Overview of Reconfigurable Computing Platforms and Their Applications in Electromagnetics Applications

Ozlem Kilic[1], Miaoqing Huang[2]
[1]Department of Electrical Engineering and Computer Science
The Catholic University of America, Washington, DC 20064, USA
kilic@cua.edu

[2]Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701, USA
mqhuang@uark.edu

***Abstract*—This paper investigates the utilization of field programmable gate arrays (FPGAs) in the acceleration of numerically intensive electromagnetics applications. We investigate the speed improvement by employing FPGAs for two different applications: (i) the optimization of a phased array antenna pattern by amplitude control using the ant colony optimization algorithm, (ii) implementation of the rigorous coupled wave (RCW) analysis technique for the design of engineered materials. The first application utilizes FPGAs as the only processor; i.e., all functionalities of the algorithm reside on the FPGA. The second one employs a hybrid hardware/software approach where the FPGA serves as a coprocessor to the CPU. The hybrid approach identifies the most numerically intensive part of the RCW algorithm and implements it on the FPGA. In both applications we demonstrate orders of magnitude of improvement in speed proving that FPGAs are highly flexible platforms suited well for the challenging electromagnetics problems. An overview of available FPGA platforms for scientific computing and how they compare are also presented in the paper.**

***Index Terms*—Field programmable gate array, Reconfigurable computing, Electromagnetics applications, Rigorous coupled wave analysis, Eigenvalue solver, Bio-inspired optimization, Ant colony optimization (ACO), Phased array.**

I. INTRODUCTION

The recent commercial and military applications for communications, imaging and remote sensing demand high mobility and multi-functionality. For instance, military applications require improved performance of their communication, radar and tracking systems while reducing size, cost and radar cross-section. Similarly, commercial communication devices are expected to perform seamlessly on the move for both voice and data exchange. In response, the research community has been investigating the use of advanced and engineered electromagnetic materials. We have witnessed the emerging of new classes of materials, such as meta-materials, photonic crystals, and plasmonics, etc [1]–[5]. These are typically complex heterogeneous mixtures of dielectric and metallic structures, which require rigorous electromagnetic simulation tools for an optimal design. Other applications involve smart antennas that can steer a beam electronically. The combined performance of the antenna with the beamformer can be a tedious task to simulate as the structure can consist of fine features with large overall dimensions, i.e., multiple wavelengths. However, the computations for such complex materials are often very cumbersome and time consuming. As a consequence, iterative design of advanced materials and simulations of antenna performance is often too slow to be of practical use.

There are many electromagnetic software packages that allow users to model complex 3-D structures. Many of these use one of the full-wave solutions such as Finite Difference Time Domain (FDTD) Method, Finite Element Method (FEM), Method of Moments (MoM), or asymptotic techniques like geometrical theory of diffraction (GTD), unified theory of diffraction (UTD), etc. The full wave solutions are limited to low-frequency applications or electrically small structures since they involve discretization of the geometry and the size of the problem becomes prohibitive for finer resolutions. The asymptotic methods are based on the assumption that the wavelength is much smaller than the finest part of the geometry. However, many of the practical applications involve modeling structures that possess fine details on large surfaces. The finer details suit well for full-wave solutions while the large surfaces are more appropriate for asymptotic approaches. Some typical applications are antennas on vehicle platforms, electrically large structures such as Rotman lens beam-formers [6], advanced RF material such as electronic band gap (EBG) and frequency selective surface (FSS) structures, and scattering properties of a medium with small and large features with respect to the wavelength.

This paper investigates the utilization of field programmable gate arrays (FPGAs) in the acceleration of numerically intensive electromagnetics applications as described above. FPGAs render themselves to parallel computing and can be customized to optimally fit the problem at hand, creating a highly efficient computing machine for the particular application. With the advancement of semiconductor technology, FPGAs have become mature enough to accommodate complicated computations. Due to the intrinsic parallelism of hardware

1054-4887 © 2010 ACES

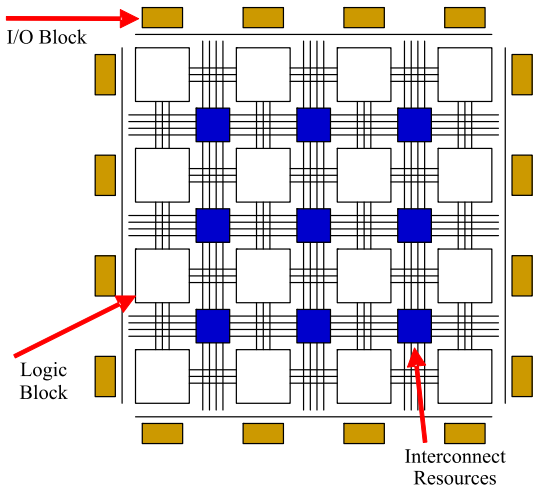


Fig. 1. The internal architecture of an FPGA device.

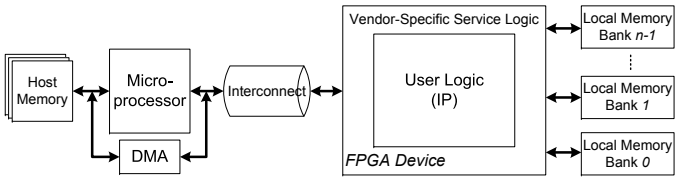


Fig. 2. The general architecture of a reconfigurable computer.

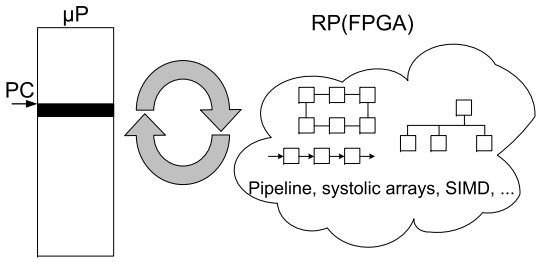


Fig. 3. The execution model of a reconfigurable computer.

implementation on FPGA devices, it is possible to achieve several orders of magnitude performance speedup compared with the corresponding implementation in software [7]. Therefore, FPGA devices have been integrated into the traditional workstations as co-processors. Generally, these workstations with addition of FPGA co-processors are called reconfigurable computers. As opposed to the specially designed ASICs, the functionality of the co-processor can be switched in milliseconds by downloading different configuration files (so the name "reconfigurable computing") into the FPGA device so that it can perform different types of operations.

The basic architecture of an FPGA device is shown in Fig. 1. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together". Logic blocks can be configured to perform complex combinational functions, or merely simple logic functions like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. The I/O blocks surrounding the logic blocks provide the interface to communicate with the outside world. The two leading FPGA manufactures as of 2010 are Xilinx [8] and Altera [9]. FPGA devices from both companies are quite visible in reconfigurable computers as co-processors.

