Multigrid FDTD with Chombo

Zdzisław Meglicki [a,*], Stephen K. Gray [b], and Boyana Norris [c]

[a] *Office of the Vice President for Information Technology, Indiana University, Bloomington, IN, 47405-1223*

[b] *Chemistry Division and Center for Nanoscale Materials, Argonne National Laboratory, Argonne, IL, 60439*

[c] *Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 60439*

Abstract

We describe a parallel, multiscale, multigrid, finite-difference time-domain (FDTD) code for simulating electromagnetic wave propagation in two-dimensional systems involving Lorentz and Drude media. We compare multigrid leapfrog time-stepping procedures and analyze the efficacy and scalability of multigrid use in FDTD. We have implemented the code using Chombo, an object-oriented toolkit for finite-difference methods on block-structured, adaptively refined rectangular grids. We discuss the advantages of this high-level approach, as opposed to the direct use of message-passing libraries.

Key words: finite-difference methods, multigrid methods, adaptive mesh refinement, wave optics, nanoscale structures, nanophotonics, multiscale simulations
PACS: 02.70.Bf, 41.20.Bs, 78.67.-n

* All correspondence should be sent to Zdzisław Meglicki, Office of the Vice President for Information Technology, Indiana University, 601 East Kirkwood Avenue, Room 116, Bloomington, IN, 47405-1223, USA, ph: (812) 856-5597, fax: (812) 855-3310, email: gustav@indiana.edu.

1 Introduction

Finite-difference time-domain (FDTD) simulations [1] can involve several length and time scales. For example, in nanophotonics simulations, in which we are particularly interested, the overall dimensions of a device can be on the scale of microns, but structures within the device, for example, metal nanoparticles or nanoholes, can have dimensions on the scale of tens of nanometers [2–4]. Having to resolve light waves at the same level, as demanded by the smallest nanostructures, results in excessively large grids, very long computation times, and, just as important, enormous output files that may be difficult or even impossible to postprocess.

A natural solution to this challenge is to apply orthogonal gridding of such density as is required in specific areas of the computational domain. Solutions based on this approach are known as multigrid methods. Keeping the grid orthogonal lets us use standard FDTD within each high-density grid region without the complexities, but also without the advantages, of curvilinear coordinates.

Various approaches exist for implementing multigrids. Many of these differ in subtle ways—each in need of its own analysis because the differences may have implications regarding stability and accuracy.

If the problem is highly dynamic, for example, with the micro- and nanostructures moving with respect to each other and with speeds comparable to that of signal propagation, the multigrid may have to adapt to the changing geometry of the system as the computation unfolds. Such situations may not arise in nanophotonics often, although it is not impossible to transfer enough momentum to a nanograin to cause it to move. But they are often enough encountered in fluid dynamics, the area in which the method of adaptive mesh refinement (AMR) arose [5–8].

Development of AMR, or even multigrid applications for parallel processors, is a difficult undertaking. Perhaps for this reason, this methodology has not been universally embraced by the FDTD community yet. Relatively few papers discuss multigrid FDTD. Taflove and Hagness cover multigrid methods under the heading of *nonuniform orthogonal grids* [1], citing literature from the mid-1980s to the mid-1990s. Govan et al. discuss multigrid FDTD in their review [9]. Burr [10] mentions the option of using multigrid to mitigate staircase problems but complains about difficult logistics. Multigrid approaches to FDTD were demonstrated also by Zhu and Carin [11], White, Yun and Iskander [12], Schuhmann, Mayer and Weiland [13], and Chow, Kubota and Namiki [14].

In the past decade, the computational fluid dynamics community developed a

variety of flexible programming tools for implementation of AMR codes. Since multigrid methods can be considered a static version of AMR, the tools are just as applicable to multigrid implementations, including FDTD. Chombo [15] is an object-oriented library package for solving partial differential equations on multigrids. Although designed primarily for finite differences and computational fluid dynamics, it has been used in other areas of computational science and engineering, by exploiting its generality and flexibility. Chombo does not provide implementations of specific scientific computations; these have to be provided by Chombo users as Fortran subroutines. Instead, Chombo provides C++ objects and methods for building and managing multigrids, for providing input data to various parts of the computational procedure, and for handling I/O—all these in either sequential or parallel environments.

We used the Chombo framework to implement a highly scalable and flexible FDTD code, called SHAPES, that operates on either static or dynamically reconfigurable multigrids. The code can be executed sequentially on a PC under Cygwin or Linux or on a massively parallel supercomputer such as the Cray XT3, in the latter case writing its output files in parallel, too. Also, Chombo enabled us to design simple, yet powerful, input semantics that let SHAPES users define complex media layouts, media properties, incident signals, and output formats including spectral response for an arbitrary number of frequencies. The Chombo layer of SHAPES *takes care of allocating and building* data structures needed to process a problem that has been specified by the SHAPES user.

2 Mathematical Foundations

SHAPES mathematics is simple and is based on the well-established FDTD methodology as presented, for example, by Taflove and Hagness [1] or by Sullivan [16].

SHAPES solves the following equations:

$$\partial_t \vec{D} = \vec{\nabla} \times \vec{H} \tag{1}$$

$$\vec{D}(\omega) = \kappa(\omega)\vec{E}(\omega) \tag{2}$$

$$\partial_t \vec{H} = -\vec{\nabla} \times \vec{E}, \tag{3}$$

in the two-dimensional configuration defined by the (E_x, E_y, H_z) triad and referred to as a TE_z mode in [1]. In spatial regions occupied by a nondispersive dielectric medium, κ is independent of ω and equal to the relative dielectric

constant. In a dispersive (and absorbing) medium, we take

$$\kappa(\omega) = \epsilon_\infty + \sum_k \frac{\epsilon_k}{\alpha_k + \mathrm{i}2\delta_k(\omega/\omega_k) - (\omega/\omega_k)^2}. \tag{4}$$

The fields and time in equations (1)–(3) are defined to absorb the fundamental constants ϵ_0, μ_0, and c and are related to those in standard SI units, $\vec{E}_{\mathrm{SI}}$, $\vec{D}_{\mathrm{SI}}$, and t_{SI} by

$$\vec{E} = \sqrt{\frac{\epsilon_0}{\mu_0}}\vec{E}_{\mathrm{SI}} \tag{5}$$

$$\vec{D} = \sqrt{\frac{1}{\epsilon_0\mu_0}}\vec{D}_{\mathrm{SI}} \tag{6}$$

$$t = ct_{\mathrm{SI}}. \tag{7}$$

Coefficients ω_k in equations (1)–(3) are Lorentz or Drude resonance frequencies, and ϵ_k and δ_k are the other two Lorentz parameters that are related to refraction and dissipation at ω_k, respectively. The reader should note that the parameters ϵ_∞, ϵ_k, α_k, and δ_k are all dimensionless; the only parameters that aren't are the resonance frequencies ω_k. For a Drude resonance we have $\alpha_k = 0$ and for a Lorentz resonance $\alpha_k = 1$.

The Drude-Lorentz model is well suited to describing the relative dielectric constant of metallic systems [17].

The system of partial differential equations (1)–(3) is solved explicitly by placing E_x and E_y on the sides of every grid cell and H_z in its center and approximating both the curl and time derivatives with central differences [1,16].

