FPGA Accelerated Phased Array Design Using the Ant Colony Optimization

Ozlem Kilic
Department of Electrical Engineering and Computer Science
The Catholic University of America, Washington, DC, USA
kilic@cua.edu

Abstract **–** The objective of this paper is to investigate the utilization of field programmable gate arrays (FPGA) in the field of electromagnetics by applying the ant colony optimization (ACO) method in the design of phased array antennas for multiple beam satellite communication systems. The amplitudes of the array elements are optimized to reduce the co-channel interference in a multiple beam satellite communication system. The potential gains in the speed of the calculations are investigated in comparison to conventional simulation techniques of the same application on a regular PC. Two different FPGA platforms and implementation approaches are compared for performance to two software developments implemented using Matlab and C languages. It has been shown that significantly accelerated performance can be achieved for the particular application. This kind of speed improvement can enable handling more complex requirements and constraints for the same application in a very reasonable amount of time, which would otherwise be impossible with conventional computational platforms and techniques. This magnitude of speed improvement is due to the configurable nature of the FPGAs. Unlike central processing units (CPU) in a conventional computer, which have to deal with a preset set of instructions to properly function; FPGAs are completely programmable to carry out a set of functions in the most efficient manner for the particular algorithm at hand. In this study, the FPGA has been configured to function as an efficient "ACO machine." Both parallelization and pipelining have been utilized to achieve this performance. The details of the implementation on the FPGA platform and the achieved acceleration are discussed in the paper.

Index Terms **-** FPGA, parallel computing, reconfigurable programming, HPC, ant colony optimization, phased arrays, satellite communications, interference.

I. INTRODUCTION

The need for faster computations in the electromagnetics community has been a bottleneck for some of the modern applications such as smart antennas, advanced rf materials, etc. These devices utilize complex structures and demand ambitious performance within their operational environment. It is often necessary to simulate the performance of components and platforms as a single system in the design stage. As a result, accurate and fast modeling of large scale structures with fine features often becomes a challenge. Conventional full wave simulation techniques typically are not capable of solving such problems due to limitations in computational resources. Often, researchers resort to asymptotic or hybrid techniques in order to obtain a "reasonably accurate" solution.

The challenge becomes even bigger when the performance of these complex designs needs to be optimized over a set of constraints and parameters. Often the classical optimization techniques are not suitable because they typically require an initial estimate reasonably close to the final result in order to avoid stagnation at a local optimum point. They also tend to require analytical calculations such as derivatives that take computational time. Recently nature based heuristic optimization methods have gained attention in the electromagnetics community due to their robust random search mechanisms which have long been utilized for survival by different species.

Furthermore, these algorithms are inherently parallel in nature, which allows for accelerated computing.

The supercomputing systems, which can potentially handle numerically intensive problems, are not commonly available because of their cost. Hardware accelerated computing has been gaining momentum over the last decade due its applicability to parallel computing while using a fraction of the power requirements of the conventional microprocessors and requiring much less cost in comparison to supercomputers.

The objective of this paper is two folds: (i) investigate the use of field programmable gate arrays (FPGAs) in numerically intensive electromagnetic simulations, (ii) utilize the parallel nature of the ant colony optimization to accelerate the optimization of complex electromagnetic problems. Since FPGAs can be instantly reconfigured to carry out different tasks simultaneously, they offer a natural choice for this application.

II. RECONFIGURABLE COMPUTING WITH FPGAs

An FPGA is a type of programmable chip that can be configured to behave in just about any way the programmer wishes enabling them to be highly efficient platforms. Over the last decade, FPGAs have established themselves as the third programmable platform after microprocessors and digital signal processor (DSP) chips, [1]. While in the past the use of DSPs was ubiquitous, the utilization of FPGAs is growing rapidly due to the need for processing millions of instructions per second (MIPS). The primary reason FPGAs are preferred over DSPs is in fact driven by the application's MIPS requirement, [2]. Three factors have driven the interest on these devices: performance, cost and their reconfigurable nature. Their high performance relies on the parallel implementation that they naturally offer. This feature allows packing massive amounts of processing performance in a single package, eliminating the need to utilize different hardware components for different applications. Furthermore, the algorithms can be optimized over the reconfigurable hardware to avoid any overhead associated with the fixed instruction sets of microprocessors.