The rest of the paper is organized as follows. In Section II, we provide an overview of the available FPGA based platforms for scientific computing applications. Programming these devices involve understanding of parallelized and pipelined computing techniques. The details on how programming can be approached are provided in Section III, with a brief description of the differences between the currently available platforms. We discuss examples of electromagnetics applications and their implementation on FPGAs in Section IV. Finally, Section V concludes this work.

II. AVAILABLE RECONFIGURABLE COMPUTERS FOR SCIENTIFIC COMPUTING

Using FPGA devices as co-processors to microprocessors in reconfigurable computers (RC) has been an industrial interest and academic research topic for many years. Fig. 2 shows a simplified architecture of a reconfigurable computer including one FPGA and one microprocessor. The FPGA co-processor is equipped with several local memory banks, usually SRAM, acting as cache. An interconnect is used to transfer data between the FPGA and the microprocessor. An application implemented on reconfigurable computers is divided into two parts. The main flow is executed on the microprocessor. The computation intensive parts of the application can be implemented on the FPGA device by taking advantage of pipelining and parallelism, as shown in Fig. 3. FPGAs differ from a single-core microprocessor in their ability to execute thousands of operations concurrently. This is achieved by programming the logic blocks in the device. A single-core microprocessor, on the other hand, is only able to perform one operation at a time.

The reconfigurable (or hybrid) computers can be divided into two subcategories based on the different integration technology used. In the first subcategory, a PCI or PCI-Express based FPGA expansion card is inserted into a conventional workstation. In most cases, the FPGA card and the workstation are from different vendors. For the second subcategory, the same vendor will design both the FPGA board and the workstation, and integrate them together using a proprietary interconnect. Since these reconfigurable computers provide more computing capacity than those in the first subcategory, they are typically called high-performance reconfigurable computers (HPRCs). In this paper, we will focus on the use of HPRCs in the field of electromagnetics. Three example systems discussed in this paper are Cray XD1 [10], SRC-6 [11] and SGI RC100 [12].

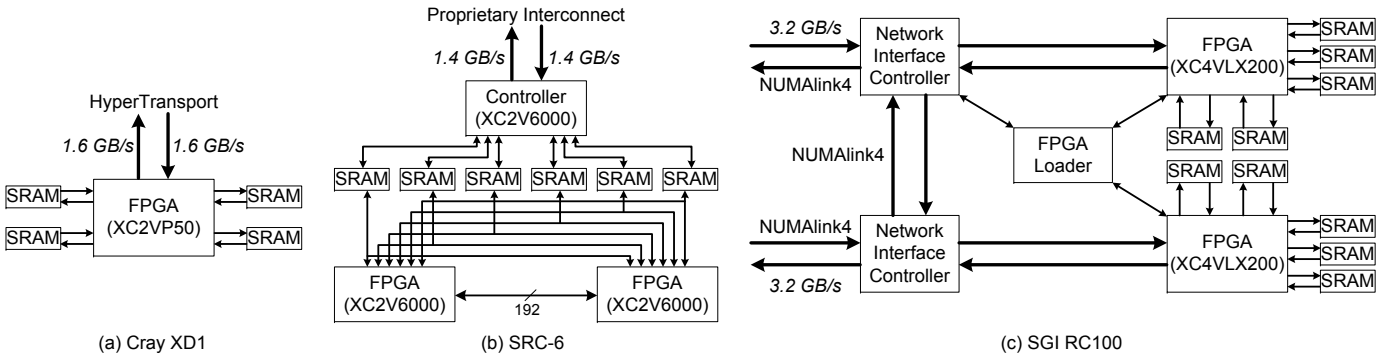


Fig. 4. The local architecture of the FPGA co-processor.

On the Cray XD1 platform, the FPGA co-processor resides on the same board as the microprocessor. This layout is different from the other two reconfigurable computers, which consist of separate FPGA board and microprocessor board. In spite of this difference, they share two similarities. (i) Multiple local SRAM modules are directly connected to the same FPGA device so that the hardware implementation can access and process multiple data blocks simultaneously, as shown in Fig. 4. On all three platforms, each memory access port is 64-bit wide. However, on both Cray XD1 and SGI RC100, the FPGA device has two separate read and write ports for each memory bank. In other words, the user logic on the FPGA device can read from and write to the same memory bank concurrently. On the other hand, the user logic on the SRC-6 platform has only one port for both reading from and writing to the same memory bank. This single-port access will degrade the performance for some applications. (ii) The FPGA co-processor is connected to the microprocessor and the host memory using high-speed interconnect in order to reduce the transportation overhead. These interconnects generally provide shorter latency and higher bandwidth for the data transfer between the FPGA and the microprocessor.

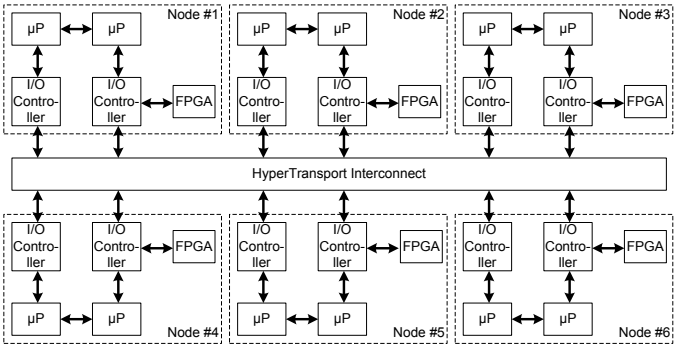
For an application that needs to use multiple FPGA devices at the same time in an RC system, different platforms deal with it differently.

- The Cray XD1 is a cluster-based reconfigurable computer. In other words, it may consist of dozens of FPGA co-processors, each of which belongs to a separate workstation. As shown in Fig. 5(a), 6 workstations (i.e., nodes) compose a chassis, which is the basic unit in a Cray XD1 system. If the user intends to use more than one FPGA co-processor, it has to cross the boundary of the operating system. One approach to use multiple FPGA devices in a single application is to use MPI (Message Passing Interface). Apparently, all the communication between any two FPGA co-processors has to be handled explicitly by software.
- The SRC-6 is a cluster-based platform as well. As shown in Fig. 4(b), there are two FPGA devices in one workstation. The user can program these two FPGA devices simultaneously in one application. These two FPGA devices can communicate to each other using the dedicated 192-bit channel. Once the communication is beyond the boundary of an operating system, MPI can be used for data transfer among different systems.
- The SGI RC100 is different from the other two platforms. On SGI RC100, different types of processing boards, i.e., microprocessor boards and FPGA boards, are connected to a same network and are visible in one single operating system, as shown in Fig. 5(c). However, as shown in Fig. 4(c), there is no communication channel between two FPGA devices on the same board. If the raw data can be divided into independent pieces, each of which is to be processed by one FPGA device, the user can allocate multiple FPGA co-processors in one software thread, and the co-processor driver will distribute the data evenly across different FPGAs. On the other hand, if the user wants to use multiple FPGAs and there is communication among these FPGAs during the data processing, a multiple-thread application is required to deal with this scenario.

