while supporting GPU operation to enable fast calculations. In addition, it supports reverse-mode automatic differentiation, so it has the feature of calculating the gradient for system optimization. Therefore, this package is suitable for producing a simulation program supporting fast calculation and optimization.

The overall simulation program sequence with Python code is described in Table 1. The first step is setting the simulation environment. Light frequency, lattice constants, and Truncated Fourier orders must be determined in this step. Also, simulation data type, device, and stabilized gradient should be determined, but their defaults are utilized, such as 32bits, CUDA if available, and true, respectively. This step does not determine the incident angle since it is necessary to decide which material the incident angle is referenced.

The next step is setting input and output layers. Since these layers are considered free space by default, this step can be skipped if these two are free space. The incident angle must be determined after the input and output layers are decided. The reference layer

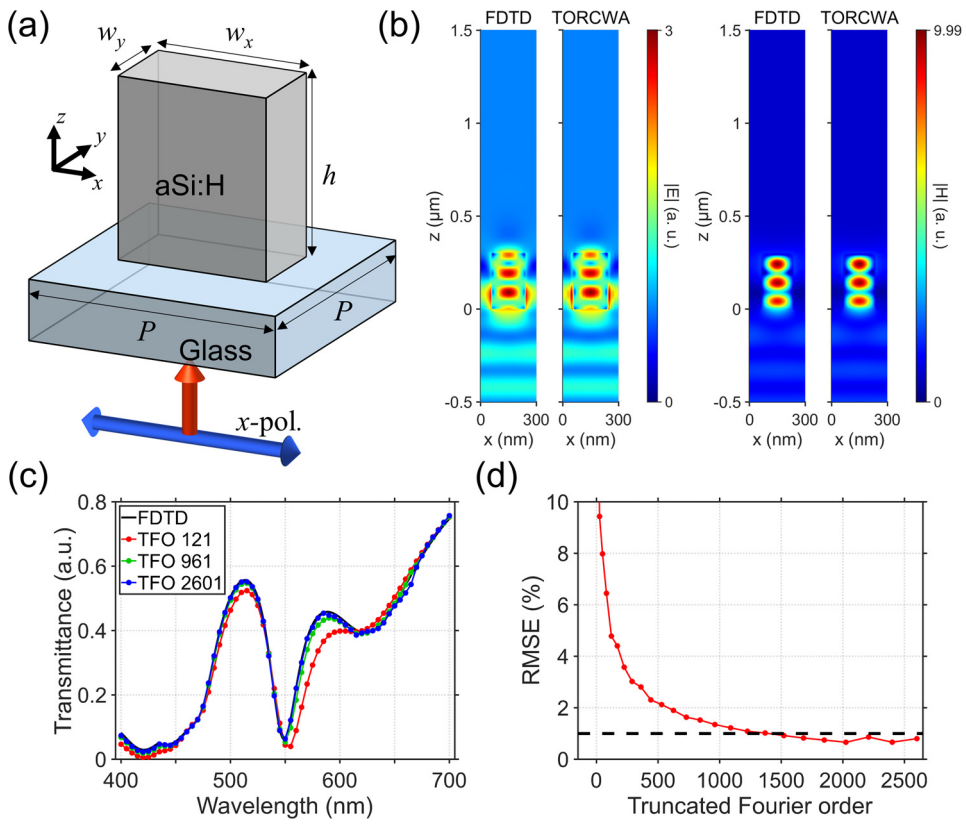

Fig. 5. Convergence test for normally incident light (a) Structure schematic. Geometric parameters of $w_x = 180$ nm, $w_y = 100$ nm, $h = 300$ nm, and $P = 300$ nm are utilized. (b) Electromagnetic field distribution in xz-plane at the center of nanofin. The field distribution is calculated with 532 nm light. TFOs are both 961. (c) Transmittance spectra for the selected TFO. (d) RMSE with respect to truncated Fourier order. The black dashed line indicates 1%.

of the incident angle, the slope of the zeroth-order k-vector, is determined together in this step. Once the angle of incidence has been determined, it automatically generates k-vectors for Fourier expansion internally.

The internal layer structures are added for the next step. The structure of the layer is considered to be on a uniform grid with relative permittivity and permeability. However, it is possible to input simple values for a homogenous layer instead of a grid. Layer adding order starts from the layer closest to the input layer. Whenever a layer is added, a Toeplitz matrix is created by Fourier expansion of the permittivity and permeability distribution of the input layer. Then, the eigenmode, S-matrix, and coupling coefficients of the added single layer are calculated sequentially.

When the addition of the internal layer is completed, the global S-matrix and coupling coefficients for the entire system are calculated. The user can stop here if the user needs only in and out relations without field information. If the user needs information about the field, the light source and the complex amplitude and forward or backward incidence according to polarization must be determined next. Then, when the window to view the field is selected, the electromagnetic fields are calculated internally using the S-matrix, coupling coefficients, and light source.

4.2. Convergence test

In the RCWA simulation, the higher the Fourier order used, the higher the accuracy of the solution. However, instead of increasing the accuracy, the calculation time is increased. It is verified whether the written program produced accurate results for structures frequently used in metasurface design. Convergence according to Fourier order is shown, and the results a compared with commercial simulation tools.

Three RCWA simulations are performed with 32bit floating point datatype, Intel Xeon Gold 5118, and NVIDIA GeForce RTX 3090. NVIDIA GeForce RTX 3090 and higher GPUs support TensorFloat-32 (TF32) operation. TF32 operation supports faster computation for matrix multiplication, but the computational error becomes relatively large. The details will be dealt with in section 3.3. Here, the simulations are performed using TF32 core disabled.

