2019-01-01

Formulation and Implementation of Iterative Method for Generating Spatially-Variant Lattices

Manuel Fernando Martinez
University of Texas at El Paso, manuel.f.m.soto@gmail.com

FORMULATION AND IMPLEMENTATION OF ITERATIVE METHOD FOR GENERATING SPATIALLY-VARIANT LATTICES

MANUEL FERNANDO MARTINEZ JR.

Master's Program in Electrical Engineering

APPROVED:

Raymond C. Rumpf, Ph.D., Chair

Rodrigo Romero, Ph.D.

Natasha Sharma, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

Dedication

To my girlfriend, for always keeping me focused on the goal. To my friends and family, who stood by me in this constant pursuit of knowledge. To my sibling, Aileen, for the stern words on how not to write a thesis.

FORMULATION AND IMPLEMENTATION OF ITERATIVE METHOD FOR GENERATING SPATIALLY VARIANT LATTICES

by

MANUEL FERNANDO MARTINEZ JR., B.S. E.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 2019

Acknowledgements

Special thanks to Dr. Raymond C. Rumpf for his support and guidance throughout this research endeavor. I would also like to express my great appreciation to Dr. Rodrigo Romero for his instrumental help in developing the software necessary to complete this task. Assistance provided by Alejandro Martinez in the area of parallel computing was greatly appreciated. Further software development aid was provided by Dr. Jimmy Touma from the Airforce Research Lab.

Abstract

The use of a matrix-free, memory-efficient approach to generate large-scale spatially variant lattices (SVL) was explored. A matrix-free iterative SVL generation algorithm was formulated and then implemented with a tremendous memory reduction observed. The algorithm consists of solving first-order central finite-differences along the entirety of the problem space point-by-point to obtain the grating phase function $\Phi(\vec{s})$ to which all desired spatially variant lattice properties are applied to. The algorithm was studied to identify key areas of data and task parallelism to exploit in heterogeneous computing systems consisting of clusters of central processing units (CPU) and graphics processing units (GPU) combinations. A sequential version of the algorithm was implemented in MATLAB to study memory usage; the sequential implementation proved to cut memory usage in the generation of large-scale SVLs when compared to the traditional matrix approach. Some preliminary efforts were made to map the algorithm to a GPU with the use of the *Compute Unified Device Architecture* (CUDA); these efforts are presented as the basis to improve the scaling problems inherent with traditional matrix approaches.

Table of Contents

List of Figures

Literature Review

PERIODIC STRUCTURES IN ELECTROMAGNETICS

The control over the behavior of all the properties of an electromagnetic wave has become a desired feature in fabricated structures and devices. For any control of an electromagnetic wave to occur, the medium through which it propagates cannot be homogeneous; a gradient of material properties, a set of device interfaces, or the periodic arrangement of elements is necessary to obtain control of a wave. Interest in the behavior of an electromagnetic wave when in contact with man-made structures with a periodic arrangement of macroscopic scatterers led to the development of mathematical foundations that explain the existence of stop-bands, pass-bands, and even band-gaps [1]. The use of spatially variant lattices (SVL) in electromagnetics has allowed for the generation of metamaterials [2], self-collimating photonic crystals (PCs) [3][4], apodized antennas used in synthetic aperture radar [5], optical metasurfaces [6], and photonic crystal optics [7][8], that can be bent, twisted, and functionally graded with minimization of unit cell deformation.

GENERATION OF SPATIALLY VARIANT LATTICES

The direct synthesis of volumetrically-complex spatially variant lattices (SVL) has been a largely ignored topic. Approaches to synthesize SVLs have dealt with the use of coordinate transform techniques, such as transformation optics [9][10] and optical conformal mapping [11]. Both of these techniques incorporate spatial variance with the use of coordinate transformations that result in the generation of permittivity and permeability tensors functions. These methods, however, do not provide a direct description of the physical geometry of the devices necessary to satisfy the material tensors.

Rumpf [12] derived a method that directly calculates the geometry of SVLs and prototyped the algorithm using the finite-difference method (FDM) [12]. Spatial transforms map points in one space to points in a second space so they fundamentally deform unit cells. The SVL algorithm is not based on a spatial transform at all and essentially builds a lattice from scratch following pictures of how the user wishes to spatially vary the lattice. This allows the algorithm to generate

the lattice with dramatically minimized deformations to the unit cells. Through a discrete Fourier transform, a periodic structure is decomposed into a set of planar gratings. Each of the planar gratings is spatially varied throughout the volume of the lattice and then summed to give the overall SVL. The orientation and spacing of the planar gratings are described through the grating vector function $\vec{K}(\vec{s})$. In concept, the spatially variant grating could be calculated according to $\cos\left[\vec{K}(\vec{s})\bullet\vec{s}\right]$; however, when the grating vector varies a function of position $\vec{s}$, this calculation is no longer correct. Instead, an auxiliary function is used, called the grating phase $\Phi(\vec{s})$. The grating vector function and grating phase are related through $\nabla\Phi(\vec{s})=\vec{K}(\vec{s})$. Given the grating phase, the spatially variant planar grating is calculated according to $\cos\left[\Phi(\vec{s})\right]$. The computational bottleneck in this algorithm is solving for the grating phase given the grating vector function. To date, this was accomplished using the FDM and a lower-upper (LU) decomposition. It is important to note that this method is largely dependent on the use of large matrices and matrix algebra libraries to solve for the fundamental equation.

The method described previously suffers from the major drawback of being memory-inefficient when implemented due to its reliance on matrices and linear algebra functions, thus limiting the scale to which SVLs can be synthesized. The present thesis proposes the formulation of an iterative SVL generation algorithm based on similar approaches to those used in electromagnetic simulation tools, such as the finite-difference time-domain (FDTD) method, in which an update equation is used to explicitly solve the underlying system of partial differential equations point-by-point throughout the problem space. This approach to the generation of SVLs allows for the computation of much larger structures due to its matrix-free nature and is, by design, easily parallelizable in a computer system.

As an exercise to visualize the main drawbacks of the FDM method presented in the introduction of this body of work, the original matrix FDM algorithm is benchmarked against the iterative FDM algorithm. Figure 1 presents the results of the benchmark where two sets of data were compared. The first set of data presented is the execution time of each method for lattices of increasing size. The iterative FDM approach is observed to have longer execution times as the

lattice size increases when compared to the matrix FDM one. This longer execution time is believed to occur because the differences between the grating phase elements between two different iterations do not change enough to meet the convergence metric needed to stop iteration. In larger lattices than the ones presented, the grating phase terms change at a much faster pace, allowing the algorithm to reach convergence sooner. When sufficiently large lattice sizes are generated, the matrix FDM approach will execute slower due to the reliance of the method on matrices and computationally expensive LU decompositions to solve for the fundamental equation. The second set of data recorded the reported memory usage by MATLAB when generating lattices of growing numbers of unit cells; in this scenario, the major downside to the matrix FDM approach is shown in the form of exponentially growing memory usage due to its use of large matrices, whereas the iterative implementation exhibits a linear trend in memory usage because of only requiring simple calculations on array elements that scale with lattice size instead of memory inefficient LU decompositions.

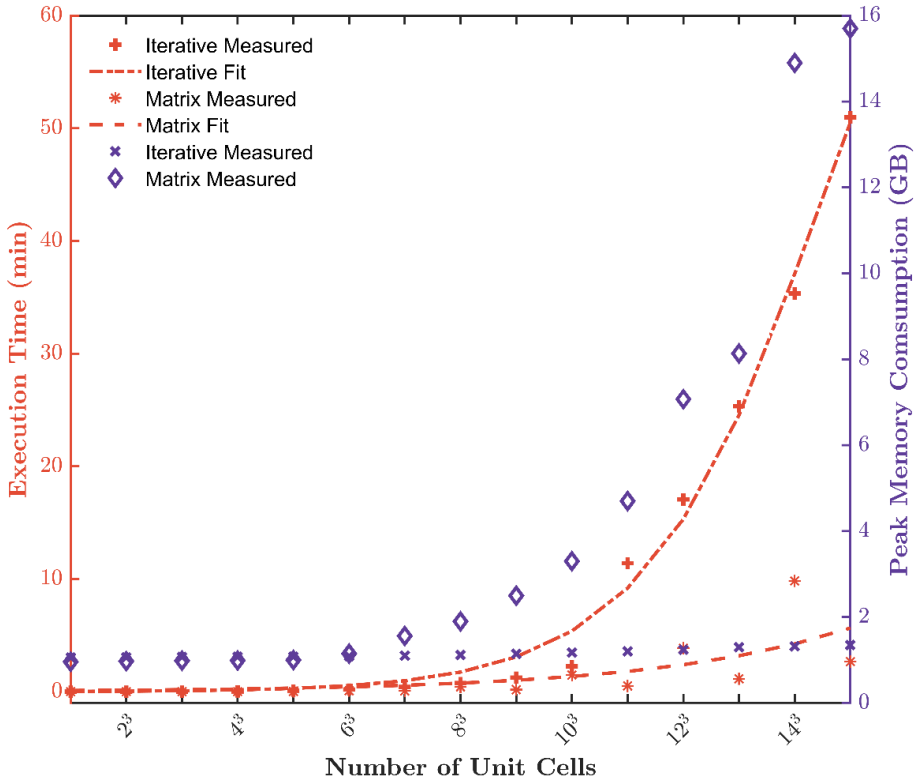


Figure 1: Execution time and memory usage data of matrix and iterative SVL solvers.