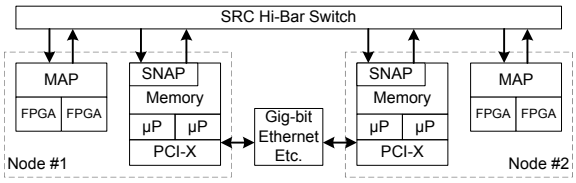
The implementation on the FPGA co-processor depends on the available resources, e.g., the available logic blocks in the FPGA device and the number of local memory banks. One example in the electromagnetics domain is the matrix multiplication, i.e., $C = AB$. Each element in C is the product of a row in A and a column in B, i.e., $c_{i,j} = \sum_{k=1}^{M} a_{ik} b_{kj}$. If matrix A and B are stored in two separate local memory banks and the result matrix C is saved in another separate memory bank, the user logic can read one pair of (a_{ik}, b_{kj}) every clock cycle assuming the multiplier and the accumulator are both fully pipelined. Therefore, it would take approximately M clock cycles to compute one element in matrix C no matter how complex the multiplication and the accumulation are.

III. PROGRAMMING RECONFIGURABLE COMPUTERS

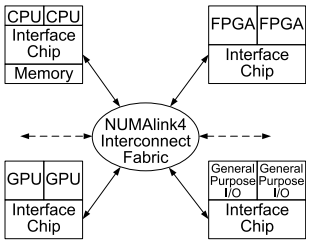
An application implemented on a reconfigurable computer consists of a hardware part and a software part as shown in Fig. 3. The user needs to program both parts and then integrate them together using vendor APIs (Application Programming Interfaces). The typical programming language for



(a) A Cray XD1 chassis.



(b) An SRC-6 consisting of 2 nodes.



(c) SGI RC100 architecture.

Fig. 5. The architecture of three representative reconfigurable computers.

```
// The block for source data reading
always @ (posedge clk) begin
    if (reset) begin
        mem_0_rd_cmd_vld     <= 0;
        mem_1_rd_cmd_vld     <= 0;
        mem_0_rd_addr        <= 0;
        mem_1_rd_addr        <= 0;
    end
    else begin
        if (mem_0_rd_addr == (MATRIX_RANK - 1)) begin
            mem_0_rd_cmd_vld <= 0;
            mem_1_rd_cmd_vld <= 0;
            mem_0_rd_addr    <= mem_0_rd_addr;
            mem_1_rd_addr    <= mem_1_rd_addr;
        end
        else begin
            mem_0_rd_cmd_vld <= 1;
            mem_1_rd_cmd_vld <= 1;
            mem_0_rd_addr    <= mem_0_rd_addr + 1;
            mem_1_rd_addr    <= mem_1_rd_addr + 1;
        end
    end
end

// The block for source data feeding
multiplier U1(.di1(mem_0_rd_data),.di2(mem_1_rd_data),
  .di_vld(mem_0_rd_data_vld),.do(product),.do_vld(pro_vld));
accumulator U2(.di(product),.di_vld(pro_vld),
  .do(ac),.do_vld(ac_vld));

// The block for result data writing
always @ (posedge clk) begin
    if (reset) begin
        mem_2_wr_cmd_vld     <= 0;
        mem_2_wr_addr        <= 0;
    end
    else begin
        mem_2_wr_cmd_vld     <= ac_vld;
        mem_2_wr_data        <= ac;
        if (mem_2_wr_cmd_vld) begin
            mem_2_wr_addr    <= mem_2_wr_addr + 1;
        end
        else begin
            mem_2_wr_addr    <= mem_2_wr_addr;
        end
    end
end
```

Fig. 6. Compute one element in matrix C using Verilog.

the software part is the C language in most cases. The more challenging part is the hardware part and it typically requires some hardware design expertise to gain the full benefit of using the FPGA co-processor.

It has been mentioned before that an FPGA co-processor is capable of performing thousands of operations concurrently. In order to achieve this concurrency, all the logic blocks in one FPGA device have to be programmed into a specific status by using a configuration file. Since the implementation depends on the available hardware resources on the FPGA device (e.g., memory, built-in multipliers, logic blocks), it might be necessary at times to distribute the hardware part into multiple FPGA configurations, each of which is called a *bitstream*. At runtime, different configurations are downloaded into the FPGA device following a pre-defined order in an application.

There are two different approaches for the user to implement the functions running on the FPGA co-processor, i.e., hardware description languages (HDLs) and high-level languages (HLLs). The default languages to program the FPGA device are HDLs, i.e., VHDL and Verilog HDL. In the meantime, there are several HLLs available, e.g., Carte-C [11], Impulse C [13], Handel-C [14], and Mitrion-C [15], bringing the ease of use at the expense of efficiency.

If HDLs are used to design the bitstream, it will require three parallel blocks to implement the matrix multiplication example, as shown in Fig. 6.

- One block is to control the source data reading from two memory banks saving matrix A and B. The functionality of this block involves the generation of reading addresses and reading commands.
- One block is to check the arrival of source data and feed them into the multiplier and the accumulator.
- One block is to control the result data writing to another memory bank for matrix C. The functionality of this block involves the generation of writing addresses and writing commands.

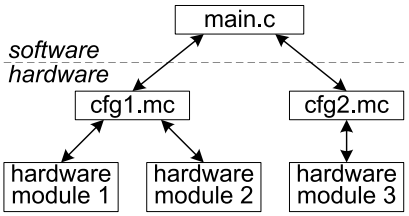


Fig. 7. Implement an application on SRC-6 using Carte-C.

```
/* Define three 2D arrays of 400x400 in three local *
 * memory banks                                      */
OBM_BANK_A_2D (A, double, 400, 400)
OBM_BANK_B_2D (B, double, 400, 400)
OBM_BANK_C_2D (C, double, 400, 400)
......
ac = 0;
for (k=0; k<400; k++) {
    Multiplier(A[i][k], B[k][j], &product);
    Accumulator(product, &ac);
}
C[i][j] = ac;
```

Fig. 8. Compute one element in matrix C using Carte-C.

On both Cray XD1 and SGI RC100 platforms, either VHDL or Verilog can be used to generate the bitstream. In order to reduce the complexity of communications with the local memory, the vendor generally provides the service logic (as shown in Fig. 2), which gives a simplified interface to access the local memory as well as the interconnect.