FPGAs are reprogrammable silicon chips in a two dimensional array of logic cells. A logic-cell is essentially made up of a small lookup table (LUT), a flip-flop and a 2-to-1 multiplexer, which can be used to bypass the flip-flop if necessary. Each logic-cell can be connected to other logic-cells through interconnect resources; i.e. wires placed around the logic-cells. Complex logic functions can be created by connecting hundreds or thousands of these logic cells together. In addition to these interconnect resources; FPGAs also have fast dedicated lines in between neighboring logic cells allowing the efficient creation of arithmetic functions. A schematic of the FPGAs is demonstrated in Figure 1.

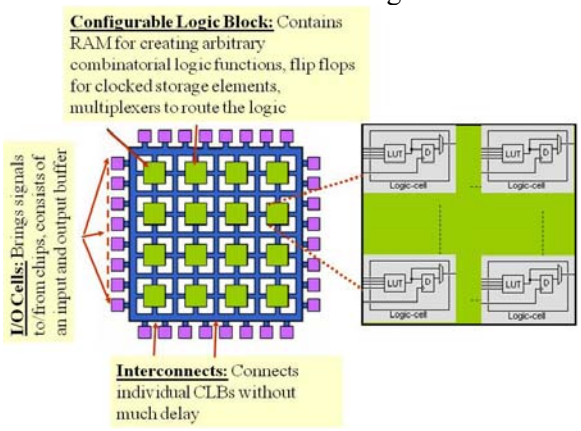

Fig. 1 Schematic diagram of FPGA components and functionality.

FPGAs can offer significant speed improvement compared to CPUs, [3] and have better price per performance ratio. They are still keeping up with Moore's law, roughly doubling their performance every 18 months. Furthermore, they have lower power consumption per computation, which makes them very attractive for very large problems. One of the main advantages of FPGAs is that they can dynamically create processing engines that fit the algorithm problem rather than fitting the algorithm to particular processor architecture. Despite all the advantages, FPGAs are finding their way to scientific computing rather slowly due to two challenges: (i) The programming on the chip requires significant hardware knowledge as well as understanding of the parallel nature of the algorithm to be implemented, (ii) FPGAs are best suited for

integer calculations and floating point calculations are often required in scientific applications. Nevertheless, researchers have been utilizing this platform for electromagnetic applications, [4] [5].

III. THE ANT COLONY OPTIMIZATION (ACO) ALGORITHM

The ACO algorithm mimics the behavior of ants in their search for the shortest path between their nest and the food. Although ants are nearly blind animals, they demonstrate the capability to establish the shortest path between their nest and food. The ethologists have discovered that ants deposit a chemical substance called pheromone on their paths, which is used by other ants in their search 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, [6]. The algorithm for this concept is demonstrated in Fig. 2. The ants serve as agents that search the optimization space for a satisfactory solution. The cost function is a measure of how satisfactory a solution is, with low cost implying a "better" solution. The description of the cost function is application dependent and is one of the most critical parts of the algorithm in terms of efficiency and accuracy. The random search is iteratively applied by the ants until one of the chosen paths satisfies the required convergence criteria.

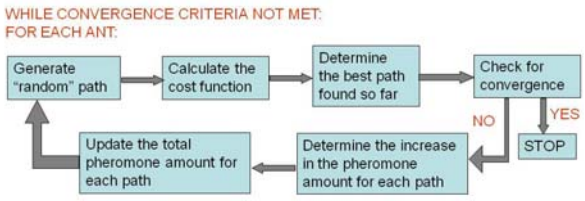

Fig. 2. ACO algorithm.

The ACO algorithm is based on selection of different paths, and therefore inherently applies to a discrete set of choices at each decision point. As a consequence, ACO is suitable for non-continuous optimization domains. However, for electromagnetics and antenna problems, the optimization domain usually consists of a continuous range of choices. Continuous problems have been solved for by modifications to the ACO algorithm, [7]-[10] This paper utilizes the Touring ACO by Hiroyasu, [7] where the solution is represented as a string of bits so that the path to decide is the bit values for this binary string, as shown in Fig. 3.

