Accelerating Finite Difference Time Domain Simulations with Reconfigurable Dataflow Computers

Heiner Giefers
IBM Research – Zurich
hgi@zurich.ibm.com

Christian Plessl, Jens Förstner
University of Paderborn, Germany
{christian.plessl,jens.foerstner}@uni-paderborn.de

ABSTRACT

Finite difference methods are widely used, highly parallel algorithms for solving differential equations. However, the algorithms are memory bound and thus difficult to implement efficiently on CPUs or GPUs. In this work we study the implementation of the finite difference time domain (FDTD) method for solving Maxwell's equations on an FPGA-based Maxeler dataflow computer. We evaluate our work with actual problems from the domain of computational nanophotonics. The use of realistic simulations requires us to pay special attention to boundary conditions (Dirichlet, periodic, absorbing), which are critical for the correctness of results but detrimental to the performance and thus frequently neglected. We discuss and evaluate the design of two different FDTD implementations, which outperform CPU and GPU implementations. To our knowledge, our implementation is the fastest FPGA-based FDTD solver.

1. INTRODUCTION

Finite difference methods are very important and widely used methods for numerically solving partial differential equations, which describe for example, electromagnetic field propagation, heat flow, or diffusion problems. The resulting, iterative algorithms are very regular and apply the same update operation to all grid points in each time step. This operation requires access to a fixed neighborhood of the grid point and are thus denoted as stencil operations. In spite of the regularity and parallelism of stencil algorithms, their efficient implementations for CPUs and GPUs is challenging because of the low ratio of data that must to be read to the number of operations executed on the data. Hence, the performance collapses when problem sizes exceed the capacity of the CPU cache or GPU global memory. The regular operations and data access patterns of stencil codes are however ideally suited for a streaming dataflow computing approach implemented with FPGAs, which allows for building deep and highly customized processing pipelines and supports large memory with application-specific memory controllers.

The objective of this work is to study the suitability of the dataflow approach for a realistic finite difference applications. Specifically, we study the finite difference time domain algorithm (FDTD) for simulating the propagation of electromagnetic fields. We target an FPGA-based Maxeler dataflow system and evaluate our work with two case studies from the domain of computational nanophotonics:

1. We study a *microdisk cavity* in a perfect metallic environment, a structure well suited for a non-synthetic 2-dimensional benchmark. This setup uses a constant (Dirichlet) boundary condition, which greatly simplifies the implementation and evaluates the raw stencil processing performance on a regular grid.

2. We simulate a *periodic structure* in a 3-dimensional domain. Such structures are receiving great attention due to their novel applications as photonic crystals and meta-materials [10]. To take advantage of the periodicity one can model only a single unit cell of the infinite periodic structure and apply periodic boundary conditions (PBC) at the connection planes. Since the periodic repetition typically occurs only in a two dimensional plane, a different (normally absorbing) boundary condition has to be applied in the third dimension.

The handling of boundary conditions is frequently neglected in explorative studies, since they break the regularity of the code and are thus detrimental to the performance. Handling non-trivial boundary conditions is however indispensable for actual physical simulations. Hence a specific contribution of our work is to support Dirichlet, periodic, and absorbing boundary conditions. We show that high speedups can be achieved if the the boundary conditions can be handled completely in the dataflow engine. If computing the boundary conditions requires processing on the CPU, the dataflow architectures cannot reach their performance potential due to communication overheads. Our implementations outperform a previously published FPGA solution by more than 100x and the fastest GPU solvers by 2x per device.

The remainder of this paper is structured as follows. Section 2 briefly introduces Maxeler's concept of dataflow computation and Section 3 provides the theoretical background for the FDTD method. In Section 4 we preset the implementation of our 2D and 3D FDTD dataflow accelerators, which are evaluated for different scenarios in Section 5. After discussing related work in Section 6 we conclude the paper and give an outlook on future research.

2. DATAFLOW COMPUTING WITH MAXELER

Dataflow and systolic architectures for parallel computing have been proposed since the 1970s. The basic principle of all dataflow machines is the *firing rule* which means that an operation is able to proceed as soon as all required (input) data is available. A natural way to specify a program for a dataflow machine is a dataflow graph (DFG), which can be turned into a hardware circuit, resulting in a highly specialized proces-

This work was undertaken at the University of Paderborn. The authors thank the Maxeler Technologies University Program MAXUP for their support.