On SRC-6, the vendor provides a high-level language, i.e., Carte-C, to implement the hardware part. Carte-C is a rich subset of C, with non-standard extensions to control hardware instantiation and parallelism. Each FPGA bitstream is defined by a single Carte-C file, which is converted into HDL during the compilation. For instance, in the case shown in Fig. 7, the hardware part is distributed into two bitstreams, described in two Carte-C files, i.e., *cfg1.mc* and *cfg2.mc*. A single Carte-C file consists of multiple blocks, which are executed in a sequence during the runtime. The Carte-C compiler will maximize the parallelism within a single block to improve the performance. It is difficult for the compiler to achieve the maximum performance for complicated operations. In this case a hand-written HDL module can be integrated into the bitstream, leaving the main flow written in Carte-C. As demonstrated in Fig. 7, two hardware modules are integrated into the first bitstream file.

Fig. 8 shows a section of codes to compute one element in the resultant matrix, in which Multiplier and Accumulator are two pipelined HDL modules. If A, B and C are stored in three separate memory banks, the Carte-C compiler is capable of generating fully pipeline hardware codes for the maximum performance.

Carte-C is a proprietary language used on the SRC-6 platform; i.e., it does not extend to other platforms. Other HLLs can be used across different platforms. For example, both Impulse C and Mitrion-C can be used to program Cray XD1 and SGI RC100.

IV. DEVELOPING ELECTROMAGNETICS APPLICATIONS ON RECONFIGURABLE COMPUTERS

A. Background Information

Electromagnetics applications tend to be numerically intensive, with most problems requiring memory intensive implementations. Complex structures can be analyzed using numerical methods by segmenting the structure into small meshes. Often these meshes can be treated independently from the rest of the geometry with the use of appropriate boundary conditions. This allows analysis to be carried out in a parallel fashion.

In terms of utilizing hardware acceleration in electromagnetics applications, there is an increasing interest in the use of general purpose graphics processing units [16], [17] mostly due to their C-like implementation and relatively low cost. The use of VLSIs has also been suggested in [18] and [19]. However, the FPGA implementation of electromagnetics algorithms has been very scarce due to the hardware expertise required on these platforms. One area of numerical electromagnetics that has been investigated for the FPGA implementation is the FDTD algorithm [20]–[23]. FDTD expresses Maxwell's equations in difference form and renders itself to parallel implementation as each cell can be handled separately from the others. A three dimensional FDTD model on hardware has been reported in [24], where an FPGA-based accelerator has been used in conjunction with a host PC and a CAD interface. This algorithm has been applied to the analysis of a Rotman lens in [25].

In the following sections we demonstrate the FPGA implementation of two applications: (i) optimization of a linear array antenna pattern using the ant colony optimization technique, (ii) implementation of the rigorous coupled wave analysis algorithm. The first implementation utilizes the FPGA as the sole processor with the CPU functionality being to call the FPGA and retrieve the final result. The second implementation uses a hybrid hardware/software approach where only the most numerically intensive components of the algorithm resides on the FPGA.

B. Ant Colony Optimization (ACO) Implementation

The utilization of FPGAs in the field of electromagnetics was recently investigated by applying the ant colony optimization (ACO) method in the design of phased array antennas for multiple beam satellite communication systems [26]. In this application, the amplitudes of the array elements were optimized to reduce the co-channel interference in a multiple beam satellite communication system. Potential gains in the speed of the calculations in the order of 10,000 has been demonstrated for the particular application. A brief overview of the problem solved and how the FPGA was utilized is discussed in this section.

1) ACO Algorithm: The ACO is a nature inspired optimization algorithm that utilizes heuristic search principles carried out simultaneously by agents and their collective intelligence.

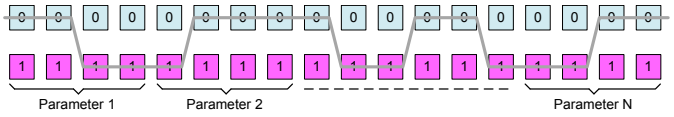


Fig. 9. Path Definition in ACO.

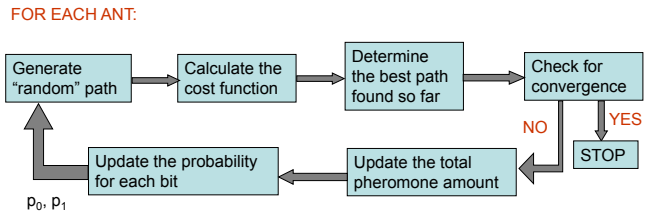


Fig. 10. Implementation of ACO on CPU - Recursive for each ant.

The ACO mimics the behavior of ants in their search for the shortest path between their nest and the food. Despite being nearly blind, ants demonstrate the capability to establish the shortest path between their nest and food. They achieve this by depositing a chemical substance called pheromone on their paths, which is used later on by other ants in their search process. During this process, the most traveled path is marked with the highest level of pheromone. This positive feedback behavior allows more ants to choose the path with the most pheromone amount [27]. The random search is iteratively applied by the ants until one of the chosen paths satisfies the required convergence criteria. The intelligence is introduced to the random search process via the cost function, which measures how far off an ant is from the desired solution. Since each solution is represented by a path in ACO, the optimization space is discretized into binary strings where a path is defined by the choice of 1 or 0 for the bit value at each bit position as shown in Fig. 9 [28].

Each path represents a possible solution and a number of ants sample the solution space at each iteration. Once all ants decide on their paths, the cost function is computed for each path. The cost is a measure of how satisfactory a solution is, with low cost values implying a "better" solution. The pheromone amount to be laid on each path is inversely proportional to the cost value associated with the path. The probability of a zero for each bit position is then calculated for all the ant paths as a function of the total pheromone levels on the path as follows:

$$p_0 = \frac{\tau_0}{\tau_0 + \tau_1} \tag{1}$$

where τ_0 and τ_1 correspond to the total pheromone levels accumulated at the bit position of interest for bit value of zero and one, respectively. The probability for bit value of one is then calculated as $1 - p_0$ for each bit. A block diagram of the algorithm is shown in Fig. 10, where each ant is processed iteratively on a typical software implementation.

2) Application - Linear Array Optimization: The ACO algorithm is used to optimize the radiation from a linear array.

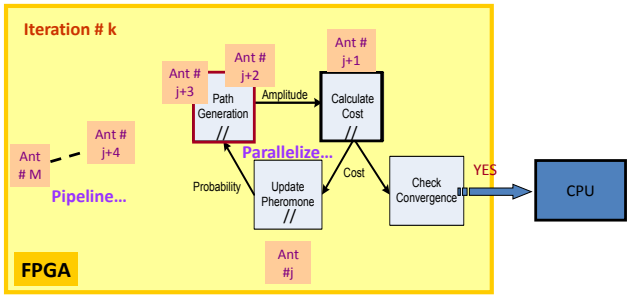


Fig. 11. Implementation of ACO on FPGA - parallelized and pipelined.

Nulls are placed at certain positions to reduce potential interference in a multiple beam satellite communication system. To achieve the desired radiation, the amplitudes of the array elements are optimized.

