Using Grids to Model Nanoscopic Devices

Geoffrey W. Burr
IBM Almaden Research Center
San Jose, California 95120

• Finite-difference time-domain (FDTD)
  • directly simulate Maxwell’s equations for nanophotonics
  • as a motivation for...

• ‘Kitchen-sink’ grid computing
  • *OptimalGrid* – IBM middleware for grid computing
  • MPI - Message-Passing Interface

• Conclusions
What is FDTD?

Model Maxwell’s Equations on a discrete grid...

\[
\begin{align*}
\mu \frac{\partial \vec{H}}{\partial t} &= -\nabla \times \vec{E} \\
\epsilon_0 \epsilon_r \frac{\partial \vec{E}}{\partial t} &= \nabla \times \vec{H} - \vec{J}_{\text{source}} - \sigma \vec{E}
\end{align*}
\]

Why would we need it?

- **design-by-nanofabrication** is too expensive...
  - need accurate modeling/simulation tools

- FDTD is rigorous yet flexible (general-purpose tool)
‘Leapfrog’ update

1) at time $t$: Update $E$ fields everywhere using spatial derivatives of $H$

$$E_x = \frac{\Delta t}{\Delta y \varepsilon_r \varepsilon_0} \left( H_{z}^{j+0.5} - H_{z}^{j-0.5} \right)$$

$$E_y = \frac{\Delta t}{\Delta x \varepsilon_r \varepsilon_0} \left( H_{z}^{i+0.5} - H_{z}^{i-0.5} \right)$$

2) at time $t+0.5$: Update $H$ fields everywhere using spatial derivatives of $E$

$$H_{z} = \frac{\Delta t}{\mu} \left( \frac{E_x^{j+1} - E_x^j}{\Delta y} + \frac{E_y^i - E_y^{i+1}}{\Delta x} \right)$$

- Every cell must get updated
- Accuracy requires $\Delta x, \Delta y, \Delta z \rightarrow 0$
- Small $\Delta x, \Delta y, \Delta z$ forces small $\Delta t$ (Courant stability)
- What happens at edges? → Boundary conditions
Boundary conditions

Infinite but periodic structures

...left side becomes the righthand “neighbor” of the right side, and vice versa...

Isolated structures (by absorbing outgoing radiation)

“Perfectly Matched Layer” (PML)

Back-reflection

Absorb wave

\[ R(\theta) = \exp(-2 \sigma_x \eta_2 d \cos \theta) \]

\[ \eta_1 = \sqrt{\frac{\mu_1}{\epsilon_1}} \quad \eta_2 = \sqrt{\frac{\mu_2 \left(1 + \sigma_x^*/j\omega\mu_2\right)}{\epsilon_2 \left(1 + \sigma_x^*/j\omega\epsilon_2\right)}} \]

\[ \epsilon_2 \frac{\partial E_x}{\partial t} + \sigma_x E_x = \frac{\partial H_z}{\partial y} \]

\[ \epsilon_2 \frac{\partial E_y}{\partial t} + \sigma_x E_y = -\frac{\partial H_z}{\partial x} \]

\[ \epsilon_1, \mu_1 \quad \epsilon_2, \mu_2, \sigma, \sigma^* \]

no back-reflection

\[ e^{-j[(\beta_{1x} x + \beta_{1y} y) - \omega t]} \]

8-12 cells of increasing \( \sigma_x \)
Sample simulation (validation)

Fresnel reflection coefficients with FDTD

But periodic boundary conditions only permit a few angles...

Solution: “Floquet” boundary conditions:
(build 2-part sim: $1@\sin(\omega t)$, $1@\cos(\omega t)$ – exchange at boundary as $\exp(\pm j \mathbf{k} \cdot \mathbf{L})$)
Fresnel reflection coefficients

Error analysis: for 1% error, need cell-size $< \lambda/50$
What happens with coarser gridding?

An extreme example: a thin-film of unphysically high index

- Cell repeats periodically...
- Ultra-high index
- \( n = 10 \)

- Small spatial features get lost
- Get wrong answer AND wrong local minimum

Thickness of high index layer [nm]

Reflectivity

Standard FDTD (2-D)

2nm cells

10nm cells

5nm cells

\( \lambda_0 \approx 632.8 \text{nm} \)
“Numerical dispersion”

Dispersion relation should be: \( \left( \frac{\omega}{c} \right)^2 = (\kappa_x)^2 + (\kappa_y)^2 + (\kappa_z)^2 \)

but instead is...

\[
\left[ \frac{1}{c \Delta t} \sin \left( \frac{\omega \Delta t}{2} \right) \right]^2 = \left[ \frac{1}{\Delta x} \sin \left( \frac{\kappa_x \Delta x}{2} \right) \right]^2 + \left[ \frac{1}{\Delta y} \sin \left( \frac{\kappa_y \Delta y}{2} \right) \right]^2 + \left[ \frac{1}{\Delta z} \sin \left( \frac{\kappa_z \Delta z}{2} \right) \right]^2
\]

Phase Velocity
[fraction of \( c/n \)]

Partial solution: when index is high, adjust local \( \mu_0 \) and \( \varepsilon_0 \) (increasing speed of light w/o affecting \( \eta \))

But only an exact fix for one angle & frequency...
“Staircasing” of small spatial features

We really want to simulate a smooth interface between regions of $\varepsilon_1$ and $\varepsilon_2$...