Methods

The methodology implemented for this study is divided into two sections: Formulation and Implementation. The Formulation section provides the mathematical framework and methods used to synthesize large-scale, volumetrically complex SVL structures computationally; it also describes the mathematical tools used to arrive to the desired SVL solver in detail, as well as introduces the basic workflow of the algorithm. The Implementation section describes the areas of the algorithm where parallelization inside of a heterogeneous computing system can be exploited.

FORMULATION

This section introduces the mathematical framework used in the formulation of the overall algorithm for generation of SVLs. The formulation process begins with the concepts of spatial variance on a single planar grating. Afterwards, the generalized process to incorporate spatial variance in arrays of periodic elements is explained. Finally, the necessary update equations to generate SVLs are derived.

Spatially Variant Planar Gratings

A simple sinusoidal planar grating is described by a grating vector $\vec{K}$. This vector has a direction that is perpendicular to the planes of the grating and has a magnitude of 2π divided by the period of the grating Λ. Figure 2 shows a uniform planar grating with an orientation of $\theta = 45°$ and a period of $\Lambda = 1$.

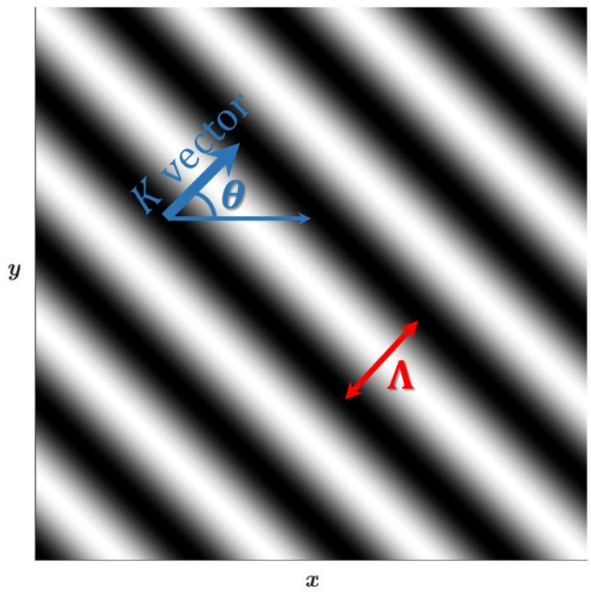


Figure 2: Illustration of how a grating vector describes the period Λ and orientation θ of a planar grating

Given a grating vector, the grating is calculated by

$$\varepsilon_a(\vec{s}) = cos(\vec{K} \cdot \vec{s}) \tag{1}$$

Where $\vec{K}$ is of the form

$$\vec{K} = K_x\hat{x} + K_y\hat{y} + K_z\hat{z} \tag{2}$$

With a magnitude

$$|\vec{K}| = \frac{2\pi}{\Lambda} \tag{3}$$

And $\vec{s}$ is

$$\vec{s} = x\hat{x} + y\hat{y} + z\hat{z} \tag{4}$$

The value $\varepsilon_a(\vec{r})$ is known as the analog grating because of its continuous variation between the values of -1 and 1.

The vector quantity $\vec{K}$ needs to become a function of position in order to spatially vary the grating by varying the period or direction. As the vector quantity $\vec{K}$ becomes $\vec{K}(\vec{s})$, the calculation in Eq. (1) fails to properly compute the analog grating [13],

$$\varepsilon_a(\vec{s}) \neq cos\left[\vec{K}(\vec{s}) \cdot \vec{s}\right] \tag{5}$$

To properly incorporate spatial variance in a planar grating, an intermediate parameter called the grating phase $\Phi(\vec{s})$ is introduced. The definition of the grating phase can be explained in the context of waves, since they have a similar mathematical form. As a wave propagates through a medium, it accumulates phase; as such, the wave can be calculated from just the phase. The grating phase is related to the grating vector through the gradient operation

$$\nabla\Phi(\vec{s}) = \vec{K}(\vec{s}) \tag{6}$$

To calculate the grating phase, it is necessary to solve Eq. (6) numerically because no analytical solution exists for this equation. Furthermore, the problem needs to be solved using a best fit because there are more attributes that are being controlled at the same time than there are degrees of freedom. In this body of work, the best fit framework being used is that of least squares due to its mathematical simplicity [14]. It is important to note that this solution as a best fit problem leads to the generation of SVLs that will usually still have some deformations, albeit greatly reduced. After numerically solving for the grating phase, the analog grating can be computed by

$$\varepsilon_a(\vec{s}) = cos[\Phi(\vec{s})] \tag{7}$$

Equation (7) is important because it provides the mathematical means through which spatial variance can be incorporated into planar gratings without failing in specific cases. In the following section, the extension from planar gratings to lattices is presented.

Spatially-variant lattices

The generalization from planar grating to full SVLs is straightforward. The single unit cell that describes the periodic structure to be spatially varied is Fourier transformed into a set of planar gratings. It is pertinent to mention that Fourier transform of a periodic array of elements is reduced into a discrete Fourier transform (DFT); therefore, the DFT is performed on the baseline unit cell to obtain each planar grating. Each of those resulting gratings are then spatially varied individually and finally summed to obtain the overall lattice.

This simple generalization to the formulation proves to be of utmost importance to generate SVLs because it allows a unit cell of arbitrary size and shape to be spatially varied throughout a lattice. This means that commonly known unit cells that show a very strong electromagnetic response can be exploited further by spatially varying them.

The update equation

At this point, we shift the focus from the underlying mathematical constructs that aide in the generation of SVLs into laying out the fundamental equations that will be used for the remainder of this body of work; the goal is to obtain an iterative solver for the $\nabla \Phi(\vec{s}) = \vec{K}$ problem. The expression in Eq. (6) can be expanded to Cartesian coordinates.

$$\frac{\partial \Phi(\vec{s})}{\partial x} + \frac{\partial \Phi(\vec{s})}{\partial y} + \frac{\partial \Phi(\vec{s})}{\partial z} = K_x(\vec{s})\hat{a}_x + K_y(\vec{s})\hat{a}_y + K_z(\vec{s})\hat{a}_z \tag{8}$$

This single vector equation can be separated into three independent scalar equations

$$\frac{\partial \Phi(\vec{s})}{\partial x} = K_x(\vec{s}) \tag{9}$$

$$\frac{\partial \Phi(\vec{s})}{\partial y} = K_y(\vec{s}) \tag{10}$$

$$\frac{\partial \Phi(\vec{s})}{\partial z} = K_z(\vec{s}) \tag{11}$$

Equation (9) can be approximated by a central finite-difference to the $i-1$ side or a central finite-difference to the $i+1$ side.

$$\frac{\Phi|_{i,j,k} - \Phi|_{i-1,j,k}}{\Delta x} = \frac{K_x|_{i,j,k} + K_x|_{i-1,j,k}}{2} \quad i-1\ side \tag{12}$$

$$\frac{\Phi|_{i+1,j,k} - \Phi|_{i,j,k}}{\Delta x} = \frac{K_x|_{i+1,j,k} + K_x|_{i,j,k}}{2} \quad i+1\ side \tag{13}$$

Solving Eqs. (12) and (13) for $\Phi|_{i,j,k}$ results in

$$\Phi|_{i,j,k} = \Phi|_{i-1,j,k} + \frac{\Delta x}{2}\left(K_x|_{i,j,k} + K_x|_{i-1,j,k}\right) \tag{14}$$

$$\Phi|_{i,j,k} = \Phi|_{i+1,j,k} + \frac{\Delta x}{2}\left(K_x|_{i+1,j,k} + K_x|_{i,j,k}\right) \tag{15}$$

Adding Eqs. (14) and (15) and solving for $\Phi|_{i,j,k}$ results in

$$\Phi|_{i,j,k} = \frac{\Phi|_{i-1,j,k} + \Phi|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_{x|_{i+1,j,k}}\right) \tag{16}$$

Equation (16) can then be interpreted as an update equation to calculate a new value of $\Phi|_{i,j,k}$ from the old values.

$$\Phi_{new}|_{i,j,k} = \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right) \tag{17}$$

By inspection, Eqs. (10)-(11) can be derived into similar equations as Eq. (16)

$$\Phi_{new}|_{i,j,k} = \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} + \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right) \tag{18}$$

$$\Phi_{new}|_{i,j,k} = \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} + \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right) \tag{19}$$

By inspection of Eqs. (17)-(19), it is clear to see that we are presented with a system of equations in which there are more equations than unknowns. Given this information, it is not possible to use this system of equations simultaneously to obtain $\Phi_{\text{new}}|_{i,j,k}$; it is necessary to solve the set of equations in the sense of least squares. To do this, Eqs. (17)-(19) are written with error terms incorporated

$$\begin{aligned}
\Phi_{new}|_{i,j,k} &= \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right) + \varepsilon_1 \\
\Phi_{new}|_{i,j,k} &= \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} + \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right) + \varepsilon_2 \\
\Phi_{new}|_{i,j,k} &= \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} + \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right) + \varepsilon_3
\end{aligned} \tag{20}$$

Solving each equation in (20) for the error term yields

$$\begin{aligned}
\varepsilon_1 &= \Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} - \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} + K_x|_{i+1,j,k}\right) \\
\varepsilon_2 &= \Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} - \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} + K_y|_{i,j+1,k}\right) \\
\varepsilon_3 &= \Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} - \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} + K_z|_{i,j,k+1}\right)
\end{aligned} \tag{21}$$

