Creating Numerically Efficient FDTD Simulations Using Generic C++ Programming

I. Valuev[1], A. Deinega[2], A. Knizhnik[2], and B. Potapkin[2]

[1] Joint Institute for High Temperatures of the Russian Academy of Sciences, Izhorskaya 13/19, Moscow, 125412, Russia
valuev@ihed.ras.ru

[2] KINTECH Kinetic Technologies, Kurchatov Sq. 1, Moscow, 123182, Russia
info@kintech.ru

Abstract. In the present work we propose a strategy for developing reusable multi-model simulation library for solving Finite-Difference Time-Domain (FDTD) problem for Maxwell's equations. The described EMTL (Electromagnetic Template Library) architecture is based on the selection of a small number of primitive low-level physical and numerical concepts which are used as parameters and building blocks for higher-level algorithms and structures. In the present work we demonstrate that a large set of FDTD techniques may be formulated using the same primitives. The basic concept for this representation is a discretized field contour entering the integral form of Maxwell's equations. We also describe the proposed architecture in terms of FDTD C++ template class library and discuss the performance and the usage of this library for various FDTD-based simulations.

1 Introduction

Interaction of electromagnetic fields with periodical materials is an important phenomenon in different device applications, such as radio antennas, wave converters, filters, etc. Recently a new field of application of periodical systems in visible optical wavelength range was proposed [1]. These systems are generally called as photonic crystals, that is, systems which are composed of a periodic arrangement of dielectric or metal material in two or three dimensions [2]. Computer modeling of such periodical systems is a power tool in design of new optoelectronic devices with the desired properties.

A general solution method of Maxwell's equations, which describe the evolution of electromagnetic waves in periodical materials, is the finite difference time domain (FDTD) method [3]. Since introduction in 70th years of the previous century this method became popular due to it certain advantages: (a) simplicity of explicit numerical scheme, (b) applicability to different geometries (only grid should be adapted), (c) explicit description of non-linear materials properties, (d) natural description of impulsive regimes. The inevitable disadvantages of this method evidently result from the above advantages: (a) large memory requirements for fine meshes; (b) strong relationship between time and space steps. The fine features of a considered system can require dense mesh for accurate description, which in turn, results in large memory requirement. For example, accurate

O. Gervasi and M. Gavrilova (Eds.): ICCSA 2007, LNCS 4707, Part III, pp. 213–226, 2007.

description of inverse opal structures with thin metal necks between voids may require fine mesh with cell sizes significantly smaller than characteristic field decay length in the metal. This results in large memory resources for fine mesh and makes corresponding calculations quite expensive.

Several approaches were proposed to overcome the disadvantages of the FDTD method [3]. In particular, using contour-path formulation [4] of Maxwell's equations it was possible to treat accurately sub-cell features without reducing FDTD cell sizes. The more general solution of efficient treatment of strongly non-homogeneous systems is to use non-uniform grids [5]. However, available widespread implementations of FDTD method (such as Meep [6], XFDTD [7], CFDTD [8]) have only limited support of these advanced FDTD methods.

In this work we present a general systematic implementation of FDTD method (called EMTL, Electromagnetic Template Library) based on the contour path formulation of Maxwell's equations, which can simultaneously describe sub-cell features and non-uniform grids. In particular, this implementation of FDTD supports both multi-block grid containers and multi level grids. Thus the total simulation grid can be represented as a set of different grids (which can in turn be non-uniform), covering efficiently each part of the simulated system. The interpolation and connection between different grids in this implementation is done using the contour path formalism. The multi-grid basis of this FDTD implementation provides natural parallelization strategy in the framework of domain decomposition method. Moreover, this implementation also provides full support of modern FDTD method simulation features, such as periodical and perfect matched layer (PML) boundary conditions, total field/scattered field subdivision of the simulation cell, extended library of optical materials properties etc.

The paper is organized as following: in the second section we describe the contour-path FDTD approach and geometry building blocks of EMTL; in the third section the mesh interplay and parallel strategies are presented; the fourth section gives brief idea of the EMTL implementation of the basic FDTD algorithms such as TF/SF, PML, etc.; the fifth section presents some recent benchmarks; conclusions are given in the sixth section.

2 Geometry Components of the Contour-Path FDTD

The integral form of charge-free Maxwell's equations ($c = \epsilon_0 = \mu_0 = 1$):

$$\frac{\partial}{\partial t}\int_S \boldsymbol{B}d\boldsymbol{S} = -\oint_l \boldsymbol{E}d\boldsymbol{l},\ \frac{\partial}{\partial t}\int_S \boldsymbol{D}d\boldsymbol{S} + \int_S \boldsymbol{J}d\boldsymbol{S} = \oint_l \boldsymbol{H}d\boldsymbol{l} \tag{1}$$

$$\oint_S \boldsymbol{D}d\boldsymbol{S} = 0,\ \oint_S \boldsymbol{B}d\boldsymbol{S} = 0 \tag{2}$$

can be used as a fundamental starting point for the formulation of FDTD discretization schemes. Indeed, following contour-path approach [4], let us consider a fixed contour in space, incircling flat polygon (Fig 1 a). For the second-order in space approximation, usually sufficient for linear materials, equations (1) may be rewritten in discretized central-difference form:

$$S(\frac{\partial}{\partial t}\boldsymbol{F}^c_{1,2} perm_{1,2} + \boldsymbol{J}^c_{1,2} - loss_{1,2}\boldsymbol{F}^c_{1,2}) = \pm \sum_i \boldsymbol{F}^l_{2,1}\boldsymbol{l}_i \tag{3}$$