Fig. 5 shows the convergence and comparison of simulations for anisotropic nanofins with normally incident light. The same simulation is performed with the commercial FDTD simulation tool Lumerical. The normalized electromagnetic field is plotted in Fig. 5b, and it can be seen that FDTD and TORCWA have nearly identical field distributions. Fig. 5c shows the transmittance from 400 nm to 700 nm according to each Truncated Fourier order (TFO) and the FDTD. As the TFO increases, it can be seen that the transmittance spectrum converges. The root-mean-square errors (RMSEs) are quantitatively calculated, and the RMSEs according to the TFO are shown in Fig. 5d, assuming that the FDTD result is the actual value. The RMSEs decrease as the TFO increases, and the RMSEs are less than 1% when TFO is 1369 or more. RMSE is about 1.35% when TFO is 961.

Fig. 6 shows the convergence and comparison of simulations for isotropic nanopillars with obliquely incident light. The same simulation is performed with the commercial FEM tool COMSOL Multiphysics 5.3a. Transmittance convergence is observed while increasing the TFO

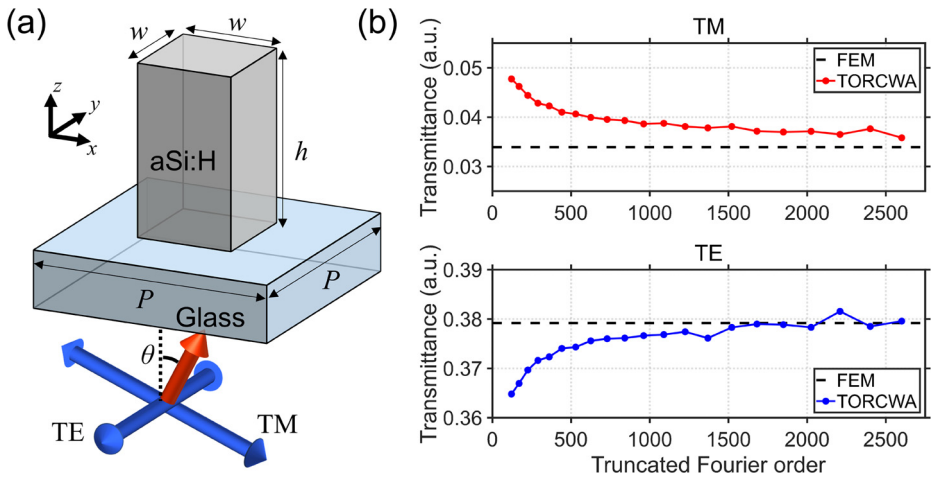

Fig. 6. Convergence test for obliquely incident light. (a) Structure schematic. Geometric parameters of $w = 120$ nm, $h = 300$ nm, and $P = 300$ nm are utilized. (b) Transmittance with respect to TFO for TM and TE light. All simulations are performed with 532 nm light with an incident angle of 15 degrees.

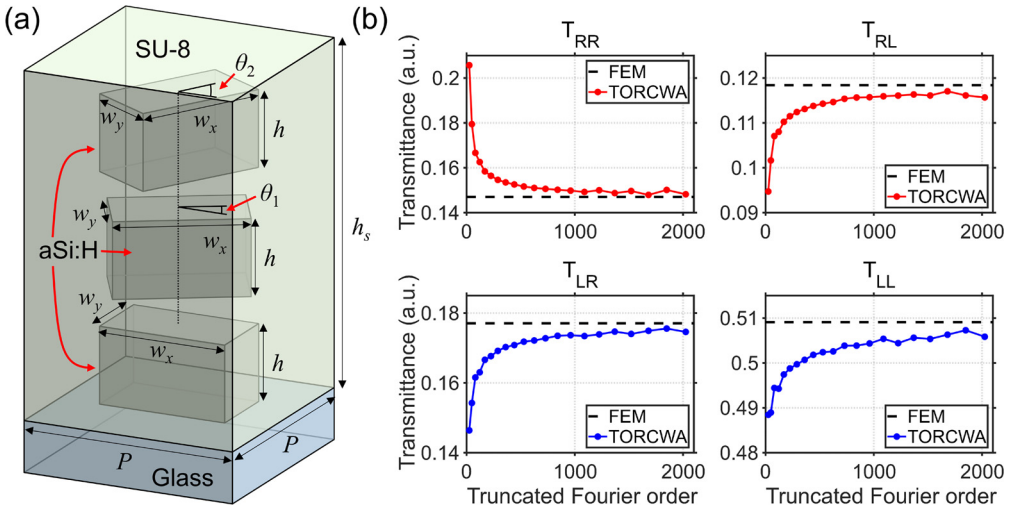

Fig. 7. Convergence test for a multi-layered structure. (a) Structure schematic. Geometric parameters of $w_x = 180$ nm, $w_y = 100$ nm, $h = 200$ nm, $h_s = 900$ nm, $\theta_1 = 30$ degree, $\theta_2 = 60$ degree, and $P = 300$ nm are utilized. The gap between the nanofins is 100 nm. SU-8 photoresist is used as a protection layer. (b) Transmittance with respect to TFO for each Jones matrix component in the circular polarization basis. All simulations are performed with 650 nm light with normal incidence.

of the TORCWA simulation for both TM and TE polarized light. As the TFO increases, it can be seen that convergence almost coincides with the FEM results.

