

A Prototype FPGA Finite-Difference Time-Domain Engine for Electromagnetics Simulation

Robert Culley, Ashish Desai, Sachin Gandhi, Shugaung Wu, and Karen Tomko
Electrical and Computer Engineering and Computer Science
University of Cincinnati
Cincinnati, OH USA
ktomko@ececs.uc.edu

Paul Sotirelis and David Richie
High Performance Technologies, Inc
Reston, VA USA

Abstract—Several Field-Programmable Gate Array (FPGA) based Reconfigurable High-Performance computing systems have recently become available. In this paper we describe our experience developing an application accelerator for a traditional HPC application, a Finite-Difference Time-Domain (FDTD) electromagnetics simulation, for two such systems. We describe the design of our FDTD calculation engine and demonstrate that FPGA can accelerate a single precision floating point implementation. We discuss the current limitations of our design and describe design improvements to overcome them.

I. INTRODUCTION

Creators of High Performance Computing (HPC) systems are always searching for new ways to improve computational power, so it is not surprising to see companies adding Reconfigurable Computing (RC) resources to their systems. Field-Programmable Gate Array (FPGA) chips have been growing in speed, size, and computational features. Resource savers such as multiply units and Block RAM (BRAM) for buffers are examples of computational features added to FPGAs [1]. Although FPGAs do not run at the same speed as the microprocessors, an efficient pipeline at a lower speed can be just as effective for certain repetitive tasks. In addition, any desired computation engine can be loaded into the FPGA device in milliseconds.

Other researchers have done work with RC to improve Finite-Difference Time-Domain (FDTD) calculations. FDTD was first used in an FPGA by researchers using bit-serial arithmetic to improve computation speeds in a one-dimensional calculation [2]. However, they did not have enough space to run the full design on one chip. A fixed-point method was used to implement a 24 times speed-up of a two-dimensional FDTD [3]. This method was not as accurate as floating point but showed tremendous speed-up over the software calculations. Another researcher demonstrated how the design decisions regarding core size and pipelining depth affect the speed of the design [4]. They also looked at the effect of design decisions on the energy

efficiency and compared their core to some commercial cores.

Over the past 18 months we have explored the feasibility of accelerating a floating-point (FDTD) solver used in electromagnetic modeling and simulation. Our target environment is a Linux Cluster enhanced with FPGA-based co-processors. FDTD is a convenient way to do simulation of electrical and magnetic fields using Maxwell's Equations in differential form and can be easily pipelined and parallelized amongst processors. We will show the speedup in the FDTD calculation that Reconfigurable HPC can supply.

This paper is organized as follows. The platform section of this paper outlines the equipment and the software used in the development of the prototype. The next section addresses the Computational Engine used for the FDTD calculation and the specific circuit used for the prototype. In the results section, we show the performance differences between the software and hardware implementation and introduce extensions to our prototype model aimed at overcoming some current limitations.

II. PROTOTYPE PLATFORMS

We have targeted two reconfigurable HPC systems for our prototype implementation. Both are Linux clusters augmented with Xilinx FPGA co-processing subsystems. Details of each are given below.

A. Heterogeneous High Performance Computer

We developed software to run on two different hardware platforms for the prototype systems. The first system is the Heterogeneous High Performance Computer (HHPC) located at the Air Force Research Laboratory/Information Directorate (AFRL/ID) Distribution Center in Rome, New York. This system has 48 nodes, each of which has two Intel Pentium4 Xeon processors, 4 GB of RAM, and an Annapolis Micro Systems WildstarII for PCI board. The Annapolis board has two Xilinx Virtex-II FPGA (XC2V6000) and

This work was funded in part by the CEN-004-006 project as part of the Programming Environment and Training (PET) Program of the High Performance Computing Modernization Program Office (HPCMPO) under GSA Contract No. DAAD05-01-C-0033.

12MB of DDR II SRAM. The data are communicated between the processors and the FPGA using a 66MHz PCI interface. The node to node communication is handled by Myrinet -2000 and Gigabit Ethernet [5]. The software uses Message Passing Interface (MPI) to communicate between the nodes and the Wildstar II's application program interface (API) for controlling to the FPGA board.

