CUDA are presented in [6], which can be used as guidelines while programming FDTD using CUDA. The discussions are based on FDTD updating equations in its simplest form: updating equations consider only dielectric objects in the computation domain, the cell sizes are equal in *x*, *y*, and *z* directions, thus the updating equations include a single updating coefficient. The efficient use of shared memory is discussed, however the presented methods limits the number of threads per thread block to a fixed size. The coalesced memory access, which is a necessary condition for efficiency on CUDA, is inherently satisfied with the given examples; however its importance has never been mentioned.

In this current contribution a CUDA implementation of FDTD is provided. The FDTD updating equations assume more general material media and different cell sizes. A thread-to-cell mapping algorithm is presented and its performance is provided.

FDTD Using CUDA

The unified FDTD formulation [8] considered for CUDA. The problem space size is $Nx \times Ny \times Nz$, where Nx, Ny, and Nz are number of cells in *x*, *y*, and *z* directions, respectively. Thus, for instance, the updating equation that updates *x* component of magnetic field is given in [8] as

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,j,k) &= C_{hxh}(i,j,k)\, H_x^{n-\frac{1}{2}}(i,j,k) \\ &+ C_{hxey}(i,j,k)\left(E_y^n(i,j,k+1) - E_y^n(i,j,k)\right). \\ &+ C_{hxez}(i,j,k)\left(E_z^n(i,j+1,k) - E_z^n(i,j,k)\right) \end{aligned} \tag{1}$$

In order to achieve parallelism, the threads are mapped to cells to update them. For the mapping, a thread block is constructed as a one-dimensional array, as shown on the first two lines in Listing 1, and the threads in this array are mapped to cells in an *x*-*y* plane cut of the FDTD domain as illustrated in Fig. 1. In the kernel function, each thread is mapped to a cell; *thread index* is mapped to *i* and *j*. Then, each thread traverses in the *z* direction in a *for* loop by incrementing *k* index of the cells. Field values are updated for each *k*, thus the entire FDTD domain is covered.

```
block_dim_x = number_of_threads; block_dim_y = 1;
n_blocks_y = 1;
n_blocks_x = (nxx*nyy)/number_of_threads +
                              ((nxx*nyy)%number_of_threads ==0?0:1);
```

Listing 1. CUDA code to define block and grid sizes.

Unfortunately in FDTD updates the operations are dominated by memory accesses. In order to ensure high performance all global memory accesses shall be coalesced. In general an FDTD domain size would be an arbitrary number. In order to achieve coalesced memory access, the FDTD domain is extended by padded cells such that the number of cells in *x* and *y* directions is an integer