We begin each time step by updating $\vec{D}$ following the discretized form of equation (1). The next step is to obtain $\vec{E}$ from $\vec{D}$. This transformation is trivial for a nondispersive dielectric medium. In regions of space occupied by a dispersive medium, the transformation between $\vec{D}(t)$ and $\vec{E}(t)$, which is given in the frequency domain by equation (2), is accomplished as follows. We define

$$\vec{S}_k(\omega) = \frac{\epsilon_k}{\alpha_k + \mathrm{i}2\delta_k(\omega/\omega_k) - (\omega/\omega_k)^2}\,\vec{E}(\omega) \tag{8}$$

so that

$$\vec{D}(\omega) = \epsilon_\infty\vec{E}(\omega) + \sum_k \vec{S}_k(\omega). \tag{9}$$

Interpreting $\mathrm{i}\omega$ as $\mathrm{d}/\mathrm{d}t$, we arrive at the following auxiliary differential equation (ADE) for each point of the computational domain:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\vec{S}_k(t) + 2\omega_k\delta_k\frac{\mathrm{d}}{\mathrm{d}t}\vec{S}_k(t) + \alpha_k\omega_k^2\vec{S}_k(t) = \epsilon_k\omega_k^2\vec{E}(t). \tag{10}$$

Upon discretization, and assuming that $t_n - t_{n-1} = \Delta t$ is constant for all n, we solve the ADE trivially for $\vec{S}_k(t_n)$ in terms of $\vec{S}_k(t_{n-1})$, $\vec{S}_k(t_{n-2})$, and $\vec{E}(t_{n-1})$:

$$\begin{aligned}\vec{S}_k(t_n) = {} & \frac{2-\alpha_k\omega_k^2\Delta t^2}{1+\omega_k\delta_k\Delta t}\vec{S}_k(t_{n-1}) - \frac{1-\omega_k\delta_k\Delta t}{1+\omega_k\delta_k\Delta t}\vec{S}_k(t_{n-2}) \\ & + \frac{\epsilon_k\omega_k^2\Delta t^2}{1+\omega_k\delta_k\Delta t}\vec{E}(t_{n-1}).\end{aligned} \tag{11}$$

We can do this for each resonance separately because the assumed medium is linear and the resonances do not interact with each other. In effect, by solving equation (9) for $\vec{E}$ in the time domain, we obtain for each point of the computational domain

$$\vec{E}(t_n) = \frac{1}{\epsilon_\infty}\left(\vec{D}(t_n) - \sum_k \vec{S}_k(t_n)\right). \tag{12}$$

This then leads to the following pseudo-code implementation for the ADE.

$\vec{E}_{\mathrm{old}} \leftarrow \vec{E}$
$\vec{E} \leftarrow \vec{D}$
`where` `media are present`
  `repeat for` $k = 1$ `to` *number of resonances*
    `hold` $\leftarrow \vec{S}_k$
    $\vec{S}_k \leftarrow \frac{2-\alpha_k\omega_k^2\Delta t^2}{1+\omega_k\delta_k\Delta t}\vec{S}_k - \frac{1-\omega_k\delta_k\Delta t}{1+\omega_k\delta_k\Delta t}\vec{S}_{k\_\mathrm{old}} + \frac{\epsilon_k\omega_k^2\Delta t^2}{1+\omega_k\delta_k\Delta t}\vec{E}_{\mathrm{old}}$
    $\vec{S}_{k\_\mathrm{old}} \leftarrow$ `hold`
    $\vec{E} \leftarrow \vec{E} - \vec{S}_k$
  `end repeat`
  $\vec{E} \leftarrow \vec{E}/\epsilon_\infty$
`end where`

The number of $\vec{S}_k$ and $\vec{S}_{k\_\mathrm{old}}$ fields required is equal to the number of resonances. For example, a formula with three resonances will require six $\vec{S}$ fields including the "old" ones. But multiple media can be covered by the same set of $\vec{S}$ fields; that is, we do not need to proliferate them if we have multiple media. All that must be done is to let the α_k, δ_k, ϵ_k, ω_k, and ϵ_∞ coefficients vary with position. The C++ Chombo shell of SHAPES dynamically allocates

all $\vec{S}$ fields and appropriate arrays of media coefficients after reading the input file.

After the ADE conversion of $\vec{D}$ to $\vec{E}$, the magnetic field H_z is updated by solving the discretized form of equation (3).

The computational domain is divided into the total field and scattered field regions. The incident signal is injected into the total field region by explicit manipulation of field derivatives at grid points near the total field region boundary in the standard way [16]. SHAPES can inject signals of many types, including Heaviside and tanh ramped harmonic waves, Gaussian pulses, and various chirps, and from any direction. On leaving the total field region and propagating across the scattered field region, the scattered signal is then absorbed on perfectly matched layer (PML) boundaries around the computational domain. This strategy is also standard and done similarly to the method presented in Sullivan [16,18]. We chose to implement various PML expressions functionally, rather than by storing them in arrays, in order to save memory. Similarly, the injected signal is propagated on the total field region boundary functionally, not numerically.

These usual FDTD complexities are not affected by the multigrid because we restrict multigrid manipulations to the interior of the total field region.

On output, SHAPES can generate images of all the fields used in the computation, that is, $\vec{E}$, $\vec{D}$, and $\vec{H}$, both individual components as well as field values, energy density, and Fourier accumulations of the fields for an arbitrary number of frequencies. Here, again, the Chombo C++ shell allocates such auxiliary storage as is required.

3 Media Distribution

SHAPES users can define an arbitrary number of Lorentz/Drude media, each with an arbitrary number of resonances. The media can then be distributed by associating them with various *shapes*, hence the name of the program. The elementary shapes provided by the program are circles, rectangles, and ellipses, which can be oriented in any direction, and rings and triangles. The elementary shapes can be repeated any number of times in any direction. Shapes are allowed to overlap. Additionally, program users can define masks that work like *lift-off* masks in microelectronics; that is, they remove whatever medium they overlap with. The masks are defined the same way as media layouts, by combining possibly multiple occurrences of elementary shapes.

Figure 1 illustrates the use of a mask and overlapping to define a waveguide

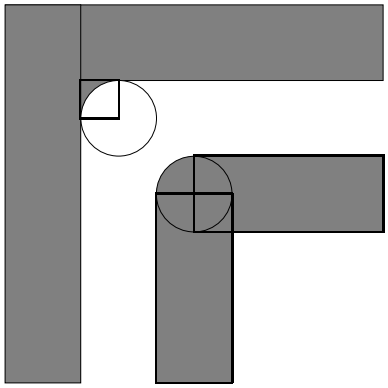

Fig. 1. A rounded corner definition in SHAPES.

bend with a rounded corner. The white circle in the upper left corner of the bend is a mask. It cuts a round portion out of the square that fills the corner. The partially overlapping circle in the lower right corner of the bend fills the square gap that results from the intersection of the two rectangles that define the waveguide from below.

The shape semantics let us define and vary with ease very complex and large configurations that may be of interest to microwave engineering and nanophotonics.

4 Multigrid and Multigrid Timestep

The Chombo multigrid is not constructed by cutting out a fragment of a low-resolution grid and filling the hole with a high-resolution grid, as is the case in some nonuniform orthogonal grid methods. Instead, in regions designated for grid refinement, an additional layer of higher-resolution grid is applied. Within this layer further refinement may be sought, producing multiple layers of ever increasing grid resolution. These layers are referred to as *levels,* with level 0 grid being the lowest-resolution grid that covers the whole computational domain, and level $k-1$ grid being the highest-resolution grid in a k-level system.

A level 1 grid is constructed by first using application-dependent criteria to *tag* level 0 grid cells for refinement. SHAPES provides three options. Cells can be tagged for refinement if

(1) the energy density within the cells exceeds a certain threshold value;
(2) the energy density within the cells changes faster than a certain threshold value;
(3) the cells are located in static regions defined by the user.

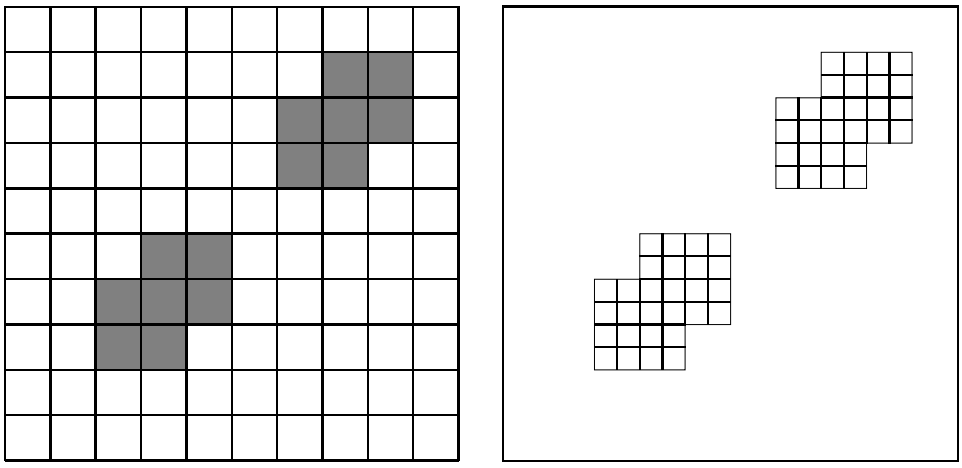

Fig. 2. Grid refinement for the refinement ratio of 2. On the left is a level 0 grid with tagged cells shaded. The result is a refined and, in this case, disconnected level 1 grid, shown on the right, that covers the shaded regions only.