B. Cray XDI

The second system is a Cray XD1 located at the Ohio Supercomputer Center (OSC) in Springfield, Ohio. This system has 18 nodes (blades). Each blade has two 64-bit AMD Opteron (200 series) processors and 4GB of main memory [6]. The OSC system has six blades populated with the optional FPGA application acceleration processor. These are Virtex-II Pro FPGA (XC2VP50-7) and four 4MB QDR II SRAM. The software utilizes Cray enhanced MPI, and a proprietary API for the FPGA [7]. The Opterons communicate with the FPGA using API calls across a proprietary bus called the RapidArray Transport Core. It should be noted that any access to the memory on either of the FPGA boards must be done through the FPGA.

III. FDTD CALCULATION

The FDTD method provides a direct time-domain solution of Maxwell's Equations in differential form [8, 9]. It is well suited to implementation on parallel computers because only nearest-neighbor interactions are involved. It also provides modeling flexibility since the method can support a wide range of model parameters and environments. Consequently, the method has been applied to a wide range of electromagnetic computational problem such as transmissions of wave guides; reception, detection and radiation from antennas; scattering of waves, etc.

The software has been designed to simulate the transient response to a delta pulse in the electric field in the center of the considered region. The medium the fields are propagating in is assumed to be an uniform free space. When the software is used with multiple processors, the MPI API is used to handle the transfer of data. The Electric field (Ez) values on the upper and left edge of any process boundary are sent to the process in the same direction. The lower and right-hand side values of the x and y components of the magnetic field (Hx and Hy) are transferred in the same manner. Depending on the platform, the host software is also designed to either fill the buffers or write to the memory across the FPGA and transfer the boundary data to the adjacent processes. The equations that are used to update the electric and magnetic field values during each iterated time-step are:

$$Hy_t[i][j] = Hy_{t-1}[i][j] + dtmudx*(Ez_{t-0.5}[i+1][j+1] - Ez_{t-0.5}[i][j+1]) \quad (1)$$

$$Hx_t[i][j] = Hx_{t-1}[i][j] - dtmudy*(Ez_{t-0.5}[i+1][j+1] - Ez_{t-0.5}[i+1][j]) \quad (2)$$

$$Ez_t[i][j] = Ez_{t-1}[i][j] + dtstepsdx*(Hy_{t-0.5}[i][j-1] - Hy_{t-0.5}[i-1][j-1]) - dtstepsdy*(Hx_{t-0.5}[i-1][j] - Hx_{t-0.5}[i-1][j-1]) \quad (3)$$

where dtmudx, dtmudy, dtstepsdx and dtstepsdy are constant coefficients which are functions of the permittivity and permeability of free space, the time step (t) and grid cell size in the discretization of the free space. The latter refers to the i and j in the equation, the cell position values in the x and y direction.

A. General Computation Engine

The FDTD Computation Engine uses parameterized VHDL floating point modules obtained for free from Dr. Miriam Leaser at Northeastern University [10]. The library is compliant with IEEE single-precision format. The basic FDTD core is laid out to implement the magnetic (Hx and Hy) updates, followed by the Electric (Ez) update. This core is shown in Fig. 1. The upper flow is repeated twice in the circuit for the Hx and Hy flows. These computations involve at least one multiply and two additions, all in floating-point format. The result of each computation is fed into the next calculation along with another value. A circuit can be designed to implement an efficient pipeline in an FPGA for this computation. The values needed for the Hx calculation are accessed in a row major format, while the Hy data is accessed in column major fashion. The ordering is crucial to take full advantage of spatial locality. The data can be stored in a FIFO and then sent to a latch for the next calculation. The Ez calculation is done completely in row major manner, which means that Hy values need to be fetched for two adjacent rows at the same time. The Hx and Hy update pipelines produce an output for every clock cycle and have a 22 cycle latency [11]. The Ez update pipeline also produces one output per clock cycle, and has a 30 cycle latency. A step not shown in the circuit is that the data is denormalized and normalized at the entry and exit of each floating point block.

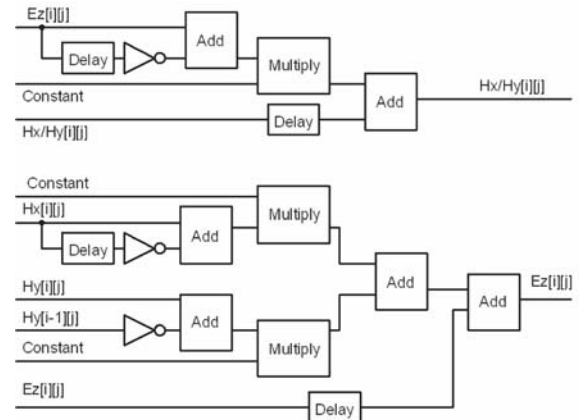


Figure 1. Block diagram of the Hx, Hy and Ez update flows.