The last RCWA simulation is a multi-layered structure shown in Fig. 7. As shown in Fig. 7a, it is a structure where three anisotropic nanofins are alternately rotated and stacked. This structure shows chirality, and a transmission coefficient matrix based on circular polarization as in Equation (36) is used to show the chirality.

$$t = \begin{bmatrix} t_{RR} & t_{RL} \\ t_{LR} & t_{LL} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} (t_{xx}+t_{yy})+i(t_{xy}+t_{yx}) & (t_{xx}-t_{yy})-i(t_{xy}+t_{yx}) \\ (t_{xx}-t_{yy})+i(t_{xy}+t_{yx}) & (t_{xx}+t_{yy})-i(t_{xy}+t_{yx}) \end{bmatrix} \tag{36}$$

The absolute square value of each matrix component in Equation (36) is simulated according to each TFO, and the result is shown in Fig. 7b. In addition, the simulation is performed using a commercially FEM tool with the same structure. As the TFO increases, the convergence of the transmittance can be seen, and the result becomes similar to FEM. Therefore, TORCWA software shows convergence even for multi-layer structures and shows nearly the same results as other simulation tools.

4.3. Performance test

Parametric sweeps of width w_x and w_y of the meta-atom structure in Fig. 5a are performed to compare the speed and accuracy of the simulation. Also, only 532 nm light is utilized, but the period, height, and polarization are the same as the conditions in Fig. 5a. Combinations of hardware and datatype in Table 2 are used to compare simulation performance.

Combinations of (1) and (2) in Table 2 are CPU and GPU combinations that can be used in personal desktop computers, and combinations of (3)–(6) are combinations that can be used in workstation computers. The simulation performances for all cases are compared in Fig. 8. Fig. 8a is the result of the FEM simulation for the same structure and shows transmittance maps for w_x and w_y. Fig. 8b plots the transmittance map for TFO of 1681 and the smallest RMSE among the performed TORCWA simulations. By comparing the two cases, the RMSE is about 0.77%, and it can be confirmed that the results are nearly identical.

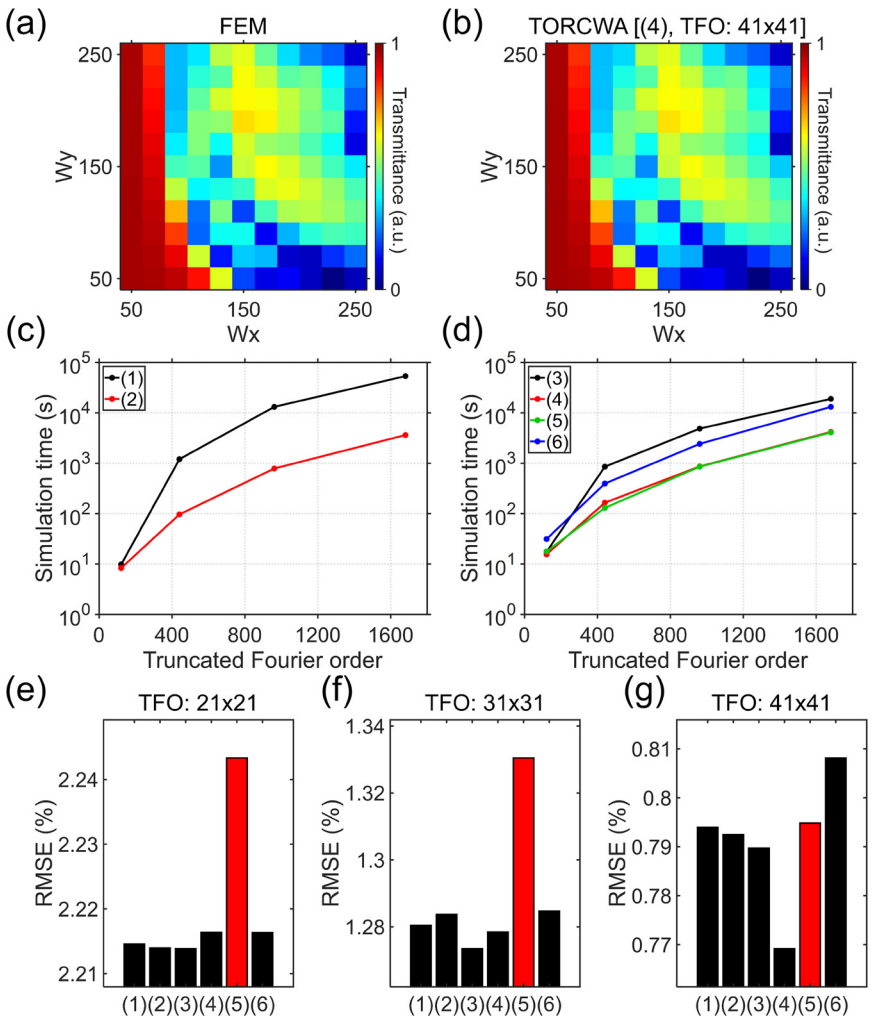

Fig. 8. Performance comparison among various simulation environments. The rectangular meta-atom transmittance map with (a) FEM and (b) TORCWA. Simulation time with a parametric sweep for (c) desktop environment, case (1) and (2) in Table 2, (d) workstation environment, case (3-6) in Table 2. RMSEs with TFO (e) 21×21 (=441) (f) 31×31 (=961) (g) 41×41 (=1681).

Table 2
Hardware, datatype combinations for performance test.

