Harnessing the cloud to understand light: A whitepaper detailing the MaxwellFDS core solver

August 16, 2012

Contents

Introduction

MaxwellFDS is a cloud-powered electromagnetic solver, tailored specifically to the field of nanophotonics. MaxwellFDS stands for Frequency-Domain Solver and delivers frequency-domain solutions to the electromagnetic wave equation directly to the Matlab programming environment, and does so in a scalable manner by harnessing the power of the Amazon Elastic Compute Cloud.

In this whitepaper, the core MaxwellFDS solver is introduced.

1 Motivation: why another electromagnetic solver?

Although many electromagnetic simulation packages are currently available, we found it necessary, for our own purposes, to create our own. Specifically, we desired a simulator which

- operates in the frequency-domain, as opposed to the time-domain.
- scales to multiple high-resolution three-dimensional domains,
- is readily accessible via Matlab.

1.1 The advantages of frequency-domain solutions

The advantages of solving Maxwell's equations in the *frequency*-domain as opposed to simulating them in the *time*-domain include

- clean excitation of input modes,
- precise characterization of material dispersion,
- calculation of figure of merits outside of the simulation,
- explicit measurement of simulation error, and
- explicit calculation of eigenmodes.

1.1.1 Input: clean excitation of modes

A clean input excitation is typically quite complex to set up in a time-domain simulation, often requiring an auxiliary simulation to be run in parallel. The finite bandwidth of the excitation makes the calculation of the mode profile problematic, while the need to suppress transients requires the use of long "build-up" excitations. These difficulties are illustrated in figure 1 where the inputs at the various frequencies of interest must be combined and converted into the time-domain.

However, in a frequency-domain simulation, since ω is explicitly defined and has no bandwidth, one can accurately compute the correct input mode. Additionally, since one does not need to deal with transient excitations, the input mode can be inserted directly into the simulation without skilled "tweaking" on the part of the user, as illustrated in figure 2 where each frequency of interest is given it's own simulation.

1.1.2 Dispersion: precise characterization of frequency-dispersion

In a similar vein, the explicitness of ω means that parameters which depend upon frequency can be accurately simulated by simply supplying the correct value of the parameter at the chosen frequency.

This is in contrast to time-domain solutions, which rely on running integrals to approximate such behavior.

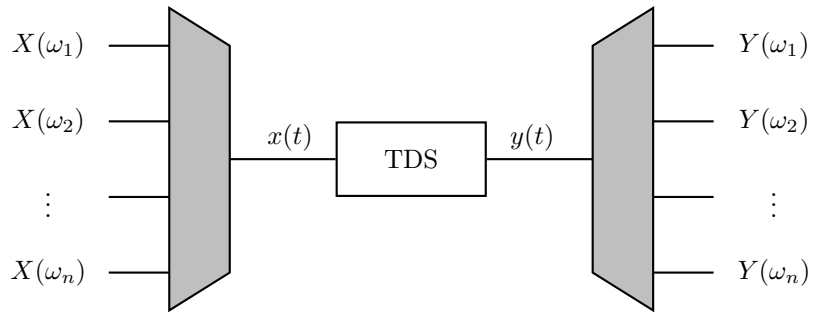

Figure 1: The difficulties of working in the time-domain. On the input side, the various input modes must be combined and converted to the time-domain. At the same time, the output measures of performance must be extracted from the time-domain output while the simulation is still running.

1.1.3 Output: complete description of device performance

An additional advantage to frequency-domain solutions is that all the information concerning the solution is contained in the output field. This means that the calculation of figures of merit and any possible measure of performance happens only after the solver has completed, as shown in figure 2.

This is in contrast to time-domain simulations, where such calculations must occur as the simulation proceeds, as shown in figure 1, since the entire field at every point in time cannot be stored in memory.

1.1.4 Error: explicit measurement of simulation error

Yet another advantage of frequency-domain solutions is the ability to explicitly calculate the error in the solution fields, which is impossible to do with time-domain simulations.

1.1.5 Eigenmodes: explicit calculation of eigenmodes

Explicitly calculating the error also allows one to explicitly calculate eigenmode solutions using frequency-domain solvers. This is particularly useful when one is attempting to discriminate between closely spaced modes, which is very difficult using time-domain approaches.

1.2 The advantages of simulating in the cloud

Although running scientific workloads in public cloud environments has not yet become the de facto standard, doing so provides many significant advantages.

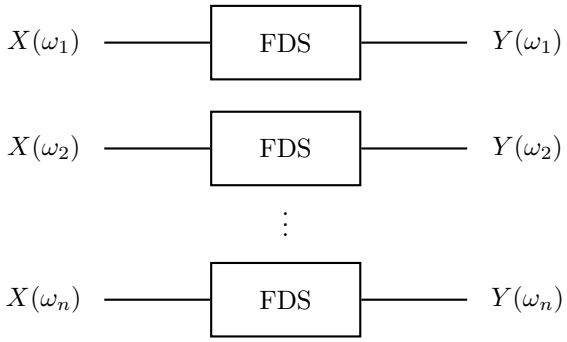

Figure 2: Using a frequency domain solver, the various input modes and output measures are computed independently of one another, greatly simplifying the user's workflow.