The user can choose one or more of these options. The static regions are defined by using semantics similar to the semantics used to define media layouts; that is, they can be defined in terms of the five elementary shapes: circles, quadrangles, ellipses, rings, and triangles, which can be repeated, overlapped, and cut into with masks.

The tagged level 0 grid cells constitute a subset of level 0. The level 1 grid is constructed on this subset by dividing the tagged level 0 cells into quarters or into smaller portions as long as the ratio of level 1 to level 0 grid constants is a power of 2. This number is called the *refinement ratio*. The collection of the subcells resulting from the division constitutes the level 1 grid. This collection may be disjoint. The level 1 grid is contained within the level 0 grid. We also say that levels 0 and 1 are *adjacent*.

When Chombo generates the level 1 grid, it may be oversized depending on the tightness of fit parameter and other logistics, for example, the distribution of the grid among processes of the MPI pool.

The grid refinement procedure is illustrated in Figure 2.

Once the level 1 grid has been constructed, the procedure may be repeated, this time by tagging level 1 grid cells and subdividing them by the same or some other refinement ratio to generate a level 2 grid, and so on, until a desired number of levels is reached. The level 2 grid is contained within the level 1 grid. Levels 1 and 2 are adjacent, but levels 0 and 2 aren't. Progressively finer grids of multiple levels are nested within each other like Russian dolls.

Initial field data for higher-level grids are generated by linear or higher-order interpolation from the adjacent coarser grids. This is normally done at the time the finer grid is constructed.

The field data defined on the hierarchy of grids so constructed must be advanced in time. The advance must be designed so that all levels step in synchrony, a non trivial task if the basic time step is to be the FDTD leapfrog. Data flow only between adjacent levels. Coarser levels provide finer levels with boundary conditions; in return, they receive field updates from the finer levels, which are then spread over the coarser levels by averaging. These replace the coarser levels' own updates—needed to generate the boundary conditions—as the field updates are deemed more accurate.

The data movement between the levels represents the major cost of the method. Nevertheless, we will demonstrate that the gains outweigh the costs considerably in cases for which the use of the multigrid is justified.

A leapfrog time step in a multigrid system can be implemented in two ways. One way is to select a time step, Δt, which is appropriate for the finest level—based on the Courant stability criterion for electromagnetic wave propagation in a vacuum:

$$\Delta t \leq \frac{\Delta x}{\sqrt{N}}, \tag{13}$$

where Δx is the finest-level grid spacing and N is the number of dimensions. We can then use the selected Δt to advance all levels at the same pace.

Let us define the following two procedures that operate on level i:

`update_e(i)`

(1) Advance the $\vec{D}^i(t_n)$ field of level i by Δt to $\vec{D}^i(t_{n+1})$.
(2) Convert $\vec{D}^i(t_{n+1})$ to $\vec{E}^i(t_{n+1})$ where media are present.
(3) Interpolate $\vec{D}^{i-1}(t_{n+1})$ and $\vec{E}^{i-1}(t_{n+1})$ from level $i-1$ onto the level i boundary.
(4) If level $i+1$ exists, call `update_e(i + 1)`.
(5) Average $\vec{D}^i(t_{n+1})$ and $\vec{E}^i(t_{n+1})$ of level i onto level $i-1$ cells that overlap with level i cells.

COMMENT 1 Level 0 has its own version of `update_e` that takes care of PML boundaries and incident signal injection and extraction at the total field region boundary. The level 0 version does not, obviously, interpolate boundary data from a coarser level, but it calls `update_e(1)` if level 1 exists.

COMMENT 2 This procedure is recursive, that is, data of level i are going to be "corrected" by level $i+1$ before `update_e(i + 1)` returns. Similar considerations hold for level $i+1$. By means of this recursion, data cascade

from the highest level back to level 0.

`update_h(i)`

(1) Advance the $\vec{H}^i(t_n + \Delta t/2)$ field of level i by Δt to $\vec{H}^i(t_{n+1} + \Delta t/2)$.
(2) Interpolate $\vec{H}^{i-1}(t_{n+1} + \Delta t/2)$ from level $i-1$ onto the level i boundary.
(3) If level $i+1$ exists, call `update_h(i + 1)`.
(4) Average $\vec{H}^i(t_{n+1} + \Delta t/2)$ of level i onto level $i-1$ cells that overlap with level i cells.

COMMENT 1 Level 0 has its own version of `update_h` that takes care of PML boundaries and incident signal injection and extraction at the total field region boundary. The level 0 version does not interpolate boundary data from a coarser level, but it calls `update_h(1)` if level 1 exists.

COMMENT 2 This procedure is recursive: that is, that data of level i is going to be "corrected" by level $i+1$ before `update_h(i + 1)` returns. Similar conditions hold for level $i+1$. By means of this recursion, data cascade from the highest level back to level 0.

Now calling `update_e(0)` followed by `update_h(0)` triggers a recursive *synchronized unistep* leapfrog advance for all levels. This procedure works well. An obvious question is, where are the savings if we advance all levels at the same short time step of the finest grid. The savings are in reducing exponentially the total number of cells that have to be advanced, because we don't spread the finest grid over the whole computational domain.

Programmers working with multigrid methods must be aware that the way data are averaged or interpolated between levels is different for cell-centered data and for face-centered data [1]. Figure 3 illustrates why this is the case.

Can we improve on the synchronized unistep procedure? An alternative approach is to advance each level at its own time step, different from one level to the other but still maintaining synchronization at shared $\vec{E}$ time slices across all levels and at shared $\vec{H}$ time slices across all levels. For the refinement ratio of 2 this approach is possible if the time step is refined by 3 between adjacent levels, not by 2, that is,

$$\Delta t^1 = \Delta t^0/3$$
$$\Delta t^2 = \Delta t^1/3 = \Delta t^0/9.$$

[1] Chombo provides methods for exchange of cell-centered data only. Methods for exchange of face-centered data were provided by Dr. Dan Martin of the Applied Numerical Algorithms Groups at the Lawrence Berkeley National Laboratory.

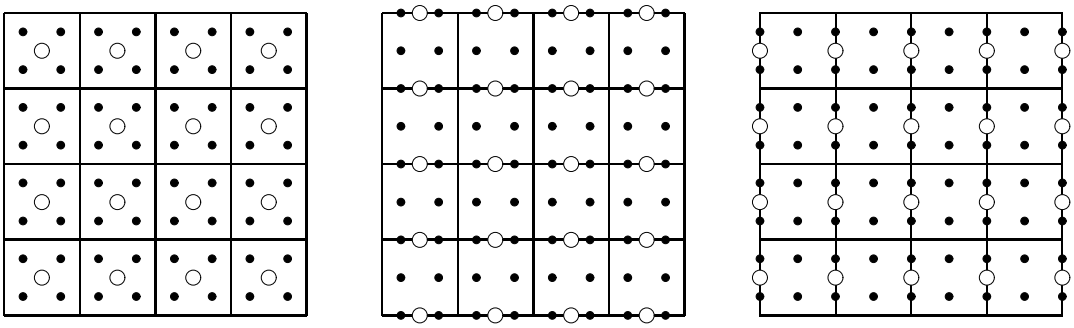

Fig. 3. Geometric relationships between adjacent level data for cell-centered data such as H_z (left panel), data centered on north-south faces such as E_x (central panel), and data centered on east-west faces such as E_y (right panel). White dots represent coarse-grid data, and black dots represent fine-grid data. Data averaging and interpolation routines developed for cell-centered data will return incorrect results if applied to face-centered data and vice versa.

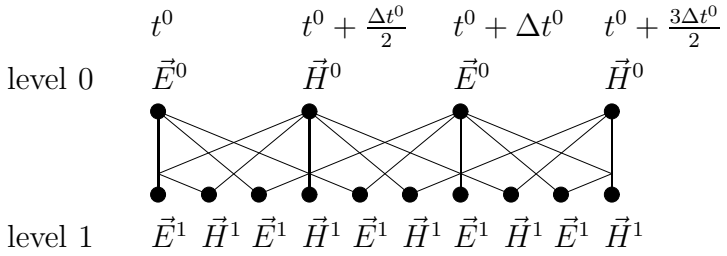

Fig. 4. The two-level synchronized multistep leapfrog showing communication lines between the levels. Communication on vertical lines is bidirectional, since these are sync lines and level 1 data can be used to correct (by averaging) level 0 data. Communication on angled lines is from level 0 to level 1. Here level 0 data are used to provide data for the level 1 boundary.