	Hardware	Datatype
(1)	Intel Core i5-7500 only	torch.complex64
(2)	Intel Core i5-7500, NVIDIA GeForce RTX 2060 Super	torch.complex64
(3)	Intel Xeon Gold 5118 only	torch.complex64
(4)	Intel Xeon Gold 5118, NVIDIA GeForce RTX 3090	torch.complex64 (FP32)
(5)	Intel Xeon Gold 5118, NVIDIA GeForce RTX 3090	torch.complex64 (TF32)
(6)	Intel Xeon Gold 5118, NVIDIA GeForce RTX 3090	torch.complex128

The graph in Fig. 8c shows the simulation time according to the TFO in the desktop environment. Here, (1) is the CPU-only environment, while (2) allows GPU operation. It can be seen that the simulation using the GPU is about ten times faster than the CPU-only simulation, except for the TFO of 121. The highest acceleration is 16.7 times faster in the TFO of 961. The graph in Fig. 8d shows the simulation time according to TFO in the workstation environment. Here, (3) is the CPU-only environment, while (4) to (6) allow GPU. In detail, (4) uses a complex 64-bits FP32, (5) uses a complex 64-bits TF32, and (6) uses a complex 128-bits datatype. In Fig. 8d, the simulations (4) and (5) show about five times faster than (3) except for the TFO of 121. The difference between (4) and (5) is whether or not a tensor float core is used. As mentioned earlier, when using the TF32 core, the matrix multiplication operation is accelerated. The simulation (5) is 1.2 times faster than (4) for the TFO of 441 and 1.03 times for the TFO of 1681. The simulation with the TF32 core is accelerated but dramatic since the matrix multiplication operation is not a significant bottleneck. The complex 128-bits datatype can be used for the high computational accuracy of the simulation. Because the computation load is doubled, it is slower than (4) and (5), but it is still faster than the CPU-only simulation except for the TFO of 121. Also, it should be noted that the acceleration may depend on the hardware combination.

Figs. 8e, f, g show the RMSE for the TFO of 441, 961, and 1681, respectively, when the FEM results are assumed to be true. The bar graph marked in red shows the case of using the TF32 core. Since it is to give up some data accuracy for acceleration, it can be seen that the RMSE is higher than the other cases. However, when the TFO is increased to 1681, RMSE of (6) is converged and almost similar to the other cases. Since there is no significant difference in RMSE, it is better to check the convergence by the TFO rather than the datatype.

Table 3
Gradient calculation process with Python code.

```
1  import torch, torcwa
2
3  rho = torch.rand((300,300),dtype=torch.float32,device=torch.device('cuda'))
4  # Make functional relationship
5  rho.requires_grad_(True)
6  eps = 4*rho + (1-rho)
7
8  # Make simulation
9  sim = torcwa.rcwa(freq=1/532, order=[15,15], L=[300,300], ...
10       dtype=torch.complex64, device=torch.device('cuda')) # Except for incident angle
11 sim.add_input_layer(eps=2, mu=1)
12 sim.add_output_layer(eps=2, mu=1)
13 sim.set_incident_angle(inc_ang=0*pi/180, azi_ang=0*pi/180, angle_layer='input')
14 sim.add_layer(thickness=200, eps=eps, mu=1)
15 sim.solve_global_smatrix()
16 S = sim.S_parameters(orders=[0,0], direction='forward', ...
17       port='transmission', polarization='xx', ref_order=[0,0])
18
19 # Backpropagation
20 S.backward()
21
22 # Get gradient
23 rho_gradient = rho.grad
```

4.4. Further extension

The TORCWA software is further extensible, and we will cover some extensible points. The first is about the boundary conditions. Basically, RCWA assumes the infinite space along the z-direction and periodic boundary conditions along the xy-direction in Fig. 1b. However, a perfectly matched layer (PML) boundary condition is also possible if we utilize the method in the reference [50]. If PML boundary condition is available, the simulation of a more general situation is possible, such as non-periodic structures. The second is about anisotropic materials. The current software is based on formulation with isotropic materials. Anisotropic materials are also used in various fields of optics, and their simulation methodology has also been studied [50,76]. More general materials can be simulated if this extension is available. The third is about algorithm improvements, such as TM mode convergence and the Fourier order truncation method. Since the RCWA simulation converges a little later for the TM mode, Enhanced algorithms have been proposed to improve TM mode convergence [51,53]. In addition, a circular truncation instead of the conventional rectangular truncation can be utilized to increase convergence or to save computer resources [77]. By embedding these algorithms, the algorithms can be manually selected by users or automatically selected suitable for the situation. The last point is the input source. Currently, two sources are supported in this software: a plane wave and a custom source whose input is Fourier amplitude. The latter source has a high degree of freedom, but the user must calculate the Fourier amplitude if a Gaussian beam or a dipole source is required. If an internal function that calculates the Fourier amplitude with parameters for frequently used sources as input is implemented as an extension, it will be more convenient for users.

5. Gradient validation and inverse design

One of the advantages of TORCWA is that it supports GPU-accelerated automatic differentiation. When the evaluation or loss function of simulations is derived as a single scalar value, gradients for the simulation conditions can be computationally obtained. Also, some physical properties, such as the phase group velocity of the meta-atom, can be calculated rather than numerical gradients. Also, it is possible to use the gradient for optimization purposes.

In actual programming, taking PyTorch as an example, the gradient by reverse-mode automatic differentiation can be calculated by adding only three lines of code. Let variable 'rho' be a geometric parameter to calculate the gradient and let variable 'S' be the final output of the calculation. First, when code 'rho.requires_grad_(True)' (line 5 in Table 3) is applied, functional relationships like a computational graph similar to the box and arrow in Fig. 4 are saved. Then, when code 'S.backward()' (line 20 in Table 3) is applied, the chain rule with reverse-mode accumulation is automatically performed. Finally, the gradient can be acquired by code 'rho.grad' (line 23 in Table 3).

The functions used in RCWA can calculate a stable gradient operation in the complex domain, but as in Section 3.2, eigendecomposition requires mathematical techniques for stability. This section looks at the accuracy of gradient calculation according to broadening parameters. In addition, we will cover optimization examples of metasurface to improve performance.