This implementation is unique in the sense that the FPGA has been utilized as the sole processer for the entire implementation. This avoids any overhead of communication between the microprocessor and the FPGA, and uses the FPGA to its full potential. The CPU is only used to call the FPGA and retrieve the results for processing. In short, an ACO machine has been implemented with this application. While this enables the ultimate parallelization and pipelining of the algorithm, there are limitations due to the problem size that can be handled by the particular FPGA at hand. The implementation was carried out on the SGI Altix 450 platform utilizing the Xilinx Virtex4LX200 FPGAs as demonstrated in Fig. 11.

Paths are produced using 8 bits for each optimization parameter (i.e., the amplitudes of the array elements), 40 parameters in each ant path (i.e., the number of array elements), 40 ant paths per iteration (i.e., 40 ants carry search for a solution simultaneously in each iteration), and as many iterations as it takes to converge, with an upper limit set by the user. With this implementation, increasing the number of bits per parameter will increase the FPGA resource requirement for this function, but will not increase the processing time. The number of nulls that can be achieved is also run in parallel, i.e., has no impact on the processing time as long as there are sufficient FPGA resources. We were able to carry out 8 bits and three nulls in parallel at a clock rate of 100 MHz for the optimization of a 40 element array on the Altix platform.

As observed in Fig. 11, there are three major sections in the algorithm: Path Generation, Cost Calculation and Pheromone Update. The details of the implementation of these sections on FPGA is given in [26].

As a test, only three null positions were required to be below -30 dB. When the algorithm was run on a standard PC (CPU: Intel Pentium M, 3 GHz and RAM: 1 GB) using Matlab, the time per a single iteration took about 0.47 seconds. The same algorithm when implemented on C and run on the same platform ran about 53.4 times faster than the Matlab version, roughly at 8.8 milliseconds per iteration. The VHDL implementation on the Altix 450 system performed at 31.3

microseconds for runs after the bit loading was completed, resulting in a factor of 15,160 in speed compared with the Matlab implementation.

C. Rigorous Coupled Wave Analysis Implementation

The previous application was small enough to be implemented fully on FPGA utilizing the platform to its full potential and achieving very promising acceleration. The ACO implementation was fully optimized to achieve this kind of acceleration. However, increasing the number of nulls or number of bits per variable are not feasible as the algorithm would cease to fit on the FPGA. The problems in electromagnetics are often complex, and require flexibility in the range of the parameters. The second application is one such example. The Rigorous Coupled Wave (RCW) algorithm applies to diffraction problems from multiple layers with periodic gratings. It is based on an extension of enhanced transmittance matrix approach [29] and adopts Lalanne's improved eigenvalue formalism [30]. A detailed discussion on the RCW algorithm can be found in these references. It has been used effectively in the design of engineered materials, such as antireflective surfaces [31]–[33]. We provide a brief overview in this section in order to describe our motivations for the hardware implementation.

The stacked multiple layer in RCW algorithm can consist of any number of gratings. However, all gratings must be periodic with the same periodicity along a given direction on the plane. The periodicity results in a spatially periodic permittivity (and inverse permittivity) within each layer and can be represented as a Fourier series expansion, as follows.

$$\varepsilon_l(x,y) = \sum_{g,h} \varepsilon_{l,gh} \exp\left(j\frac{2\pi g x}{\Lambda_x} + j\frac{2\pi h y}{\Lambda_y}\right) \tag{2a}$$

$$\varepsilon_l^{-1}(x,y) = \sum_{g,h} A_{l,gh} \exp\left(j\frac{2\pi g x}{\Lambda_x} + j\frac{2\pi h y}{\Lambda_y}\right) \tag{2b}$$

where $\varepsilon_{l,gh}$ and $A_{l,gh}$ are the Fourier coefficients for the lth layer in the stack for the permittivity and inverse permittivity respectively. The electric field inside the layers can similarly be expressed as a Fourier series in terms of spatial harmonics. Maxwell's equations for the layered structure can be written in terms of the tangential components of the electric and magnetic fields, resulting in a coupled equation set in (3), where S_l represents the amplitudes of the spatial harmonics of the electric field in the lth layer, with subscripts x and y denoting the directions of periodicity in the plane of the stack. The parameters B and D in (3b) are matrices given as $B = k_x \varepsilon_l^{-1} k_x - I$ and $D = k_y \varepsilon_l^{-1} k_y - I$.

Thus, the coupled wave equation can be solved by finding the eigenvalues of the matrix Ω_l, which is a function of the stack properties. The rank of this matrix is $M \times N$, where M and N are the number of spatial harmonics retained along the two dimensions of periodicity in the plane of stacked layers. Ideally an infinite number of them are needed for an exact solution but truncation with minimal error is possible. Despite this truncation, the rank can be in the order of magnitude of 400 or more for a typical application of AR surface design. Hence, the most numerically intensive component of the RCWA algorithm is this eigenvalue computation.

Algorithm 1: Hessenberg Reduction

Input: A square complex matrix A with rank n
Output: The reduced Hessenberg matrix H

1.1 **for** k=*0* **to** $n-3$ **do**
1.2 $v_k = \textbf{House}(A_{k+1:n-1,k})$; /*Step 1: See Alg. 2*/
1.3 $A_{k+1:n-1,k:n-1} = A_{k+1:n-1,k:n-1} - 2v_k(v_k^* A_{k+1:n-1,k:n-1})$; /*Step 2: $P_k A_{k+1:n-1,k:n-1}$, $P_k = I - 2v_k v_k^*$*/
1.4 $A_{0:n-1,k+1:n-1} = A_{0:n-1,k+1:n-1} - 2(A_{0:n-1,k+1:n-1} v_k)v_k^*$; /*Step 3: $A_{0:n-1,k+1:n-1} P_k$*/

Algorithm 2: House(x)

Input: A complex vector x
Output: The Householder vector v

2.1 $\alpha = -e^{i\varphi}\|x\|$; /*$\varphi$ is the argument of x_1*/
2.2 $u = x - \alpha e_1 = x + e^{i\varphi}\|x\|e_1$; /*$e_1 = [1, 0, ..., 0]^T$*/
2.3 $v = \frac{u}{\|u\|}$;

1) QR Eigenvalue Algorithm: Given a square matrix $A \in \mathbb{C}^{n\times n}$, an eigenvalue λ and its associated eigenvector $\mathbf{v}$ are, by definition, a pair obeying the relation $A\mathbf{v} = \lambda\mathbf{v}$. Equivalently, $(A - \lambda I)\mathbf{v} = 0$ (where I is the identity matrix), implying $\det(A - \lambda I) = 0$. This determinant can be expanded into a polynomial in λ, known as the *characteristic polynomial* of A. One common method for determining the eigenvalues of a small matrix is by finding the roots of its characteristic polynomial. However, a general polynomial of order $n > 4$ cannot be solved by a finite sequence of arithmetic operations and radicals. Therefore, many numerical iterative algorithms have been proposed [34] to solve the eigenvalue problem of high-rank square matrices, such as Power Method, Inverse Iteration, Jacobi Method, etc. Among these, the shifted Hessenberg QR algorithm [35]–[37] is accepted as a practical solution adopted in most applications to deal with general square matrices.