In the framework of least squares, the overall error E is

$$E = \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 \tag{22}$$

It is necessary to minimize the value in Eq. (22) by calculating $\Phi_{\text{new}}|_{i,j,k}$ with the use of the first-derivative rule

$$0 = \frac{dE}{d\Phi_{new}|_{i,j,k}}$$

$$0 = \frac{d}{d\Phi_{new}|_{i,j,k}}(\varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2)$$

$$0 = \frac{d\varepsilon_1^2}{d\Phi_{new}|_{i,j,k}} + \frac{d\varepsilon_2^2}{d\Phi_{new}|_{i,j,k}} + \frac{d\varepsilon_3^2}{d\Phi_{new}|_{i,j,k}}$$

$$0 = \frac{d}{d\Phi_{new}|_{i,j,k}}\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} - \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right)\right]^2 + \cdots$$
$$+ \frac{d}{d\Phi_{new}|_{i,j,k}}\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} - \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right)\right]^2 + \cdots$$
$$+ \frac{d}{d\Phi_{new}|_{i,j,k}}\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} - \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right)\right]^2$$

Using the derivative chain rule $\frac{df(g(x))}{dx} = \frac{df(x)}{du} \cdot \frac{du}{dx}$

$$0 = 2\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} - \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right)\right] + \cdots$$
$$+2\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} - \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right)\right] + \cdots$$
$$+2\left[\Phi_{new}|_{i,j,k} - \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} - \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right)\right]$$

Solving for $\Phi_{\text{new}}|_{i,j,k}$ yields

$$\Phi_{new}|_{i,j,k} = \frac{1}{3}\left[\frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right)\right]$$
$$+\frac{1}{3}\left[\frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} + \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right)\right] \quad (23)$$
$$+\frac{1}{3}\left[\frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} + \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right)\right]$$

Comparing the terms inside the square brackets in Eq. (23) to Eqs. (17)-(19) shows that the system of equations in Eq. (23) by least squares equals to the arithmetic mean of the value of $\Phi_{\text{new}}|_{i,j,k}$ generated by solving Eqs. (17)-(19) individually. Therefore,

$$\Phi_{new}|_{i,j,k} = \frac{\Phi_x|_{i,j,k} + \Phi_y|_{i,j,k} + \Phi_z|_{i,j,k}}{3} \tag{23}$$

Where,

$$\begin{aligned}
\Phi_x|_{i,j,k} &= \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right) \\
\Phi_y|_{i,j,k} &= \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} + \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right) \\
\Phi_z|_{i,j,k} &= \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} + \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_z|_{i,j,k+1}\right)
\end{aligned} \tag{25}$$

Incorporating boundary conditions

It is important to note that the terms in Eq. (25) fail to properly compute at the edges of the grid. The first condition is when $i = 1$ since the calculation for Eq. (25) becomes

$$\Phi_x|_{1,j,k} = \frac{\Phi_{old}|_{0,j,k} + \Phi_{old}|_{2,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{0,j,k} - K_x|_{2,j,k}\right) \tag{26}$$

This is attempting to compute $\Phi_x$when $x = 0$, which exists outside of the grid range of $1 < x < Nx.$. In a similar manner, the second condition for improper computation occurs when $i = N_x$. The calculation then becomes,

$$\Phi_x|_{N_x,j,k} = \frac{\Phi_{old}|_{Nx-1,j,k} + \Phi_{old}|_{N_x+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{N_x-1,j,k} - K_x|_{N_x+1,j,k}\right) \tag{27}$$

Where $\Phi_{\text{old}}|_{N_x+1,j,k}$ and $K_x|_{N_x+1,j,k}$ are attempting to access values of Φ_{old} and K_x when x is outside of the upper limit set by N_x. To compensate for this, it is necessary to reformulate the update equation in (25); this expression was obtained by adding Eqs. (14) and (15). To properly reformulate, only Eq. (15) is interpreted as being the update equation.

$$\Phi_x|_{1,j,k} = 4\Phi_{old}|_{2,j,k} - 2\Delta x\left(K_x|_{2,j,k} + K_x|_{1,j,k}\right) \tag{28}$$

In a similar fashion, the infringing terms in Eq. (27) are ignored in Eqs. (14) and (15); Eq. (15) is then interpreted as the update equation:

$$\Phi_x|_{N_x,j,k} = 4\Phi_{old}|_{N_x-1,j,k} + 2\Delta x\left(K_x|_{N_x,j,k} + K_x|_{N_x-1,j,k}\right) \tag{29}$$

To complete the formulation for Φ_x, the terms in Eqs. (25), (28), and (29) are used to yield

$$\Phi_x|_{i,j,k} = \begin{cases} 4\Phi_{old}|_{2,j,k} - 2\Delta x\left(K_x|_{2,j,k} + K_x|_{1,j,k}\right) & \text{for } i = 1 \\ \frac{\Phi_{old}|_{i-1,j,k} + \Phi_{old}|_{i+1,j,k}}{2} + \frac{\Delta x}{4}\left(K_x|_{i-1,j,k} - K_x|_{i+1,j,k}\right) & 2 \le i \le N_x - 1 \\ 4\Phi_{old}|_{N_x-1,j,k} + 2\Delta x\left(K_x|_{N_x,j,k} + K_x|_{N_x-1,j,k}\right) & \text{for } i = N_x \end{cases} \quad (30)$$

By inspection of Eq. (30), the expressions for Φ_y and Φ_z become,

$$\Phi_y|_{i,j,k} = \begin{cases} 4\Phi_{old}|_{i,2,k} - 2\Delta y\left(K_y|_{i,2,k} + K_y|_{i,1,k}\right) & for\ j = 1 \\ \frac{\Phi_{old}|_{i,j-1,k} + \Phi_{old}|_{i,j+1,k}}{2} + \frac{\Delta y}{4}\left(K_y|_{i,j-1,k} - K_y|_{i,j+1,k}\right) & 2 \le j \le N_y - 1 \\ 4\Phi_{old}|_{i,N_y-1,k} + 2\Delta y\left(K_y|_{i,N_y,k} + K_y|_{i,N_y-1,k}\right) & for\ j = N_y \end{cases} \quad (31)$$

$$\Phi_z|_{i,j,k} = \begin{cases} 4\Phi_{old}|_{i,j,2} - 2\Delta z\left(K_z|_{i,j,2} + K_z|_{i,j,1}\right) & for\ k = 1 \\ \frac{\Phi_{old}|_{i,j,k-1} + \Phi_{old}|_{i,j,k+1}}{2} + \frac{\Delta z}{4}\left(K_z|_{i,j,k-1} - K_x|_{i,j,k+1}\right) & 2 \le k \le N_z - 1 \\ 4\Phi_{old}|_{i,j,N_z-1} + 2\Delta z\left(K_z|_{i,j,N_z} + K_z|_{i,j,N_z-1}\right) & for\ k = N_z \end{cases} \quad (32)$$

IMPLEMENTATION

In this section, the step-by-step description of the overall algorithm to generate SVLs is presented. The first subsection discusses the input maps that define the spatial variance to be implemented. The following sub-sections define the steps necessary to synthesize a SVL as well as presenting the flow-diagram of the algorithm for ease of implementation on a computer system.

Algorithm inputs

There are two pieces of data needed by the algorithm to synthesize SVLs, the baseline unit cell and the functions defining the spatial variance of the lattice parameters.

Construction of baseline unit cell

The baseline unit cell describes the geometry of the singular element in a periodic array to be spatially varied. Figure 3 shows an example of what type of unit cell data is expected as an

input for the algorithm. To simplify the implementation of the algorithm, the function describing the unit cell $\varepsilon_{\mathrm{uc}}(\vec{s})$ is initialized to all zeros and any coordinate where dielectric exists is set to 1.

Figure 3: Baseline unit cell

The constructed unit cell is then decomposed into a complex Fourier series via a fast Fourier transform (FFT). The resultant coefficients in each term of the series can be interpreted as a one-dimensional (1D) sinusoidal grating. Since the Fourier expansion results in an infinite number of terms, the decomposition of the unit cell needs to be truncated to a finite number of harmonics, M, for it to be feasible to implement on a computer. Thus, the unit cell function $\varepsilon_{\mathrm{uc}}(\vec{s})$ becomes,

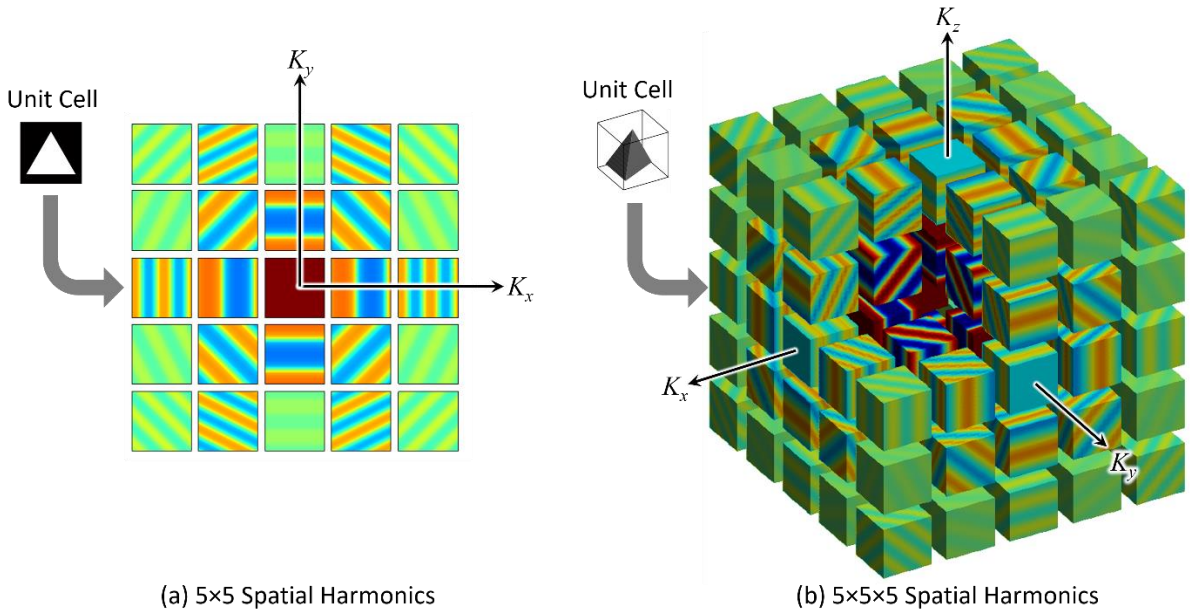
$$\varepsilon_{uc}(\vec{s}) = \sum_{m=1}^{M} \alpha_m e^{j\vec{K}_m \cdot \vec{s}} \tag{33}$$