5.1. Broadening parameter and gradient validation

The gradients derived by numerical and automatic differentiation with several broadening parameters are calculated for comparison. The numerical gradient is calculated with a slight deviation of a parameter such as $\partial L/\partial x = [L(x+\Delta x) - L(x-\Delta x)]/(2\Delta x)$. However, if a slight change Δx is similar or smaller than 10^{-7} of the data value x with FP32 datatype, the difference in L is significantly affected by the floating-point error. Also, increasing the scale of Δx causes a large error. For this reason, to reduce the effect of floating-point error in gradient calculation, a gradient test is conducted using a 64-bits data type.

The silicon nitride nano cylinder shown in Fig. 9a is used. This structure has resonance according to the radius change, as shown in Fig. 9b. The differentiation of transmittance with respect to R is calculated in four ways: (1) automatic differentiation without broadening parameters, (2) numerically differentiation with and $2\Delta x = 0.01$ nm, (3) automatic differentiation with broadening parameter of 4.9×10^{-324}, and (4) automatic differentiation with broadening parameter of 10^{-10}. Fig. 9c shows the calculated differentiation for each case. It can be seen that the computed differential values are almost identical. The relative errors compared to case (1) are displayed to distinguish

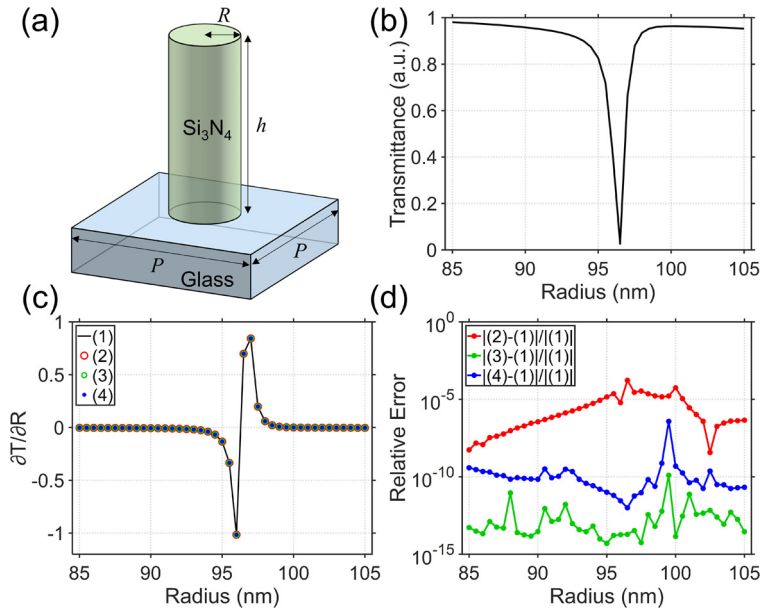

Fig. 9. (a) Structure schematic. (b) Transmittance with respect to radius. (c) Differentiation of transmittance with respect to radius. (d) Relative error. Simulation is performed with the wavelength of 473 nm, $h = 600$ nm, and $P = 300$ nm.

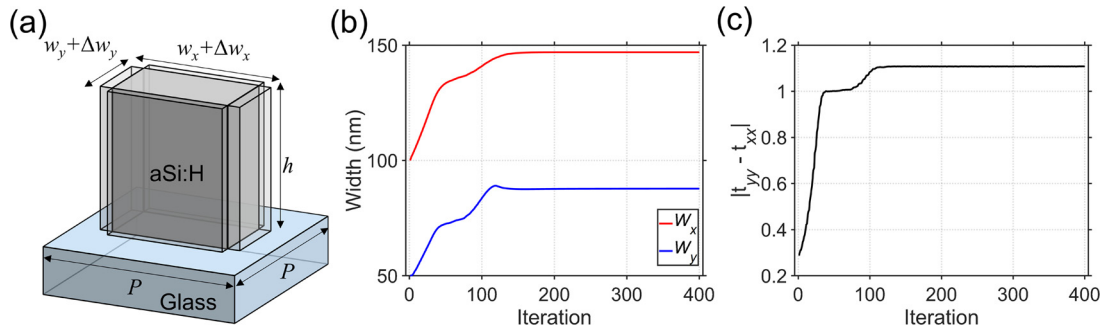

Fig. 10. (a) Structure schematic. (b) Change of width according to iteration. (c) Performance evaluation according to iteration. Optimization is performed with the wavelength of 532 nm, $h = 250$ nm, and $P = 300$ nm.

the difference clearly, as shown in Fig. 9d. Numerically calculated gradients are almost identical, with a maximum relative error of 10^{-4}. In the case of (4), when a minimum broadening parameter is inserted, the relative errors of 10^{-13} are generally shown. Even when a broadening parameter of 10^{-10} is inserted, case (3) shows the relative errors of 10^{-10}. Therefore, it can be confirmed that the gradient calculation is almost accurate even if the broadening parameter is added. In addition, even though the broadening parameter is 0, it is stable and nearly coincides with the numerical gradient. Theoretically, the nano cylinder structure has degenerated eigenvalue, but a numerical error causes a slight difference in the eigenvalue calculation. Since the denominator of Equation (32) is not completely 0, unstable calculation results do not come out.

5.2. Meta-optics inverse design

$$\nabla \times \mathbf{E} = j\omega\mu\,(x, y)\,\mathbf{H} \tag{37}$$

$$\nabla \times \mathbf{H} = -j\omega\varepsilon\,(x, y)\,\mathbf{E} \tag{38}$$

The desired functionality of the photonic device is the function of **E** and **H**. In Equations (37) and (38), the shape of the device or the distribution of permittivity and permeability, the unknown variable, should be inversely calculated from **E** and **H**. For inverse computation, various optimization techniques can be applied. TORCWA provides the gradient of how the distribution of permittivity and permeability should change to make close to the desired electromagnetic field distributions. Therefore, gradient-based inverse design is possible for nanophotonic devices, including metasurfaces.