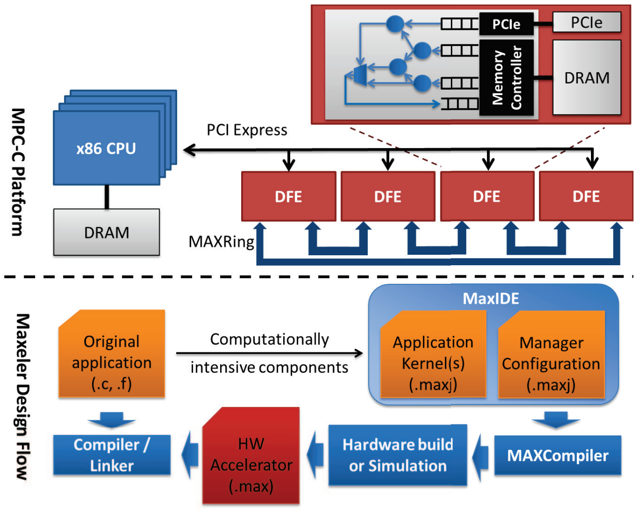


Figure 1: MPC-C platform architecture and Maxeler design flow.

sor. If reconfigurable hardware is used as the target platform the processor can be adapted to various applications.

We illustrate this transformation for the reconfigurable dataflow engines from Maxeler. Figure 1 shows a sketch of a Maxeler MPC-C node and the corresponding design flow. This FPGA based accelerator cards (called *dataflow engine*, DFE) communicate via PCIe to x86 server CPUs. The DFEs in a node are additionally connected through MAXRing, a high-bandwidth, low-latency interconnect. A compute node, consisting of two Xeon CPUs, four DFE cards and up to 384GB of memory consumes about 500-600W under load [8].

The Maxeler system is a hybrid architecture, i.e., only the compute intensive parts of an application are processed on the accelerator cards, while the main part (in terms of lines of code) stays on the CPUs [2]. Typically, HPC applications involve one to a few timing critical operations (kernels) which consume a vast amount of the runtime. To accelerate such an application on Maxeler's platforms, the programmer has to implement dataflow specifications for the particular kernels. These dataflow programs can be formulated in Java combined with Maxeler's language extensions [7] leading to a high-level specification of the hardware accelerator. An accelerator specification is typically composed of two parts. The *kernel* comprises the data-path implementation of the application. With a *manager*, the programmer arranges the data streams between kernels, off-chip memory, and CPUs.

The MaxCompiler tool transforms the kernel and manager code into a hardware design which is further processed by the FPGA vendor tools to generate a bitfile. The bitfile is merged with meta information into a *maxfile*, which can be linked into a CPU application. For controlling and configuring the kernel execution from the host application, MaxCompiler generates a dedicated interface library for the kernel.

For 3D finite-difference applications, Maxeler provides a domain-specific compiler called *MaxGenFD* [8]. MaxGenFD is built on top of MaxCompiler and provides further abstractions such as stencil operations, convolutions, or stimuli injection. MaxGenFD automatically decomposes the 3D domain into blocks and instructs the memory controllers to process the blocks by streaming the particular data from/to accelerator memory through the FPGA. MaxGenFD supports IEEE 754 single precision, a multitude of reduced precision, and fixed-point data formats. The compiler automatically generates a customized datapath for the selected data types.

3. THE FDTD METHOD

The prevalent method for electromagnetic simulations of nanophotonic structures like photonic crystals and plasmonic devices is the FDTD technique originally proposed by Yee [14]. As an alternative to solving the electric (or magnetic) field alone by means of a wave equation, the FDTD method solves for both fields in time and space using the coupled Maxwell curl equations. The light is represented by the electric (E) and magnetic (H) vector fields, which are discretized on an interleaved cartesian grid for numerical simulations. For every grid point in the computational domain, the FDTD calculates the E and H fields at a current time step so that the overall simulation describes the progression of the electromagnetic field movement through the model over time. The FDTD method allows for modeling a multitude of linear and nonlinear dielectric and magnetic materials. It has been proven to be one of the most effective numerical methods for studying metamaterials and is used in a wide range of applications. For a state of the art review of the FDTD method and its and applications we refer to *Taflove and Hagness* [13].

A main drawback of the FDTD method is the high computational requirements which are typical of time-domain simulation. The original FDTD algorithm requires that the entire computational domain is modeled as a regular grid and that the resolution of grid points is sufficiently fine-grained to model the smallest geometrical features and wavelengths of the experiment. Adequate discretizations can lead to large computational domains which results in very long solution times and enormous memory requirements. Even advanced approaches like nonuniform or adaptive grids can only partially compensate this scaling.