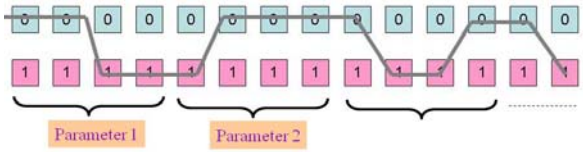

Fig. 3 The binary path for ACO in continuous domain.

The probability of a zero or one for each bit position is calculated from the total pheromone levels for the path for bit value of zero and one at each position as a function of the pheromone levels on the path as follows:

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

In equation (1), τ_0 denotes the total pheromone amount for bit value of zero at a given stage, and is computed from the sum of all pheromone amounts laid by all ants in the current iteration plus an "evaporated" amount of pheromone laid before in previous iterations, as shown in equation (2). The total pheromone amount for bit value of 1, τ_1 is calculated in a similar fashion. The coefficient ρ represents the evaporation parameter, and 1-ρ is the evaporation amount. This simulates nature's influence on the pheromone amounts that have been laid a while back. The increase in the total pheromone amounts due to each ant is inversely proportional to the cost function, $C_{k_{ant}}$. The incremental pheromone values $\Delta\tau_0^{k_{ant}}$ and $\Delta\tau_1^{k_{ant}}$ correspond to the bit value zero and one, respectively.

$$\tau_0(t+1) = \rho\tau_0(t) + \sum_{k_{ant}} \Delta\tau_0^{k_{ant}}(t, t+1) \tag{2}$$

$$\Delta\tau_0^{k_{ant}}(t, t+1) = \begin{cases} \dfrac{1}{C_{k_{ant}}} & \text{if } k_{ant} \text{ chooses 0 for the current bit} \\ 0 & \text{else} \end{cases}$$

IV. THE APPLICATION: OPTIMIZATION OF PHASED ARRAYS FOR MULTIPLE BEAM SATELLITE COMMUNICATIONS

In cellular satellite communications systems, a given coverage area is typically filled with a number of contiguous spot beams, which carry concentrated radiation along preferred directions. Since large areas are served in satellite communications, many beams need to be generated by the satellite antenna. Due to limited available bandwidth, the same frequency bands are often reused in cells separated apart from each other to accommodate the traffic. The frequency reuse approach results in co-channel interference due to the energy leaking from beams operating at the same frequency into each other. The concept of frequency reuse for a multiple beam satellite communication system is demonstrated in Fig. 4, where the beams operating at the same frequency are denoted with the same color. A reuse factor of 7 is shown in this figure; i.e. the same frequency is repeated every seven beams in the coverage area. In such a configuration, there is a potential of energy leak into beams operating at the same frequency through the side lobes of the radiation for an intended beam. Such a design often relies on the spatial isolation of these co-channel beams for reduced interference. However, if there are a substantial number of them, the resultant noise can be detrimental to the operation of the system.

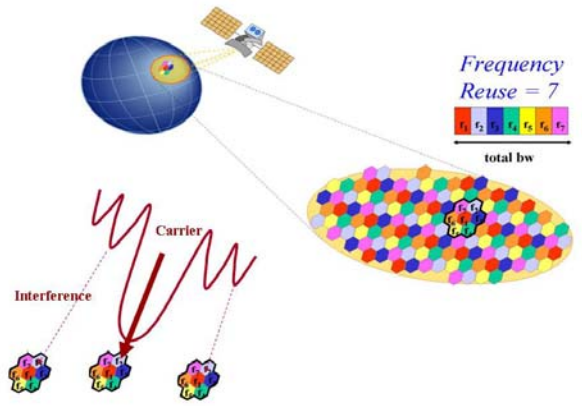

Fig. 4. Co-channel interference concept for multiple beam satellite communications systems.

This paper discusses the optimization of phased array antenna patterns to minimize this co-channel noise for multiple beam satellite systems. The noise the paper is concerned with is due to the interference from other beams in the system. The antenna pattern is manipulated by changing the amplitudes of the array elements so that the radiation along the direction of the co-channel beam centers is reduced below a threshold placing nulls in the antenna pattern along these directions. A linear array will be assumed for ease of computations with the understanding that the algorithm can be easily modified to adapt to a planar array. The only anticipated challenge in modifying the algorithm to planar arrays is the requirement to solve for a larger number of unknowns.