Where $\vec{s}$ is the position, α_m is the complex Fourier coefficient of the m^{th} spatial harmonic, and $\vec{K}_m$ is the grating vector associated with the m^{th} spatial harmonic. The associated grating vectors are calculated with

$$\vec{K}_m = \frac{2\pi p}{\Lambda_x}\hat{x} + \frac{2\pi q}{\Lambda_y}\hat{y} + \frac{2\pi r}{\Lambda_z}\hat{z} \tag{34}$$

Where p, q, and r are integers that represent the indices of the spatial harmonics and Λ_x, Λ_y, and Λ_z are the unit cell dimensions in the x, y, and z directions.

The decomposition of the unit cell shown in Figure 3 into its spatial harmonics is shown in Figure 4 for both two-dimensional (2D) and three-dimensional (3D) cases. For the remainder of this body of work, the final generated lattice will consist of the summation of three planar gratings, resulting in a simple cubic unit cell.



(a) 5×5 Spatial Harmonics (b) 5×5×5 Spatial Harmonics

Figure 4: Spatial harmonic decomposition for a) 2D and b) 3D unit cells [12]

Define spatially variant functions

Of all the lattice parameters that can be spatially varied, the period, orientation, and fill fraction of each unit cell seem to be the most common. A function that defines each parameter of interest needs to be constructed before the implementation of the algorithm. This is done by generating maps, either two-dimensional or three-dimensional, where each point in space describes the variation that is desired. For illustration purposes, the maps shown in Figure 5 represent sample spatial variance maps for a 2D lattice in which the lattice spacing is spatially varied radially, the orientation of each unit cell is slightly bent along the x-axis, and the unit cell's fill fraction is

linearly graded from left to right. It is important to note that any arbitrary pattern can be used to control the spatial variance of these parameters.

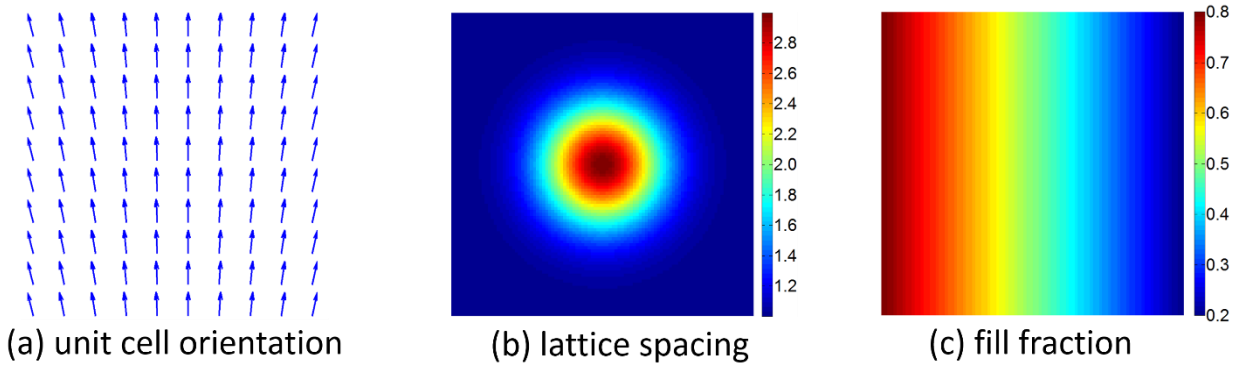


Figure 5: Spatially variant input functions for 2D cases [12]

Iterative Synthesis of SVLs

The proposed algorithm generates the overall lattice by spatially varying each planar grating for the decomposed unit cell. The algorithm described below illustrates the steps to spatially vary each harmonic.

Generate spatially variant $\vec{K}$ function

To compute the spatially variant K function, a grid that encompasses the problem space is constructed. The $\vec{K}_m$ grating vector associated with the m^{th} spatial harmonic is extracted and applied to the entirety of the grid. A simple addition operation is then performed to incorporate the tilt of the orientation function and obtain an intermediary K function. Finally, the magnitude of this intermediary K function is multiplied by the nominal lattice spacing and then further divided by the spatially variant period information to produce the final spatially variant K function. This process is summarized in Figure 6 for a 2D case.

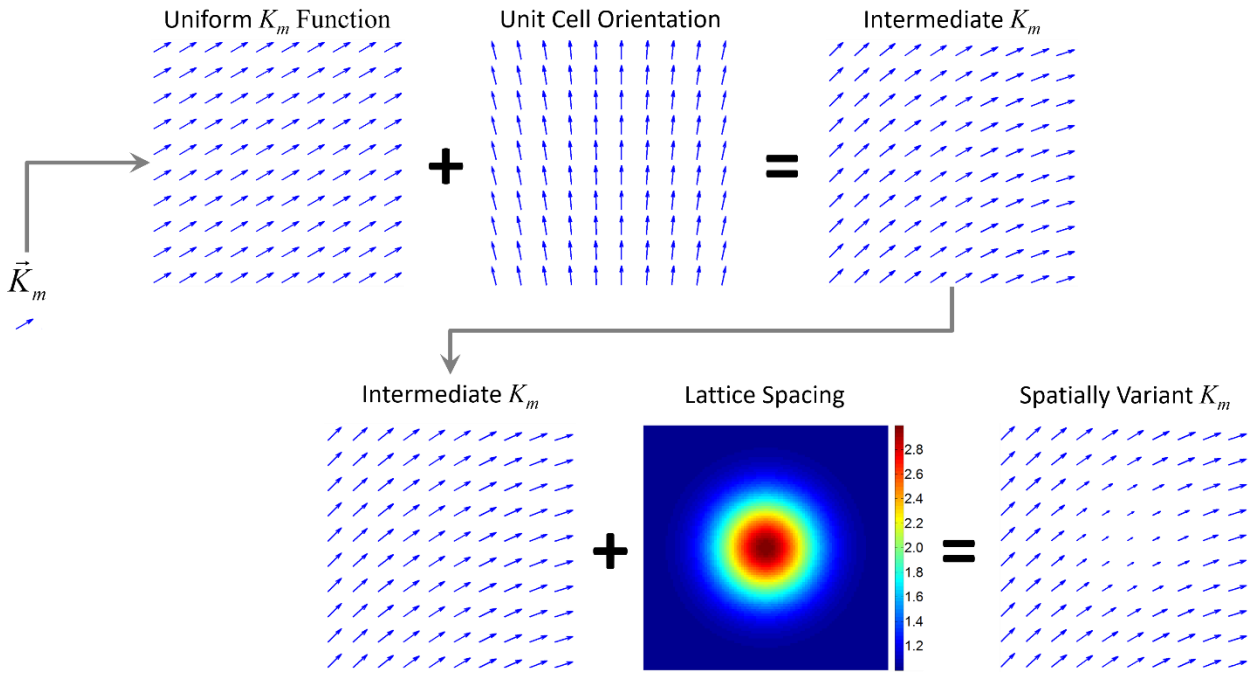


Figure 6: Generation of the spatially variant K function [12]

Calculate grating phase $\boldsymbol{\Phi(\vec{s})}$

The computation of the grating phase is carried out in three independent calculations per iteration. As shown in Eqs. (30)-(32) each individual component of $\Phi(\vec{s})$ can be calculated independently from each other. The grating phase is initialized to be all zeros; more sophisticated initialization schemes that minimize the amount of iterations needed to converge to a result might be used but are not within the scope of this work. Each point in the problem space is iterated through to calculate Φ_x, Φ_y, and Φ_z. Following the calculation of each phase component, the updated grating phase is calculated using Eq. (1). The newly obtained grating phase is then compared to the grating phase one previous iteration to compute the step increment of the problem space; the process is repeated until the problem space step increment converges to a preset threshold.

Calculate step increment

The newly obtained grating phase is then compared to the grating phase from the previous iteration to compute the overall step increment of the problem space; the process is repeated until

the problem space step increment in the grating phase falls below some predefined threshold. In this regard, the error is calculated with the use of the following set of equations:

$$\Phi_{old} = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} \Phi_{old}|_{i,j,k} \tag{35}$$

$$\Phi_{new} = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} \Phi_{new}|_{i,j,k} \tag{36}$$

$$err = \frac{(\Phi_{old} - \Phi_{new})}{I * J * K} \tag{37}$$

Compute spatially variant 1-D grating

After computing the overall grating phase for the problem space, the spatially variant grating, $\varepsilon_{\mathrm{v}}(\vec{s})$, is calculated with the use of the following equation:

$$\varepsilon_v(\vec{s}) = \alpha_m e^{i\Phi(\vec{s})} \tag{38}$$

Where α_m represents the Fourier coefficient of the m^{th} spatial harmonic, obtained from the decomposition of the baseline unit cell.