3.1 FDTD Algorithm

The starting point for the FDTD are Maxwell's curl equations for linear, isotropic, non-dispersive, lossy materials:

$$\frac{\partial H}{\partial t} = -\frac{1}{\mu}\nabla \times E - \frac{1}{\mu}(M_{source} + \sigma^{*} H) \qquad (1)$$

$$\frac{\partial E}{\partial t} = \frac{1}{\epsilon}\nabla \times H - \frac{1}{\epsilon}(J_{source} + \sigma E) \qquad (2)$$

H and E represent the magnetic and electric field, μ is the electric permeability and ϵ is the electrical permittivity. M_{source} and J_{source} are the magnetic and electric current densities, acting as independent sources and σ^{*} and σ depict the magnetic and electric conductivity loss, respectively.

The FDTD simulation proceeds by iteratively updating the spatially and temporally interleaved H and E fields. The stencil definition for H_z is given below, the stencils for H_x and H_y can bee obtained by cyclic rotation of the coordinates x, y, z. The stencils for E are structurally very similar [13]. n is a unit-less index for the simulation step, which is related to the simulation time by $t = \Delta t \cdot n$.

$$\begin{aligned} H_z\big|^{n+1}_{i,j+1,k+\frac{1}{2}} &= d_a(m_{i,j+1,k+\frac{1}{2}}) \cdot H_z\big|^{n}_{i,j+1,k+\frac{1}{2}} + d_b(m_{i,j+1,k+\frac{1}{2}}) \\ &\cdot \Big[E_x\big|^{n+\frac{1}{2}}_{i,j+\frac{3}{2},k+\frac{1}{2}} - E_x\big|^{n+\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}} + E_y\big|^{n+\frac{1}{2}}_{i-\frac{1}{2},j+1,k+\frac{1}{2}} - E_y\big|^{n+\frac{1}{2}}_{i+\frac{1}{2},j+1,k+\frac{1}{2}} \Big] \\ &- M_{source,z}\big|^{n+\frac{1}{2}}_{i,j+1,k+\frac{1}{2}} \end{aligned}$$

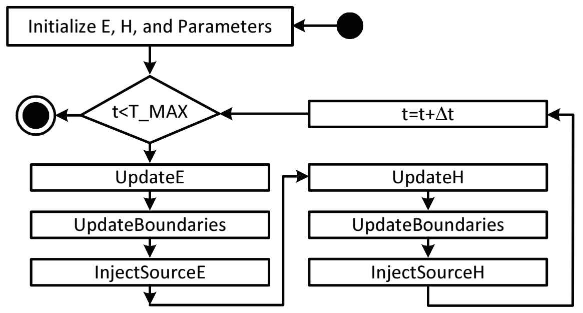


Figure 2: Flow chart for the FDTD algorithm.

For easier indexing of grid points we can align the H and E fields by shifting the non-integer H field indices by $+\frac{1}{2}$ and the non-integer E field indices by $-\frac{1}{2}$. In a similar way, the half step rule in the time dimension can be avoided. At the beginning of the simulation, all fields get initialized with zeros. In every time step, the next field value in time is computed using the current values (stored in memory) and the updated results are stored back to memory. Both, the H and E field values are computed in the same time step. However, as the updated E field values are required for updating the H field we have to compute the E and H field updates in sequence. The complete FDTD algorithm is depicted in Figure 2.

3.2 FDTD Boundary Conditions

As the computational domain must be finite, the algorithm has to apply certain conditions at the boundaries. Depending on the represented physical system, various boundary conditions can be used. The simplest of which is the Dirichlet boundary which assumes a fixed value (generally zero) at the boundary points of the domain. Forcing the fields to zero at the boundaries makes the outside of the grid a perfect electric conductor (PEC) or a perfect magnetic conductor (PMC). PEC and PMC layers will cause reflections into the FDTD domain which are typically unwanted. In order to model an infinite unbounded computational domain absorbing boundary conditions can be applied at the outer lattice boundary. The nowadays most commonly used absorbing boundary condition for the FDTD method is the convolutional perfect match layer (CPML) [9]. The CPML gradually damps the electromagnetic wave at the boundary layers of the domain without causing reflections effectively simulating an open infinite space. CPML boundaries provide very good absorption properties and lead to most accurate simulation results. For structures and excitation conditions with translational symmetry in one or more cartesian dimensions, periodic boundary conditions can be applied.