More generally, for a space refinement ratio of n, the time-step refinement between adjacent levels should be $n + 1$. In the following considerations we will focus on the space refinement ratio of 2.

Figure 4 illustrates why this is so. We have only two levels in this figure, but its meaning can be extended by simply relabeling the levels. We can see that the level 0 $\vec{E}^0$ coincides with the level 1 $\vec{E}^1$ and that the level 0 $\vec{H}^0$ coincides with the level 1 $\vec{H}^1$. We call this procedure a *synchronized multistep*. In this case every level maintains its own separate time variable, t^0 for level 0, t^1 for level 1, and so on.

Figure 4 also shows interlevel communication for the synchronized multistep. For the level 1 points, where two angled lines meet, time interpolation is implied between the level 0 points from which the lines originate.

Let us consider two adjacent levels, i and $i - 1$. Let us assume that the data are defined for the following time slices.

$$
\begin{array}{rcl}
\vec{D}^{i-1}_{\text{old}} & \text{at} & t^{i-1} \\
\vec{D}^{i-1} & \text{at} & t^{i-1} + \Delta t^{i-1} \\
\vec{E}^{i-1}_{\text{old}} & \text{at} & t^{i-1} \\
\vec{E}^{i-1} & \text{at} & t^{i-1} + \Delta t^{i-1} \\
\vec{H}^{i-1}_{\text{old}} & \text{at} & t^{i-1} - \Delta t^{i-1}/2 \\
\vec{H}^{i-1} & \text{at} & t^{i-1} + \Delta t^{i-1}/2 \\
\vec{D}^{i} & \text{at} & t^{i} = t^{i-1} \\
\vec{E}^{i} & \text{at} & t^{i} = t^{i-1} \\
\vec{H}^{i} & \text{at} & t^{i} + \Delta t^{i}/2 = t^{i-1} + \Delta t^{i-1}/3/2
\end{array}
$$

We can provide boundary data to $\vec{E}^i(t^i)$ because we have $\vec{E}^{i-1}(t^{i-1}) = \vec{E}^{i-1}(t^i)$, and we can provide boundary data to $\vec{H}^i(t^i + \Delta t^i/2) = \vec{H}^i(t^{i-1} + \Delta t^{i-1}/3/2)$ by time interpolating between $\vec{H}^{i-1}_{\text{old}}(t^{i-1} - \Delta t^{i-1}/2)$ and $\vec{H}^{i-1}(t^{i-1} + \Delta t^{i-1}/2)$ at $t = t^{i-1} + \Delta t^{i-1}/3/2$

We can implement the two-level time step for this system and at the same time extend it recursively to an arbitrary number of levels by redefining our level update routines `update_e` and `update_h` as follows.

`update_e(i)`

(1) Advance the $\vec{D}^i(t^i)$ field of level i by Δt^i to $\vec{D}^i(t^i + \Delta t^i)$.
(2) Convert $\vec{D}^i(t^i + \Delta t^i)$ to $\vec{E}^i(t^i + \Delta t^i)$ where media are present.
(3) Time and space interpolate between $\vec{D}^{i-1}_{\text{old}}(t^{i-1})$ and $\vec{D}^{i-1}(t^{i-1} + \Delta t^{i-1})$ and between $\vec{E}^{i-1}_{\text{old}}(t^{i-1})$ and $\vec{E}^{i-1}(t^{i-1} + \Delta t^{i-1})$ at $t^{i-1} + \Delta t^{i-1}/3$ to fill the boundary of level i.
(4) If level $i + 1$ exists, call `update_e(i + 1)`.
(5) Advance the $\vec{H}^i(t^i + \Delta t^i/2)$ field of level i by Δt^i to $\vec{H}^i(t^i + \Delta t^i/2 + \Delta t^i)$;
(6) Space interpolate $\vec{H}^{i-1}$ onto the boundary of level i. Because $t^i + \Delta t^i/2 + \Delta t^i = t^{i-1} + \Delta t^{i-1}/3/2 + \Delta t^{i-1}/3 = t^{i-1} + \Delta t^{i-1}/2$ we find that $\vec{H}^i$ and $\vec{H}^{i-1}$ are synchronized, and so there is no need to time interpolate.
(7) If level $i + 1$ exists, call `update_h(i + 1)`.
(8) Because $\vec{H}^i$ and $\vec{H}^{i-1}$ are in sync, average $\vec{H}^i$ onto level $i - 1$ cells that overlap with level i cells.
(9) Advance the $\vec{D}^i(t^i + \Delta t^i)$ field of level i by Δt^i to $\vec{D}^i(t^i + 2\Delta t^i)$.
(10) Convert $\vec{D}^i(t^i + 2\Delta t^i)$ to $\vec{E}^i(t^i + 2\Delta t^i)$ where media are present.
(11) Time and space interpolate between $\vec{D}^{i-1}(t^{i-1})$ and $\vec{D}^{i-1}(t^{i-1} + \Delta t^{i-1})$ and between $\vec{E}^{i-1}(t^{i-1})$ and $\vec{E}^{i-1}(t^{i-1} + \Delta t^{i-1})$ at $t^{i-1} + 2\Delta t^{i-1}/3$ to fill the boundary of level i.
(12) If level $i + 1$ exists, call `update_e(i + 1)`.

COMMENT 1 Level 0 has its own version of `update_e` that takes care of PML boundaries and incident signal injection and extraction at the total field region boundary. The level 0 version does not, obviously, interpolate boundary data from a coarser level, but it calls `update_e(1)` if level 1 exists.

COMMENT 2 This procedure is mutually recursive with `update_h`. The first call to `update_e(i + 1)` corrects by averaging higher-resolution data $\vec{H}^i(t^i + \Delta t^i/2)$ before it is used in the level i update. The following call to `update_h(i + 1)` similarly corrects $\vec{E}^i(t^i + \Delta t^i)$. The second call to `update_e(i + 1)` corrects $\vec{H}^i(t^i + \Delta t^i/2 + \Delta t^i)$. Similar conditions hold for level $i + 1$. By means of this mutual recursion, both $\vec{E}$ and $\vec{H}$ data cascade from the highest level back to level 0.

`update_h(i)`

To understand this routine, we should assume that `update_e` has already been called. So $\vec{H}^i$ is now $\vec{H}^i(t^i + 3\Delta t^i/2) = \vec{H}^i(t^{i-1} + 3\Delta t^{i-1}/3/2) = \vec{H}^i(t^{i-1} + \Delta t^{i-1}/2)$, meaning that it is synchronized with $\vec{H}^{i-1}$. But by this time the lower-level calls to `update_e` and `update_h` would have advanced $\vec{H}^{i-1}$ to $\vec{H}^{i-1}(t^i + \Delta t^i/2 + \Delta t^i)$ and $\vec{H}_{\rm old}$ to $\vec{H}^{i-1}_{\rm old}(t^i + \Delta t^i/2)$, so $\vec{H}^i$ is, in fact, synchronized with $\vec{H}^{i-1}_{\rm old}$, not with $\vec{H}^{i-1}$.

Let us then advance t^i to $t^i = t^{i-1} \leftarrow t^{i-1} + \Delta t^{i-1}/2$, so as to align it with the slice where $\vec{H}^{i-1}_{\rm old}$ and $\vec{H}^i$ are defined.

(1) Advance the $\vec{H}^i(t^i)$ field of level i by Δt^i to $\vec{H}^i(t^i + \Delta t^i) = \vec{H}^i(t^{i-1} + \Delta t^{i-1}/3$.
(2) Time and space interpolate between $\vec{H}^{i-1}_{\rm old}(t^{i-1})$ and $\vec{H}^{i-1}(t^{i-1} + \Delta t^{i-1})$ at $t^{i-1} + \Delta t^{i-1}/3$ to fill the boundary of level i.
(3) If level $i + 1$ exists, call `update_h(i + 1)`.
(4) Advance the $\vec{D}^i(t^i + \Delta t^i/2)$ field of level i by Δt^i to $\vec{D}^i(t^i + \Delta t^i/2 + \Delta t^i)$.
(5) Convert $\vec{D}^i(t^i + \Delta t^i/2 + \Delta t^i)$ to $\vec{E}^i(t^i + \Delta t^i/2 + \Delta t^i)$ where media are present.
(6) Space interpolate $\vec{D}^{i-1}$ and $\vec{E}^{i-1}$ onto the boundary of level i. Because $t^i + \Delta t^i/2 + \Delta t^i = t^{i-1} + \Delta t^{i-1}/3/2 + \Delta t^{i-1}/3 = t^{i-1} + \Delta t^{i-1}/2$, we find that $\vec{E}^i$ and $\vec{E}^{i-1}$ are in synchrony, as are $\vec{D}^i$ and $\vec{D}^{i-1}$, and so there is no need to time interpolate.
(7) If level $i + 1$ exists, call `update_e(i + 1)`.
(8) Because $\vec{E}^i$ and $\vec{E}^{i-1}$ are in sync, as are $\vec{D}^i$ and $\vec{D}^{i-1}$, average $\vec{E}^i$ and $\vec{D}^i$ onto level $i - 1$ cells that overlap with level i cells.
(9) Advance the $\vec{H}^i(t^i + \Delta t^i)$ field of level i by Δt^i to $\vec{H}^i(t^i + 2\Delta t^i)$.
(10) Time and space interpolate between $\vec{H}^{i-1}_{\rm old}(t^{i-1})$ and $\vec{H}^{i-1}(t^{i-1} + \Delta t^{i-1})$ at $t^{i-1} + 2\Delta t^{i-1}/3$ to fill the boundary of level i.
(11) If level $i + 1$ exists, call `update_h(i + 1)`.