The fields of different conjugated types $\boldsymbol{F}_{1,2} = \boldsymbol{E}, \boldsymbol{H}$ are separated: the fields $\boldsymbol{F}^c$ and source currents $\boldsymbol{J}^c$ measured at the center of the contour and multiplied by the polygon area S are at the left-hand side of (3), while the fields $\boldsymbol{F}^l$ contributing to curl and observed at the edge centers along the contour are entering the right-hand side multiplied by edge vectors $\boldsymbol{l}_i$. The permeability and loss ($perm_1 = \varepsilon$, $loss_1 = \sigma$, $perm_2 = \mu$) are related to the left (surface) side of the equation. They represent material properties:

$$\boldsymbol{D} = \varepsilon\boldsymbol{E}, \qquad \boldsymbol{B} = \mu\boldsymbol{H}, \qquad \boldsymbol{J}_1 = \sigma\boldsymbol{E} + \boldsymbol{J}_{source}, \qquad \boldsymbol{J}_2 = 0. \tag{4}$$

In order to make the explicit time-stepping possible, some kind of approximation, like semi-implicit approximation

$$\boldsymbol{F}(t) = \left[\boldsymbol{F}(t + 0.5\Delta t) + \boldsymbol{F}(t - 0.5\Delta t)\right]/2 \tag{5}$$

$$\boldsymbol{F}'(t) = \left[\boldsymbol{F}(t + 0.5\Delta t) - \boldsymbol{F}(t - 0.5\Delta t)\right]/\Delta t \tag{6}$$

is needed in the left-hand side of (3). Then (3) can be used to express the fields at the next time level via the fields at the previous levels:

$$\boldsymbol{F}^c_{1,2}(t + \frac{1}{2}\Delta t) = \alpha_{1,2}\boldsymbol{F}^c_{1,2}(t - \frac{1}{2}\Delta t) + \beta_{1,2}(\sum_i \boldsymbol{F}^l_{2,1}(t)\boldsymbol{l}_i - \boldsymbol{J}_{source}S), \tag{7}$$

where $\alpha_{1,2}$ and $\beta_{1,2}$ are material-dependent update coefficients for each type of the field. For approximations of greater orders in time or space, equations similar to (3) may be written containing more "reference" points at the polygon where fields have to be known to compose the surface and the curl part of the equation; or containing more "time layers".

If the contour passes the interface between different materials (Fig 1 a), an approximation of the material properties at the left-hand side is also required. The simplest approximation of that kind is taking the material property at the center of the contour (staircase model for Yee meshes). However, for some geometries, subcell averaging models [9] appear to be extremely useful (see Section 5.3).

The numerical data for fields at reference contour points is stored in computer memory which can be represented as many-dimensional vector $\mathbf{M}$. To bind the field value at any given space point $\boldsymbol{x}$ and memory locations (indices of $\mathbf{M}$) the following general interpolation expression is used in EMTL:

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{i \in I(\boldsymbol{x})} \mathbf{c}_i(\boldsymbol{x}) \cdot \hat{L}_i(\mathbf{M}), \tag{8}$$

where $I(\boldsymbol{x})$ gives (normally small) index set for a given point, $\mathbf{c}_i$ is interpolation coefficient. The "memory layout operator" $\hat{L}_i$ binds the interpolation term of index i with the memory array (Fig 1 b). Usually this operator is linear with respect to memory elements: $\hat{L}_i(\mathbf{M}) = \sum_k \alpha_{ik} M_k$. Different mesh layouts, interpolation

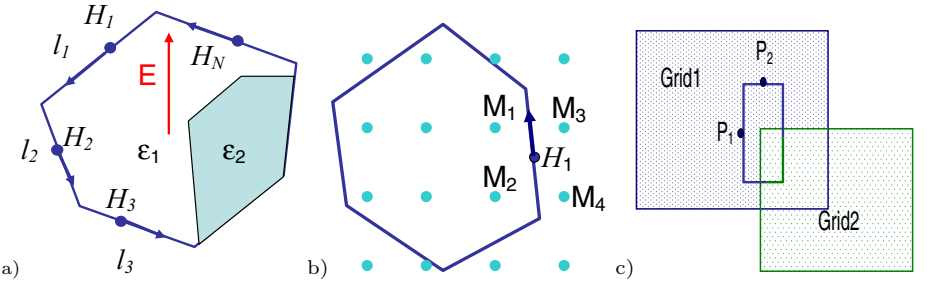

Fig. 1. a) — **H**-contour used to update **E** field in its center; b) — grid points interpolating the field along the contour; c) a contour crossing region boundaries