4. IMPLEMENTATION OF FDTD KERNELS

We study two nanophotonic devices, a microdisk cavity in the plane and a periodic 3-dimensional domain with CPML and periodic boundaries. For the microdisc cavity, we use the general MaxCompiler and follow a heavily pipelined design approach. The accelerator for cubic domains was developed with the help of the MaxGenFD domain-specific compiler. Correctness of the accelerators was proven by comparing the computed results to software generated solutions.

4.1 Microdisk Cavity

The FDTD algorithm has an evident dataflow nature, since the electromagnetic fields have to be iteratively computed in sequence. Our Maxeler implementation exploits dataflow techniques to drastically reduce DRAM memory accesses. Figure 3(a) shows how the calculation of the E and H fields works in sequence. To compute both fields in one sweep as depicted in Figure 3(b), our implementation buffers the required E values on-chip and directly computes the H field update before the results are written back to memory. We could further exploit data locality by computing several simulation steps in one sweep (Figure 3(c)). Instead of sending the results to memory, we connect the outputs of one E/H updater module to the inputs of a second module. The MaxCompiler can be used to replicate blocks in order to implement the module pipeline. Global synchrony is achieved by scheduling appropriate FIFO memories on the connections.

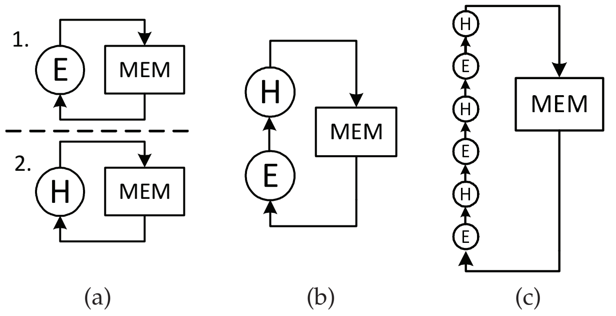


Figure 3: Dataflow exploration of 2D FDTD.

Figure 4 shows a schematic of one 2D E/H updater module. Since we tackle a 2D problem the FDTD equations must be solved for only the three (non-zero) components E_x, E_y, and H_z. The fields are linearized in row-major order and streamed to to the incoming ports *ex*, *ey*, and *hz* of the module. The fourth stream *h2* is used to integrate the squared values of the *hz* field over the time, which represents the intensity (energy density) of the fields and can be used, for example for visualizing the results of the simulation run like depicted in Figure 5(f). The MaxCompiler generates proper memory command streams for the access patterns and instructs the on-chip memory controllers to fetch the data. If all required data is available at the memory controllers for a certain time step t, the 4-tuple $(ex_t, ey_t, hz_t, h2_t)$ enters the kernel. The kernel itself gets configured by setting scalar parameters before execution. *ca*, *cb*, *da*, and *db* are the material coefficients and are globally defined for the whole domain (due to the considered experiment). If the physically represented system requires different material coefficients for the grid points, one could transform the scalar parameters into stream inputs of the kernel. In that case, instead of reading a fixed value, a 4-tuple of parameter values stored in DRAM would enter the kernel in any time step. Other scalar parameters are used to configure the size of the domain (*dimX, dimY*), the source stimuli (*source*), the radius of the microdisc (*rad*), and the actual *phase* of the simulation. The phase is one-hot coded and either 1 (*source injection*), 2 (*hz field integration*), or 0 (*general simulation*).

In the kernel specification, each statement is always related to a current time step t. The MaxCompiler ensures global synchrony of the dataflow implementation by scheduling pipeline registers into lower latency paths. In addition to the delay elements scheduled by the compiler, the programmer can access stream elements which enter the system with

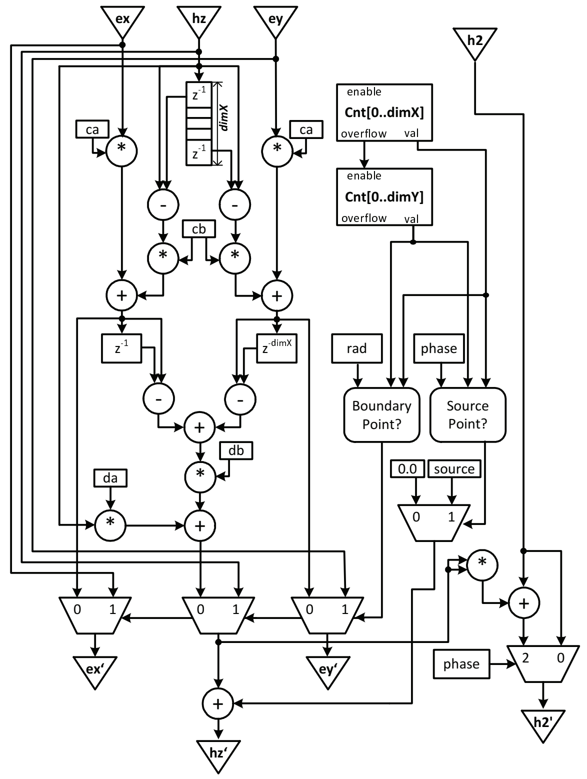


Figure 4: 2D FDTD updater module (without scheduled delay elements of MaxCompiler).