COMMENT 1 Level 0 has its own version of `update_h` that takes care of PML boundaries and incident signal injection and extraction at the total field region boundary. The level 0 version does not interpolate boundary data from a coarser level, but it calls `update_h(1)` if level 1 exists.

COMMENT 2 This procedure is mutually recursive with `update_e`. The first call to `update_h(i + 1)` corrects by averaging higher resolution data $\vec{E}^i(t^i+\Delta t^i/2)$ and $\vec{D}^i(t^i+\Delta t^i/2)$ before they are used in the level i update. The following call to `update_e` similarly corrects $\vec{H}^i(t^i+\Delta t^i)$. The second call to `update_h` corrects $\vec{E}^i(t^i+\Delta t^i/2+\Delta t^i)$ and $\vec{D}^i(t^i+\Delta t^i/2+\Delta t^i)$. Similar conditions hold for level $i+1$. By means of this mutual recursion, $\vec{E}$, $\vec{D}$, and $\vec{H}$ data cascade from the highest level back to level 0.

Calling `update_e(0)` followed by `update_h(0)` triggers a recursive synchronized multistep leapfrog advance for all levels.

As with many complex procedures, the synchronized multistep is not without problems.

The first problem arises from having to divide the time step by a number that is larger than the refinement ratio. We have to set our Δt^0 to satisfy the stability criterion for level 0. But then Δt^i for the higher levels ends up being shorter than needed for those levels, and this may become costly for systems with a large number of levels. For example, for a 10-level system, $\Delta t^9 = \Delta t^0/3^9 = \Delta t^0/19683$ compared with $\Delta t^0/2^9 = \Delta^0/512$. This is some 38 times shorter than it would really have to be. A remedy here could be to refine not by 2 but, say, by 4. Then $\Delta t^0/\Delta t^1 = 5$, the difference is no longer this large, and we no longer need so many levels, either.

The second problem is caused by time interpolation, which is costly and introduces error that can destabilize the procedure. We have observed emergence of high-frequency noise on the level boundaries. Because the lower-resolution grid of the lower level cannot propagate the high-frequency noise away to the perfectly matched layer, where it would be absorbed, the noise becomes trapped within the high-resolution mesh and is continuously fed by the interpolation. This is a common problem for multigrid methods when applied to hyperbolic equations. In computational fluid dynamics various devices such as artificial viscosities can be deployed to absorb the noise. But we do not have artificial viscosities in FDTD. The remedy here can be to resort to a more elaborate, higher-order time interpolation (we use linear interpolation in the code), which would make the procedure even costlier, or to deploy a noise elimination method, for example, based on the Fourier analysis of the field.

Luckily, the synchronized unistep method, which is free of time interpolations,

works well and affords us considerable savings. We have implemented both methods in the code, and the user is free to choose between them.

5 Parallelization and Output

Chombo parallelizes its operations by dividing all cells of a given level into boxes and then distributing the boxes of cells among the MPI processes. Thus, a Chombo box of cells is a *quantum* of parallelism. But a given MPI process may end up with more than one box for a given level, and then it may have to compute on boxes covering other levels, too. Chombo attempts to distribute the load evenly. This process, however, depends also on the specific shape of the subgrids. Intricate shapes may require a large number of small boxes to cover them.

A SHAPES user can specify the smallest size of a box to be used in covering the subgrids, the largest size of a box, and the extent to which the covering is allowed to be relaxed. A very relaxed covering may simply place a single large box on the designated area of the grid. A very tight covering may end up tracing the shape of the area closely—generating many small boxes in the process.

Chombo takes care of communicating data between the boxes and between the levels, both in a sequential and in a parallel context.

We have tested SHAPES on IA32 and IA64 Linux clusters, as well as on the Cray XT3 supercomputer, varying the number of MPI processes between 4 and 1024. When SHAPES runs in parallel, all I/O is done in parallel, too. SHAPES diagnostics are written by participating MPI processes on their own process-specific files. Field data is written on HDF5-structured MPI-IO files, which are accessed by all MPI processes simultaneously. Such files are best created on a parallel file system, for example, PVFS, GPFS, or Lustre.

When the code is run sequentially, the data can also be dumped in a human-readable format suitable for display and postprocessing with Gnuplot. In both cases, the data files are substantially annotated.

The HDF5 files can be perused with standard HDF5 tools that list their content; but in order to visualize the data, a special tool is needed because the structure of HDF5 data within HDF5 files is application dependent.

Chombo provides such a tool, called ChomboVis. It is a Python/VTK package that understands the way Chombo writes its multigrid data on the HDF5 files. The tool provides a considerable number of visualization options that can be

Table 1
Drude/Lorentz parameters for silver

Symbol	Value	Unit
ϵ_∞	2.36461	
ω_D	1.3257×10^{16}	Hz
Γ_D	1.136268×10^{14}	Hz
ω_1	6.6457856×10^{15}	Hz
g_1	0.266286	
δ_1	4.2487×10^{14}	Hz
ω_2	7.863688×10^{15}	Hz
g_2	$1 - g_1$	
δ_2	8.316865×10^{14}	Hz
$\Delta\epsilon$	1.1830916	

additionally enhanced or customized by Python scripting.

6 Scattering off a Drude/Lorentz Dispersive Metallic Cylinder

A good example is the interaction of radiation with a metallic cylinder. We illustrate in this section how this problem is handled by SHAPES, at the same time indicating where our code is unique.

We assume the cylinder to be made of silver, and we use the following phenomenological model for the material:

$$\vec{D}(\omega) = \epsilon_0 \left(\epsilon_\infty - \frac{\omega_D^2}{\omega^2 - \mathrm{i}\Gamma_D\omega} + \frac{g_1\omega_1^2\Delta\epsilon}{\omega_1^2 + 2\mathrm{i}\omega\delta_1 - \omega^2} + \frac{g_2\omega_2^2\Delta\epsilon}{\omega_2^2 + 2\mathrm{i}\omega\delta_2 - \omega^2} \right) \vec{E}(\omega), \quad (14)$$

where the values of the parameters are given in Table 1. These parameters were shown to accurately describe empirical data for the complex dielectric constant of silver over a wide range of frequencies [17].

We *normalize* this material model to that given by equation (4) by dividing both the numerator and the denominator of each resonance term by the square of its resonance frequency. The result is the following transformation of the coefficients for the Lorentz terms:

$$\alpha_k \leftarrow 1 \quad (15)$$

$$\delta_k \leftarrow \frac{\delta_k}{\omega_k} \tag{16}$$
$$\epsilon_k \leftarrow g_k \Delta\epsilon \tag{17}$$

and the following transformation for the Drude term:

$$\alpha_k \leftarrow 0 \tag{18}$$
$$\delta_k \leftarrow \frac{\Gamma_D}{2\omega_D} \tag{19}$$
$$\epsilon_k \leftarrow 1. \tag{20}$$

Because our unit of time is the same as the unit of length on account of the $t = ct_{\mathrm{SI}}$ substitution, we have to scale the resonance frequencies accordingly. To this end, we multiply each frequency by the time it takes for light to traverse a unit length chosen for the simulation. In this example we have chosen the unit of length to be 15 nm. Light traverses this distance in 5×10^{-17} seconds (0.05 fs), so this is the number we need to multiply the resonance frequencies by.