types and memory storage patterns may be fit into this expression. For example, for 3-linear interpolation in equidistant orthogonal 3D grid $N_0 \times N_1 \times N_2$, with space step Δg and per-component memory layout, the interpolation set consists of 24 elements:

$$I(\boldsymbol{x}) = \{i_{8k+4m_0+2m_1+m_2}\}_{k=0,1,2;\, m_0,m_1,m_2=0,1} \tag{9}$$

$$i_{8k+4m_0+2m_1+m_2} = N_0N_1N_2k + N_1N_2(n_0+m_0) + N_2(n_1+m_1) + n_2 + m_2$$

$$\boldsymbol{c}_i(\boldsymbol{x}) = \boldsymbol{c}_{8k+4m_0+2m_1+m_2} = \boldsymbol{e}_k \prod_{j=0}^{2}((\frac{x_j}{\Delta g} - n_j)(-1)^{m_j} + m_j), \quad \hat{L}_i(\mathbf{M}) = M_i,$$

where $n_j = x_j \bmod \Delta g$ are basic grid indices for the position $\boldsymbol{x}$, the increment of memory pointer by one corresponds to the change in z-coordinate of the grid (array index $j = 2$ changes first in memory). As it will be discussed later (Section 5.1), memory layout may have significant influence on the code performance.

Note that (8) is used both for collecting the input field values from memory before each field update according to (7) and for placing the (different) updated output fields into memory. Using the linearity of (8) and (3) with respect to $\hat{L}_i(\mathbf{M})$, each ith component, corresponding to the memory location $\hat{L}_i(\mathbf{M})$, may be updated separately. This allowed us to implement the interpolation expression (8) in EMTL in most general way. However, special mesh layouts may simplify (8) greatly, reducing the number of the input/output field update combinations. For example, in the standard Yee mesh construction [10], the index set $I(\boldsymbol{x})$ is reduced to one component for each update location $\boldsymbol{x}$ in the center of the contours and their edges, thus resulting in one update combination per contour with four input memory elements (edge centers) and one output element (contour center).

3 Design of the FDTD Simulation in EMTL

3.1 Simulation Stages and Basic Components

As any FDTD simulation, the EMTL-based application has three main computation stages: (1) geometry setup and pre-processing of update coefficients;

(2) main loop with iterative time propagation and recording of output data; (3) processing of output data. Before describing the first two stages of the simulation in more detail, we list briefly the key components of EMTL. The building blocks of EMTL are components that are used as template parameters in higher-level structures. These building concepts are STL-like iterators, points (vectors), contours, bodies, interpolation forms, mesh blocks. The methods of these components are used from the inner loops at geometry setup or time propagation stages and implemented as inline functions for performance reasons.

The basic *geometry* components in EMTL are points in 3D, contours (flat convex polygons) and continuous regions (bodies) in 3D. All bodies implement a concept of `SpaceRegion`, requiring the corresponding C++ classes to define the following functions, necessary for contour path analysis:

- testing whether a point (3D vector) is inside the region;
- for a given flat contour, return the area of a subcontour which is inside the region;
- for a line segment connecting given inside and outside points, return the intersection point of the body surface with the segment and the surface normal vector at the intersection.

The last function is used in subcell models where the interface direction between two materials is important [9]. All regions, except for the main bounding region, may be infinite in one or more dimensions, for example a `Polyhedron` can contain only one plane selecting a half-space with some material properties. Simulation in one or two dimensions is possible by reducing the size of non-existent dimensions to two contours and introducing periodic boundary conditions in the reduced directions.

The meshes, or `emMeshBlock` objects, are the building blocks of the FDTD model and define a set of update contours, memory layout and interpolation. Thus, EMTL separates the mesh design task (which is itself a complicated problem) from the FDTD simulations which use a particular mesh. Practically, any FDTD method available in EMTL may be used with arbitrary meshes and even their combinations. The `emMeshBlock`-conforming classes must implement the following key functionality:

- define a "control region" where the mesh can provide an interpolation of fields;
- return an interpolation for a given field type ($\boldsymbol{E}$ or $\boldsymbol{H}$ at a given location $\boldsymbol{x}$ in the form (8), binding memory array indices and coefficients;
- define an iterator of all $\boldsymbol{E}$ and $\boldsymbol{H}$ contours composing the mesh;
- provide optimized basic bulk update function and accept basic update coefficients α, β (7) for contours
- optionally accept indications to "unused" (not updated) contours to save memory.

Various kinds of meshes can be designed to conform to these model, including non-uniform, unstructured and non-orthogonal ones. Note that the chain of contours defined by the mesh must satisfy the second pair of the Maxwell's equations (2).