a certain offset to the current element. The Maxeler specification of the 2D FDTD update rule for E_y looks as follows, the resulting circuit is depicted in Figure 4.

```
ey' = (BoundaryPonit) ? ey : ca * ey + cb *
      (stream.offset(hz, -dimX) - hz);
```

To update the E_x and E_y components at position i, the wave equations access the values $ex[i]$, $ey[i]$, $hz[i]$ as well as the left ($hz[i-1]$) and top ($hz[i-dimX]$) neighbors of of $hz[i]$. As an offset of $-dimX$ refers to a value that entered the kernel $dimX$ time steps before, the compiler has to buffer the first $dimX$ values of the stream. The compiler will add additional buffers on the stream channels between computation nodes to to compensate for delays.

The (x, y) position of the current stream values is traced by two counter modules connected in a chained fashion. If the current value is outside of the microdisc cavity, the incoming data is directly forwarded to the output. The programmer can also set the position coordinates (not shown in Figure 4) and the value of the point source for the H field.

4.2 Periodic 3D Domain with CPML

For 3-dimensional domains, the module chaining approach is not feasible because the on-chip memories are far too small to buffer complete planes of realistic 3D domains, as would be needed for chaining. Therefore, we decompose the domain into smaller blocks that can be processed on the FPGA in terms of compute pipelines. We use the *MaxGenFD* programming environment for specifying the 3D FDTD code. The 3D kernel implements all 6 updater equations for the FDTD. As the implementation of the equations is very similar to the 2D microdisc example, we focus on the more advanced aspects of the 3D FDTD accelerator.

Source injection: To allow for injecting source signals into the 3D domain in a most flexible manner, we added a controllable PCI-express source input stream to the kernel. On the host, the programmer can generate a stimuli vector and inject the data at any arbitrary cuboid block inside the domain. This way, the accelerator supports arbitrary point- and plane wave sources without recompilation.

Material parameters: In contrast to the planar microdisk example, it should be possible to define the material parameters for each individual point in the 3D domain. The easiest way would be to add a separate stream from DRAM for each material parameter, however, the additional streams would dramatically consume memory bandwidth. As the domain parameters are typically not highly variant, one could use compression techniques to optimize the memory accesses. Parameter values can be kept in on-chip look-up memories and only the indices into the lookup tables are streamed from DRAM. If the domain consists of, e.g. 8 different materials, the stream of indices needs to be only 3-bit wide and thus would consume very limited bandwidth.

Periodic boundary conditions (PBC): A PBC can be implemented in software easily by swapping the specific boundary planes after each E and H field update. We have implemented PBCs with the help of host streams. Instead of keeping the relevant boundary planes of the domain in the DRAM of the accelerator card, the PBC planes streamed from and to the host memory via PCI-express. After each simulation step, the boundary planes are swapped on the CPU and streamed into the kernel for the next step. Normally, a swapping of boundaries would be needed after each E and H field update. To halve the number of required swapping steps, we extended the domain by one at each PBC plane. This way we can compute the E field results required for a subsequent H field update on the boundaries.

CPML boundary conditions: The CPML is an absorbing boundary condition that uses a layer of artificial absorbing material at the edges of the domain. Waves that enter the absorbing layer decrease to zero without significant reflections back into the domain. We apply CPML boundaries at the top and bottom of the domain. The thickness P of the CPML configurable by the programmer and set to 10 in our simulations. A CPML boundary condition requires to solve four convolution terms $\Psi_{E_{x,y}}$, $\Psi_{E_{x,z}}$, $\Psi_{H_{x,y}}$, and $\Psi_{H_{x,z}}$ for the boundary layer grid points. The results of these terms get added to the E_x, E_z, H_x, and H_z wave fields equations, respectively (see [13])

The Ψ equations also require additional parameters be_j, ce_j, bh_j, and ch_j. As these parameters are only dependent on the level $j \in \{0..P-1\}$ of the CPML layer, we can use a look-up table for keeping them on-chip. For the convolution, we need to stream four additional wave fields through the chip and the equations circuits consume a significant amount of the FPGA logic resources.