B. HHPC Specific Details

The HHPC system feeds and retrieves the data with the FDTD update Engine through 2K FIFO. The FIFO is implemented using the BRAM on the Virtex-II device on the Wild star II board which allows 511 values to be computed in one pass. Since the data is not a continuous stream, dummy values are inserted at the start of the buffer and any start of a row or column. This is done to prime the delays with the correct value corresponding to the calculation that is to take place. The host application must pack and unpack the data sent to the FPGA board, and also pass any required boundary data to adjacent processors. The FDTD core utilizes only 25% of the Virtex-II part, so this circuit can be repeated four times within the FPGA. The FDTD core is clocked at just over 50MHz. The communication from the host software to the FPGA and the links between processors are the chief bottleneck in this prototype.

IV. RESULTS AND CRAY PROTOTYPE IMPLEMENTATION

A. Results

Simulation runs on the HHPC yielded the results shown in Table 1. The two dimensional grids of the following sizes were used in our experiment: 16x26, 46x51, 105x117, and 1003x1012. The resulting number of 511 word packets communicated to the FPGA for each calculation was 1, 5, 25, and 1999. The time taken to get FPGA results of the computations grows with the increase in the number of buffers used to transfer the data. A partial data packet requires the same amount of time as a full packet. The speed-up was not as great on runs in which data did not fill the full packet and when the number of packets is small. Some data values are transferred to the FPGA multiple times, in particular at row and column boundaries. For the Hx and Hy calculations, if the data set was not square, the smallest dimension has the most insertions of repeat values to restart on the next column or row. This would determine the number of packets that had to be sent. For the single node computation, the FPGA execution time for Ez is twice as fast as the software code. The Hx/Hy update is four times faster on the FPGA. This shows that FPGA can implement the computation two to four times faster if the host did not have

to transfer the information back and forth with the FPGA. The overhead of packaging and unpacking the data for the magnetic and electric calculations was 40 and 28 times greater than the time to calculate the data inside the FPGA. The values in the table for the host software have the 1.02 μ s cost of the call and the calculation of the elapsed time.

Running the FDTD calculation with multiple nodes yielded much the same results as the single node. The performance speed-up was worse in the situations where the buffers are not completely filled. Here the FIFO must be completely clocked through the system where the software can complete without having the extra work. The delay due to passing data to adjacent processors was approximately the same, and the host software incurs similar latencies.

The largest bottleneck by far was to pack and unpack the data for repeated transfer to and from the FPGA co-processor via the PCI bus. The design would show an overall improvement if the communication was quicker and/or the data was stored on the board.

B. Cray XDI Prototype Implementation

In order to reduce the communication bottleneck between the processor and the board, the pipeline design has been modified from using FIFO to using the SRAM connected to the FPGA. This reduces the communication of the data with the CPU but does not completely eliminate it. The interface to the RAM is shown in Fig. 2. The solid lines show the flow for the Hx/Hy update, and the dashed lines show the data path for the Ez update. This circuit is designed to run at 100MHz and has the same latencies as mentioned above.

The boundary values still must be passed between processors for multiprocessor computation. Therefore the data must pass through the FPGA to the host application, then through an MPI function, and then finally back down to the adjacent processor's FPGA. The data communication between the FPGA and the host code is formed into nine 64bit packets, for write operations [7]. The first quad word is the address followed by 8 data quad words. This is very efficient for continuous data transfer, such as along rows, but the column transfers require more overhead.

TABLE I. EXECUTION TIMES FOR THE FDTD ENGINE AND THE SOFTWARE IMPLEMENTATION OF THE CALCULATION.

	Number of Processors	Ez Calculation in micro seconds				Hx/Hy Calculation in micro seconds			
		Number of Buffers							
		1	5	25	1999	1	5	25	1999
FDTD Engine (FPGA)	1	10.9	54.5	272.5	21790	10.7	53.4	267.0	21349
	4	10.9	21.8	76.3	5494	10.7	21.36	74.76	5340
	8	10.9	10.9	43.6	2758	10.7	10.7	42.72	2670
	9	10.9	10.9	32.7	2453	10.7	10.7	32.04	2382
FDTD Software	1	18.0	91.2	462	39000	41.3	209	1055	87600
	4	6.14	23.8	117	9870	12.5	52.3	265	22100
	8	3.52	12.2	59.8	4970	6.80	27.4	134	11400
	9	3.18	11.2	52.2	4450	5.62	24.6	121	10010