Numerous nature inspired optimization algorithms, including ACO, has been successfully applied to this problem before using conventional programming techniques on CPU, [11]. The optimization problem involves the computation of the array factor for a given array geometry and reducing the radiation levels along the co-channel beam directions. For a linear array with equally spaced elements, the array factor, $f(\theta)$ and the normalized radiation pattern, $U_n(\theta)$ can be calculated as given in [12] as follows:

$$f(\theta) = \left| \sum_{m=0}^{N-1} I_m e^{j\varphi_m} e^{+jmkd\cos\theta} \right| \tag{3}$$

$$U_n(\theta,\varphi) = \left|f_n(\theta)\right|^2 \left|e(\theta,\varphi)\right|^2 \tag{4}$$

where m is an index over element number, N is the total number of elements, I_m, φ_m are the amplitude and phase of the mth element, d is the center-to-center separation between elements, $k = 2\pi/\lambda$ is the wave number, and θ, φ are the observation angles with respect to the array axis. To further simplify the analysis, the individual elements are assumed to be isotropic sources; i.e. $e(\theta,\varphi) = 1$. With these assumptions, the normalized radiation pattern is the square of the array factor, and the optimization can be based solely on the array factor calculations. A uniform phase distribution is assumed, and the

optimization searches for a suitable set of I_m values to achieve the desired radiation performance.

V. FPGA IMPLEMENTATION OF ACO ALGORITHM FOR PHASED ARRAY OPTIMIZATION

Two target platforms were chosen for running the ACO on an FPGA: (i) Silicon Graphics (SGI) Altix 450 system, (ii) ML510 development board based on Xilinx Virtex-5 FX130T FPGA. The Altix system is configured with two Itanium processor based compute blades and a Reconfigurable Application-Specific Computing (RASC) blade. The RASC blade comprises of two Xilinx Virtex 4 XC4VLX200 FPGAs. The ML510 board is an embedded development platform with a 512 MB Compact Flash card and two 512 MB DDR2 DIMMs. Both platforms have a CPU connected to an FPGA, but using different I/O mechanisms in the hardware (NUMAlink and Core Services for SGI, APU for the ML510).

Mapping such an algorithm to an FPGA is different than programming a Van Neumann machine. Rather than a program counter controlling sequencing of instruction execution, data counters are used to control the streaming of data through a pipelined array of processing elements. Functions of the processing elements are fixed and data is passed from one processing element to the next, eliminating the need to move data in and out of memory as a shared processing resource steps through the processing sequence of the algorithm. One must also be aware of the resources available (i.e. internal registers, look-up tables RAM, multipliers and accumulators) when determining the way in which parallelism is achieved in an FPGA.

The ACO algorithm has multiple processing functions that are repeatedly performed as described earlier in the flowchart in Fig. 2. There are three major sections to the algorithm: Path Generation, Cost Calculation and Pheromone Update, which comprise a recursive pipeline. Additional logic monitors the process to determine when the algorithm has converged, and forwards the resulting data to the application running on the compute blade. Researchers have successfully implemented ACO on FPGA platform before, where in [13] a simplified form of ACO, namely P-ACO, was used to be able to fit the code on the Virtex-II Pro Platform utilizing FPGA XC2VP125. In [14] the authors utilize the FPGA as a coprocessor to the CPU, which carries out the controller evaluation functionalities.

This paper implements the ACO algorithm entirely on a single FPGA by using two different approaches. The first approach uses a highly efficient VHDL code on the SGI Altix platform. The second approach utilizes a software interface to the VHDL, ImpulseC, to implement the code in a C-like environment on the ML510 board. Details on the development for both approaches and a comparison of their performances are provided in the following sections.

V.1 VHDL Implementation on Altix 450

Path Generation updates the binary paths as discussed in Fig. 3. For this simulation, 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. These data are generated at the bit level. For each bit, the probability is maintained as to whether that bit is a one or a zero. The new path is generated based on these probability values. 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.