Compute overall lattice

Having calculated each spatially variant 1-D grating, the overall lattice is obtained from their sum.

$$\varepsilon_v(\vec{s}) = Re\left[\sum_{m=1}^{M} \varepsilon_v(\vec{s})\right] \tag{39}$$

The numerical noise caused by the use of the FFT and the construction of the lattice via the use of (33) may cause the dielectric values in $\varepsilon_{\mathrm{v}}(\vec{s})$ to contain a small complex component, which is negligible and is ignored with the use of only the real component of the summation.

Algorithm flowchart

Having defined the steps necessary to generate SVLs iteratively, it is useful to visualize the flow of the algorithm through a block diagram. This visualization allows for the quick understanding of the key points of the workflow and the necessary parameters that need to be

generated to properly compute SVLs. The diagram in Figure 7 shows the basic workflow for generating SVLs.

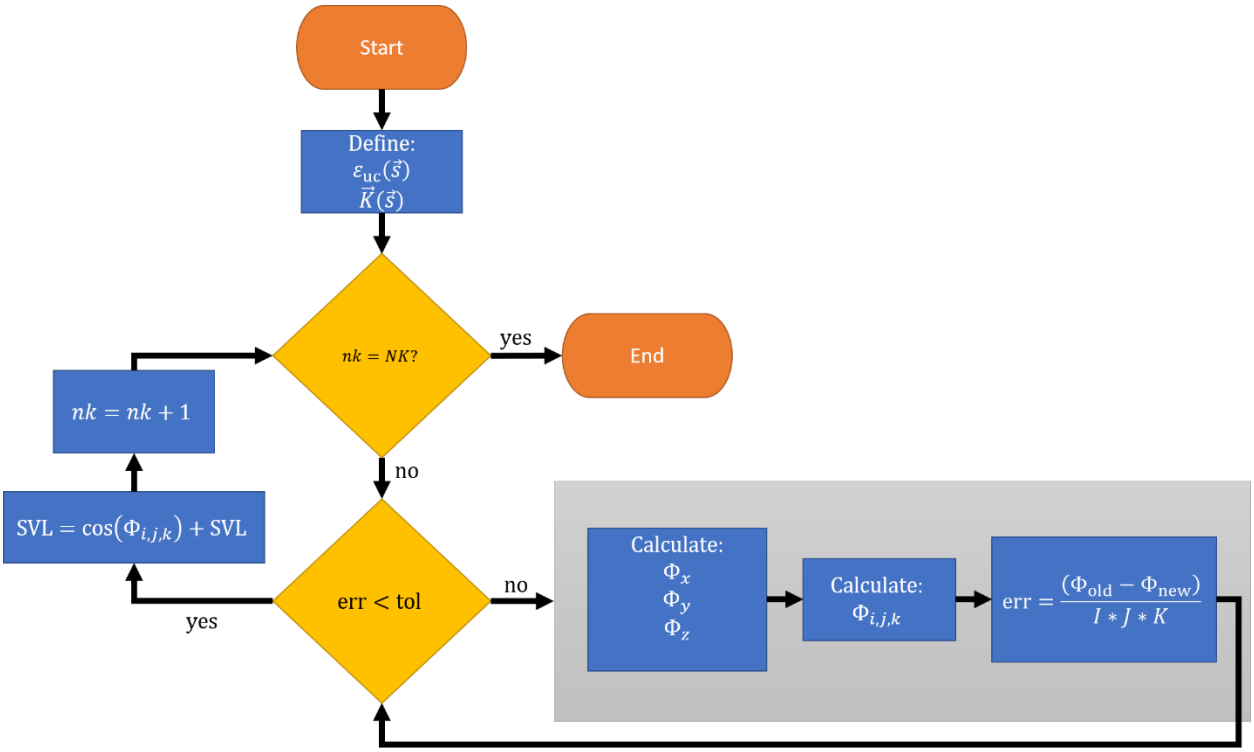


Figure 7: Algorithm flowchart

Identifying key areas of parallelism

To facilitate the implementation of the presented algorithm inside of a heterogeneous computing system, the key areas of data parallelism need to be identified. The algorithm is solving for the underlying partial differential equations in Eq. (6) throughout the entire problem space. This distribution of the calculations means that the bulk of the computations occurs whenever the algorithm is rasterizing through each point in the volume of the problem. At first glance, the computation of the auxiliary Φ_x, Φ_y, and Φ_z parameters, as well as the computation of the overall grating phase $\Phi(\vec{s})$ and the calculation of the final SVL pose ideal candidates for parallelization. Given these considerations, it is easy to begin the exploration of areas of potential parallelism by focusing on these terms.

The study of possible areas of parallelism begins with the computation of the auxiliary Φ terms. Given that these calculations in equations (30)-(32) can be performed independent from one another over each iteration, they pose a very attractive candidate for task parallelism, where each CPU core in a multi-core system handles execution of these calculations in parallel. Furthermore, to study the level of data parallelism for these equations, it is necessary to study how the algorithm handles data during these computations. For each intermediate term, the algorithm rasterizes for each point in the problem space to evaluate the expressions in equations (30)-(32); these expressions depend only on previous values of the overall $\Phi(\vec{s})$ term, the grid resolution, and information about the spatial variance maps absorbed under the K function. These calculations are therefore a prime candidate for mapping inside of a massively parallel processor such as a GPU because they can be performed as a distribution of threads that compute each individual voxel in the problem space. A thoughtful implementation of these terms inside of a GPU breaks apart the problem space into smaller sub-sections that can be loaded in fast-access memory to save the computational overhead of host-to-device memory accesses.

The following snippets of code, presented in Figure 8, aim to contextualize the areas in which the computation of these intermediate Φ pose an ideal candidate for parallelization. In Figure 8a, the sequential implementation of the iterative method for calculating Φ_x is presented as a triple *for* loop that calculates Φ_x for every point in the problem space. This sequential version of the implementation has the major drawback that for large values of $Nx, Ny,$ and Nz, the triple loop implemented takes a long time to execute. Furthermore, these computations do not depend on other values of Φ_x, they can be carried independently from each other as a GPU kernel. In Figure 8b, the GPU kernel implementation is presented where each thread in a CUDA grid is tasked with calculating a Φ_x value in space.

a)

```
for(int z = 0; int z < Nz; z++) {
    for(int y = 0; int y < Ny; y++) {
        for(int x = 0; int x < Nx; x++) {
            phix = 2 * (oPHI[z + Nz * (y + Ny * (x - 1))] + \
            oPHI[z + Nz * (y + Ny * (x + 1))]) + \
            dx*((Kx[z + Nz * (y + Ny * (x - 1))]) - \
            (Kx[z + Nz * (y + Ny * (x + 1))]));
        }
    }
}
```

```
_global__ void getPhiVol(int Nx, int Ny, int Nz, float dx, float * d_PHI, float * phix, float * d_Kx) {

    // Array Access Variables
    int x = blockIdx.x*blockDim.x + threadIdx.x + 1;
    int y = blockIdx.y*blockDim.y + threadIdx.y + 1;
    int z = blockIdx.z*blockDim.z + threadIdx.z + 1;

    if ((x < Nx-1) && (y < Ny-1) && (z < Nz-1)) {

        // Calculate phix for volume
        phix[z + Nz * (y + Ny * x)] = 2 * (d_PHI[z + Nz * (y + Ny * (x - 1))] + \
                                           d_PHI[z + Nz * (y + Ny * (x + 1))]) + \
          dx*((d_Kx[z + Nz * (y + Ny * (x - 1))]) - \
              (d_Kx[z + Nz * (y + Ny * (x + 1))]));
        __syncthreads();
    }
}
```

b)

Figure 8: a) Sequential and b) parallel kernel implementations to calculate intermediate grating phase parameters

Following the study of the intermediate grating phase terms, the focus shifts to studying the levels of parallelism available for exploitation in the calculation of the overall grating phase term in equation (24). In a similar fashion to the calculation of the intermediate grating phase terms, the overall $\Phi(\vec{s})$ term is calculated by rasterizing through the entire problem space size; the main difference lies with the terms that this calculation is dependent on. As evidenced in equation (24), the grating phase depends only on the values at each coordinate in space for the intermediate $\Phi_x, \Phi_y,$ and Φ_z terms. This equation is therefore also a prime candidate for data level parallelism and mapping inside of a GPU because it only requires memory accesses to certain points in space that are not dependent on one another.