3.2 Geometry Analysis Stage

The contour collection that is returned by `emMeshBlock` is analyzed at geometry setup stage when the update coefficients for each contour are calculated. The analysis is performed only once before the iterative update stage, therefore this part of a simulation is less performance critical and may be generalized. The contour analysis algorithm is the main algorithm of EMTL, it is implemented in the most general way and does not depend on the mesh construction details. The only requirements for the mesh is the conformal definition of a contour: (a) contour is a sequence of points; (b) contour is flat and convex; (c) here is a binding between contour characteristic points (center for output and edge centers for input) and memory locations provided by `emMeshBlock`; (d) contour is connected with some "control" volume. The last contour property is used in smoothing models only where volume averaging is required. The contour analysis algorithm can be easily extended to include new types of equations, materials and subcell smoothing models.

To construct the whole task geometry, the components are assembled in the object of type `emBlockContainer<mesh_type1,mesh_type2>`. This is the main "access object" of the model, it can contain arbitrary number of mesh blocks, each of them belonging to one of the two mesh types. The `emBlockContainer<>` can itself be treated as an aggregate `emMeshBlock`, allowing to combine arbitrary number of mesh types in one model. The top level container must have a bounding region for which boundary conditions are defined. The boundary conditions determine the behavior when a certain contour crosses the container boundary and some of its input points are outside. In case of periodic boundaries, these points are reverted to the other side of the container using "transfer fix", while for reflective conditions the input fields located outside are assumed to be zero.

Multiple `emMeshBlock` objects may be placed in the container having control regions with different interpolation levels assigned to them. In case the meshes intersect, the actual field update is performed in the mesh of higher level, while the lower level meshes use interpolated values for the input points of their contours (Fig 1, c). The same mechanism works for partially intersecting meshes or when transfer between different CPU domains is needed. In the later case, the MPI transfer is taking place along with data interpolation.

3.3 Separation of Duties: Basic and Specialized Updates

The equation (7) with constant material properties (4) is the basic field update relation in EMTL. Any mesh (the object of class `emMeshBlock`) must be capable of performing this type of update on a specified range of mesh contours. This repeatedly executed basic bulk update takes usually the major part of CPU time during the simulation. Although a generic EMTL update step implementation based on (7) and (8) may be utilized, the usage of specialized algorithms taking memory layout for a particular mesh into account can greatly improve the code performance. Because of its relatively simple localized pattern, the basic update may be decomposed into the processing of independent memory streams. The

minimal number of streams (four for input and one for output in homogenous medium case) is achieved in the Yee mesh layout. The advance of the streams (read or write) with unit memory increment is interchanged with stream rebinding when contour iterator reaches mesh boundaries (see Section 5.1).

Possible modifications to (3), resulting in applying different update equations to some contours are introduced in the form of corrections to (7) or *Fixes* in terms of EMTL. The modified equations are usually needed to include various features in the FDTD model, such as source field generation, absorbing boundary conditions, dispersive materials, etc. Note that unlike the traditional approach of including the specialized updates into the main update loop and testing the equation type flags on each iteration, these updates are performed as separate *Fix loops* in EMTL. The identifiers of the contours requiring special processing are recorded in the packed fix lists. There are fixes that are performed before (changing the input fields) and after (correcting the output fields) the basic field update of all contours. For example, the data transfer between different meshes in the area of mesh intersection is implemented as a pre-step fix. The transfers between different CPU domains in case of MPI parallel execution require both pre-step fix (initializing the boundary sends) and post-step fix (completing receives and updating the boundary-dependent contours). Unlike the basic update, the fixes contain model-specific algorithms and are implemented in a general mesh-independent way. However, when the number of contours affected by a certain fix is large, the mesh-specialized implementation may be useful to gain performance. EMTL provides a means for the code developer designing a mesh to define such specialized updates.

As for all explicit time integration schemes, the maximal time step dt for field update in FDTD is connected with the space resolution dx and propagation speed c by the Courant condition: $dt \approx dx/c$. Thus, when meshes with different space resolutions coexist in the model, it is necessary to propagate them with different time steps. In EMTL, the meshes are advanced in the order of their time steps, the mesh with the largest possible time step is updated first determining the next target time. All other meshes are advanced in order till they reach the target time. Special "synchronization fix" is used for transfers between meshes with different time layers. For that transfers, the field data from the previous time level is also required to interpolate the input fields both in time and space. This pre-step fix stores the required for interpolation to other meshes field values of current time level, before they are overwritten by the next field update.

3.4 Optimizing Memory Usage

The fix loops iterate over a subset of the contour set provided by the mesh. The sequence of "fixed" contours depends mainly on large-scale geometric setup of the model and is in most shows regular patterns. Moreover, the data utilized by contour fixes often contains the same corrections for different contours. To account for this regularity, a simple data packing techniques are used in EMTL. The packing is based on the detection of linear trend subsequences (identical increment) in the sequences of integers. The subsequences are then encoded by

replacing them with an indicator code (-1) followed by the (increment, number) pair. Although very primitive, this compression permits to save much memory for bulk corrections and does not induce significant overheads when unpacking. Both the iterator sequences and data index sequences (pointers to identical data values) can be effectively packed using this strategy. For complex correctors such as UPML fix, utilizing large amount of data, the fix update sequence reordering can be utilized to improve cache efficiency when accessing the packed data. In this case, the order of contour iterations may be adjusted corresponding to the order of unique data values stored in memory.

