Hardware Implementation of a Three-Dimensional Finite-Difference Time-Domain Algorithm

James P. Durbano, *Member, IEEE*, Fernando E. Ortiz, *Member, IEEE*, John R. Humphrey, *Member, IEEE*, Mark S. Mirotznik, *Member, IEEE*, and Dennis W. Prather, *Member, IEEE*

***Abstract*—In order to take advantage of the significant benefits afforded by computational electromagnetic techniques, such as the finite-difference time-domain (FDTD) method, solvers capable of analyzing realistic problems in a reasonable time frame are required. Although software-based solvers are frequently used, they are often too slow to be of practical use. To speed up computations, hardware-based implementations of the FDTD method have been recently proposed. Although these designs are functionally correct, to date, they have not provided a practical and scalable solution. To this end, we have developed an architecture that not only overcomes the limitations of previous accelerators, but also represents the first three-dimensional FDTD accelerator implemented in physical hardware. In this paper, we present a high-level view of the system architecture and describe the basic functionality of each module involved in the computational flow. We then present our implementation results and compare them with current PC-based FDTD solutions. These results indicate that hardware solutions will, in the near future, surpass existing PC throughputs, and will ultimately rival the performance of PC clusters.**

***Index Terms*—Electromagnetic analysis, finite-difference methods, field programmable gate array (FPGA), hardware acceleration.**

I. Introduction

NO LONGER relegated to RF engineers, antenna designers, and military applications, electromagnetic analysis has become a key factor in many areas of advanced technology. From 3-GHz PCs and wireless computer networks, to personal digital assistants (PDAs) with Internet capabilities and the seemingly ubiquitous cell phone, it seems that electronic designs increasingly require electromagnetic characterization. To facilitate such analysis, numerical techniques have been developed that allow computers to easily solve Maxwell's equations. Among the most common computational techniques is the finite-difference time-domain (FDTD) method [1].

Although FDTD methods are accurate and well defined, current computer-system technology limits the speed at which these operations can be performed. Run times on the order of hours, weeks, months, or longer are common when solving problems of realistic size. Some problems are even too large to be effectively solved due to practical time and memory constraints. The slow nature of the algorithm primarily results from the nested for-loops that are required to iterate over the three spatial dimensions and time.

To shorten the computational time, people acquire faster computers, lease time on supercomputers, or build clusters of computers to gain a parallel processing speedup [2], [3]. These solutions can be prohibitively expensive and frequently impractical. As a result, an approach that increases the speed of the FDTD method in a relatively inexpensive and practical way is required. To this end, people have suggested that an FDTD accelerator, i.e., special-purpose hardware that implements the FDTD method, be used to speed up the computations [4]–[9]. However, none have succeeded in developing a practical implementation, nor a full three-dimensional (3-D) solver.

The purpose of this paper is to describe our acceleration architecture, the first successful 3-D FDTD accelerator to be implemented in physical hardware. We begin with a brief discussion of three of the most important areas associated with a hardware-based implementation. Next, we provide a high-level view of the computational flow of our architecture. We conclude by presenting our implementation results and comparing them with current PC-based FDTD solutions.

II. Possible Approaches

Hardware accelerators are usually part of an overall system, consisting of a host PC and the accelerator. In an FDTD acceleration system, the host computer initially describes the problem to be solved and sends all relevant information to the hardware accelerator for computation. The accelerator then updates the fields at every timestep, periodically sending the results back to the host computer for postprocessing and visualization.

Outside of this general form, there are many tradeoffs to consider when implementing an FDTD accelerator. In this section, we look at three of the most important design decisions that must be made and present the two extremes within each possible approach. The three major areas of concern are the computation engine, data storage, and the handling of special nodes. We begin by discussing the computation engine.

A. Computation Engine

The computation engine implements the field-update equations, which represent the discretized form of Maxwell's equations. These equations are responsible for propagating the electromagnetic fields and are at the heart of any hardware-based

Manuscript received February 13, 2003. This work was supported in part by the U.S. Department of Defense Small Business Innovative Research Program.

J. P. Durbano is with EM Photonics, Inc., Newark, DE 19711 USA (e-mail:durbano@emphotonics.com).

F. E. Ortiz, J.R. Humphrey, and D. W. Prather are with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716 USA (e-mail:ortiz@ee.udel.edu; humphrey@ee.udel.edu; dprather@ee.udel.edu).

M. S. Mirotznik is with the Electrical Engineering and Computer Science Department, The Catholic University of America, Washington, DC 20064 USA (e-mail:mirotznik@cua.edu).