In Figure 9a, the sequential implementation to calculate the $\Phi(\vec{s})$ parameter rasterizes through the problem space with the use of a triple loop. In a similar manner to the calculation of the intermediate Φ terms, this approach can take long execution times with large grid sizes. Given that the parameters needed to calculate $\Phi(\vec{s})$ are distributed independently through the volume of the problem gird, they can be parallelized inside of a CUDA kernel by launching a grid of threads in which each thread reads the Φ_x, Φ_y, and Φ_z values throughout space and then calculates $\Phi(\vec{s})$ as shown in Figure 9b.

a)

```
for (int z = 0; z < Nz; z++) {
    for (int y = 0; y < Ny; y++) {
        for (int x = 0; x < Nx; x++) {
          d_PHI2[z + Nz * (y + Ny * x)] = (phix + phiy + phiz) / 12;
        }
    }
}
```

b)

```
__global__ void getPhiTot(int Nx, int Ny, int Nz, float *d_PHI2, float *phix, float *phiy, float *phiz) {

    // Array Access Variables
    int x = blockIdx.x*blockDim.x + threadIdx.x;
    int y = blockIdx.y*blockDim.y + threadIdx.y;
    int z = blockIdx.z*blockDim.z + threadIdx.z;

    if ((x < Nx) && (y < Ny) && (z < Nz)) {
        // Calculate phix for volume
        d_PHI2[z + Nz * (y + Ny * x)] = (phix[z + Nz * (y + Ny * x)] + phiy[z + Nz * (y + Ny * x)] + phiz[z + Nz *
        (y + Ny * x)]) / 12;
        __syncthreads();
    }
}
```

Figure 9: a) Sequential and b) parallel implementations to calculate the overall grating phase on a volume.

The next term of interest to study for parallelism is the calculation of the error term that determines if the grating phase calculation continues or if it is interrupted. In this case, data parallelism is the main mechanism that can be exploited since it depends only on addition of terms of $\Phi(\vec{s})$ of two consecutive iterations. This section of the algorithm can be easily mapped to a GPU capable of executing the concurrent accesses to allocated space in memory. In this case, to avoid race conditions [15] on data accesses, the use of *atomic* operations was performed at the cost

of execution time. In Figure 10a, the sequential implementation, in a trend similar to the two previous calculations explored, uses triple loops to rasterize through the problem space to obtain the absolute maximum difference between two successive iterations to obtain the $\Phi(\vec{s})$ parameter. In the parallel case, shown in Figure 10b, the kernel exploits the GPU's capabilities to use many different threads to access variables to add all of the contents of Φ_{old} and Φ_{new} into single numbers. These variables are then further used in host code to obtain the absolute maximum difference between these two parameters. It is important to note that the use of *atomic* operations in this implementation has a negative effect on the performance of the algorithm, but were necessary to avoid race conditions that destroy the data present in these arrays. Further improvements to the algorithm can be obtained by completely avoiding the use of *atomics*.

a)

```
for (int z = 0; z < Nz; z++) {
    for (int y = 0; y < Ny; y++) {
        for (int x = 0; x < Nx; x++) {
            h_err = h_err + abs(d_PHI2[z + Nz * (y + Ny * x)] - d_PHI[z + Nz * (y + Ny * x)]);
          }
      }
}
```

b)

```
// Device Code
__global__ void getErr(int Nx, int Ny, int Nz, float * d_PHI, float * d_PHI2, float * d_oPhi, float * d_nPhi) {

    // Array Access Variables
    int x = blockIdx.x*blockDim.x + threadIdx.x;
    int y = blockIdx.y*blockDim.y + threadIdx.y;
    int z = blockIdx.z*blockDim.z + threadIdx.z;

    if ((x < Nx) && (y < Ny) && (z < Nz)) {

        // Sum all terms of phi
        atomicAdd(&d_nPhi[0], d_PHI2[z + Nz * (y + Ny * x)]);
        atomicAdd(&d_oPhi[0], d_PHI[z + Nz * (y + Ny * x)]);
    }
}

// Host Code
h_err = abs(h_nPhi[0] - h_oPhi[0]);
```

Figure 10: a) Sequential and b) parallel implementation to calculate overall error

The final term to explore for possible parallelism is the calculation of the overall SVL structure with the use of equation (39). In a very similar vein to the terms discussed thus far, the calculation of the overall SVL can be parallelized via the mapping of the calculations to a GPU

because of its high level of data parallelism. The SVL is initially set to all zeroes for all points in the problem space and each planar grating iteration, the cosine calculation of each element of the grating phase array is added to it; this process is repeated for NK number of spatial harmonics. In Figure 11a, the sequential implementation happens in two separate steps: the first one calculates the analog grating for the nk^{th} harmonic with a cosine operation on a point-by-point basis, the second one calculates the overall SVL by adding the nk^{th} grating to the overall SVL array. In Figure 11b, the parallel implementation is also comprised of two separate kernels: the first kernel distributes the calculation of the analog grating such that each thread in the grid calculates the cosine of the grating phase. The second step adds each value of the analog grating point-by-point to the overall SVL array.

a)

```
// Calculate Analog Grating
for (int z = 0; z < Nz; z++) {
    for (int y = 0; y < Ny; y++) {
        for (int x = 0; x < Nx; x++) {
            h_ERA[z + Nz * (y + Ny * x)] = cos(d_PHI[z + Nz * (y + Ny * x)]);
        }
    }
}

// Add to SVL
for (int z = 0; z < Nz; z++) {
    for (int y = 0; y < Ny; y++) {
        for (int x = 0; x < Nx; x++) {
            h_SVLA[z + Nz * (y + Ny * x)] = h_SVLA[z + Nz * (y + Ny * x)] + h_ERA[z + Nz * (y + Ny * x)];
        }
    }
}
```

b)

```
// Analog calculator kernel
__global__ void getERA(int Nx, int Ny, int Nz, float * d_ERA, float * d_PHI) {

    // Array Access Variables
    int x = blockIdx.x*blockDim.x + threadIdx.x;
    int y = blockIdx.y*blockDim.y + threadIdx.y;
    int z = blockIdx.z*blockDim.z + threadIdx.z;

    // Calculate ERA
    if ((x < Nx) && (y < Ny) && (z < Nz)) {
        d_ERA[z + Nz * (y + Ny * x)] = cos(d_PHI[z + Nz * (y + Ny * x)]);
    }
}

// Calculate Overall Lattice
__global__ void getSVLA(int Nx, int Ny, int Nz, float * d_ERA, float * d_SVLA, float * d_PHI) {

    // Array Access Variables
    int x = blockIdx.x*blockDim.x + threadIdx.x;
    int y = blockIdx.y*blockDim.y + threadIdx.y;
    int z = blockIdx.z*blockDim.z + threadIdx.z;

    if ((x < Nx) && (y < Ny) && (z < Nz)) {
        // Calculate overall analog SVL
        atomicAdd(&d_SVLA[z + Nz * (y + Ny * x)], d_ERA[z + Nz * (y + Ny * x)]);
        __syncthreads();
    }
}
```

Figure 11: a) Sequential and b) parallel implementations to calculate the overall SVL

The appendix contains the first steps taken to map the iterative SVL solving algorithm to a CUDA-enabled GPU.

Results

For illustration purposes, a lattice with orientation maps following a meandering line in a volume of space and a constant lattice constant function (as shown in Figure 12) was generated.

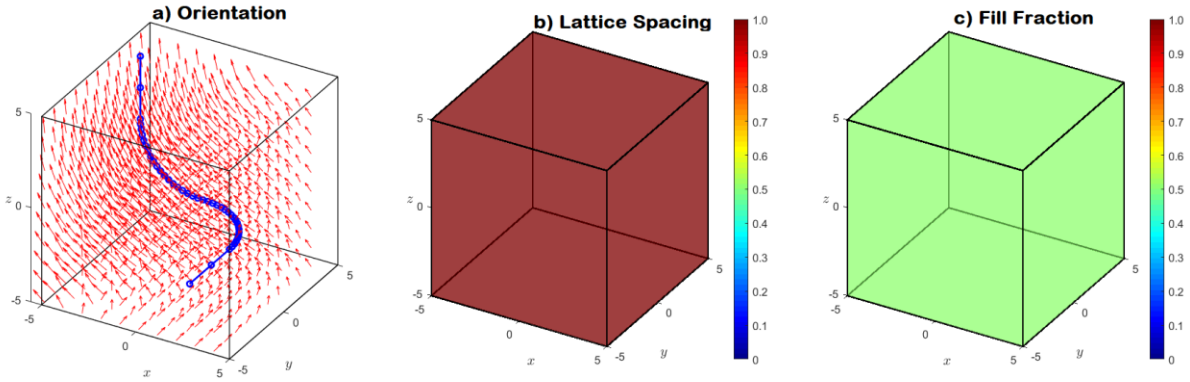


Figure 12: 3D spatial variance maps

Figure 13 shows the generated lattice with $10 \times 10 \times 10$ unit cells. As can be seen, the lattice output shows no signs of discontinuities, or changes to the overall geometry of the unit cells that compose it. As referenced in Figure 1, the total memory utilized for this lattice never exceeded 2 GB, which is an ideal characteristic of the iterative SVL solver that allows for the generation of large-scale devices.

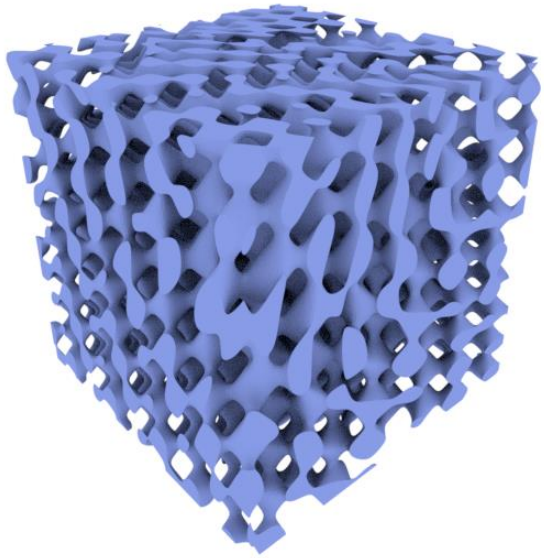


Figure 13: Double-bend lattice generated iteratively

Conclusions

The generation of SVLs with conventional finite-difference methods has proven to be memory inefficient, thus limiting the synthesis of large-scale lattices that can be of special importance in electromagnetics. In this body of work, an iterative, matrix-free, approach to compute these periodic structures was derived and the main points of parallelization were identified.