There are two phases in the practical QR algorithm, as described in (4). In the first phase, the original matrix A is reduced to the upper Hessenberg form H using the Householder transformation [38]. The second phase involves applying the implicit QR iteration with shifts on the unreduced Hessenberg matrix H until it converges to a triangular matrix, i.e., the Schur form S. The eigenvalues of a triangular matrix are listed on the diagonal, i.e., the $\otimes$s in (4), and the eigenvalue problem is solved once this form is achieved.

2) Implementation on SGI RC100 Reconfigurable Computer: The RCW algorithm in the most general sense creates a square matrix with complex values. Both real and imaginary parts of a matrix entry are represented in double precision (64-bit) floating-point format. In the hardware implementation of QR eigenvalue algorithm on FPGA device, we combine the two physical local memory banks into a 128-bit wide logical memory bank so that each memory entry can store one com-

$$\begin{bmatrix}\partial^2 S_{l,y}/\partial z'^2\\ \partial^2 S_{l,x}/\partial z'^2\end{bmatrix} = \Omega_l \begin{bmatrix}S_{l,y}\\ S_{l,x}\end{bmatrix} \tag{3a}$$

$$\Omega_l = \begin{bmatrix}k_x^2 + D[\alpha\varepsilon_l + (1-\alpha)A_l^{-1}] & k_y\{\varepsilon_l^{-1}k_x[\alpha A_l^{-1} + (1-\alpha)\varepsilon_l] - k_x\}\\ k_x\{\varepsilon_l^{-1}k_y[\alpha\varepsilon_l + (1-\alpha)A_l^{-1}] - k_y\} & k_y^2 + B[\alpha A_l^{-1} + (1-\alpha)\varepsilon_l]\end{bmatrix} \tag{3b}$$

$$\underset{A}{\begin{bmatrix}\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times\end{bmatrix}} \xrightarrow{\text{Phase 1}} \underset{\text{Hessenberg } H}{\begin{bmatrix}\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times\\ 0&\times&\times&\times&\times\\ 0&0&\times&\times&\times\\ 0&0&0&\times&\times\end{bmatrix}} \xrightarrow{\text{Phase 2}} \underset{\text{Triangular } S}{\begin{bmatrix}\otimes&\times&\times&\times&\times\\ 0&\otimes&\times&\times&\times\\ 0&0&\otimes&\times&\times\\ 0&0&0&\otimes&\times\\ 0&0&0&0&\otimes\end{bmatrix}} \tag{4}$$

Table 1. Calculation breakdown of iteration k in Hessenberg reduction.

Step	Sub-step	Calculation	Number of clock cycles for computation*	
1	1.1	$\|x\|,\ \|x_1\|$	$n-k-1$	$3k^2-9nk+6n^2-3n-2$
	1.2	$x_{1\_r} + \|x\|\cos\varphi,\ x_{1\_i} + \|x\|\sin\varphi$	1	
	1.3	$\|u\|$	$n-k-1$	
	1.4	$u/\|u\|$	$n-k-1$	
2	2.1	$m = v_k^* A_{k+1:n-1,k:n-1}$	$(n-k)(n-k-1)$	
	2.2	$N = v_k m$	$(n-k)(n-k-1)$	
	2.3	$A_{k+1:n-1,k:n-1} - 2N$	$(n-k)(n-k-1)$	
3	3.1	$m' = A_{0:n-1,k+1:n-1} v_k$	$n(n-k-1)$	
	3.2	$N' = m' v_k^*$	$n(n-k-1)$	
	3.3	$A_{0:n-1,k+1:n-1} - 2N'$	$n(n-k-1)$	

*Ignoring all latencies.

plete matrix entry. Therefore, the real part and the imaginary part of a complex variable can be accessed simultaneously.

As described earlier, there are two phases in the QR algorithm. These phases are implemented in two separate FPGA configurations. The first phase, Hessenberg reduction, is carried out by applying the Householder reflection for $n-2$ iterations (see Alg. 1), where n is the rank of the original matrix A. Each iteration comprises three steps, as shown in Table 1. Each step further includes multiple sub-steps. In our hardware design, Steps 1, 2 and 3 comprise 4, 3 and 3 sub-steps, respectively. All iterations, the steps in each iteration, and the sub-steps within every step have to be carried out sequentially due to the data dependency among them. The advantage of hardware implementation comes from the parallel processing within each sub-step. For example, Sub-step 1.1 involves multiplication, addition, accumulation and square root operation to calculate the norm of a vector. If all the basic operators, e.g., multipliers and adders, are fully pipelined, it will take roughly $n-k-1$ clock cycles to finish this sub-step (if we ignore all potential latencies). By putting everything together, the total number of clock cycles required to reduce a matrix of rank n to its Hessenberg form can thus be computed as:

$$\sum_{k=0}^{n-3}(3k^2 - 9nk + 6n^2 - 3n - 2) = \frac{5}{2}n^3 - \frac{9}{2}n - 11. \tag{5}$$

The second phase of the QR algorithm is to convert the upper Hessenberg matrix to its upper triangular form, which is implemented as a hardware/software co-design (see pp. 359 in [39] for a detailed description of the algorithm). The main step of the second phase is the Francis QR Step, in which the most computation demanding part is implemented in hardware as a separate FPGA configuration. Its functionality is shown in Alg. 3. By comparing Alg. 1 and Alg. 3, it can be found that Alg. 3 is a shrunk version of Alg. 1. In other words, the implementation of both algorithms will share the majority of their logic such as the floating-point operators and the control flow. Some small modifications are required to reduce the scope of the computation in Alg. 1 to match the functionality of Alg. 3. In general, the computation in the Francis QR Step is significantly less than the computation in the Hessenberg reduction phase.

Algorithm 3: Francis QR Step (hardware part)

Input: A square complex matrix H with rank n
Output: Matrix H
3.1 **for** k=0:$n-5$ **do**
3.2 $v_k = \textbf{House}(H_{k+1:k+3,k})$;
3.3 $H_{k+1:k+3,k:n-1} = H_{k+1:k+3,k:n-1} - 2v_k(v_k^* H_{k+1:k+3,k:n-1})$;
3.4 $H_{0:k+4,k+1:k+3} = H_{0:k+4,k+1:k+3} - 2(H_{0:k+4,k+1:k+3}v_k)v_k^*$;

3) Results: The hardware implementation of Hessenberg reduction occupies 56,520 (63%) slices on the target FPGA device and runs at 100 MHz. The basic operators, i.e., mul-

Table 2. Performance improvement of Hessenberg reduction.