In this section, the broadening parameter for the gradient calculation utilizes 1.4×10^{-45}, the default broadening parameter of FP32 (torch.complex64). The first optimization example is the shape derivative method that updates the width parameters of the intuitively shaped meta-atoms. The optimization is performed with the TFO of $441 (= 21 \times 21)$. Here, two width parameters of the rectangular structure are set as optimization parameters, as shown in Fig. 10a. The evaluation function is set as the distance between two transmission coefficients, t_{xx} and t_{yy}. For the circularly polarized light, the transmission efficiency of the cross-polarized light is determined by this evaluation function. Two width parameters are optimized by the ADAM optimizer with 400 iterations. The evolution of two width parameters and evaluation function are shown in Figs. 10b and 10c, respectively. It can be seen that w_x and w_y change to increase the distance

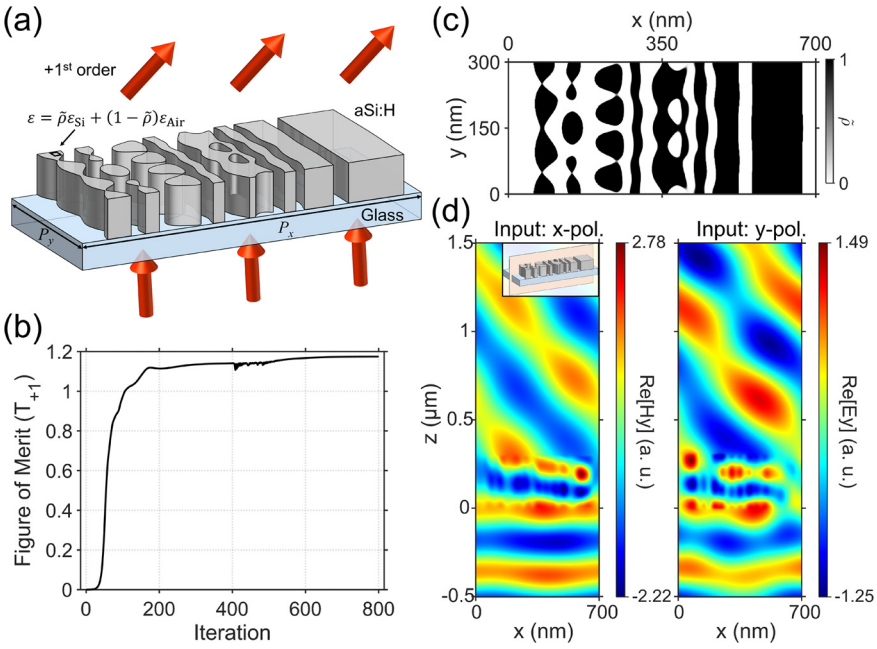

Fig. 11. (a) A schematic illustration of a freeform structure and desired functionality. Optimization is performed with the wavelength of 532 nm, $h = 300$ nm, $P_x = 700$ nm, and $P_y = 300$ nm. (b) Performance evaluation according to iteration. (c) Optimized metasurface device. (d) Electromagnetic field distribution of the optimized devices.

and converge at the end of the iterations. Here, optimization is shown only for two widths, but it is possible to optimize with height as a parameter.

$$g = A\exp\left[\frac{-\left(x\,(m)^2 + y\,(n)^2\right)}{r_0^2}\right] \tag{39}$$

$$\overline{\rho} = \rho * g \tag{40}$$

$$\widetilde{\rho} = \frac{1}{2} + \frac{\tanh\left[2\beta_t\overline{\rho} - \beta_t\right]}{\tanh\left[2\beta_t\right]} \tag{41}$$

The second optimization example is metasurface with a freeform structure. The second example utilizes the TFO of 527(=31x17, 31 in the x-direction and 17 in the y-direction). The permittivity ratio distribution is updated to improve the result. As shown in Fig. 10a, the relative permittivity at each point is defined by the ratio $\widetilde{\rho}$ between silicon and air. Before, $\rho\,(x, y) \in [0, 1]$ is defined as an optimization parameter at each point. This parameter can be utilized as permittivity ratio distribution instead of $\widetilde{\rho}$ in Fig. 11a, but additional operations are performed to consider the fabrication feasibility. Since small structures and narrow gaps are challenging to fabricate, pattern ρ is blurred with Gaussian kernel as Equations (39) and (40) [8,27]. Here, r_0 is blurring radius, and A is the normalization factor making the sum of ρ and the sum of $\overline{\rho}$ are equal. Also, the permittivity ratio value must be separated into 0 or 1 for the actual metasurface device. Inserting a binary push function such as Equation (41) would be the solution [8,27]. Here, β_t is a binarization factor and increases with iteration, making it close to 0 or 1.

The evaluation function is defined as the $+1^{st}$ order transmission efficiency for arbitrary polarization. The diffraction efficiency of the freeform metasurface for each iteration and the final $\widetilde{\rho}$ distribution are shown in Figs. 11b and c, respectively. The FoM is improved and converged according to iteration. Figs. 11d shows the electromagnetic field distribution on the xz-plane, and it can be confirmed that the electromagnetic wave propagates with the targeted diffraction order. Both optimization problems showed improvement for the evaluation function. It indicates that the TORCWA correctly computes the gradient of the FoM and shows that it can be used for optimization problems.