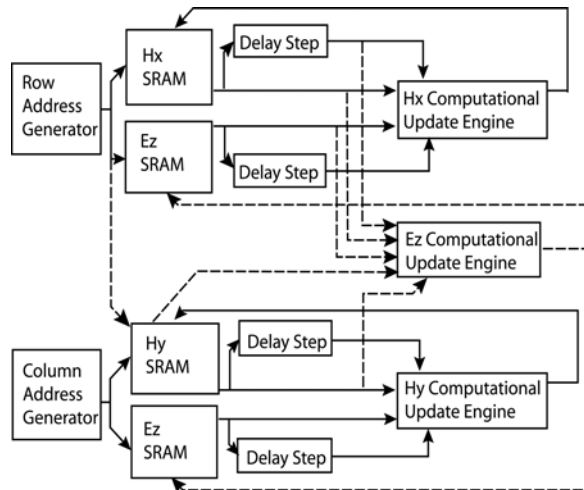


Figure 2. Block diagram of SRAM interface to the Computational Update Engine.

V. CONCLUSION

In conclusion, our prototype implementation demonstrates that FPGAs can accelerate FDTD computation on HPC platforms. We see that the calculation throughput can be doubled even with a modest clock rate of 50 MHz. There are several things that can be done to improve this result further. If the circuit is replicated on the FPGA, there is a possibility of speeding up the calculation by 8 times. Additionally, a higher performing floating-point library would allow a faster clock [4]. Of course, the most necessary improvement is to reduce the amount of communication required between the host application and the FPGA. With the cost of 28 to 40 times the computation to form and unpack the data sent to the FPGA, currently prohibits practical use. Our Cray XD1 implementation is addressing this issue. Firstly, the RapidArray transport between host and FPGA allows two 32-bit floating point values to be transferred to achieve full bandwidth. In addition, the FDTD is kept in the SRAM adjacent to the FPGA and only boundary data and final results need be transferred to the host.

A. Future Work

The Cray System interface is currently being developed and will be tested shortly. The Cray system has the feature of FPGA to FPGA communication. Once this feature is enabled the FPGAs will be programmed to pass boundary

data directly to each other, further reducing the communication overhead. The host software will supply the direction the data needs to travel.

ACKNOWLEDGMENT

This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at ARL, MSRC-ASC, and AFRL/ID. This work was also supported in part by computer time from the Ohio Super Computer Center, and also required the support of the EDA community. The Xilinx University Program supplied Xilinx ISE. Synplicity's University Program supplied SynplifyPro, and Mentor Graphics' Higher Education Program (HEP) supplied Modelsim.

REFERENCES

- [1] Xilinx, Inc., "Virtex-II platform FPGAs: complete data sheet", DS031, v3.4, March 1, 2005.
- [2] R. Schneider, L. Turner, and M. Okoniewski, "Application of FPGA technology to accelerate the Finite-Difference Time-Domain (FDTD) Method", Proceedings of the 2002 ACM/SIGDA tenth international symposium on field-programmable gate arrays. pp. 97-105.
- [3] W. Chen, P. Kosmas, M. Leeser, C. Rappaport, "An FPGA implementation of the Two-Dimensional Finite-Difference Time-Domain (FDTD) Algorithm", International Symposium on Field Programmable Gate Arrays, Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004, pp. 213-222
- [4] G. Govindu, L. Zhuo, S. Choi, and V. K. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs". In Proceedings of the 11th Reconfigurable Architecture Workshop, Santa Fe, NM, April 2004.
- [5] "Heterogeneous FPGA cluster data sheet", High Performance Technologies, Inc.
- [6] Cray XD1 Datasheet, <http://www.cray.com/products/xd1>
- [7] Cray Inc., "Cray XD1™ FPGA development", Release 1.2, S-6400-12, 2005
- [8] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media", IEEE Trans. Antennas and Propagation, 16 (1966), pp. 302-307.
- [9] K. S. Kunz, R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*, CRC Press, 1993.
- [10] P. Belanović and M. Leeser. "A library of parameterized floating point modules and their use". In Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL), Montpellier, France, September 2002, pages 657-666.
- [11] S. Gandhi, "An FPGA implementation of FDTD codes for reconfigurable high performance computing", M.S. Thesis, Dept. of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Nov. 2004.