1.2.1 Cloud computing allows for "free" parallelism

The most fundamental difference in running simulations in a cloud computing environment is its cost structure. Specifically, in the cloud, one is billed based on the amount of computational resources used and the time which one uses those resources. In contrast, the cost of simulating on one's own hardware depends most heavily on the amount of hardware needed, at least in most scientific environments.

The disruptive cost structure of cloud environments has profound effects on scientific workloads as illustrated in figure 3; in essence, allowing scientific workloads access to "free" parallelism. The parallelism is free in the sense that the cost of running N jobs sequentially on 1 computer is equivalent to the cost of running N jobs in parallel on N computers.

1.2.2 Cloud computing takes care of all messy hardware details

A secondary benefit in using cloud computing is that many details no longer need to be handled such as

- physically installing, maintaining, and fixing computational hardware,
- ensuring identical software configurations on every cluster node, and
- installing and configuring the cluster networking.

Of course, these conveniences come at the expense of being able to customize hardware, and networking resources to ideally suit an application's needs. How-

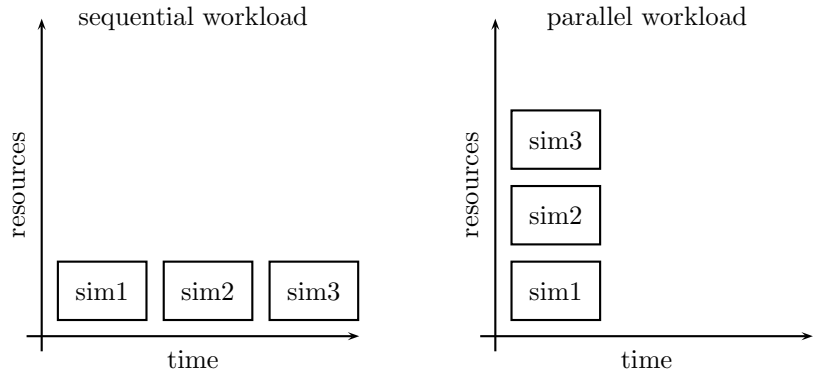

Figure 3: The fundamental difference in going to the cloud is that both sequential and parallel workloads cost the same amount of money, since one is billed based on time × resources. In contrast, simulating on one's own hardware would mean that parallel workloads are much more expensive since one is billed on resources alone. In essence, therefore, cloud computing allows for parallelism at no cost.

ever, considering that most scientific folk are not experts in these issues, this can be considered a very small disadvantage.

1.3 The advantages of being built within Matlab

The primary advantage of building MaxwellFDS within Matlab is accessibility.

Often simulation packages are run as stand-alone programs which are controlled either by custom user interfaces or custom input and output file formats. The result of this approach, however, is

- a steep learning curve for the user,
- the need to create "connection" software in order to link the simulation package with the other packages or environments the user needs, and
- increased unreliablility, since the simulation software is inevitably less "bullet-proof" than standard platforms such as Matlab.

For these reasons, the core solver of MaxwellFDS is built

- to resemble a simple Matlab command,
- never requires the user to leave the Matlab environment (assuming the user has a valid Amazon Web Services ID).

MaxwellFDS was built with extreme simplicity in mind, so much so that its core solver is even contained in a single Matlab file,

2 Harnessing Amazon's Elastic Compute Cloud

2.1 Accessing

3 Solving the electromagnetic wave equation using MaxwellFDS

3.1 Analytic derivation of the electromagnetic wave equation

The electromagnetic wave equation can be derived from the differential form of Maxwell's equations, that is,

$$\nabla \times E = -\mu \frac{\partial H}{\partial t} \tag{1a}$$

$$\nabla \times H = J + \epsilon \frac{\partial E}{\partial t}, \tag{1b}$$

where E, H, and J are the electric, magnetic and electric current vector fields, respectively, ϵ is the permittivity and μ is the permeability.

Assuming the time dependence $\exp(-i\omega t)$, where ω is the angular frequency, these become

$$\nabla \times E = -i\mu\omega H \tag{2a}$$

$$\nabla \times H = J + i\epsilon\omega E, \tag{2b}$$

which we can combine to form the time-harmonic wave equation for E,

$$\nabla \times \mu^{-1} \nabla \times E - \epsilon\omega^2 E = -i\omega J, \tag{3}$$

which MaxwellFDS solves

Note that the alternative wave equation for H, where we consider the magnetic current source M instead of J,

$$\nabla \times \epsilon^{-1} \nabla \times H - \mu\omega^2 H = -i\omega M, \tag{4}$$

can also be solved using MaxwellFDS .