Digital Object Identifier 10.1109/LAWP.2003.812245

1536-1225/03$17.00 © 2003 IEEE

implementation of the FDTD method. There are two extremes to consider when implementing the computation engine: a massively parallel approach and a serial approach.

A massively parallel approach to the computation engine implements the field-update equation in hardware N times, where N is the number of nodes in the computational mesh. For example, in a $100 \times 100 \times 100$ node mesh, one million field-update equations would be created in hardware. At the opposite extreme is a purely serial approach, in which only one field-update equation is placed in hardware, regardless of problem size. Because there is only one equation, nodes must wait in a queue to have their fields updated. Although both parallel and serial computation engines have been implemented with varying degrees of success, both approaches have significant disadvantages [4]–[9].

A massively parallel approach, while highly efficient in terms of speed, is highly inefficient in terms of area, requiring too much hardware to be practical [6], [9]. In contrast, the serial computation engine, which utilizes minimal area, does not fully exploit the parallelism inherent to the problem, thus sacrificing potential speedups. Our system utilizes a hybrid approach that balances these two extremes. This results in an accelerator that consumes enough resources to exploit parallelism, but does not require so many resources that area concerns render the design impractical.

B. Data Storage

The second area of concern in the design of an FDTD accelerator is data storage. Because each node in the mesh has associated electric fields, magnetic fields, and material properties, a great deal of storage is required to hold all of this information. There are two primary alternatives to consider when determining where to store the necessary data: utilizing the host computer's memory hierarchy or storing the data on-board with the computation engine.

When using the host computer's memory hierarchy, the majority of the problem data are stored in the host computer via cache, main memory, and the hard disk. In this case, the hardware accelerator and host computer are connected via peripheral component interconnect (PCI), universal serial bus (USB), or some other high-speed interconnect. Data are transferred from the host computer to the accelerator, the fields are updated within the accelerator, and the resultant data are sent back and stored in the computer. Alternatively, data can be stored entirely inside the hardware accelerator. In this approach, the computation engine and data storage units are co-located within the accelerator, thereby minimizing the overhead associated with transferring data between the accelerator and the computer. Because speed is the primary motivation for the design, our implementation utilizes on-board data storage to minimize the latency associated with data transfer operations. Although resultant data will still need to be passed back to the host PC in this configuration, if the intermediate data values are not needed, such as in a steady-state analysis, this overhead does not present a problem.

C. Special Nodes

The final area of concern in the development of an FDTD hardware accelerator to be discussed is the handling of special nodes. Special nodes are nodes that are not part of the standard computational mesh. These include source nodes and nodes that enforce the appropriate boundary conditions, such as the absorbing or perfect electric conductor (PEC) walls. As with data storage, special nodes can be handled by the host computer or the hardware accelerator.

As mentioned above, because of the communication overhead, it is necessary to minimize computer-to-accelerator transactions. This observation suggests that special nodes be handled within the hardware accelerator. Unfortunately, hardware designs are most efficient when they are used to perform only one task. To incorporate functionality within the accelerator to detect special nodes and handle them differently increases hardware logic and slows the overall design. Thus, the designer must choose between the communication overhead associated with computer-to-accelerator transactions, and reduced speeds due to the increased logic required to handle special nodes within the accelerator. Our system handles all computations, including special nodes, within the accelerator.

As has been shown in this paper, there is a variety of approaches to implement a hardware-based FDTD solver. In this section, we briefly examined several of these approaches with respect to three decisions that must be made by an FDTD hardware designer and described our approach. In the next section, we present a high-level description of the computational flow of our architecture.

III. Architecture

Although the design architecture contains modules that are used for clock synchronization, RAM and PCI controllers, source field generation, and overall system control, the majority of the components are used to implement the computational datapath. In this section, we describe the flow of the computational datapath (see Fig. 1).

The computational datapath begins with the counting and control unit (CCU). In addition to containing global system data, the CCU produces the coordinates and type (electric or magnetic) of the next field to be computed. These coordinates are then passed to the data dependence unit (DDU). The DDU is responsible for determining all values necessary to update the individual field components at this node (i.e., which surrounding field values are required).

The coordinates output by the DDU are then passed into a RAM address decoder (RAD). This unit takes a given field component [e.g., Ex(i,j,k)] and determines its location in memory. By including multiple DDU and RAD units in the design, several field components can be updated simultaneously. Because this generates numerous read requests, the memory switching unit (MSU) was developed to coordinate all memory transactions.