3.5 Parallelization Strategy

EMTL supports parallelization using space domain decomposition which is a common technique for time domain solvers. The CPU domains may be arbitrary `SpaceRegion`s, subdividing the main simulation region. The EMTL contour analysis algorithm detects contours crossing domain boundaries and registers "transfer fixes" for the contour input points belonging to non-local meshes or CPU domains. The data interpolation coefficients required to represent data from a source different from the mesh the contour originates from (different domain or different mesh) are recorded in transfer fix. They are convolved with the data taken from the remote source on each field update iteration. The only difference for parallel execution is that the recorded data transfer requires MPI message rather than direct memory copy.

The automated balanced decomposition based on bisection algorithm is implemented in EMTL and may be utilized to decompose the main region into box-shaped domains according to the number of processors detected at program start. To split the region into balanced domains, the algorithm uses a number of recursive region bisections by planes in axial directions. The bisections are performed in such a way that two resulting subregions are balanced in terms of $n_i \int_{V_i} w(\boldsymbol{x}) dV$, where n_i is the number of processors assigned to subregion i, w and V_i are "workload density" and the subregion volume correspondingly. The workload density represents the cost of computations for a given space location. It can be approximated by roughly sampling the fraction of different materials/equations along the bisection axis. The same contour path algorithm, determining the crossing fraction of a contour with a body is used for this purpose. The correctors are assigned experimentally determined workload weights (the basic bulk update has the weight of 1.). For example, for UPML absorbing boundary condition fix (see Section 4.3) the measured weight is approximately 2.5. The bisection algorithm is then applied to the subdivisions and stops recursion when the number of assigned processors for a subregion reaches one.

Techniques trying to optimize cache efficiency of the basic bulk update by reducing the number of memory stream rebindings are possible within EMTL decomposition algorithm. For example, the authors of [11] argued that the "pencil" memory layout can save up to 60 per cent of computational time on Beowulf type architectures. In this technique the space dimension corresponding to the most rapidly changing data array index is kept as extensive as possible, while

domain decomposition is performed in the remaining dimensions. The EMTL bisection algorithm permits to specify the number of bisections (including zero) along selected axis explicitly, and the effects of pencils may be studied directly (see Section 5.1).

4 Implementing Basic FDTD Algorithms as Correctors

In this section we briefly describe the FDTD techniques available in EMTL in the form of "fixes", or corrections to the basic bulk update equations (7).

Wave Generation by the Total Field/Scattered Field Method. The TF/SF technique [3] is a very effective source field generation method when the simulation domain may be split into the Total Field (TF) and Scattered Field (SF) regions. It is assumed, that the incident wave propagation is known analytically for the case there are no scattering objects. The update equations in the SF zone are solved for the *scattered field* which is a difference between the total field and the known unscattered incident field. The TF zone contains scattering objects and the update equations for the total field are solved there. The update equations have exactly the same form (7) in both zones, except for the places where the update contour passes the TF/SF boundary. The output point and the input points of such a contour may be classified to be TF or SF points according to their positions. If the update involves input points of the type different from the output, then a corresponding correction (incident field at the contour edge) is added or subtracted:

$$\boldsymbol{F}^c(t+\frac{1}{2}\Delta t) = \boldsymbol{F}_{bulk} \pm \beta(\sum_{diff.\ zone} \boldsymbol{F}^l_{inc}(t)\boldsymbol{l}_i), \tag{10}$$

where $\boldsymbol{F}_{bulk}$ is the basic update at the right hand of (7), "+" sign is used when the output point belongs to the TF zone and "−" otherwise. The TF/SF fix records the update coefficients $\pm\beta\boldsymbol{l}_i$ for each combination of input/output memory indices composing the interpolations of $\boldsymbol{F}$. These indices are also classified to be of TF or SF type.

Absorbing Boundary Conditions. This technique prevents reflection of the wave from the container boundaries thus simulating a finite object (container) in an infinite surrounding medium. We implemented the Uniaxial Perfectly Matched Layer [3] (UPML) technique as a fix within EMTL. To guarantee absorbtion of a signal on some interface at any incidence angle, the "matched" frequency-dependent dielectric/magentic tensors are introduced in absorbtion zone instead of ε and μ in (4):

$$\boldsymbol{D} = \varepsilon\, s(\omega)\, \boldsymbol{E}, \qquad \boldsymbol{B} = \mu\, s(\omega)\, \boldsymbol{H}, \tag{11}$$

where $s(\omega)$ is a diagonal tensor. The frequency dependence of its diagonal elements is a rational function of the form $(a_1 + jb_1\omega)/(a_2 + jb_2\omega)$, which leads to

the differential equation of the first order linking the components of the contour-centered fields $\boldsymbol{D}^c$ and $\boldsymbol{E}^c$ in the time domain, for example:

$$k_1 \frac{\partial}{\partial t} D_x^c + k_2 \frac{\partial}{\partial t} E_x^c + k_3 D_x^c + k_4 E_x^c = 0. \tag{12}$$

This equation supplements the basic update (7), which in this case is formally performed for $\boldsymbol{D}$. The equations for $\boldsymbol{B}^c$, $\boldsymbol{H}^c$ pair have the similar form. The function of the EMTL fix for the contours entering UPML region is to reserve a space in memory for extra independent variables ($\boldsymbol{D}$, $\boldsymbol{B}$) and coefficients k_i, and to solve (12) at each update iteration using explicit time stepping.