5. EXPERIMENTAL RESULTS

This section presents the runtime results for the FDTD simulations as presented in Section 4. The measurements were executed on a Maxeler MPC-C node equipped with two Intel Xeon X5650 2.7GHz 6-core CPUs and four MAX3 DFE cards coupled through a MAXRing interconnect (see Figure 1). All measurements involve the whole application runtime including initialization of accelerator memory and data read-back.

5.1 Microdisk Cavity

The measurements for the microdisk cavity example are compared against an optimized OpenMP-parallelized CPU version, running on a two socket quad-core server [5]. Figure 5(a) provides the runtime for the microdisk FDTD, Figure 5(f) shows the energy density for an exemplary simulation run. Regarding the Maxeler implementation, we could build a pipelined chain of 15 E/H updater modules (as shown in Figure 4). Each stage needs a separate stimulus value as the stages compute different time steps of the simulation. The stimuli are precomputed on the host and are forwarded as one 15-entry vector to the kernel before each iteration. We use a IEEE 754 double precision float type in this example. With four 8-byte streams for input and output, the kernel can generate 64 byte requests per cycle. As the design is running at 100MHz the bandwidth requirement for the kernel is only 6.4 GB/s. The memory controllers of the MAX3 DFE card could achieve much higher bandwidth. However, this design gives enough space for further parallelization (multiple pipelines in parallel) and for more input streams, e.g., separate material parameter streams, as mentioned above. The normal parallelization approach is to decompose the domain into blocks and solve the FDTD on the blocks in parallel. For a 15-fold parallelized version, one would require at least $2 * 15 * 4 * 8Byte * 100MHz = 96GB/s$ of bandwidth.

For larger data sets, our implementation archives a peak performance of $1486 \cdot 10^6$ Mcells/s (grid point updates per second) which corresponds to a speedup of about 2.4x compared to the optimized multi-core version.

Our 2D FDTD updater module with 15 pipelined units causes the following utilization of the Xilinx XC6VSX475T FPGA on the MAX3 card: LUTs 71%, FFs 48%, BRAMs 80%, DSPs 52%. Albeit the resources are not stressed to more than 80% in any category, the design is already packed. In fact, it is difficult to place and route circuits with more than 50% LUT or Flip-Flop utilization for a timing constraints of 100MHz or more without significant optimization effort. The compilation time (MaxCompiler, Xilinx XST, Xilinx PAR) for the presented 2D FDTD accelerator was more than 40h.

5.2 Periodic 3D Domain with CPML

We have conducted several experiments with the 3D FDTD accelerator using varying boundary conditions, the results are presented in Figure 5(b)–(e). The data type for these simulations is IEEE 754 single precision floating point and the experiments were conducted on cubes with 2^{12} to 2^{30} grid points consuming up to 80GB of memory. For domain sizes beyond 2^{27}, the memory requirements exceeded the main memory of the CPU node and of 2 DFEs. Thus, the largest domains are only computed on the 4 DFE Maxeler system. The performance for 3D FDTD simulations in terms of Mcells/s is typically lower than in the 2D case, as the implementation needs the doubled amount of of wave field components and involves more and more complex convolution terms.

We compare the runtime of the accelerated applications to OpenMP-parallelized versions using up to 24 threads on a dual X5650@2.7GHz server node. The parallel CPU solutions were carried out by parallelizing the software versions that are used to verify the accelerator implementations and thus compute exactly the same results.

When using Dirichlet (PEC) boundary conditions (Fig. 5(b)) the Maxeler implementations outperforms the multi-core version by 7.5x for the largest data sets. The accelerators use 6 parallel pipelines per DFE and peak at 1820 Mcells/s. In this case, the memory bandwidth is stressed to 26.8 GB/s. The single core performance for the 3D algorithm is around 20 Mcells/s (irrespective of the applied boundary condition) and up to 91x slower than the hardware accelerated version.

When using a combination of PEC and CPML boundary conditions, we still archive a significant speedup of 5.1x compared to the fastest CPU implementation (Figure 5(c)). The performance drops to 1193 Mcells/s compared to PEC version, mainly because we can only place 4 pipelines onto the FPGA due to the additional logic for carrying out the CPML boundary condition. Albeit we apply less processing pipelines, the exploited memory bandwidth increases to 29.9 GB/s, because of the additional CPML fields that have to be streamed.