Streaming data from the Path Generation is fanned-out to parallel Multiply-Accumulators (MAC) in the Cost Calculation, as shown in Fig. 5. Separate MACs provide simultaneous updates to the cost function for each null, processing each parameter in every path. Note that the coefficients that describe the desired null pattern are stored in the FPGAs Block RAM. Four of these Block RAMs are used for each coefficient to provide the required 32-bit data width. After these MACs, the number of computations decreases to just the number of nulls. A multiplexer is employed to funnel data into divider that normalizes the data that has been accumulated. These numbers are accumulated to a single sum and a cost function is applied to generate the total cost for each ant path.

Based on this cost, the pheromone levels along each path and the probability of 0/1 at each bit position in the binary string is updated as shown in Fig. 6.

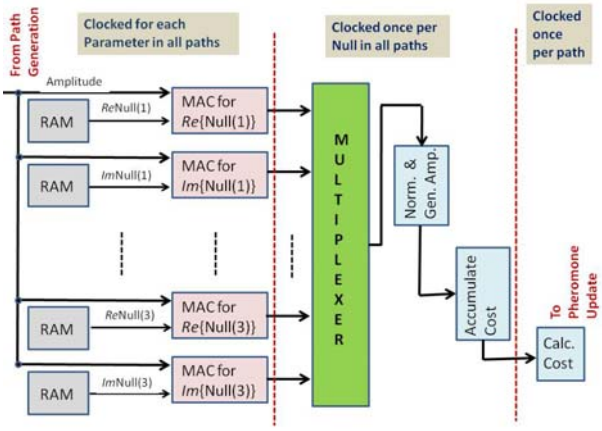

Fig. 5. Cost calculation block diagram.

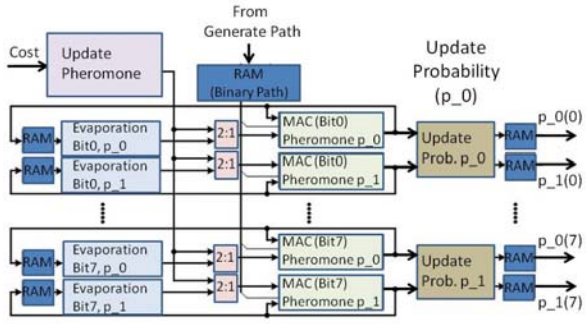

Fig. 6. Pheromone update block diagram.

The time required to process data with this FPGA implementation is shown in the timing diagram in Fig. **7**, where Clk is the 100 MHz system clock. The Reset signal starts the first iteration of the algorithm, while the Restart signal starts the subsequent iterations of the algorithm. A timing strobe denoted by pSOF is used to increment the AntCount, which keeps track of the path being processed. GPFirstAntOut and GPLastAntOut are timing strobes that mark the start of the first and last path outputs from Generate Path. The flow of probability data from Pheromone Update to Generate Path is controlled by the data counter pCount, and AmpSOF is a timing strobe that marks the beginning of data flow out of the Path Generation section.

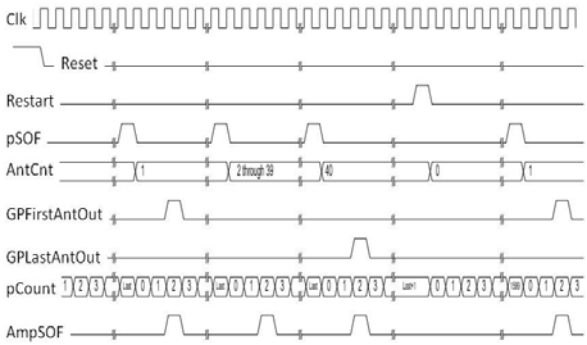

Fig. **7.** Timing diagram of ACO.

By running the FPGA at 100 MHz clock rate, the data is processed at a rate of 10 ns per clock. Parameters are processed on each clock cycle, with a one path delay at the beginning, due to amplitude normalization in Path Generation, and a one path delay at the end due to probability update in Pheromone Update. It should be noted that this path delay at the end is not shown in the timing diagram, but is enforced by the Restart, which is issued by Pheromone Update once the last path computations have completed.

