FPGA-accelerated Finite-Difference Time-Domain Simulation on the Cray XD1 using Impulse C

Peter Messmer, David Smithe, Paul Schoessow, *Tech-X Corporation***,** *and* **Ralph Bodenner**, *Impulse Accelerated Technologies*

ABSTRACT: *The Finite-Difference Time-Domain (FDTD) algorithm is a well established tool for modelling transient electromagnetic phenomena. Using a spatially staggered grid, the algorithm alternately advances the electric and magnetic fields, according to Faraday's and Ampere's laws. These simulations often contain millions of grid cells and run for thousands of time-steps, requiring highly efficient grid update algorithms and high-performance hardware. Here we report on the implementation of the FDTD algorithm on the application accelerator of a Cray XD1. The FPGA is programmed using the Impulse C tool suite. These tools translate C code into VHDL and therefore enable the domain scientist to develop FPGA-enhanced applications. Different optimization strategies, ranging from algorithmic changes to exploitation of the high degree of parallelism on the FPGA, will be presented.*

KEYWORDS: XD1, FDTD, FPGA, ImpulseC

1. Introduction

Transient electromagnetic problems are often too complicated to be treated analytically and numerical simulations are required. Examples of such problems are electromagnetic wave propagation in complex geometries or scattering of waves off dielectric objects. The finite-difference time-domain (FDTD) algorithm [1] is a well established technique for treating these problems efficiently. In addition to pure electromagnetic propagation problems, the FDTD algorithm is at the heart of more complex algorithms, including e.g. the particle-in-cell (PIC) algorithm for the kinetic simulation of plasmas. A result of such a model, computed with the electromagnetic particle-in-cell code VORPAL [2] is shown in Figure 1: A particle bunch propagating in a cavity creates a wake-field in the cavity. This code uses the FDTD algorithm to compute the evolution of the electromagnetic field.

Finite-difference time-domain discretizes the electric and magnetic field in the computational domain on a rectangular mesh and updates each cell via finite differences. Each cell of the computational grid can be updated independently of each other and this large amount of parallelism makes it an ideal candidate to be computed on distributed memory systems. However, the update operation per grid cell is complex enough that they may benefit from additional acceleration, e.g. via the application accelerator FPGAs on the Cray XD1.

Modelling FDTD using FPGAs has been investigated for several years. However, these systems are usually restricted to custom built hardware or require hardware engineers to program them. The advent of ready-made clusters with hardware acceleration put FPGA computing within reach of computational physicists without access to custom built hardware.

A second major recent development is the availability of C-to-gates translators which enable non-experts in hardware engineering to take advantage, or at least explore, FPGA computing.

In this paper, we describe the different optimizations of an FDTD implementation on the application accelerator of the Cray XD1 using ImpulseC [3]. In the next paragraph,

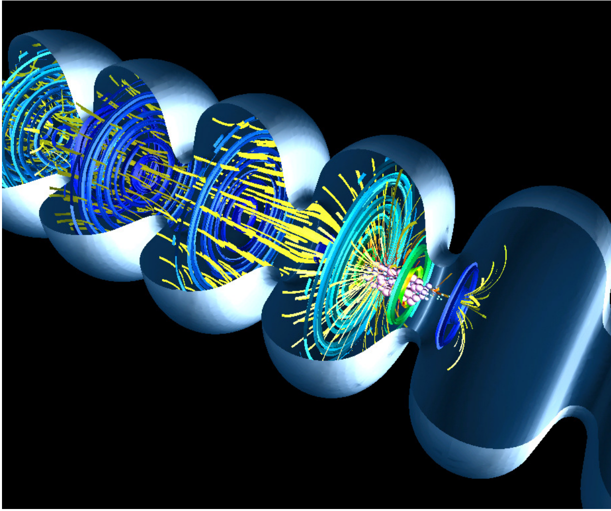

Figure 1: Electromagnetic field in a superconducting cavity, computed with the FDTD code VORPAL.

we will introduce the FDTD algorithm, and the ImpulseC environment. Then we will describe the initial algorithmic optimizations. We will then present different optimization strategies of the FDTD algorithm on the FPGA. We will summarize the work and draw conclusions.

2. The FDTD algorithm

The FDTD algorithm computes the update of the electromagnetic field based on Ampere's and Faraday's law,

$$\frac{dE}{dt} = c^2 \nabla \times B \text{ and } \frac{dB}{dt} = -\nabla \times E$$

using finite differences. For the single component of the E field, this reduces to

$$E_z^{t+1} = E_z^t + ((B_x^{y+1} - B_x^y) / dy - (B_y^{x+1} - B_y^x)/dx) dt/c^2$$

This operation has to be performed for the entire computational domain. In order to obtain second order accuracy, the E and B fields are staggered in space and time on a so-called Yee Grid. The algorithm is well understood and an extensive literature on it has been published. For an introduction, see [1].