Using a similar approach to that of the finite-difference time-domain method[16], the fundamental underlying equations that allow for the generation of SVLs were derived. These equations were then organized into a cohesive algorithm that serves as the foundation for a computational implementation. This algorithm was further analyzed to understand the main areas of heavy computational tasks as well as the overall memory consumption. Finally, the possibility to map these key areas of parallelism to a heterogeneous computing system consisting of many CPU cores and GPU cores was explored to allow data scientists and distributed computing architects to easily implement the presented algorithm.

The main focus of this body of work was to develop a novel approach to generating large-scale spatially variant lattices. The algorithm derived was implemented in a sequential MATLAB script and memory usage reductions of upwards of 15 GB were observed, thus proving the viability of the proposed method to generate large-scale lattices. The method proposed succeeded at this task given its matrix-free nature and due to the avoidance of use of expensive lower-upper decompositions that the original matrix method extensively used. An execution speedup is expected at larger lattice sizes due to the tendency of the intermediate parameters calculated iteratively ($\Phi(\vec{s})$) to change more abruptly, allowing for convergence to occur at a faster pace.

Future Work

Although preliminary work was done to map the current algorithm to a CUDA-capable GPU, the expected performance boost and results were not obtained. As part of future work to continue this research, and following the identification of the key areas of parallelism within the algorithm, an efficient heterogeneous computing system application of the algorithm needs to be implemented. To achieve this with the best performance boost, a few considerations need to be accounted for. The first is that large size computations need to be automatically broken apart into smaller sections to allow the hardware to use shared memory to improve computational overhead, the second is to efficiently distribute the computing tasks between CPU cores and GPU architectures.

References

[1] E. Yablonovitch and T. J. Gmitter, "Photonic band structure: The face-centered-cubic case," *Phys. Rev. Lett.*, vol. 63, no. 18, pp. 1950–1953, 1989.

[2] C. R. Garcia, M. D. Irwin, H. H. Tsang, R. C. Rumpf, and J. E. Padilla, "Electromagnetic Isolation of a Microstrip By Embedding in a Spatially Variant Anisotropic Metamaterial," *Prog. Electromagn. Res.*, vol. 142, pp. 243–260, 2014.

[3] J. L. Digaum *et al.*, "Tight control of light beams in photonic crystals with spatially-variant lattice orientation," *Opt. Express*, vol. 22, no. 21, p. 25788, 2014.

[4] R. C. Rumpf, J. J. Pazos, J. L. Digaum, and S. M. Kuebler, "Spatially variant periodic structures in electromagnetics," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 373, no. 2049, 2015.

[5] H. C. Stankwitz, R. J. Dallaire, and J. R. Fienup, "Spatially variant apodization for sidelobe control in SAR imagery," in *Proceedings of 1994 IEEE National Radar Conference*, 1994, pp. 132–137.

[6] S. Chen, Y. Cai, G. Li, S. Zhang, and K. W. Cheah, "Geometric metasurface fork gratings for vortex-beam generation and manipulation," *Laser Photon. Rev.*, vol. 10, no. 2, pp. 322–326, Mar. 2016.

[7] Y. Jiao, S. Fan, and D. A. B. Miller, "Designing for beam propagation in periodic and nonperiodic photonic nanostructures: Extended Hamiltonian method," *Phys. Rev. E - Stat. Physics, Plasmas, Fluids, Relat. Interdiscip. Top.*, vol. 70, no. 3, p. 9, 2004.

[8] P. Halevi, A. A. Krokhin, and J. Arriaga, "Photonic crystal optics and homogenization of 2D periodic composites," *Phys. Rev. Lett.*, vol. 82, no. 4, pp. 719–722, 1999.

[9] D. H. Spadoti, L. H. Gabrielli, C. B. Poitras, and M. Lipson, "Focusing light in a curved-space," *Opt. Express*, vol. 18, no. 3, p. 3181, 2010.

[10] A. V Kildishev and V. M. Shalaev, "Engineering space for light via transformation optics," *Opt. Lett.*, vol. 33, no. 1, pp. 43–45, Jan. 2008.

[11] U. Leonhardt, "Optical Conformal Mapping," *Science (80-.).*, vol. 312, no. 5781, pp. 1777–1780, 2006.

[12] R. C. Rumpf and J. Pazos, "Synthesis of spatially variant lattices," *Opt. Express*, vol. 20, no. 14, p. 15263, Jun. 2012.

[13] R. C. Rumpf, "Lecture #18a--Synthesis of Spatially-Variant Planar Gratings." El Paso, p. 6, 2018.

[14] R. F. Ling, C. L. Lawson, and R. J. Hanson, "Solving Least Squares Problems.," *Journal of the American Statistical Association*, vol. 72, no. 360. p. 930, 2006.

[15] NVIDIA, "CUDA C Programming Guide (https://docs.nvidia.com/cuda/cuda-c-programming-guide/)," no. March, 2015.

[16] A. Taflove and S. C. Hagness, *Computational electrodynamics: the finite-difference time-domain method*, 3rd ed. Boston, MA: Artech House, 2005.

[17] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 2nd ed. Waltham, MA: Elsevier Inc., 2013.

[18] D. De Donno, a. Esposito, L. Tarricone, and L. Catarinucci, "Introduction to GPU Computing and CUDA Programming: A Case Study on FDTD [EM Programmer's Notebook]," *IEEE Antennas Propag. Mag.*, vol. 52, no. 3, pp. 116–122, 2010.

Appendix

CUDA Program Anatomy

At the core of the CUDA programming paradigm, the coexistence of a host (CPU) and one or more devices (GPUs) is expressed with the use of the programming toolkit provided by NVIDIA. The structure of a CUDA program is, therefore, a combination of pure C++ code—meant to be executed sequentially in the host—and device code that exhibits a great level of data parallelism. These pieces of data-parallel code, called *kernels,* are executed one at a time by the host code. Figure 14 shows a typical execution diagram for a CUDA C++ program; it is important to note that the programming model allows for the distribution of data parallelism inside of the kernel to be made in a *grid* containing *blocks* of *n threads* that can be mapped to n^{th} dimensional data.

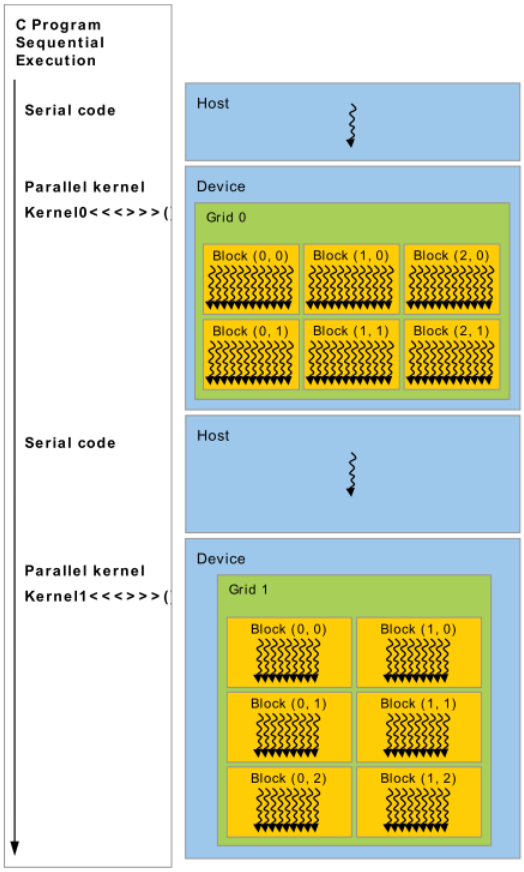


Figure 14: Heterogeneous programming model in CUDA [15]

CUDA HARDWARE IMPLEMENTATION

With the advent of high-demand, high-definition 3D graphics, graphics processing units have evolved into a highly parallel, multi-threaded, manycore processor [15]. Any CUDA-enabled GPU contains a high number of streaming multiprocessors (SMs) which in turn contain a set of streaming processors (SPs) [17]. By design, each of these processors is capable of processing large amounts of data concurrently (given the intended use of these to draw images on a computer screen) by following a single-instruction multiple-thread (SIMT) approach to scheduling the processing of data. This SIMT approach is achieved by executing multiple threads of data in *warps* that execute common instructions one at a time [15].

IMPLEMENTING CUDA-ENABLED ALGORITHM

The first step towards implementing the workflow presented in Figure 7 is to identify the sections of high data parallelism in the algorithm. At a first glance, the bulk of data processing occurs in the calculation of the $\Phi_x, \Phi_y, and\ \Phi_z$ components for the grating phase as well as the calculation of the overall $\Phi(\vec{s})$ term representing the grating phase of the entire problem space. Computation of the $\Phi_x, \Phi_y,$ and Φ_z components—according to equations (30), (31), and (32)—are independent from each other and can be therefore be easily parallelized with the use of CUDA. In the following sub-sections, the iterations taken to parallelize these sections of the algorithm are presented in chronological order.