For 40 parameters per path and 40 paths per iteration, the best expected run time per iteration is at about 16.8 μs (= 10 ns/ parameter * 40 parameters/path * 42 paths/iteration). It should be noted that 42 paths were used to account for the one path delay at the beginning, due to amplitude normalization in Path Generation, and a one path delay at the end due to probability update in Pheromone Update. For the planar array case (i.e. 40x40 array), run time is expected to be about 672 ms/iteration (= 10 ns/ parameter * (40*40) parameters/path * 42 paths/iteration).

V.2 Impulse C Implementation on ML510

SGI's Altix 450 platform is a highly efficient structure that integrates CPU with FGAs and utilizes shared memory to reduce any bottlenecks for data access. However, it is a highly sophisticated platform that requires expertise in programming on such platforms. Since expertise on such customized platforms are not common for the researched who is not in the field of FPGA computing, it was deemed of interest to implement the same algorithm on a more readily found FPGA card utilizing a software interface that helps simplify FPGA programming. The block diagram below shows the CPU-FPGA architecture for the

ML510 development board using Impulse C language as an interface. A C-like code is used with interpreted commands that translate into HDL implementation of the code by utilizing parallelism and pipelining. The development of the code utilizes the functions Impulse C provides for pipelining and parallelization, and the user can avoid working with the detailed timing diagrams as in the VHDL implementation. Impulse C generates the necessary timing for the specific FPGA platform it supports. Compiling the Impulse C code to the FPGA device involves two main steps: (i) Generating HDL and exporting from Impulse CoDeveloper, (ii) Synthesizing and mapping to the FPGA board using Xilinx ISE. Since the first step is automated by Impulse C, the user has limited control on the specifics of the HDL code generated in comparison to the first approach where the VHDL code was developed manually. The algorithm was compiled at 100 MHz as in the SGI Altix implementation and a bit file was created successfully. Therefore, the devices are running at the same speed for identical algorithms for a fair comparison of efficiency. The algorithm was run numerous times and an average run time of 0.3 milliseconds per iteration was observed. While the time for implementation of the code can be significantly reduced by using a software interface, a significant price in the run time efficiency is paid for as a result.

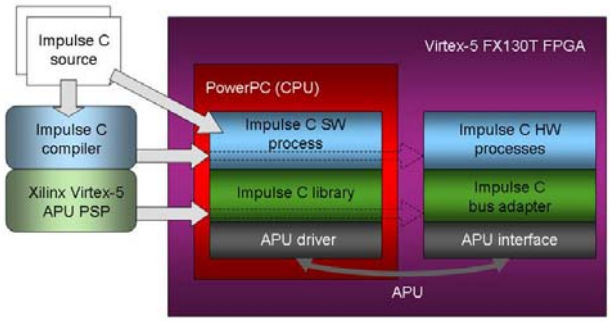

Fig. 8. CPU-FPGA architecture for the ML510 implementation.

VI. SIMULATION RESULTS – COMPARISONS OF FPGA, C AND MATLAB IMPLEMENTATIONS

As in all heuristic optimization methods, the performance of the algorithm depends on how the convergence criteria and cost function are defined. The cost function is defined such that a 25% drop in the peak gain is allowed while requiring the power levels along the direction of co-channel beams to be at least 40 dB down. For simulation purposes in this investigation, the centers of six co-channel beam locations were considered; at 3.75, 6.34 and 9.00 degrees off the main direction on either side. A linear array of 40 elements, with center-to-center element separation of half a wavelength was considered, and symmetry was employed; i.e. amplitude and phase values of the array elements were assumed symmetric with respect to the center of the array. The optimization space was sampled by 40 ants using eight bits per each optimization parameter. Due to the symmetry assumption, the number of unknowns is 20, half of the number of array elements. Therefore, the binary string generated by each ant is 160 (=20x8) bits long.