Matrix Rank	Computation Time (s)		Speedup	Matrix Rank	Computation Time (s)		Speedup	Matrix Rank	Computation Time (s)		Speedup
	Hardware*	Software			Hardware*	Software			Hardware*	Software	
20	0.007	0.062	**9.4**	180	0.161	428.911	**2663.3**	340	1.019	5476.617	**5375.0**
40	0.008	1.020	**123.4**	200	0.217	654.289	**3013.0**	360	1.206	6964.269	**5773.9**
60	0.013	5.209	**410.2**	220	0.285	957.911	**3355.5**	380	1.415	8696.029	**6144.3**
80	0.021	16.553	**789.6**	240	0.367	1358.445	**3696.9**	400	1.647	10717.100	**6505.7**
100	0.034	40.516	**1184.9**	260	0.464	1870.955	**4035.2**	420	1.904	13055.131	**6858.1**
120	0.054	84.318	**1574.2**	280	0.576	2516.849	**4371.2**	440	2.185	15750.099	**7207.7**
140	0.080	156.366	**1944.8**	300	0.705	3318.075	**4707.9**	460	2.493	18859.268	**7563.6**
160	0.116	267.548	**2309.5**	320	0.852	4293.784	**5038.9**	480	2.829	22393.864	**7914.8**

*Including data transportation time and data processing time.

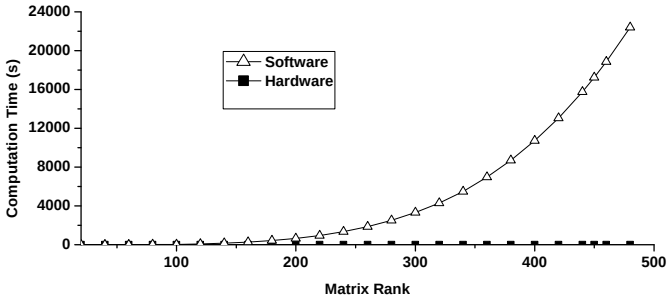


Fig. 12. Computation time of Hessenberg reduction.

tipliers, adders, subtractor, dividers and square rooters, are generated using CORE Generator, which is a tool included in the Xilinx ISE package. The rank of the object matrix is passed to hardware design as a parameter through a register. Before the FPGA starts processing, the original matrix as well as its rank are transferred from host memory to FPGA local memory. After the processing is finished, the upper Hessenberg matrix is transferred back to host memory. We tested matrices of different ranks and collected their corresponding hardware computation times, as listed in Table 2 and Fig. 12. The hardware computation time consists of both data transportation time and data processing time. It is found that the measured time matches the estimation using (5) in all cases.

For a comparison of acceleration over a pure software based implementation, we coded the Hessenberg reduction phase in C++ and ran it on a PC with Itanium 2 using a 1.6 GHz microprocessor. The speedup between is in the order of thousands (as shown in Fig. 12), which is mainly due to two factors. (i) The hardware implementation is fully pipelined, which means that multiple operations can be processed concurrently. On the other hand, the microprocessor has to process these operations in a sequential means. (ii) FPGA devices are equipped with large amount of directly accessible local memory, e.g., 40 MB on Altix RASC RC100. The local memory of FPGA devices can be compared to the L1/L2 cache of microprocessors, which are much smaller in terms of capacity. As we can see from Alg. 1, the Hessenberg reduction operation spans on all the matrix, along both columns and rows. Since the local memory of the FPGA device is quite large, it is able to accommodate the whole matrix. On the other hand, the Hessenberg reduction operation on the microprocessor is accompanied by frequent data swapping among the L1 cache, the L2 cache and the main memory, which contributes a lot of overhead in the software implementation.

The hardware implementation of Alg. 3 takes almost the same resources (i.e., 56,327 (63%) slices) on the FPGA device and runs at the same frequency. The interface to the second bitstream is the same as the first one. We applied the same type of comparison between the hardware implementation and the software version on the Francis QR Step. For a 480×480 matrix, the computation time is 0.450 s for software and 0.063 s for hardware, respectively. In other words, the hardware implementation is able to outperform the corresponding software version by 7.2 folds for those matrices we are interested. The comparatively small performance improvement is mainly due to the dramatic reduction of computation in Alg. 3. For example, the computation in line 3.4 in Alg. 3 only involves $3k + 15$ matrix elements. The corresponding line (i.e., line 1.4) in Alg. 1 involves $\mathcal{O}(n^2)$ matrix elements. Therefore, the advantage of a deep pipeline is more evident in the hardware implementation for Hessenberg reduction.

V. CONCLUSIONS

The potential use of FPGAs in electromagnetics has been demonstrated in the context of two applications: (i) the optimization of a linear array using the ant colony optimization (ii) implementation of rigorous coupled wave analysis method. The first application renders itself to parellel computing as the ant colony optimization is based on sampling of the optimization space simultaneously by a set of "ants". Like in many other heuristic search algorithms, the simultaneous search is independent of each other in each iteration while the agents gather collective intelligence. The problem investigated was small enough to fit fully on a single FPGA, enabling remarkable speed improvement (in the order of 15,000). This application demonstrated the ultimate power of FPGAs when the platform and problem are a perfect fit. The second application involved a more challenging task, where the FPGA was utilized as a co-processor to the CPU, mainly carrying out the most numerically intensive part of the algorithm. The task was the computation of eigenvalues of a complex matrix with rank of 400 or more. We have used the QR eigen value algorithm and implemented

the Hessenberg reduction and Francis QR methods on the FPGA. We have observed speed improvement in the order of thousands, with increased efficiency as the matrix rank increases. While FPGAs are finding their way slowly in the scientific computing area due to the challenges in being able to implement code using hardware description languages, their potential in providing reconfigurable parallelism make them an attractive platform.

ACKNOWLEDGMENT

The authors would like to thank Charles Conner for the software implementation of Hessenberg reduction and the integration of the FPGA bitstreams into the QR algorithm.

REFERENCES

[1] J. Witzens, M. Lončar, and A. Scherer, "Self-collimation in planar photonic crystals," *IEEE J. Sel. Topics Quantum Electron.*, vol. 8, no. 6, pp. 1246–1257, Nov. 2002.

[2] S. Y. Lin, E. Chow, S. G. Johnson, and J. D. Joannopoulos, "Demonstration of highly efficient waveguiding in a photonic crystal slab at the 1.5-μm wavelength," *Optics Letters*, vol. 25, no. 17, pp. 1297–1299, Sep. 2000.

[3] M. Notomi, K. Yamada, A. Shinya, J. Takahashi, C. Takahashi, and I. Yokohama, "Extremely large group-velocity dispersion of line-defect waveguides in photonic crystal slabs," *Physical Review Letters*, vol. 87, no. 25, pp. 253 902–1–253 902–4, Dec. 2001.