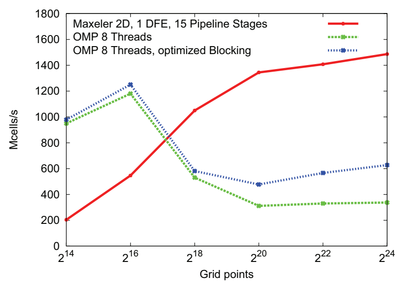
When the experiment comprises periodic boundary conditions, the Maxeler performance dramatically decreases to a slowdown of 0.44x (Figures 5(d) and (e)) compared to the fastest CPU implementation. The problem lies in the streaming of boundary points from and to the host CPU. The reordering of data is currently done in-between the iteration steps of the FDTD algorithm. There is a lot of room for optimization, though, it would be best if the reordering of data could be done on the DFE itself. Such reordering is currently not supported by MaxGenFD and would require massive changes on the memory interface.

6. RELATED WORK

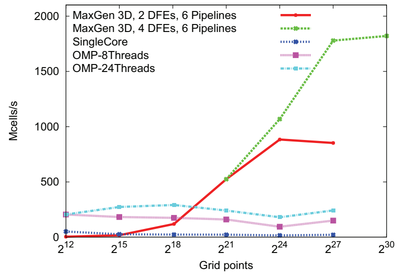
Numerous researchers used GPUs to accelerate the FDTD algorithm. On an Nvidia Quadro FX 5600 Sypek *et al.* achieve a peak performance of 430 Mcells/s [12]. One drawback of GPU-based solvers is the limited accelerator memory. Realistic simulation domains generate memory requirements that exceed the 4–6 GB that a single GPU currently can store locally. For that reason, most FDTD solvers decompose the domain and employ multiple GPUs. Nagaoka and Watanabe propose a multi-GPU cluster in order to simulate larger domains and achieve from around 690 Mcell/s on 2 GPUs up to around 2500 Mcell/s when the system is scaled to 21 GPUs [6]. The MPI-CUDA multi GPU solver presented in [4] peaks at 5000 Mcells/s for a 2-dimensional FDTD simulation with PEC boundaries. A mixed MPI/OpenCL implementation for a hybrid CPU/GPU system incorporates Mur absorbing boundary conditions and reaches nearly 3000 Mcells/s on a 16 GPU system [11]. The FPGA accelerator presented in [1] uses fixed-point data representation and peaks at 13.8 Mcells/s for a 2D domain of size 100x100. Compared to this work, our 2D double precision FDTD solver is 108x faster. In [3] the authors attempt to generate an FPGA-based 3D FDTD solver for the Cray XD1 with the Impulse C framework. After several optimizations, the solver reaches 8 Mcells/s and is still 4-5 times slower than the pure Software reference implementation and 228x slower than our 3D solver.

7. CONCLUSION

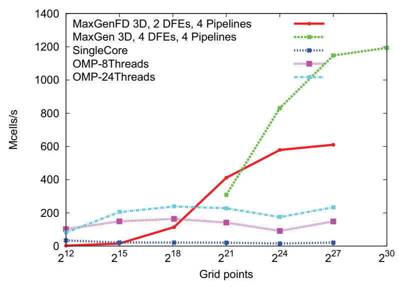
In this paper we present dataflow implementations for 2D and 3D FDTD nanophotonics simulations targeting the Maxeler dataflow computer. Our 2D accelerator follows a deep pipelining approach and shows, how a custom dataflow processor can overcome the memory bandwidth limitations. The 3D accelerator was implemented using the MaxGenFD domain-specific compiler for finite difference applications. Using a high-level specification, the accelerator can be rapidly adapted to the applications requirements. Material parame-



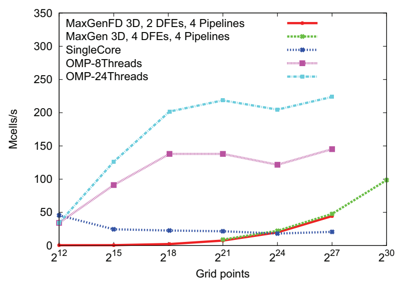
(a) 2D microdisk, 1 DFE



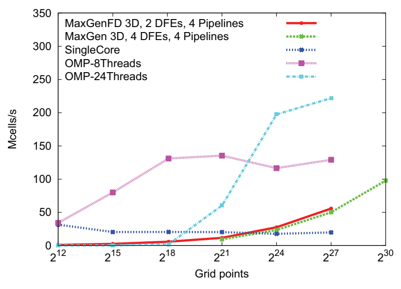
(b) 3D PEC, 2/4 DFEs.