The majority of the problem data are stored in three RAM banks. Each RAM contains x-, y-, or z-directed fields and the material type of each node (e.g., air, water, silicon). As data are fetched from RAM, they are stored in register banks until all

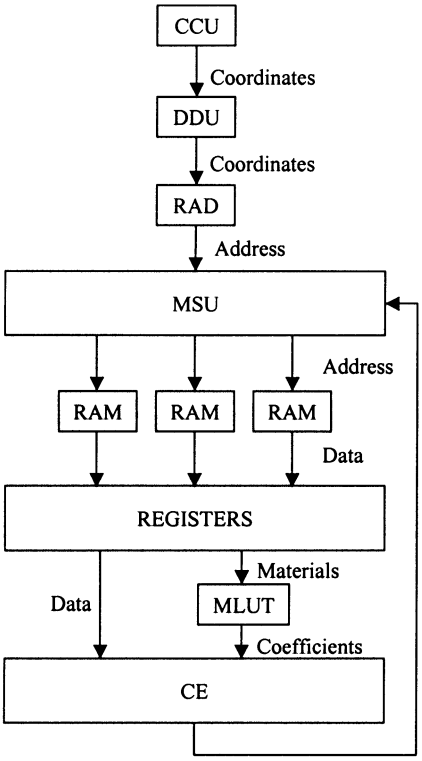

Fig. 1. Block diagram of the computational datapath. This figure shows, at a very high level, some of the key modules in the accelerator and how the data flow between them. For simplicity, only one CE is shown. Our actual design utilizes more than one CE to exploit parallelism.

necessary data have been retrieved and the system is ready to update the field.

Before the field-update computation occurs, however, several material coefficients must be determined. These coefficients are used in the computation of the field-update equation and take into account the material properties of the medium (e.g., permittivity, permeability, conductivity). To determine the coefficients, the material types are passed to the material lookup table (MLUT). The MLUT reads in bit vectors representing a given material and returns the various coefficients corresponding to those materials.

The surrounding field values and material coefficients must then be routed to the appropriate computation engine (CE), which updates the given field component based on the field-update equation. Several CEs are included in the design, allowing the system to update multiple field components in parallel. The updated values are then passed back to the MSU for storage in RAM.

In this section, we described the flow of the computational datapath. In the next section, we present the overall system implementation results and compare our FDTD accelerator with standard, PC-based implementations.

IV. Implementation Results

In order to test our architectural ideas, a prototyping board with a Xilinx Virtex-II 6000 FPGA, several RAM banks, and a PCI interface was acquired.[1] The user describes the design to analyze by means of a CAD front end developed by EM Photonics, Inc. This general-purpose CAD software allows complex designs to be modeled and includes several toolboxes for specific applications, such as photonic bandgaps and diffractive optical elements. The front-end software then sends the appropriate data, such as the mesh size and the number of timesteps to execute, to the hardware via the PCI bus. The FDTD accelerator proceeds to update the fields, periodically sending the results back to the host computer for postprocessing and visualization.

The benchmark problem was an air-filled cavity surrounded by PEC walls. The cavity was excited by a z-directed, sinusoidal point source (of unity amplitude) located at the center of the resonator. The simulation was run for 5000 timesteps with a mesh size of $43 \times 43 \times 43$. In order to analyze the error, a point detector was placed in the corner of the cavity.

The hardware results were then compared with the results obtained from C and MATLAB 6.1 programs solving the same problem on 1.13- and 2.0-GHz PCs. A C implementation was chosen to perform the speed analysis, whereas a MATLAB implementation was chosen for error analysis. This allowed us to optimize the C program for speed and the MATLAB program for error measurements.

The average absolute error was on the order of 10^{-7} with the average percentage error around 0.13%. This numerical error is a result of two primary factors. First, MATLAB is a double-precision (64 bit) language whereas the hardware implementation supports only single-precision (32 bit) arithmetic units. If precision is of the utmost importance, however, double-precision arithmetic units can easily be implemented. The second factor that contributes to the error is the computation of the source field. For simplicity, the sine function was implemented as a lookup table (LUT) with only 16 K entries. In future implementations, more entries will be included in the LUT to increase resolution or other techniques, such as the CORDIC algorithm, will be used to generate the source [10].