The cylinder itself is located in the center of the computational domain, and its radius is 1.25 units of length, that is, 18.75 nm. Table 2 lists the resulting simulation parameters.

The injected signal is an x-polarized Gaussian wave packet moving in the y-direction. The Gaussian envelope has a half-width σ of 900 nm (i.e., 60 units of length), and the wavelength λ of the harmonic component of the signal is 300 nm (i.e., 20 units of length), in the near ultraviolet range. The metallic optical properties determined by $\kappa(\omega)$, equation (14), are such that surface plasmon resonances exist in the 300 to 500 nm spectral range [2]. Our injected signal's frequency falls right at the edge of this region, where scattering, extinction, and absorption cross-sections are still minimal (cf. Figure 2 in [2]). Nevertheless, we observe intensification of electromagnetic fields near the metallic surfaces of the cylinder in agreement with observations made in [2]. The exact formula describing the signal is

$$\begin{aligned}
&f(\zeta) = \exp\left(-\frac{\zeta^2}{2\sigma^2}\right) \sin\frac{2\pi}{\lambda}\zeta \\
&\zeta = n_x(x - x_0) + n_y(y - y_0) - (t - t_0) \\
&n_x^2 + n_y^2 = 1 \\
&H_z = f(\zeta) \\
&E_x = -n_y f(\zeta) \\
&E_y = n_x f(\zeta),
\end{aligned}$$

where the parameters n_x, n_y, x_0, y_0, t_0, σ, and λ are shown in Table 2. The

Table 2
Signal and material parameters used in the cylinder simulation.

Units			
Length	15 nm		
Time	0.05 fs		
Geometry of the Region			
	Computational Domain	*PML-Free Region*	*Total Field Region*
$(x_{\min}, y_{\min})$	$(0, 0)$	$(5, 5)$	$(30, 30)$
$(x_{\max}, y_{\max})$	$(100, 100)$	$(95, 95)$	$(70, 70)$
Injected Signal			
(n_x, n_y)	$(0, 1)$		
(x_0, y_0)	$(0, 30)$		
t_0	300		
λ	20		
σ	60		
Material Parameters			
ϵ_∞	2.36461		
	$k = 1$	$k = 2$	$k = 3$
ω_k	0.66285	0.33229	0.39318
α_k	0.00000	1.00000	1.00000
δ_k	0.00429	0.06393	0.10576
ϵ_k	1.00000	0.31504	0.86805
Layout Parameters			
(x_c, y_c)	$(50, 50)$		
r_c	1.25000		
Grid Refinement Parameters			
$(x_{\min}, y_{\min})$	$(44, 44)$		
$(x_{\max}, y_{\max})$	$(56, 56)$		
Levels	5		
Refinement ratio	2		

signal is injected and extracted at a rectangular total field region boundary defined by $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$, also shown in Table 2.

Dispersive problems such as the present one do not trivially scale with system size as simple dielectric ones do. However, the solution discussed here could be related to problems of other scale under appropriate circumstances because the unit of length does not appear in the computations explicitly and can be changed arbitrarily. The material properties are characterized here by dimensionless ratios of damping coefficients (Γ_D, δ_1, and δ_2) to their respective resonance frequencies and by the ratio of the resonance frequency to the frequency of the incident signal. If a material with similar ratios could be found, or made, for example, in the microwave regime, the solution would apply just as well.

We solved this problem on five levels, with level 0 $\Delta x = \Delta y = 0.25$ (i.e., 3.75 nm) and Δt the same for all levels—we used the synchronized unistep in the computation—and set somewhat below what is dictated by the stability criterion for level 4, to 0.00390625, which is $\Delta x_0/(4 \times 2^4)$.

The level 0 grid had 400×400 nodes. It was refined within the rectangle defined by its lower left corner at $(44, 44)$ and its upper right corner at $(56, 56)$, and the refinement ratio between adjacent levels was 2. The level 4 grid would have 6400×6400 nodes if it were to cover the whole computational domain. We performed single-level computations on such a grid, too, to demonstrate the gains that resulted from using the multigrid method in this context.

The scattered field region was left quite wide in order to display the long-range scattered field for this configuration. For the same reason the diameter of the cylinder was made very small. It is not resolved sufficiently within the level 0 grid.

Figure 5 illustrates the geometry of the computational domain. The cylinder is represented by the small circle in the center. It is enclosed within the tag box, which is, in turn, enclosed within the total field region box. The scattered field region around it is wide. Eventually we get to the PML boundary and finally the boundary of the computational domain.

Figure 6 shows how the cylinder is resolved at various levels, beginning with level 0 on the left through level 4 on the right. To resolve the edge of the cylinder very finely helps mitigate staircase spikes, which become smaller with increasing resolution.

Figure 7 shows images of the E_x field for $t = 325$ (16.25 fs), which corresponds to the peak of the Gaussian packet traversing through the cylinder. The leftmost panel shows the whole computational domain, $1,500\,\text{nm} \times 1,500\,\text{nm}$, with the total field region and the incident signal clearly visible in the center. The

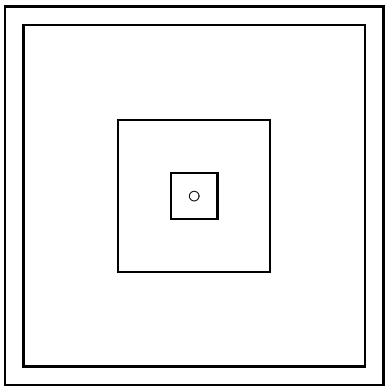

Fig. 5. Definition of various regions in the simulation. The innermost box is the tag region; the next box from the center is the total field region. It is surrounded by a fairly wide scattered field region. Then we have the PML boundary and finally the boundary of the computational domain. The circle in the center represents the cylinder. All objects are drawn to scale.

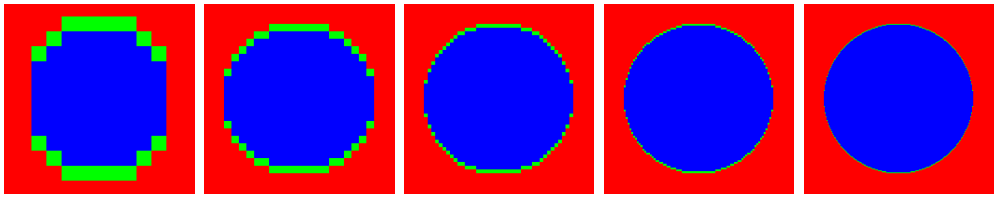

Fig. 6. Resolving the cylinder at 5 levels.

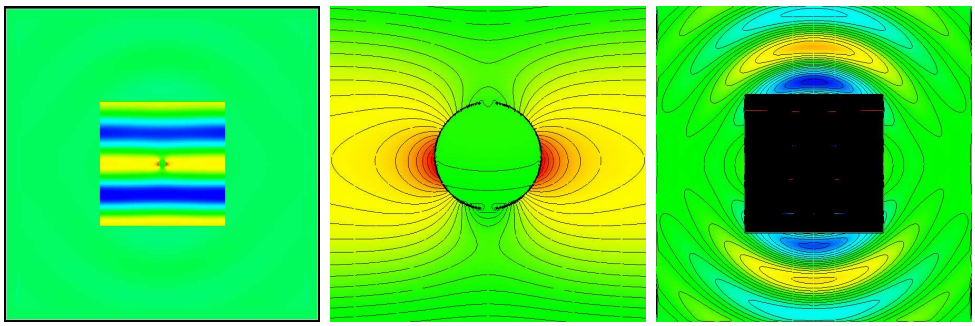

Fig. 7. ChomboVis images of the E_x field for the 5-level run for $t = 325$ (16.25 fs). The leftmost panel shows the whole computational domain with the incident wave itself visible in the total field region. The center panel shows the magnified image of the E_x field in the middle of the computational domain. The rightmost panel shows the field in the scattered field region. In order to see the field in the scattered field region, the colormap had to be adjusted. This adjustment resulted in the blotting out the total field region.

rightmost panel shows the scattered field. Because the scattered field is much weaker than the incident field, we had to blot out the total field region, which appears black. Finally the center panel zooms on the cylinder itself to illustrate the highly resolved features of the field around and inside the cylinder. The diameter of the cylinder is 37.5 nm, and the field in its vicinity is resolved down to about 0.2 nm. On the other hand, the field in the scattered field region

is resolved down to about 4 nm. An important feature of this solution is that in this single simulation we compute *simultaneously* the far field and the near field, in both cases with such resolution as is needed.