Dispersive Materials. The addition of the auxiliary update equations to the contour basic update is also used when frequency-dependent dispersion is introduced in the dielectric function. This representation is usually originating from the fits of the dielectric function of real materials in frequency domain, approximating it with multiple terms of the Debye, Drude or Lorentz form [12]. These terms may be generalized by the formula [3]:

$$\varepsilon(\omega) = \sum_i \frac{a_i + j b_i \omega}{c_i + j d_i \omega + e_i \omega^2}, \tag{13}$$

where a_i ... e_i are real coefficients. Each term of (13) is associated with a partial polarization $\boldsymbol{P}_i = \varepsilon_i(\omega)\boldsymbol{E}$. The equations for $\boldsymbol{P}_i$ are converted to the time domain, resulting in the differential equations of the order determined by the highest power of ω in the denominators of $\varepsilon_i(\omega)$. The EMTL fix managing the variables $\boldsymbol{P}_i$ and their update equations for the contours crossing dispersive material regions is organized in the way similar to the UPML fix described in the previous subsection.

5 Illustrative Examples and Benchmarks

In this section we present the actual simulation results and comparisons obtained so far with EMTL-based applications. In the benchmark subsection we compare EMTL with GPL-licensed MEEP code developed in MIT [6]. Although not implementing non-Yee meshes and multiple meshes, this parallel code offers a variety of FDTD techniques and can be used as a library tool.

5.1 Mesh Memory Layout Testing

Mesh memory update is the central procedure in the time domain equation solvers. The mesh data itself is related to physical properties measured at certain locations in space. This localization is often used as a basis for memory layout, where the data entries responsible for physical properties characterizing a given mesh cell are stored in adjacent memory locations. Usually the equations imply update loops of multiple kinds, when only a limited number of physical properties

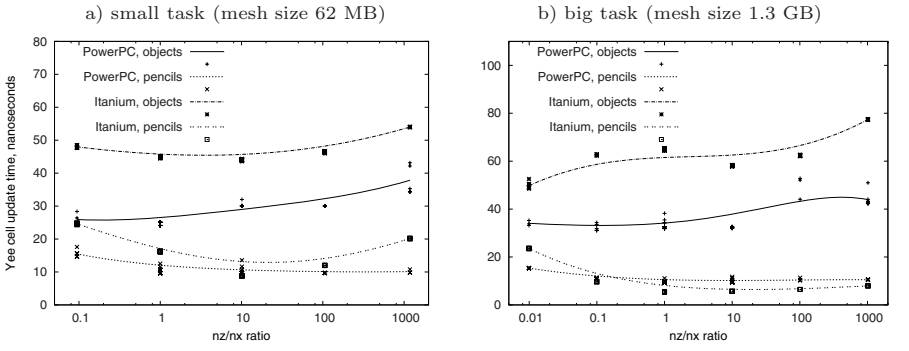

Fig. 2. Effects of the memory layout and simulation domain proportions ($nx = ny$ by varying nz) on FDTD performance in vacuum update (constant material properties). The code was compiled on PowerPC with IBM xlc 7.0 (-q64 -O5) compiler and on Itanium with Intel icc 9.0 (-O3) compiler. The tests were performed at the Joint Supercomputer Center of RAS [13].

are affected by the individual update loop, applied for all cells. In this case the "cell as an object" memory organization may reduce cache efficiency because it increases the memory pointer stride when accessing mesh data. An alternative is the memory layout where mesh data is grouped by the physical property in possibly long and contiguous arrays, or "pencils" [11].

Being a template library, EMTL supports arbitrary memory layout strategies and allowed us to study the memory layout effects for FDTD (Fig. 2). Two orthogonal Yee meshes of size ($nx \times ny \times nz$) were used for testing: the one where six field components per each Yee cell are stored in adjacent memory ("objects"), and the other where each component type is stored in a separate array ("pencils"). In both cases the most rapidly changing spatial mesh index was the one along z-direction of the simulation domain. Choosing different nz by keeping $nx = ny$ and constant domain volume, we measured the update time per Yee cell. On both considered architectures (Itanium 2 and PowerPC 970) the "pencil" layout resulted in much higher performance. The memory stream length (nz/nx) parameter also affects performance, but to less extent than the layout itself.

5.2 Parallel Benchmarks

To study the parallelization efficiency of our code, we performed a set of scaled and unscaled tests on different platforms (Fig. 3). For the *unscaled* tests the task size is kept constant and the execution time $t(N)$ is measured while varying the number of processors N, the parallel efficiency is calculated as $t(1)/Nt(N)$. For the *scaled* tests the mesh size is proportional to the number of processors and the parallel efficiency is computed as $t(1)/t(N)$.