3.2 Numerical discretization of the wave equation

3.2.1 Use of the Yee cell

To solve (3) we discretize our vector fields based on a primitive Yee cell, as shown in figure 4. Similarly to the finite-difference time-domain simulation technique, the use of the Yee cell allows the $\nabla\times$ operators to be well-defined.

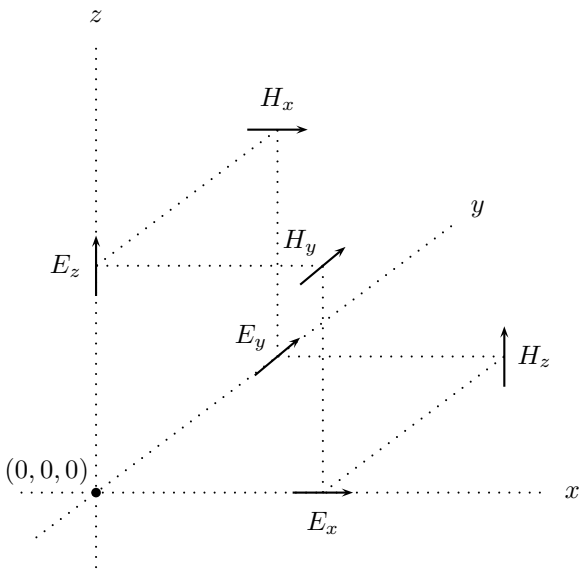

Figure 4: The primitive Yee cell. The computational grid is formed by tiling this pattern in three dimensions such that no two field components of E or H are co-located. The grid used in MaxwellFDS has periodic "wrap-around" boundary conditions, and the relevant distances between adjacent field components are denoted by $d^{\text{prim}}_{x,y,z}$ and $d^{\text{dual}}_{x,y,z}$.

In this configuration, the E, J, and ϵ vector fields are positioned on the $E_{x,y,z}$ locations while the H, M (in the case of (4)) , and μ vector fields are placed on the $H_{x,y,z}$ locations.

To define the distances between adjacent field components, MaxwellFDS uses the following convention:

- d^{prim}_w denotes the distance in the direction i between E_w components for $w = x, y, z$, and
- d^{dual}_w denotes the distance in the direction i between H_w components for $w = x, y, z$.

This definition allows us to precisely define the numerical derivatives found in the $\nabla\times$ operators.

3.2.2 Use of a periodic "wrap-around" grid

MaxwellFDS features periodic "wrap-around" boundary conditions in the definition of the $\nabla\times$ operators in (3). For example, this means that the operation $\frac{\partial}{\partial x}H_y$ is still well-defined at the $x = 0$ boundary as $(H_y|^{x=0} - H_y|^{x=x_{\max}})/d_x^{\text{prim}}$.

More specifically, MaxwellFDS calls the elements in the Yee cell at (i, j, k) as $E_x(i, j, k)$, $E_y(i, j, k)$, ... and then denotes

- $d_x^{\text{prim}}(0)$ as the distance between $E_x(x_{\max}, j, k)$ and $E_x(0, j, k)$,
- $d_y^{\text{prim}}(0)$ as the distance between $E_y(i, y_{\max}, k)$ and $E_y(i, 0, k)$,
- $d_z^{\text{prim}}(0)$ as the distance between $E_z(i, j, z_{\max})$ and $E_y(i, j, 0)$,
- $d_x^{\text{dual}}(x_{\max})$ as the distance between $H_x(x_{\max}, j, k)$ and $H_x(0, j, k)$,
- $d_y^{\text{dual}}(y_{\max})$ as the distance between $H_y(i, y_{\max}, k)$ and $H_y(i, 0, k)$, and
- $d_z^{\text{dual}}(z_{\max})$ as the distance between $H_z(i, j, z_{\max})$ and $H_y(i, j, 0)$.

It should be noted that MaxwellFDS 's strictly periodic grid still allows the use of Bloch periodic, mirror and perfectly-matched layer boundary conditions. This is accomplished by setting the $d^{\text{prim,dual}}$ values to the appropriate complex values or even ∞, both of which MaxwellFDS is able to understand.

3.2.3 The wave equation in terms of matrices and vectors

With these definitions we now can see how MaxwellFDS formulates the wave equation in the language of linear algebra. Specifically, MaxwellFDS formulates (3) as

$$(A_1 \text{diag}(\mu^{-1}) A_2 - \omega^2 \text{diag}(\epsilon))x = b, \tag{5}$$

where

- A_1 and A_2 represent the first and second $\nabla\times$ operators respectively,
- $x \to E$, and
- $b \to -i\omega J$.

The vector fields E is converted into vector x as

$$x = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \tag{6}$$

where

$$e_w = \begin{bmatrix} E_w(0,0,0) \\ E_w(1,0,0) \\ \vdots \\ E_w(x_{\max}, y_{\max}, z_{\max}) \end{bmatrix}, \tag{7}$$

and so on for all vector fields.