6. Conclusion

In this paper, we introduce an RCWA-based electromagnetic wave simulation tool, TORCWA, featuring GPU-accelerated simulation and reverse-mode automatic differentiation. TORCWA is written with Python programming language and version 1.10.1 of PyTorch library. It can simulate nanophotonic devices with a significantly improved speed due to the utilization of GPU. Therefore, it is particularly suitable when a large amount of simulation is required. In addition, it provides the calculation of the gradient of optical device parameters for the entire simulation and the evaluation function of the simulation.

It has been verified that TORCWA agrees with conventional FEM or FDTD simulations. While providing almost the same results, it supports simulation operations about ten times faster in a typical desktop environment and five times faster in a workstation environment. In addition, it calculates the gradient for the nanophotonic device parameters and optimizes them by GPU-accelerated reverse-mode automatic differentiation.

Although metasurface is shown as an example in this paper, simulation and optimization are possible for periodic structures such as 3D periodic metamaterial, photonic crystal, and diffraction grating. Moreover, further extensions of the algorithms make this software possible to address more general photonic problems such as waveguides. With GPU-accelerated simulation and gradient calculation characteristics, TORCWA is a suitable program for nano-optics for accelerated simulation and optimization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2B5B02002730) and Samsung Electronics Co., Ltd (IO201214-08164-01).

References

[1] N. Yu, F. Capasso, Nat. Mater. 13 (2014) 139–150.
[2] J. Sung, G.-Y. Lee, B. Lee, Nanophotonics 8 (2019) 1701–1718.
[3] P. Lalanne, S. Astilean, P. Chavel, E. Cambril, H. Launois, Opt. Lett. 23 (1998) 1081.
[4] S. Astilean, P. Lalanne, P. Chavel, E. Cambril, H. Launois, Opt. Lett. 23 (1998) 552.
[5] P. Lalanne, J. Opt. Soc. Am. A 16 (1999) 2517.
[6] P. Lalanne, S. Astilean, P. Chavel, E. Cambril, H. Launois, J. Opt. Soc. Am. A 16 (1999) 1143.
[7] M.-S.L. Lee, P. Lalanne, J.-C. Rodier, E. Cambril, Opt. Lett. 25 (2000) 1690.
[8] D. Sell, J. Yang, S. Doshay, R. Yang, J.A. Fan, Nano Lett. 17 (2017) 3752–3757.
[9] G. Zheng, H. Mühlenbernd, M. Kenney, G. Li, T. Zentgraf, S. Zhang, Nat. Nanotechnol. 10 (2015) 308–312.
[10] A. Arbabi, Y. Horie, M. Bagheri, A. Faraon, Nat. Nanotechnol. 10 (2015) 937–943.
[11] G.-Y. Lee, G. Yoon, S.-Y. Lee, H. Yun, J. Cho, K. Lee, H. Kim, J. Rho, B. Lee, Nanoscale 10 (2017) 4237–4245.
[12] J. Sung, G.-Y. Lee, C. Choi, J. Hong, B. Lee, Nanoscale Horizons 5 (2020) 1487–1495.
[13] J. Jang, G. Lee, J. Sung, B. Lee, Adv. Opt. Mater. 9 (2021) 2100678.
[14] D. Lin, P. Fan, E. Hasman, M.L. Brongersma, Science 345 (2014) 298–302.
[15] M. Khorasaninejad, W.T. Chen, R.C. Devlin, J. Oh, A.Y. Zhu, F. Capasso, Science 352 (2016) 1190–1194.
[16] G.-Y. Lee, J.-Y. Hong, S. Hwang, S. Moon, H. Kang, S. Jeon, H. Kim, J.-H. Jeong, B. Lee, Nat. Commun. 9 (2018) 4562.
[17] S. Wang, P.C. Wu, V.-C. Su, Y.-C. Lai, M.-K. Chen, H.Y. Kuo, B.H. Chen, Y.H. Chen, T.-T. Huang, J.-H. Wang, R.-M. Lin, C.-H. Kuan, T. Li, Z. Wang, S. Zhu, D.P. Tsai, Nat. Nanotechnol. 13 (2018) 227–232.
[18] W.T. Chen, A.Y. Zhu, V. Sanjeev, M. Khorasaninejad, Z. Shi, E. Lee, F. Capasso, Nat. Nanotechnol. 13 (2018) 220–226.
[19] C. Kim, S.-J. Kim, B. Lee, Opt. Express 28 (2020) 18059–18076.
[20] Z. Li, P. Lin, Y.-W. Huang, J.-S. Park, W.T. Chen, Z. Shi, C.-W. Qiu, J.-X. Cheng, F. Capasso, Sci. Adv. 7 (2021) eabe4458.
[21] Y. Kim, G. Lee, J. Sung, J. Jang, B. Lee, Adv. Funct. Mater. 32 (2022) 2106050.
[22] R.L. Haupt, IEEE Antennas Propag. Mag. 37 (1995) 7–15.
[23] M.P. Bendsøe, O. Sigmund, Topology Optimization, Theory, Methods, and Applications, Springer, 2004.
[24] J.S. Jensen, O. Sigmund, Laser Photonics Rev. 5 (2011) 308–321.
[25] M.M.R. Elsawy, S. Lanteri, R. Duvigneau, J.A. Fan, P. Genevet, Laser Photonics Rev. 14 (2020) 1900445.
[26] J.A. Fan, Mater. Res. Soc. Bull. 45 (2020) 196–201.
[27] P. Camayd-Muñoz, C. Ballew, G. Roberts, A. Faraon, Optica 7 (2020) 280.
[28] M. Mansouree, H. Kwon, E. Arbabi, A. McClung, A. Faraon, A. Arbabi, Optica 7 (2020) 77.
[29] M. Minkov, I.A.D. Williamson, L.C. Andreani, D. Gerace, B. Lou, A.Y. Song, T.W. Hughes, S. Fan, ACS Photonics 7 (2020) 1729–1741.
[30] W. Jin, W. Li, M. Orenstein, S. Fan, ACS Photonics 7 (2020) 2350–2355.
[31] Z. Shi, A.Y. Zhu, Z. Li, Y.-W. Huang, W.T. Chen, C.-W. Qiu, F. Capasso, Sci. Adv. 6 (2020) eaba3367.
[32] S. Colburn, A. Majumdar, Commun. Phys. 4 (2021) 65.
[33] Z. Li, R. Pestourie, J.-S. Park, Y.-W. Huang, S.G. Johnson, F. Capasso, Nat. Commun. 13 (2022) 2409.
[34] P.R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, V. Paillard, Nat. Nanotechnol. 12 (2017) 163–169.
[35] Z. Jin, S. Mei, S. Chen, Y. Li, C. Zhang, Y. He, X. Yu, C. Yu, J.K.W. Yang, B. Luk'yanchuk, S. Xiao, C.-W. Qiu, ACS Nano 13 (2019) 821–829.
[36] J. Hong, H. Son, C. Kim, S.-E. Mun, J. Sung, B. Lee, Opt. Express 29 (2021) 3643.
[37] D. Liu, Y. Tan, E. Khoram, Z. Yu, ACS Photonics 5 (2018) 1365–1369.
[38] S. So, J. Mun, J. Rho, ACS Appl. Mater. Interfaces 11 (2019) 24264–24268.
[39] S. So, J. Rho, Nanophotonics 8 (2019) 1255–1261.
[40] J. Jiang, J.A. Fan, Nanophotonics-Berlin 9 (2019) 1059–1069.
[41] J. Jiang, J.A. Fan, Nano Lett. 19 (2019) 5366–5372.
[42] J. Jiang, D. Sell, S. Hoyer, J. Hickey, J. Yang, J.A. Fan, ACS Nano 13 (2019) 8872–8878.
[43] S. So, T. Badloe, J. Noh, J. Bravo-Abad, J. Rho, Nanophotonics 9 (2020) 1041–1057.
[44] O.D. Miller, arXiv:1308.0212, 2013.
[45] T.W. Hughes, I.A.D. Williamson, M. Minkov, S. Fan, ACS Photonics 6 (2019) 3010–3016.
[46] M.G. Moharam, T.K. Gaylord, J. Opt. Soc. Am. 71 (1981) 811.
[47] R. Bräuer, O. Bryngdahl, Opt. Commun. 100 (1993) 1–5.
[48] M.G. Moharam, T.K. Gaylord, D.A. Pommet, E.B. Grann, J. Opt. Soc. Am. A 12 (1995) 1077.
[49] R.C. Rumpf, Prog. Electromagn. Res. B 35 (2011) 241–261.
[50] H. Kim, J. Park, B. Lee, Fourier Modal Method and Its Applications in Computational Nanophotonics, CRC Press, 2017.
[51] G. Granet, B. Guizal, J. Opt. Soc. Am. A 13 (1996) 1019.
[52] L. Li, J. Opt. Soc. Am. A 13 (1996) 1870.
[53] P. Lalanne, G.M. Morris, J. Opt. Soc. Am. A 13 (1996) 779.
[54] P. Lalanne, J. Opt. Soc. Am. A 14 (1997) 1592.
[55] T. Schuster, J. Ruoff, N. Kerwien, S. Rafler, W. Osten, J. Opt. Soc. Am. A 24 (2007) 2880.
[56] L. Li, J. Opt. Soc. Am. A 13 (1996) 1024.
[57] D.M. Whittaker, I.S. Culshaw, Phys. Rev. B 60 (1999) 2610–2618.
[58] E.L. Tan, J. Opt. Soc. Am. A 19 (2002) 1157.
[59] L. Li, J. Opt. Soc. Am. A 20 (2003) 655.
[60] H. Kim, I.-M. Lee, B. Lee, J. Opt. Soc. Am. A 24 (2007) 2313.