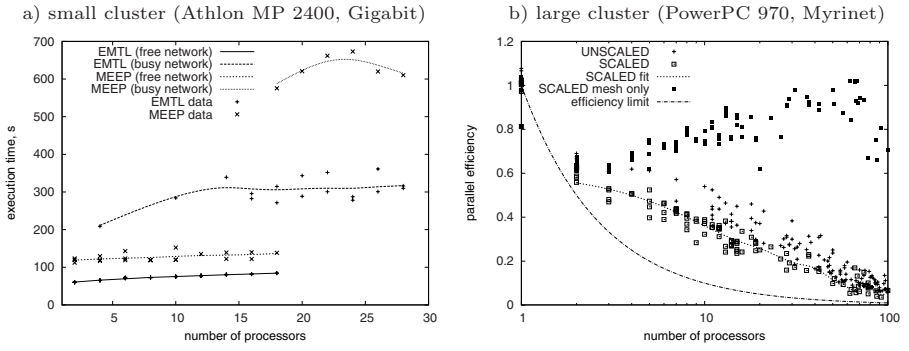

Fig. 3. Parallel benchmarks. a) — scaled test compared to MEEP [6] code on a small cluster [14], the basic task size is 140^3 cells vacuum box, scaled in one dimension proportionally to the number of processors, 100 timesteps; b) — tests on a large cluster [13] in its regular multi-user production regime, basic task size is 300×100^2, limiting minimal efficiency curve $1/N$ is drawn for comparison. The efficiency of memory update (not taking network transfer times into account) is shown by filled squares.

The execution time trends for a small cluster, entirely dedicated to the task, may be branched according to the two regimes of cluster network load (Fig. 3, a), the interconnecting Gigabit switch presumably being the critical device limiting performance. The difference in performance between MEEP and EMTL growth in inverse proportion to the bandwidth of the switch, thus it can be mainly attributed to the parallelization. The parallel efficiency tests of the typical 1000-timesteps task on the large production system (Fig. 3, b) also show strong sensitivity of the code performance to the network state of the whole system. Large variation of the results, while *unscaled* efficiency is surprisingly higher on average than the scaled one, contributes to this conclusion. For the particular cluster studied, using medium number of processors (10–40) rather than small number (2–10) in parallel execution is still profitable (relative efficiency factor is about 0.7).

5.3 Testing Subcell Averaging Methods

The contour architecture of EMTL simplifies implementation of various subpixel averaging methods, used to improve FDTD accuracy when modeling discontinuous materials or small objects (Figs. 4, 5). We performed a series of tests using a) contour/volume averaging of material properties for metals; b) contour/volume averaging of ε or ε^{-1} for dielectrics; c) mixed tensor averaging [9]. In the latter case the ε for the contour is replaced by the dielectric tensor of the form $\epsilon^{-1} = P < \varepsilon^{-1} > +(1 - P) < \varepsilon >^{-1}$, where $P_{ij} = n_i n_j$ is the projection matrix onto the normal to material interface. We tested currently the mixed averaging using only the diagonal part of P (referred to as `dTensor` in the Fig 4). A gaussian pulse with characteristic width of 20 Yee space steps, impinging a single metallic or dielectric sphere with radius of 0.5–7 space steps, was used for testing. Scattering cross section from the sphere obtained for different subcell averaging methods was compared with the Mie

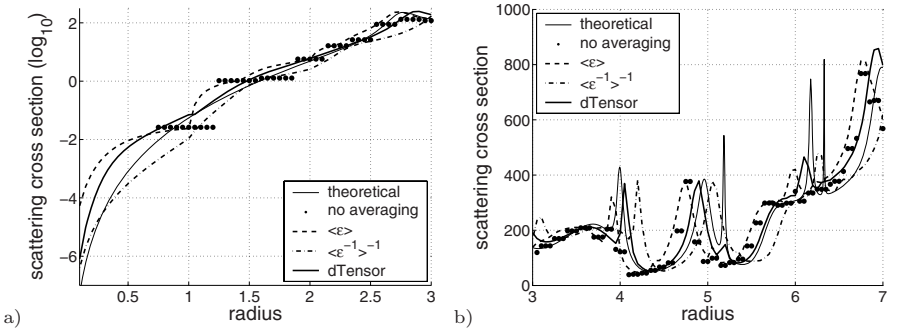

Fig. 4. Scattering from a small dielectric sphere ($\varepsilon = 12$) at $\lambda = 20$ using various subcell dielectric function smoothing methods compared to the Mie theory. The averaging is performed over "contour surface", except for dTensor case, where control volume average is used. The unit length is the Yee space step.