First Iteration

As a first step towards parallelizing the algorithm, the flowchart provided in Figure 7 needs to be modified to incorporate which areas are executed in host code and device code respectively. As mentioned in the previous section, the sections of highest-data parallelism are those where the $x, y, and\ z$ components of the Φ terms are calculated. Therefore, as shown in Figure 15, all the calculations involving Φ and $\Phi(\vec{s})$ are absorbed under one kernel.

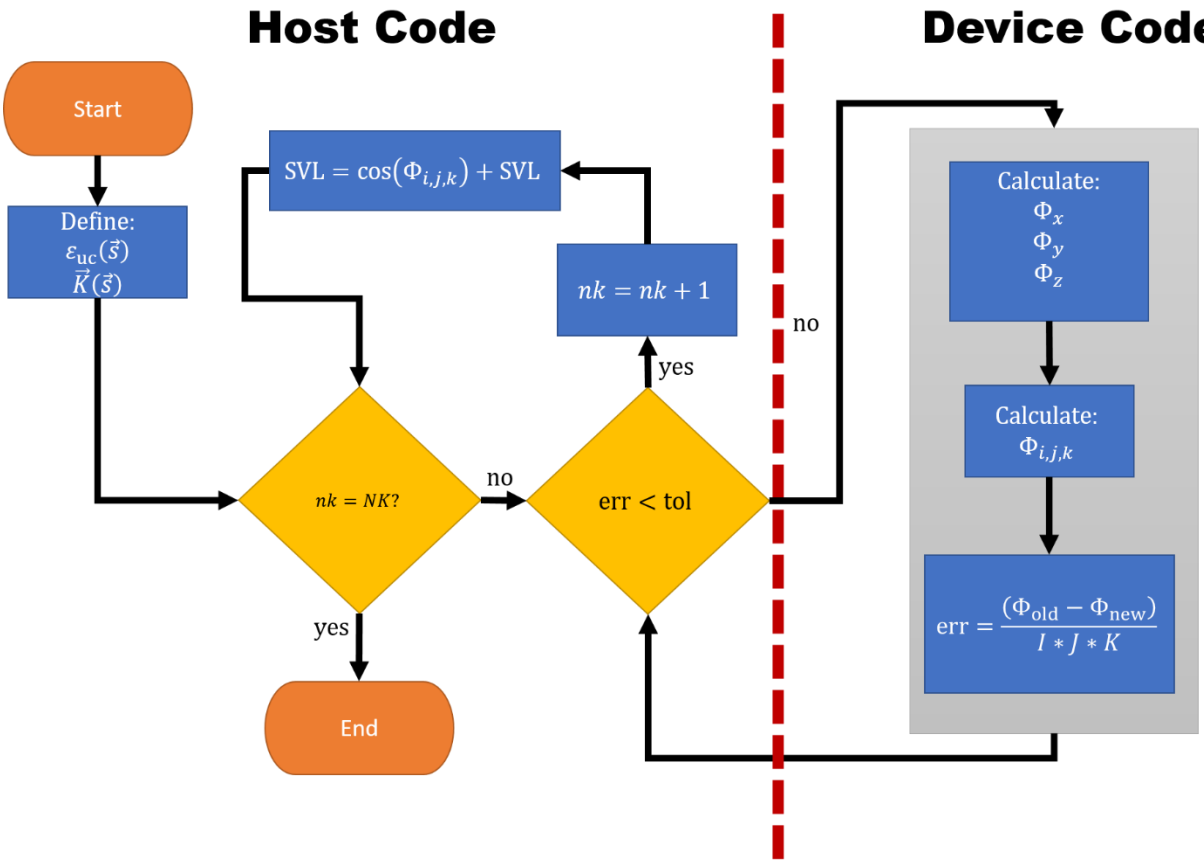


Figure 15: First iteration of parallel SVL generator

Although this provided a first good approach to parallelize the algorithm, the implementation exhibits bottlenecks on high-performance computing capabilities of a GPU.

By observing the data flow in the code, two main points of performance bottleneck can be easily identified. The first is the use of *if* statements to determine the appropriate array positions to implement Neumann boundary conditions; this causes branching divergence of the threads executed within a warp and has a negative effect on performance because the thread scheduler needs to re-launch the computational grid to accommodate for all of the divergent paths [15]. The second performance bottleneck comes from the read-write operations on large chunks of global memory space to compute the error terms, as well as to copy values of Φ_{new} to Φ_{old}.

Second Iteration

Following the identification of the main areas of performance bottleneck, it was necessary to break down the modified algorithm in Figure 15 to accommodate for the concepts of data

coalescence [18]. To perform these modifications, the calculation of all intermediate terms necessary to compute the overall SVL are separated in individual kernels to allow for better data coalescence and thread coalescence. In Figure 16,the overall algorithm is broken apart into sections of code that execute in host code and sections that execute in device code. An important difference between the information presented in Figure 15 and Figure 16 is that the computation of infringing terms at the boundaries are implemented as their own kernel to better coalesce the threads in the grid.

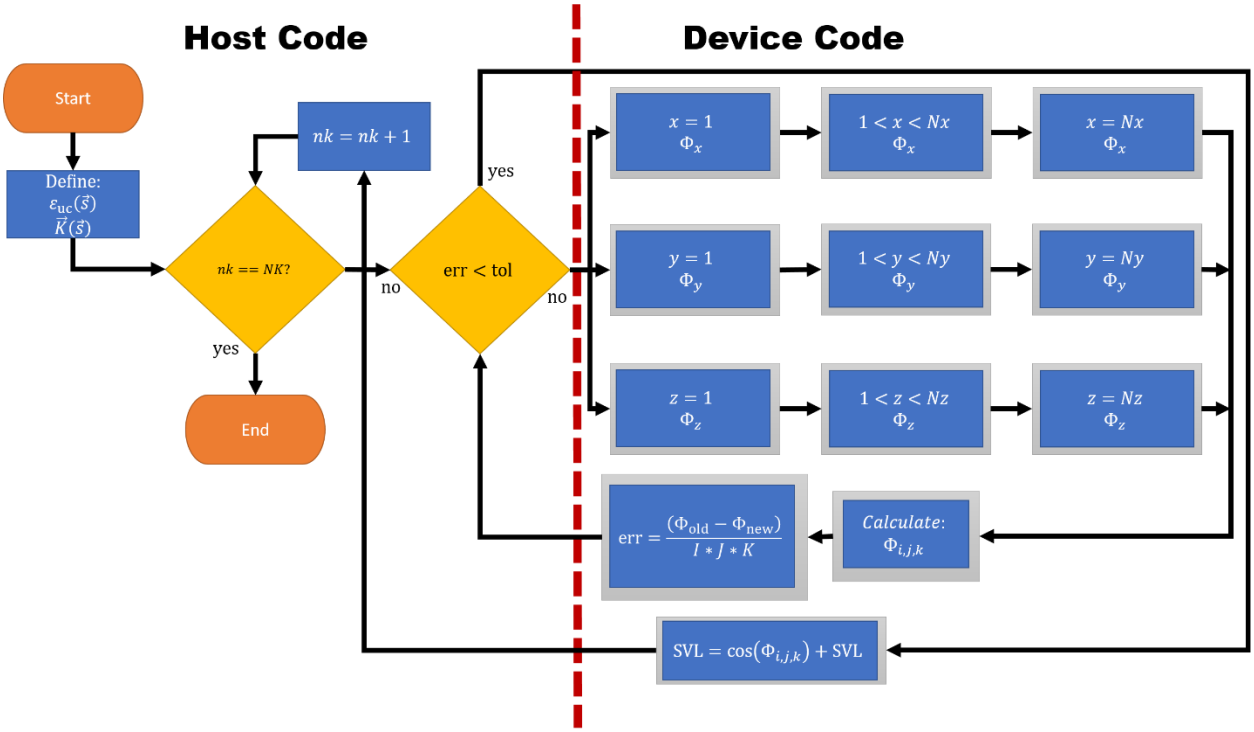


Figure 16: Adapted algorithm flowchart

Vita

Manuel Fernando Martinez Jr., born in El Paso, Texas, raised in Ciudad Juarez, Chihuahua. Studied kindergarten through high school in Juarez; during high school, he became interested in electronics and science. Upon graduating from high-school, he decided that pursuing a career in electrical engineering would provide with the necessary preparation to exploit his interests to the fullest potential. On Fall 2012, he began his college career at the El Paso Community College, where he took his first steps towards understanding the field of electrical engineering. Two semesters after, he had his first exposure to teaching in a formal environment. He became a math tutor for the remedial mathematics department at the El Paso Community College, where he taught students the basic concepts of arithmetic, college level algebra, and problem solving techniques. Soon after this first exposure, he transferred to the University of Texas at El Paso (UTEP) to narrow the scope of his studies. During his tenure at UTEP, he became interested in research, landing a position as an undergraduate research assistant under the tutelage of Dr. Deidra Hodges in the thin-film photovoltaics (TFPV) group at the electrical and computer engineering department. During his time as an undergraduate researcher, he became more passionate about the minute details of experimental setup and data analysis. This interest led to him presenting a poster at the Materials Research Society (MRS) spring meeting and symposium in Phoenix, AZ, as well as publishing his research findings in the MRS Advances Journal under the tile “Effects of Processing Parameters on Zinc Oxide Thin-Films Prepared by Single Solution Deposition.” This first approach in research led him to obtain internships in the University of Wisconsin-Madison as an undergraduate student and the Air-Force Research Lab as a graduate student.

Contact Information: mfmartinez4@miners.utep.edu