3. FPGA programming

Programming of FPGAs used to require knowledge of hardware definition languages and was therefore limited to hardware engineers or scientists with hardware design knowledge. While this is still the case nowadays if maximum performance should be obtained from FPGAs, the growing number of tools for C-to-gates translation makes the power of these chips accessible to more software oriented scientists and engineers.

For this project, we used the ImpulseC systems, developed by Impulse Accelerated Technologies. In ImpulseC, an application is composed of multiple processes which communicate among each other via streams, shared memories or signals. ImpulseC is based on standard C language, extended by an application programming interface (API) for the communication primitives and directives (pragmas) for fine-tuning the translation. Each ImpulseC process is essentially a C function. Special functions then allow the programmer to place these functions either on the host CPU or on the FPGA. The ImpulseC compiler then generates the appropriate code and interfaces expressed in hardware definition language.

While the ImpulseC code is portable between different systems, the ImpulseC compiler needs to generate system specific code via so called *platform support packages*. One of these support packages targets the Cray XD1.

4. Hardware

For this project we were using Cray Inc.'s XD1 system *pacific*. The system consists of 2 XD1 chassis with a total of 12 nodes. Each node consists of 2 dual core AMD Opteron 275 processors, acting as four individual CPUs. The clock speed per CPU is 2.2 GHz.

One chassis of this system is equipped with Application Accelerators, fully user programmable Field-Programmable Gate Arrays (FPGA). On the particular system used for this project, the application accelerator is a Xilinx Vitrex-II Pro FPGA, XC2VP50-7. The clock speed of each FPGA is user selectable, with a maximum speed of 200 MHz. This relatively low clock rate of the FPGA requires that the main advantage of the FPGA is exploited in order to beat the Opteron processor, running at a 10 times higher clock rate.

5. Algorithm optimizations

Before trying to accelerate the application with sophisticated hardware, we were looking into accelerating the algorithm itself. By choosing the right units of B and E and assuming uniform grid spacing, one can eliminate the need for divisions and multiplications in the update altogether. The update described in the previous section then becomes

$$E_z^{t+1} = E_z^t - (B_x^{y+1} - B_x^y - B_y^{x+1} + B_y^x)$$

The advantage of these units is that the field values can be expressed simply as integer numbers rather than floating-point numbers.

The overall algorithm can be expressed in pseudo code as follows:

```
for t = 0, nsteps {        // time stepping loop
   for x, y, z in domain { // E field update
      compute index of all contributing
      elements, e.g.
        Ez^t.  Bx^y+1, Bx^y , By^x+1, By^x
      compute curl, e.g.
        Ez^t+1 = Ez^t - ( Bx^y+1 - Bx^y - By^x+1 + By^x)
   }

   for x, y, z in domain { // B field update
      compute index of all contributing elements
      compute curl, e.g.
        Bz^t+1 = Bz^t - ( Ex^y - Ex^y-1 - Ey^x + Ey^x-1)
   }
}
```

For all the subsequent experiments, we used a benchmark problem of 400 x 200 x 1 cells, running for 100 time-steps. The pure software implementation ran on this system in about 0.2s.

6. Tool chain

In order to utilize the application accelerator, the FPGA has to be configured with the appropriate logic. This configuration is programmed via a configuration file (bit-stream). In order to generate the bit-stream from an ImpulseC source file, a variety of tools have to be run. In this section, we briefly describe this process:

Once the algorithm has been expressed in ImpulseC, it can be translated into a hardware definition language, in this case VHDL, using the ImpulseC compiler CoDeveloper. This compiler generates all the interfaces necessary to connect the user generated logic to the logic provided by Cray. E.g. the Rapid Array Transport is used to transmit data between the host CPU and the FPGA. In addition, the ImpulseC compiler generates project files for the tools subsequently used.

This hardware-definition code is then synthesized, placed and routed and finally converted into a bit-stream using the ISE/XST toolchain from Xilinx [4]. The project files generated by CoDeveloper make this step almost a black box. However, this process is very time consuming and, depending on the complexity of the logic, can take hours to complete.

Once the bit-stream and the software is built, it can be ported to the XD1 system. ImpulseC generates a Makefile, making the build process for the software processes also a black box.

Before the FPGA accelerated application can be executed, the bit-stream has to be prepared by pre-pending Cray proprietary header information. This is accomplished with a utility program *fcu*, which takes the FPGA device number and the clock speed and creates this header.

The *fcu* utility also offers additional functionality, like resetting the application accelerator or loading a bit-stream. However, the ImpulseC program generated host application takes care of all these steps.