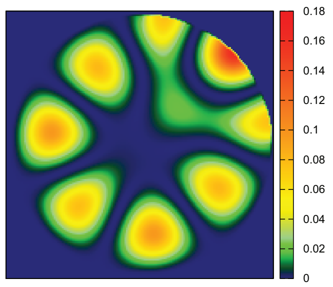
(c) 3D CPML/PEC, 2/4 DFEs.



(d) 3D PBC/PEC, 2/4 DFEs.



(e) 3D PBC/CPML, 2/4 DFEs.



(f) 2D Simulation Result

Figure 5: Runtime for various FDTD experiments: 2D microdisk setup with PEC boundary condition. The accelerator uses pipelined design with 15 chained E/H updater modulesMaxGenFD generated 3D FDTD accelerator running on up to four DFEs with several combinations of boundary conditions: PEC on all 6 boundary planes (b), PEC on xz and yz boundaries and CPML on xy boundaries (c), PBC on xz and yz boundaries and PEC on xy boundaries (d), PBZ on xz and yz boundaries and CPML on xy boundaries (e). Energy density result image for an exemplary 2D simulation run (f).

ters can be set per grid point and are stored in on-chip look-up tables to reduce bandwidth requirements. Our examples comprise Dirichlet, periodic, and absorbing boundary conditions and we present benchmarks to compare the runtime of the experiments to multi-threaded software.

These results from the exploration of FDTD solvers provide major insights on how to efficiently accelerate stencil and convolution operations using dataflow systems, which are transferable to other domains. In ongoing work, we try to formalize this implementation and performance optimization knowledge and to make it available to users from computational sciences without requiring detailed hardware knowledge by further leveraging the abstraction level with domain specific languages and visual programming approaches.

8. REFERENCES

[1] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport. An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm. In *Int. Symp. on Field Programmable Gate Arrays*, 2004.

[2] O. Lindtjorn, R. Clapp, O. Pell, H. Fu, M. Flynn, and O. Mencer. Beyond traditional microprocessors for geoscience high-performance computing applications. *IEEE Micro*, 31:41–49, 2011.

[3] P. Messmer, D. Smithe, P. Schoessow, and R. Bodenner. FPGA-accelerated Finite-Difference Time-Domain Simulation on the Cray XD1 Using Impulse C. In *Cray User Group Meeting*, 2006.

[4] B. Meyer, C. Plessl, and J. Förstner. Transformation of scientific algorithms to parallel computing code: Subdomain support in a MPI-multi-GPU backend. In *Symp. on Application Accelerators in High Performance Computing*, 2011.

[5] B. Meyer, J. Schumacher, C. Plessl, and J. Förstner. Convey Vector Personalities - FPGA acceleration with an OpenMP-like programming effort? In *Int. Conf. on Field Programmable Logic and Applications*, 2012.

[6] T. Nagaoka and S. Watanabe. Accelerating three-dimensional FDTD calculations on GPU clusters for electromagnetic field simulation. In *Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, 2012.

[7] O. Pell and V. Averbukh. Maximum Performance Computing with Dataflow Engines. *Computing in Science & Engineering*, 14:98–103, 2012.

[8] O. Pell, J. Bower, R. Dimond, O. Mencer, and M. J. Flynn. Finite difference wave propagation modeling on special purpose dataflow machines. *IEEE Trans. Parallel Distrib. Syst.*, 24:906–915, 2013.

[9] J. A. Roden and S. D. Gedney. Convolution PML (CPML): An efficient FDTD implementation of the CFS PML for arbitrary media. *Microwave and Optical Technology Letters*, 27:334?–339, 2000.

[10] D. R. Smith, J. B. Pendry, and M. C. K. Wiltshire. Metamaterials and negative refractive index. *Science*, 305:788–792, 2004.

[11] T. Stefanski, N. Chavannes, and N. Kuster. Hybrid OpenCL-MPI parallelization of the FDTD method. In *Int. Conf. on Electromagn. in Advanced Applications*, 2011.

[12] P. Sypek, A. Dziekonski, and M. Mrozowski. How to render FDTD computations more effective using a graphics accelerator. *IEEE Trans. Magn.*, 45(3), 2009.

[13] A. Taflove and S. C. Hagness. *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech House, 2005.

[14] K. S. Yee. Numerical solution of inital boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas Propagat.*, 14:302–307, 1966.