7 Mitigating the Cost of High Resolution

How much do we save by running this example in multigrid, rather than in the target resolution covering the whole region?

This example was run on four nodes of a typical IA32 cluster, and field snapshots were dumped every 7 minutes. The target resolution is very high. The $\Delta x_0 = 0.25$ is refined 4 times down to $\Delta x_4 = \Delta x_0/2/2/2/2 = 0.25/16 = 0.015625$. As we remarked earlier, at this resolution the 100×100 domain would be covered by $6,400 \times 6,400$ grid cells, which is more than 40 million cells. This would be a very large problem, and it would no longer fit in the memory of four nodes. When run on sixteen nodes, the program takes 70 minutes between the snapshots, instead of just 7 minutes for the 5-level simulation, and the produced data files are 3.36 GB each. One hundred and thirty of these files fills more than 436 GB of disk space.

If the problem could somehow be executed on four nodes instead of sixteen, it would take four times longer to run (the single-level computation parallelizes well, and the scalability is almost linear), that is, 4 hours and 40 minutes per snapshot. In order to complete the run, nearly a month would be needed.

The Fortran subroutines used with the program at present are not highly optimized, but even if we managed to optimize them, say, five times—which would be a lot—the single-level configuration would still take days to complete when run on four nodes.

But the size of the resulting data files is the worst problem. It is very difficult to postprocess data files as large as 3.36 GB. Computer memory is still expensive, and we seldom have more than 2GB in our desktops, even though the EM64T architecture does away with the previous 32-bit limitation.

Table 3 compares resources used by five SHAPES input configurations, all of which model the example discussed in this section. For all configurations the computational domain is $1,500\,\text{nm} \times 1,500\,\text{nm}$, the cylinder is located in the center, and its diameter is 37.5 nm. For all multilevel configurations, the tag regions are defined in the same way. The *target* resolution for all runs is the same, but in the Cylinder-5 configuration it is reached within level 5, as discussed above. In the Cylinder-4 configuration we have 4 levels only, and the target resolution is reached within level 4. To accomplish this, we must en-

Table 3
Resources consumed by cylinder runs with varying numbers of levels but with a fixed target resolution of the finest level, identical physical configuration, and a fixed number of processors.

Configuration	No. of levels	$n_x \times n_y$	Snapshot	
			Time (min)	Size (MB)
Cylinder-5	5	400×400	7	28.55
Cylinder-4	4	800×800	17	94.32
Cylinder-3	3	1600×1600	32	269.43
Cylinder-2	2	3200×3200	79	909.15
Cylinder-1	1	6400×6400	280	3437.51

large the level 0 grid to 800×800 nodes. Similarly, we have to enlarge the level 0 grid for the 3-level configuration to $1,600 \times 1,600$ nodes, then for the 2-level configuration to $3,200 \times 3,200$ nodes, and finally for the single-level configuration to $6,400 \times 6,400$ nodes.

All runs were carried out on four processors. Table 3 shows that the resources were consumed exponentially with the decreasing number of levels:

- The 4-level run takes 2.4 times longer to execute than the 5-level run, and the generated data files are 3.3 times larger.
- The 3-level run takes 1.8 times longer than the 4-level run, and the generated data files are 2.8 times larger.
- The 2-level run takes 2.5 times longer to execute than the 3-level run, and the generated data files are 3.4 times larger.
- The 1-level run takes 3.5 times longer to execute than the 2-level run, and the generated data files are 3.8 times larger.

Another way to look at multigrid savings is to emphasize the cost of improving resolution. The cost of improving resolution can be exponential—improving the resolution twice in each dimension results in increasing the number of grid points $2 \times 2 = 4$ times, improving the resolution twice again increases the number of grid points 4 times again, and so on. For n such improvements, the size of the problem grows like 4^n.

Of course, this is not the only way of improving resolution. Resolution may be improved more gradually, not by dividing each cell in half but, for example, by switching from a 400×400 grid to a 500×500 one for the same domain size. But such gradual improvements do not yield significant results, and they cannot be incorporated into the multigrid scheme of Chombo.

When resolution is improved exponentially in a small patch only, the equation changes. Let us suppose the computational domain is square and the small patch to be refined is square, too, and comprises N_p nodes in each direction. Let us suppose, to make the reasoning easier, that the patch is located in the lower left corner of the computational domain. Then the total number of nodes in each direction is $N = N_p + N_0$, where N_0 is the number of nodes in each direction that are not refined. The total number of nodes in the 2D region is

$$(N_p + N_0)^2 = N_p^2 + 2N_pN_0 + N_0^2. \tag{21}$$

Refining the small patch replaces N_p with

$$N_p + 2N_p + 2 \cdot 2N_p + 2 \cdot 2 \cdot 2N_p + \ldots = N_p \sum_{k=0}^{n} 2^k = N_p \left(2^{n+1} - 1\right), \tag{22}$$

where n is the number of levels and where we have made use of the fact that $\sum_{k=0}^{n} 2^k$ is the sum of a geometric series. Hence, the size of the problem grows with the number of levels n as follows:

$$\begin{aligned} &\left(N_p \left(2^{n+1} - 1\right)\right)^2 + 2N_p \left(2^{n+1} - 1\right) N_0 + N_0^2 \\ &\quad = N_0^2 \left(1 + 2\frac{N_p}{N_0} \left(2^{n+1} - 1\right) + \left(\frac{N_p}{N_0}\right)^2 \left(2^{n+1} - 1\right)^2\right). \end{aligned} \tag{23}$$

For $N_p \left(2^{n+1} - 1\right) \ll N_0$ the growth in the size of the problem with n is negligible, but when $N_p \left(2^{n+1} - 1\right)$ approaches N_0, we encounter the original problems, with their onset merely delayed and with the added burden of having to carry out computations for intermediate levels.

This observation gives us a criterion for selecting a good number of levels in a given simulation. We are going to look for such n that

$$N_p \left(2^{n+1} - 1\right) = N_0. \tag{24}$$

From this

$$n = \frac{1}{\ln 2} \ln \left(\frac{N}{2N_p}\right). \tag{25}$$

Let us consider the example discussed in Section 6. We had there that $N_p = 12$ and $N = 400$. Using the above formula, we obtain

$$n = \frac{1}{\ln 2} \ln \left(\frac{400}{2 \times 12}\right) = 4.05889. \tag{26}$$

Table 4
Cost of adding levels. The multilevel runs on the left-hand side of the table all start with the same level 0 resolution of 400×400 nodes. The single-level runs on the right-hand side stretch the target resolution that is the same as for the runs on the left-hand side over the whole computational domain.

Multi-Level Runs				Single-Level Runs			
Configuration	No. of Levels	Snapshot		Configuration	$n_x \times n_y$	Snapshot	
		Time (min)	Size (MB)			Time (min)	Size (MB)
Cylinder-1a	1	0.15	13.44	Cylinder-1b	400×400	0.15	13.44
Cylinder-2a	2	0.29	14.25	Cylinder-2b	800×800	0.59	53.72
Cylinder-3a	3	0.74	15.71	Cylinder-3b	1600×1600	4.00	214.85
Cylinder-4a	4	2.00	19.07	Cylinder-4b	3200×3200	34.00	852.38
Cylinder-5a	5	7.00	28.55	Cylinder-5b	6400×6400	280.00	3437.51

So, our choice of the maximum level being $n = 4$ was fortuitous.

The above consideration ignores the added cost of communication, both within a node and between nodes, which is very considerable for the multigrid method.

8 The Cost of Adding Levels

Table 4 illustrates the empirical cost of improving resolution by adding levels. The configurations in the Cylinder-xa series of runs, where $x = 1, 2, 3, 4, 5$, were constructed by fixing level 0 resolution at 400×400 nodes and then adding levels. The target resolution achieved in these models is obviously different. It is 400×400 for Cylinder-1a, 800×800 for Cylinder-2a, and so on until we reach 6400×6400 for Cylinder-5a. Similarly, the length of the time step shrinks for these models, in order to fit the stability criterion for the finest subgrid. Otherwise the models are the same as the model discussed in Section 6. The numbers in the third column of Table 4 correspond to the wait, in minutes, between successive snapshots. This number grows somewhat faster than exponentially with the number of levels added, but it remains acceptably short, even for the 5-level run (7 minutes). Column 4 shows the size of the snapshot in megabytes. Here the growth is also somewhat faster than exponential, but the size of the data file remains reasonable even for the 5-level run.