7. Porting to FPGA

The first step of porting the FDTD algorithm to an FPGA was mainly to get acquainted with the overall system and not really designed for performance. We chose to move the computation of the curl operator from a pure software implementation to the FPGA. We therefore created an FPGA process which reads 9 values from an input stream (Ex, Ey, Ez, Bx, Bx,y+1, By, By,x+1, Bz, Bz,x+1) computes the curl and returns the updated three components (Ex, Ey, Ez) to the host CPU. Both the time-stepping loop as well as the loop over the entire computational domain are still executed on the host CPU.

While the implementation was working as expected, it turned out that it took about 250 s to execute. Putting both the curl B and curl E computations onto the FPGA resulted even in 450s overall execution time. This is more than 3 orders of magnitude slower than the pure software implementation.

8. Optimizations

One of the main advantages of high-level development of FPGA codes is that it enables simple experiments with different splitting between hardware and software processes. In the following, sections we describe some experiments we performed in order to reduce the time on the FPGA.

8.1 Concurrent processes for curl components

The main bottle neck of the implementation presented in the previous section is the large amount of data that has to be transferred between the host process and the FPGA. Instead of sending individual numbers from the processor to the FPAG, we now investigated the possibility to compress two 32 bit integers into a 64 bit integer. In addition, we split the computation of the curl operator into three individual processes, each computing only one component of the resulting vector

The overall time for the benchmark problem was reduced from 450 s to 230 s. While it shows that larger data transfers are important, it still is very far from the CPU based implementation.

8.2 Avoid CPU-FPGA data transfer

In order to avoid the problem of the data transfer between the host CPU and the FPGA, ImpulseC supports shared memories. However, this feature is currently under development and was not supported at the beginning of this project. We therefore had to find solutions to avoid massive transfers of data between CPU and FPGA.

One obvious choice is to put the entire FDTD algorithm onto the FPGA. While the limited size of the available memory reduced the feasible problem sizes, these experiments enabled to estimate the possible performance.

Running the entire FDTD algorithm on the FPGA resulted for the benchmark problem in an overall execution time of 5s. Experiments with pipelining the inner most loop reduced this time down to 4.7s.

8.3 *Future optimizations*

An optimization that was not completely finished at the time of this writing is to exploit system level parallelism: On distributed memory machines, FDTD is usually parallelized via domain decomposition. The update of a cell only depends on nearest neighbours, making the communication relatively simple.

The test-problem for the FDTD algorithm on FPGA required only about 8% of the available logic. We therefore ran multiple FDTD processes concurrently, each working on a separate sub-domain. While we did not yet implement the communication between the domain boundaries, the resulting speedup is encouraging.

9. Conclusion

Many problems in computational electrodynamics can benefit from accelerated FDTD computation. One possible approach to accelerate these computations is to use reconfigurable coprocessors based on Field Programmable Gate Arrays to accelerate these computations. The Cray XD1 system is an ideal platform to experiment with FPGA acceleration of algorithms. High-level development tools, like ImpulseC, enables domain scientists to experiment with different optimization strategies and helps to get a feeling for the power of the FPGAs.

We were investigating different optimizations of the FDTD algorithm on the Cray XD1. In a first step, we rescaled the units to avoid time-consuming floating-point operations. In a next step, we split the computation between the host CPU and the FPGA. However, the large amount of data transferred between CPU and FPGA resulted in poor performance. Next we put the entire FDTD algorithm onto the FPGA. The performance was about a factor of 20 slower than the pure software implementation, corresponding roughly to the ratio in clock speeds between the FPGA and the CPU. Using multiple processing pipelines, each working on a part of the computational domain, can help to overcome this discrepancy.

Acknowledgments

The authors would like to thank David Strenski (Cray) and Roy White (Xilinx) to provide access to the XD1 and the XST tool-suite.

References

[1] A. Taflove, S. Hageness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed., Artech House, 2005.
[2] J. R. Cary, C. Nieter, VORPAL: A versatile plasma simulation code, J. Comp. Phys., 196, 448, 2004.
[3] www.impulsec.com
[4] www.xilinx.com

About the Authors

Peter Messmer is research scientist at Tech-X Corp., Boulder, CO. His is working on computational plasma physics and parallel computing support tools. He can be reached at messmer@txcorp.com. David Smithe is Senior Scientist at Tech-X Corp. and working in the field of computational electrodynamics and plasma physics. He can be reached at smithe@txcorp.com. Paul Schoessow was Senior Scientist at Tech-X Corp. where he was working on computational electrodynamics. He recently has joined Euclid Tech Labs, where he is performing similar research. He can be reached at pvs@ieee.org. Tech-X' mailing address is Tech-X Corp. 5621 Arapahoe Ave., Suite A, Boulder, CO 80303, USA. Ralph Bodenner is Senior Engineer at Impulse Accelerated technologies, where he is working on the XD1 support package. He can be reached at Ralph.bodenner@impulsec.com.