[61] J.P. Hugonin, P. Lalanne, arXiv:2101.00901, 2021.
[62] V. Liu, S. Fan, Comput. Phys. Commun. 183 (2012) 2233–2244.
[63] G. Yoon, J. Rho, Comput. Phys. Commun. 264 (2021) 107846.
[64] RODIS, http://photonics.intec.ugent.be/research/facilities/design/rodis/default.htm.
[65] MRCWA, http://mrcwa.sourceforge.net/, 2008.
[66] J. Kang, X. Wang, P. Bermel, C. Liu, S4: Stanford stratified structure solver, https://nanohub.org/resources/s4sim, 2014.
[67] G. Kajtar, RawDog, https://rawdogapp.weebly.com/, 2014.
[68] L. Bolla EMpy, https://github.com/lbolla/EMpy/, 2017.
[69] C. Boeddeker, P. Hanebrink, L. Drude, J. Heymann, R. Haeb-Umbach, arXiv:1701.00392, 2017.
[70] M. Seeger, A. Hetzel, Z. Dai, E. Meissner, N.D. Lawrence, arXiv:1710.08717, 2017.
[71] H.-J. Liao, J.-G. Liu, L. Wang, T. Xiang, Phys. Rev. X 9 (2019) 031041.
[72] J.D. Jackson, Classical Electrodynamics, John Wiley & Sons, 2007.
[73] TensorFlow, https://www.tensorflow.org/, 2015.
[74] PyTorch, https://pytorch.org/, 2019.
[75] JAX, http://github.com/google/jax, 2018.
[76] E. Popov, M. Nevière, J. Opt. Soc. Am. A 18 (2001) 2886.
[77] L. Li, J. Opt. Soc. Am. A 14 (1997) 2758.