...but the discretization in FDTD gives a “staircase” effect.

Using small grid cells only where we need them sounds ideal, but the logistics are unpleasant...

...but we can get a similar effect by using an effective $\varepsilon$ in each cell.

Each E-field component gets its own effective $\varepsilon$, which makes the most sense in terms of the field-update equations.

\[
E_y' = \frac{\Delta t}{\varepsilon_r \varepsilon_0} \left( \frac{H_x' - H_x^{\text{previous cell}}}{\Delta x} \right)
\]

\[
E_x' = \frac{\Delta t}{\varepsilon_r \varepsilon_0} \left( \frac{H_y' - H_y^{\text{previous cell}}}{\Delta y} \right)
\]
Returning to our artificial example...

Cell repeats periodically... **Ultra-high index**

(n = 10)

FDTD with field-based effective $\varepsilon$

Despite coarse gridding...

- small spatial features get modeled
- get the correct answer AND the right local minimum

Standard FDTD (2-D)

Reflectivity

Thickness of high index layer [nm]

$\lambda_0 \sim 632.8\text{ nm}$

Reflectivity

Thickness of high index layer [nm]
Returning to our artificial example...

Ultra-high index
(n = 10)

FDTD with field-based effective \( \varepsilon \)
and \((\varepsilon_0, \mu_0)\) corrected for numerical dispersion

Despite coarse gridding...

- small spatial features get modeled
- get the correct answer AND the right local minimum
Measuring frequency response

Example: Mapping out a dispersion diagram...

1) Define unit cell
2) Ping it with an impulse

3) Set k-vector at boundaries
4) Monitor at non-symmetry point
5) FFT becomes a column of the dispersion diagram

- For good frequency resolution, need lots of timesteps...

Dispersion plot

\( H_z \) vs. \( t \)

\( H_z \) vs. \( f \)

827nm

\( \text{k} \)
Example: photonic crystal band-diagrams

Square Lattice

Triangular Lattice

Air holes

Dielectric

Floquet/Bloch boundary conditions on tilted unit cell

Triangular Lattice – TE bands

\[ r/a = 0.48 \]
\[ \varepsilon = 13 \]

* 2-D FDTD (w/ corrections)

Plane-wave result

* MIT “Photonic bands” code – thanks to Bob Shelby of IBM Almaden for his help...
Custom functionality by plug-in DLL

Arbitrary structures

ArbShape

“Tell me about point x,y,z?”

Tunnel Junction .DLL
descriptor

“Media #4”
same code can gauge effective index...

Arbitrary computational steps

• VisualizeMode
• TriangularLattice

ArbStep

FDTD loop
Update E
Arbstep E
Update H
Arbstep H

Update running Fourier transform at every cell

Visualize Mode .DLL
descriptor
Metals

FDTD cannot model materials with \( \text{Re}\{\varepsilon\} < 1 \)
(such as silver at visible wavelengths)
except by also modeling material dispersion.

\[ n = 0.13455 + i 3.9163 \]

@ 632.8nm

\[ n_i \]

\[ n_r \]

\( \lambda \) [\( \mu \text{m} \)]

Indices of silver

\[ n = \frac{1}{\sqrt{\varepsilon}} \]

\[ \frac{\partial \vec{E}}{\partial t} = \nabla \times \vec{H} - \vec{J}_p \]

\[ \frac{\partial \vec{P}}{\partial t} = \vec{J}_p \]

\[ \frac{\partial \vec{J}_p}{\partial t} + \vec{J}_p = f \left( \vec{E}, \vec{P}, |E|^2, \text{etc.} \right) \]

\( \varepsilon_\infty \)

\( k = -n_i \)

\( n \& k \)

\( \lambda/\lambda_p \)

- extra variables & update rules at each metal/dispersive cell
Metals - verification

Plasmon resonance is extremely sharp:

- Cell repeats periodically...
- 50nm of silver @ $\lambda=632.8$nm
- Angle $\phi$ [degrees]

Graph showing reflectivity vs. angle for different cell periods:
- 5nm cells
- 2nm cells
- 0.5nm cells(!)

[Diagram illustrating the setup with glass, Ag, and air layers, and the PML boundaries]
FDTD in summary

**Advantages:**

- **algorithm is rigorous** — potential for arbitrarily high accuracy
- **can handle dispersive material including metals** (surface plasmons, etc.)
- **time-domain simulation**: one simulation can model broad frequency response
- **simple core algorithm +**
  - **nearest-grid-neighbor dependencies** — amenable to parallelization
- **FDTD can support**
  - finite or infinitely-periodic structures
  - arbitrary spatial arrangements of materials
  - input of pulsed, CW, or impulse waveforms
  - point-source, plane-wave, or mode-profile wavefronts
  - measurement of field, intensity, Poynting vectors
FDTD in summary

**Disadvantages:** ("no pain, no gain")

- FDTD only becomes accurate as cell-spacing → zero
- small cell spacing also mandates short time-steps (Courant stability)
- finite # of cells — boundary conditions at edges of the simulation are critical
- high frequency resolution requires many timesteps (Fourier-transform relationship)
- all simulation cells must be updated each timestep — computing just a portion is inefficient

**The sum effect:**

"to be confident you will get the right answer, 3-D FDTD simulations must be large & slow"

But this means FDTD can only be a **design verification** tool, never a **design optimization** tool

**How to change this?** • error mitigation
  • careful design of numerical experiment
  • grid computing!
Parallelized C code

Matlab script

Matlab GUI
Notes for FDTD grid-implementation...