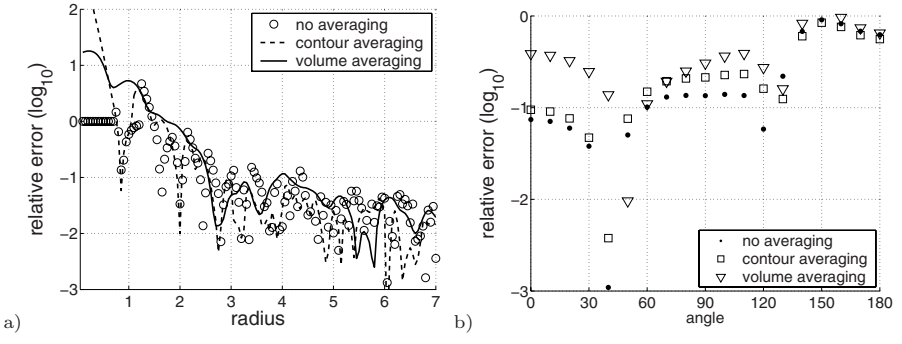

Fig. 5. Scattering from a small metallic sphere compared to the Mie theory, the metal is modeled by two Lorentz [3] terms $\delta\varepsilon_1 = \delta\varepsilon_2 = 0.5$, $\delta_1 = \delta_2 = 2\pi\times0.001$, $\omega_{01} = 2\pi\times0.04$, $\omega_{02} = 2\pi \times 0.06$. a) — relative error of the scattering cross section at ω_{02} as a function of the sphere size; b)— relative error of S_{11} scattering matrix element of $R = 5$ sphere at ω_{02} as a function of the scattering angle. The unit length is the Yee space step.

theory. For the metallic sphere, the scattering matrix element S_{11} [15] at resonant frequency was also studied as a function of observation angle (Fig. 5, b). To obtain fields at large distances and different scattering angles, Near-to-Far field transformation [16] was used in transforming the fields measured at 2.5 radius distances from the sphere to the far fields.

The results of the tests show, that averaging is important for small objects. The direct $< \varepsilon >$ and harmonic $< \varepsilon^{-1} >^{-1}$ averages lead to comparable but opposite in sign error, the proper mix of both [9], even using diagonal tensor only, leads to much smaller errors. The averaging over a "control volume" of a contour rather than over a contour "encircled surface" may be important for properly distinguishing very small objects but does not lead to any apparent improvement in the accuracy.

6 Conclusions

In the present work we demonstrated that a large set of FDTD techniques may be formulated using the same primitives. The basic concept for this representation is a discretized field contour entering the integral form of Maxwell's equations. The main aim of this paper was to describe contour-based mesh architecture utilized in EMTL and demonstrate its universality and performance. Some benchmarks and test obtained so far with EMTL are presented, the others are subject to the future work.

The library itself is a research tool oriented on the model code developers who need to use FDTD as part of their simulations. Being a library, rather than a standalone application, EMTL provides a set of template classes to implement a variety of modern FDTD techniques in the efficient parallel C++ code.

Acknowledgements. This work is partially supported by the research program #14 of the Russian Academy of Sciences. The benchmarks were performed using hardware of the Joint Supercomputer Center of RAS and the Department of Molecular and Biological Physics of the Moscow Institute of Physics and Technology.

References

1. Joannopoulos, J.D., Meade, R.D., Winn, J.N.: Photonic crystals: molding the flow of light. Princeton University Press, Princeton (1995)
2. Johnson, S.G., Joannopoulos, J.D.: Acta Materialia 51, 582 (2003)
3. Taflove, A., Hagness, S.H.: Computational Electrodynamics: The Finite Difference Time-Domain Method. Artech House, Boston (2000)
4. Jurgens, T.G., Taflove, A., Umashankar, K.R., Moore, T.G.: IEEE Trans. Antennas and Propagation 40, 357 (1992)
5. Dey, S., Mittra, R.: IEEE Microwave and Guided Wave Lett. 7, 273 (1997)
6. MIT Electromagnetic Equation Propagation, `http://ab-initio.mit.edu/wiki/index.php/Meep`
7. Remcom's full wave FDTD solver, `http://www.remcom.com/xfdtd6/index.htm`
8. RM Associates' conformal FDTD code, `http://www.rm-associates.biz/software.html`
9. Farjadpour, A., Roundy, D., Rodriguez, A., et al.: Optics Letters. 31, 2972 (2006)
10. Yee, K.S.: IEEE Trans. Antennas and Propagation 14, 302 (1966)
11. Dobbler, W., Haugen, N.E.L., Yousef, T.A., Brandenburg, A.: Phys Rev E 68 (2003) 026304, `http://www.nordita.dk/software/pencil-code`
12. Roberts, S.: Phys Rev. 114, 104 (1959)
13. Joint Supercomputer Center of the Russian Academy of Scienece, `http://www.jscc.ru`
14. Computational cluster of the Department of Molecular and Biological Physics, MIPT `http://biolab1.mipt.ru`
15. Bohren, C.F., Huffman, D.R.: Absorption and Scattering of Light by Small Particles. Wiley-Interscience, New York (1983)
16. Umashankar, K.R., Taflove, A.: IEEE Trans. Electromagnetic Compatibility 24, 397 (1982)