The right-hand side of Table 4 shows run times and snapshot sizes for single-

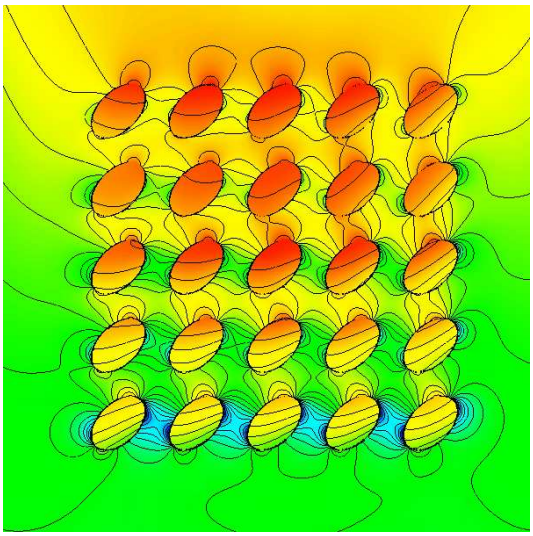

Fig. 8. Scattering on a grid of elliptical cylinders. Fourier transform of the energy density at $t = 650$ (32.5 fs) and $\omega = 0.62831853$ (12.566 PHz). A 400 nm $\times$ 400 nm portion of the full 1,500 nm $\times$ 1,500 nm grid is displayed.

level configurations, where the target resolution was stretched to cover the whole computational domain. These configurations are called Cylinder-xb, where $x = 1, 2, 3, 4, 5$. We note that the Cylinder-1a and Cylinder-1b configurations are identical, as are the Cylinder-5b configuration and the Cylinder-1 configuration from Table 3. We observe explosive growth in the snapshot size in the last column. Here, the cost of high resolution becomes especially significant. The growth in the size of the output is also reflected in the execution time, which is shown in column 7. The savings provided by the multigrid method become substantial in the high-resolution regime.

9 Conclusions

The example introduced in Section 6 is indeed simple. Yet, with only a few small changes to the input file, we can modify it into scattering on a grid of elliptical wires.

Figure 8 shows Fourier transform of the energy density at $t = 650$ (32.5 fs) and $\omega = 0.62831853$ (12.566 PHz) for this system. Here, material and incident signal parameters are the same as in the single cylinder simulation; in other words, the metal is described by a realistic dispersive and absorbing model, and the incident signal is an x-polarized Gaussian wave packet. This is a more complicated simulation and with a larger refined region. Consequently this computation was carried out on 16 IA64 nodes.

Chombo grid manipulation and I/O tools made this computation possible

and easy. Without Chombo we would have to attend to complex logistics of parallel multigrid programming ourselves—a considerable burden. Instead we were able to focus on the physics, solution methodology, and program functionality. Other utilities provided by Chombo, such as visualization and input data handling, proved equally helpful.

Chombo represents a second generation of parallel programming utilities that are now being built on top of MPI. There are still too few of these, but where they are becoming available, they have the potential to revolutionize the way we do scientific computing.

The resulting program is flexible and powerful, allowing us to simulate efficiently and with ease complex, multiscale nanophotonic structures both sequentially and in parallel on systems of varying size, from a single PC to 1024-CPU simulations on the Cray XT3.

The multigrid technique proved highly effective at reducing solution time and, just as important, the size of the output files. To match the speed of the 5-level simulation that was carried out on four PCs, we had to resort to executing the single-level $6,400 \times 6,400$ node job on 256 CPUs of the Cray XT3.

By working in two dimensions we were able to develop and debug the basic framework of SHAPES without getting bogged down in considerable difficulties of handling, analyzing, and visualizing three-dimensional data sets. Yet, the final objective of this research is to develop tools for three-dimensional simulations that will be also applicable to nonlinear materials, essential to nanophotonics. Clearly, the savings offered by the combination of FDTD and multigrid will be even more substantial in three dimensions. In some cases three-dimensional problems of interest to nanophotonics are not even tractable without multigrid. We expect that a three-dimensional extension of SHAPES—called FORMS and currently under development—will become a valuable design and exploration tool for nanophotonics engineers.

Acknowledgments

This work has been supported by the U. S. Department of Energy, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences, and Biosciences, under DoE contract W-31-109-ENG-38. We thank Dr. Tae-Woo Lee, Dr. Misun Min, and Dr. Barry Smith for helpful discussions and their many valuable suggestions.

References

[1] Allen Taflove, Susan C. Hagness, Computational Electrodynamics, Second Edition, Artech House, Boston, 2000.

[2] S. K. Gray, T. Kupka, Phys. Rev. B 68(4), pp. 045415(11), 2003.

[3] T.-W. Lee, S. K. Gray, "Controlled spatiotemporal excitation of metal nanoparticles with chirped optical pulses," Phys. Rev. B, 71, pp. 035423 (1–9), 2005.

[4] M. S. Min, T.-W. Lee, P. F. Fischer, S. K. Gray, "Fourier spectral simulations and Gegenbauer reconstructions for electromagnetic waves in the presence of a metal nanoparticle," J. Comp. Phys. 213, pp. 730–747, 2006.

[5] J. B. Bell, P. Colella, J. Trangenstein, M. Welcome, "Adaptive methods for high Mach number reacting flow," in Proceedings, AIAA 8th Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 9–11, 1987, pp. 717–725.

[6] J. B. Bell, P. Colella, J. Trangenstein, M. Welcome, "Godunov methods and adaptive algorithms for unsteady fluid dynamics," in Proceedings, 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, June 1988, Springer Lecture Notes in Physics, Vol. 323, pp. 137–141.

[7] J. B. Bell, P. Colella, J. Trangenstein, M. Welcome, "Adaptive mesh refinement on moving quadrilateral grids," in Proceedings, 9th AIAA Computational Fluid Dynamics Conference, Buffalo, New York, June 1989, pp. 471–479.

[8] M. J. Berger, P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," J. Comp. Phys., 82(1), pp. 64–84, 1989

[9] D. Govan, E. Bekker, J. D. Paul, S. Greedy, Y. Liu, K. Biwojno, J. Wykes, A. Vukovic, D. W. P. Thomas, T. M. Benson, P. Sewell and C. Christopoulos, "Computational electromagnetics: current applications and future trends," Microwave Review (Mikrotalasna revija), November 2004

[10] G. W. Burr, "FDTD as a nanophotonics design optimization tool," International Symposium on Photonic and Electromagnetic Crystal Structures V (PECS-V), March 7–11, 2004, poster Mo-P49.

[11] X. Zhu, L. Carin, "Multiresolution time-domain analysis of plane-wave scattering from general three-dimensional surface and subsurface dielectric targets," IEEE Transactions on Antennas and Propagation, 49(11), pp. 1568–1578, November 2001.

[12] M. J. White, Z. Yun, M. F. Iskander, "A new 3D FDTD multigrid technique with dielectric traverse capabilities," IEEE Transactions on Microwave Theory and Techniques, 49(3), pp. 422–430, March 2001.

[13] R. Schuhmann, F. Mayer, T. Weiland, "Consistent 3D-FDTD subgrids for microwave applications," in Proceedings of the International Conference on Electromagnetics in Advanced Applications (ICEAA 2003), pp. 125–128, 2003

[14] P. Chow, T. Kubota, T. Namiki, "A block-solve multigrid-FDTD method," The 22nd Annual Review of Progress in Applied Computational Electromagnetics, Miami, Florida, March 12–16, 2006.

[15] P. Colella, D. T. Graves, T. J. Ligocki, D. F. Martin, D. Modiano, D. B. Serafini, B. Van Straalen, "Chombo software package for AMR applications design document," Applied Numerical Algorithms Group, NERSC Division, Lawrence Berkeley National Laboratory, Berkeley, California, September 12, 2003. For more information and to download the package, see `http://seesar.lbl.gov/anag/chombo/`.

[16] Dennis M. Sullivan, "Electromagnetic simulation using the FDTD method," IEEE Press Series on RF and Microwave Technology, IEEE Press, New York, 2000, ISBN 0-7803-4747-1.

[17] T.-W. Lee, S. K. Gray, Optics Express 13, pp. 9652-9659, 2005.

[18] Dennis M. Sullivan, "An unsplit step 3-D PML for use with the FDTD method," IEEE Microwave and Guided Wave Letters, 7, pp. 184-186, July 1997.