- want to keep same Matlab script for input definition, same Matlab GUI for output analysis
- some cells (PMLs, metals) may have more data payload than others
- may need to have opposite edges act like neighbors
- sometimes need 1 ‘extra’ dimension (Floquet boundary conditions)
- support Plug-in DLLs

Also good to have:
- a GUI at input to check the simulation before running
- a quick way to estimate the memory usage, run-time, & file-space needs of a simulation
- minor feedback while a simulation is running...
- PCanywhere or equivalent, for remote administration
OptimalGrid

Matlab script → C code → Matlab GUI

Convert to text .CFG file
Convert from zipped XML files

OptimalGrid

Every cell is a Java object
coordinates, ID, class info
data payload (E-, H-fields, cell info)

Pro:
• Simpler to program
• should scale well
• may support variable grid spacing

Con:
• Space & time overheads
• Java is slow

next step here is to move to JNI
(Java Native Interface)

James Kaufman
Toby Lehman
Glenn Deen
John Thomas
Zhenghua Fu

www.almaden.ibm.com/cs/OptimalGrid.html

March APS, Y5.004
Grid computing using MPI (message-passing interface)

C code

Matlab script

Matlab GUI

Read .MAT file w/o Matlab

Recombine data files

Memory scheme

first grid cell (26 bytes)

second grid cell

Pro: • Compact, fast
    • Designed for FDTD

Con: • Only 2-D mapping to computers
    • Complexity (XY, XZ, YZ)
    • Scalability?
    • MPI implementation correct?
    • How to have variable cell spacing?

• Each XY point is a separate pointer
• Data for Z spatial dimension
  (at that x,y) is all in one array
• Additional cell-data in separate “list” structure
Grid computing using MPI

While each computer is responsible for one part of the problem-space...

every computer has the full 2-D array of pointers

Only the relevant pointers are allocated (rest are NULL)

All redundant data is updated once per timestep, but in two portions – for instance, to update E-fields we only need info from “behind” for H-fields we need the info from “ahead”

To reduce the # of data transfers, relevant memory arrays sit in contiguous memory (where possible)

Plus periodic & Floquet boundary conditions!
Two ways to use this parallelism

**Distributed mode:** 1 job split amongst 10 machines

**Master-slave mode:** 9 jobs in parallel

Problems with heterogeneous clusters:

Scaling to the worst performer:

The curse of the last job:

*Time*
Distributed mode...

...can handle really big simulations...

But how does the speedup scale?
Speed-up due to grid-computing

Heterogeneous grid:
Ten WinXP PCs,
2.4 – 3.1Ghz
Gigabit ethernet

Relative speedup

Perfect speedup

$N^{0.9}$ — ballpark figure for “good” speedup(?)

2-D mapping

1-D mapping

2 fast computers!

1 fast, 1 slow!
So where’s the missing speedup?

- Need 3-D mapping to computers?
- Coding problems?
  - portioning of cells between slow & fast computers?
  - ordering of data exchanges?
  - incorrect use of synchronous/asynchronous MPI features?
- Shared-memory multi-processor system

I prefer Master-slave mode if

- each simulation fits on 1 machine, and
- I’m going to run multiple jobs anyway (optimization, band diagrams, etc.)

Advantages:

- no redundant grid points
- little network overhead
- less post-execution re-assembly of output files
- easier to combine heterogeneous machines

<table>
<thead>
<tr>
<th>Problem size [# cells]</th>
<th>$150k</th>
<th>$15k</th>
<th>$1.5k</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^5</td>
<td>fits on 1 machine</td>
<td>fits on 10 machines</td>
<td></td>
</tr>
<tr>
<td>$10^6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallelism is good.

Caveats:
- only 1-2 simulations may fit in the aggregate memory-space
- may need to see 1st answer before designing 2nd simulation
Conclusions

- The **FDTD** (finite-difference time-domain) algorithm has many advantages as a numerical simulation tool for nanophotonics.

- Its disadvantages all boil down to:
  
  “to be confident you will get the right answer, simulations must be large & slow”

- **My motivation**: to do *design optimization* with FDTD

  need grid-computing to help do large, long simulations faster

- **MPI implementation**
  - supports all desired FDTD features
  - demonstrated on up to 60 nodes
  - for master-slave mode: ‘perfect’ scaling with # of computers

- **Future work**
  - improve speedup for distributed problems
  - JNI version for OptimalGrid
  - get ‘more’ out of small FDTD simulations