[4] M. Lončar, D. Nedeljkovic, T. Doll, J. Vučković, A. Scherer, and T. P. Pearsall, "Waveguiding in planar photonic crystals," *Applied Physics Letters*, vol. 77, no. 13, pp. 1937–1939, Sep. 2000.

[5] S. John, "Strong localization of photons in certain disordered dielectric superlattices," *Physical Review Letters*, vol. 58, no. 23, pp. 2486–2489, Jun. 1987.

[6] O. Kilic and R. Dahlstrom, "Rotman lens beam formers for army multifunction RF antenna applications," in *Proc. IEEE AP-S International Symposium and USNC/URSI National Radio Science Meeting,* vol. 2B, pp. 43–46, Jul. 2005.

[7] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *IEEE Computer*, vol. 41, no. 2, pp. 78–85, Feb. 2008.

[8] *http://www.xilinx.com*.

[9] *http://www.altera.com*.

[10] *Cray XD1™FPGA Development (S-6400-14)*, Cray Inc., May 2006.

[11] *SRC Carte™C Programming Environment v2.2 Guide (SRC-007-18)*, SRC Computers, Inc., Aug. 2006.

[12] *Reconfigurable Application-Specific Computing User's Guide (007-4718-007)*, Silicon Graphics, Inc., Jan. 2008.

[13] *Impulse C – http://www.impulsec.com*, Impulse Accelerated Technologies, Inc., 2009.

[14] *Handel-C Language Reference Manual*, Agility Design Solutions Inc., 2007.

[15] *Mitrion C – http://www.mitrionics.com*, Mitrionics AB, 2009.

[16] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications," *IEEE Antennas Propag. Mag.*, vol. 47, no. 6, pp. 71–78, Dec. 2005.

[17] N. Takada, T. Takizawa, Z. Gong, N. Masuda, T. Ito, and T. Shimobaba, "Fast computation of 2-D finite-difference time-domain method using graphics processing unit with unified shader," *IEICE Trans. Inf. Syst.*, vol. J91-D, no. 10, pp. 2562–2564, 2008.

[18] J. R. Marek, M. A. Mehalic, J. Andrew, and J. Terzuoli, "A dedicated VLSI architecture for Finite-Difference Time Domain calculations," in *Proc. 8th ACES Conference*, 1992.

[19] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini, "A custom VLSI architecture for the solution of FDTD equations," *IEICE Transactions on Electronics*, vol. E85-C, no. 3, pp. 572–577, Mar. 2002.

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

[21] J. P. Durbano, *Hardware implementation of a 1-dimensional Finite-Difference Time-Domain algorithm for the analysis of electromagnetic propagation*. M.E.E.Thesis, Department of Electrical and Computer Engineering, University of Delaware, Newark, USA, 2002.

[22] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik, "Implementation of three-dimensional FPGA-based FDTD solvers: An architectural overview," in *Proc. 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM03)*, pp. 269–270, Apr. 2003.

[23] 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. the 10th ACM International Symposium on Field-Programmable Gate Arrays*, pp. 97–105, 2002.

[24] J. P. Durbano, J. R. Humphrey, F. E. Ortiz, P. F. Curt, D. W. Prather, and M. S. Mirotznik, "Hardware acceleration of the 3D finite-difference time-domain method," in *Proc. IEEE AP-S International Symposium and USNC/URSI National Radio Science Meeting*, pp. 77–80, Jun. 2004.

[25] O. Kilic, M. S. Mirotznik, and J. P. Durbano, "Application of FPGA based FDTD simulators to Rotman lenses," in *Proc. 22nd ACES Conference*, 2006.

[26] O. Kilic, "FPGA accelerated phased array design using the ant colony optimization," *to appear in ACES Journal*.

[27] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents,"

IEEE Trans. Syst., Man, Cybern. B, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[28] T. Hiroyasu, M. Miki, Y. Ono, and Y. Minami, "Ant colony for continuous functions," *The Science and Engineering Review of Doshisha University*, 2000.

[29] M. G. Moharam, D. A. Pommet, E. B. Grann, and T. K. Gaylord, "Stable implementation of the rigourous coupled-wave analysis for surface relief gratings: enhanced transmittance matrix approach," *Journal of the Optical Society of America A*, vol. 12, no. 5, pp. 1077–1086, 1995.

[30] P. Lalanne, "Improved formulation of the coupled-wave method for two-dimensional gratings," *Journal of the Optical Society of America A*, vol. 14, no. 7, pp. 1592–1598, 1997.

[31] M. G. Moharam and T. K. Gaylord, "Rigorous coupled-wave analysis of planar-grating diffraction," *Journal of the Optical Society of America*, vol. 71, no. 7, pp. 811–818, Jul. 1981.

[32] J. M. Jarem, "Rigorous coupled wave analysis of radially and azimuthally-inhomogeneous,elliptical, cylindrical systems," *Progress In Electromagnetics Research,* PIER 34, pp. 181–237, 2001.

[33] W. Lee and F. L. Degertekin, "Rigorous coupled-wave analysis of multilayered grating structures," *Journal of Lightwave Technology*, vol. 22, no. 10, pp. 2359–2363, Oct. 2004.

[34] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA: Society for Industrical and Applied Mathematics (siam), 1997.

[35] J. G. F. Francis, "The QR transformation, I," *The Computer Journal*, vol. 4, no. 3, pp. 265–271, 1961.

[36] ——, "The QR transformation, II," *The Computer Journal*, vol. 4, no. 4, pp. 332–345, 1962.

[37] V. N. Kublanovskaya, "On some algorithms for the solution of the complete eigenvalue problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 1, no. 3, pp. 637–657, 1963.

[38] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the ACM*, vol. 5, no. 4, pp. 339–342, Oct. 1958.

[39] G. H. Golub and C. F. V. Loan, *Matrix Computations (3rd edition)*. Baltimore, MD: The John Hopkins University Press, 1996.

Ozlem Kilic graduated from The George Washington University (1996) with a D.Sc. degree in Electrical Engineering. She is presently an Assistant Professor in the Department of Electrical Engineering and Computer Science at The Catholic University of America. Before joining CUA, she worked at the U.S. Army Research Laboratories, Adelphi, MD and COMSAT Laboratories, Clarksburg, MD. Her research areas include computational electromagnetics, hardware accelerated programming for scientific computing, antennas and propagation, and radiation and scattering problems from random media.

Miaoqing Huang is an Assistant Professor in the Department of Computer Science and Computer Engineering at University of Arkansas. His research interests include reconfigurable computing, high-performance computing architectures, cryptography, computer arithmetic, and cache design in Solid-State Drives. Huang received a B.S. degree in electronics and information systems from Fudan University, China in 1998, and a Ph.D. degree in computer engineering from The George Washington University in 2009, respectively. He is a member of IEEE.