In terms of processing power, the 14-MHz hardware had an average throughput of approximately 150 000 nodes per second (150 Knps[2]). This is 5.66 times slower than the processing power of C running on a 1.13-GHz PC (849 Knps) and 8.55 times slower than C on a 2.0-GHz machine (1282 Knps). *Note that although the PC is clocked over 142 times faster than the hardware, the hardware is less than nine times slower.*

Certainly a design that is slower than existing solutions is not desired! However, this was a proof-of-concept design that served not only to implement our basic architectural ideas, but also to achieve the first 3-D FDTD accelerator implementation in physical hardware. As such, these results were obtained on a preliminary, nonoptimized design. A detailed analysis indicates that overlapping the computations of different nodes will result in a threefold increase in speed. Also, because the throughputs of our accelerator increase linearly with clock frequency, by increasing the clock frequency from 14 to 100 MHz, a common

[1]DN3000K10S board from The Dini Group, [Online] Available: http://www.dinigroup.com

[2]Knps = thousands of nodes processed per second, where the time to process a node is the time to update all of the fields at that node.

FPGA system speed, a sevenfold increase in throughput is possible. Although the current design is almost nine times slower than a 2.0-GHz PC running optimized C code, after the above design modifications are made, the hardware throughput will be almost two and a half times that of a 2.0-GHz PC. Note that modifying the design to work at 100 MHz is not an unreasonable goal, as the most complex units in the design are the floating-point arithmetic units, which are already capable of speeds in excess of 100 MHz. The overall clock frequency had to be reduced for some nonoptimized units related to the routing of data. These units can be pipelined, thus permitting increased clock frequencies.

Finally, it should be noted that these results are from a commercial, off-the-shelf prototyping board. Because the design had to be mapped into this general-purpose board, several architectural optimizations (such as increased parallelism) could not be implemented. Initial calculations indicate that a customized board can provide speed increases of *at least two orders of magnitude* through the addition of multiple RAM banks, increased clock frequencies, and a more efficient use of memory.

V. Conclusion

In this paper, we presented three major areas involved in a hardware-based FDTD implementation, namely the computation engine, data storage, and the handling of special nodes, and briefly discussed the range of solutions within each area. Next, we described our hardware-acceleration architecture and presented the implementation results. We found that our design not only provided accurate results, but also runtimes that will be comparable to current PC implementations in the near future. To the best of our knowledge, this work represents the first successful 3-D FDTD algorithm in hardware. We are currently working on an optimized version of this architecture and a custom printed circuit board to support our design. This will provide increased computational speeds, which we predict will easily surpass desktop computers, and will ultimately rival the performance of computer clusters.

References

[1] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propagat.*, vol. AP-14, pp. 302–307, May 1966.

[2] H. Jordan, S. Bokhari, S. Staker, J. Sauer, M. ElHelbawy, and M. Piket-May, "Experience with ADI-FDTD techniques on the Cray MTA supercomputer," in *Proc. SPIE Commercial Applications for High-Performance Computing*, vol. 4528, 2001, pp. 68–76.

[3] G. A. Schiavone, I. Codreanu, R. Palaniappan, and P. Wahid, "FDTD speedups obtained in distributed computing on a Linux workstation cluster," in *Proc. IEEE Antennas Propagation Society Int. Symp.*, vol. 3, 2000, pp. 1336–1339.

[4] J. R. Marek, M. A. Mehalic, J. Andrew, and J. Terzuoli, "A dedicated VLSI architecture for Finite-Difference Time Domain calculations," in *Proc. 8th Annu. Rev. Progress Applied Computational Electromagnetics*. Monterey, CA: Naval Postgraduate School, 1992.

[5] J. R. Marek, "An Investigation of a Design for a Finite-Difference Time Domain (FDTD) Hardware Accelerator," M.S. thesis, Air Force Inst. Technol., Wright-Patterson AFB, OH, 1991.

[6] R. N. Schneider, L. E. Turner, and M. M. Okoniewski, "Application of FPGA technology to accelerate the Finite-Difference Time-Domain (FDTD) method," in *Proc. 10th ACM Int. Symp. Field-Programmable Gate Arrays*, Monterey, CA, 2002.

[7] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini, "A custom VLSI architecture for the solution of FDTD equations," *IEICE Trans. Electron.*, vol. E85-C, pp. 572–577, 2002.

[8] L. Verducci, P. Placidi, G. Matrella, L. Roselli, F. Alimenti, P. Ciampolini, and A. Scorzoni, "A feasibility study about a custom hardware implementation of the FDTD algorithm," in *Proc. 27th General Assembly URSI*, Maastricht, Netherlands, 2002.

[9] M.E.E. thesis, Dept. Elect. Comput. Eng., Univ. Delaware, Newark, DE, 2002.

[10] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Monterey, CA, 1998.