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 to the Matlab implementation. The same algorithm took 102.1 microseconds per iteration including the bit loading, resulting in a factor of 4,607 in speed compared to the Matlab implementation. It should be noted that the bit loading is only necessary when the algorithm is first run. Later runs do not need this process as the FPGA is already configured. The FPGA implementation using VHDL on Altix 450 system performed 100 times faster than the implementation on ML510 board using Impulse C language. The results of the algorithm are demonstrated for different convergence criteria (0.001, 0.07 and 0.20) in Fig. 9. The most strict case (err = 0.001) took on average 12 minutes to complete. The second case (err = 0.07) converged in about 0.4 minutes. Finally, the least strict case err = 0.20) took 0.01 minutes to complete. These times are based on average numbers for multiple runs of the same criteria.

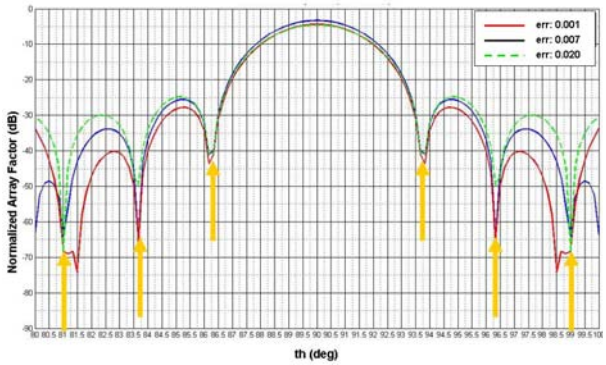

Fig. 9. Optimized antenna pattern for six co-channel beam centers - FPGA implementation on SGI Altix Platform.

VII. THE PROCESS FOR RUNNING THE ALGORITHM ON THE FPGA

There are significant differences between conventional software design flow and a hardware design flow for FPGAs, [15]. A multistage process is completed before a design can be used in an FPGA. These stages include synthesis, verification, translation, mapping, place and route. Synthesis stage is where the hardware description language code (e.g. VHDL, Verilog) is translated into a text description of a schematic. The verification step is to ensure that the specified design of the first step is functional. The translation means the conversion of this text description into a binary format. At this stage all the components and connections are mapped to the configurable logic blocks. The place and route stage is when the design is fitted onto the target FPGA. As a result of these stages a *.bit file, which is a configuration file to program the FPGA resources, is created to load the design onto the FPGA. Once all these stages are completed, the algorithm can be run repeatedly, without having to repeat these steps.

This multi-stage process in addition to the need to efficiently utilize available FPGA resources through pipelining and parallelism requires a steep learning curve for a scientist who is used to the conventional programming techniques. It is this aspect of the FPGAs that hinders the wide use of these platforms in the broader scientific community. Another key difference between FPGA implementation versus conventional programming is the compilation times. Software compilation is shorter than the hardware implementations and debugging can be done as an iterative approach. However, in the hardware approach the mapping of a defected design can cause significant delays in the place and route stage and should be avoided.

VIII. CONCLUSIONS

The utilization of FPGAs in the field of electromagnetics has been investigated by optimizing the radiation pattern of an array antenna using the ant colony optimization method. The acceleration performance in comparison with conventional programming techniques has been shown to be in the order of 15,000 for the particular application using a clock speed of 100 MHz. This order of magnitude of speed improvement can enable handling more complex requirements and constraints for the same application in a very reasonable amount of time, which would otherwise be impossible with conventional computational platforms and techniques.

This study demonstrates that FPGAs have tremendous potential for scientific computing. However, the problem investigated was small enough to be custom fit on a single FPGA, which enabled the high acceleration achieved. In more challenging electromagnetic problems, improvements at this magnitude may not be feasible. The most likely approach in such cases is utilizing the FPGA as a coprocessor to the CPU, which will reduce the acceleration factor. Furthermore, there are significant challenges to be overcome before the FPGAs can be considered as mainstream platforms for scientific computing. The overall acceleration for an application is highly dependent on the nature of the algorithm, required resources and what is available to the programmer, as well as programming skills. Interdependence and resource requirements of processes determine how the code can be parallelized. To optimally utilize the FPGA, the programmer needs to know the available resources and time required for each process. This is often a highly detailed process without access to mainstream products. The device at hand must

have the required resources for a given code. Successfully creating a bitfile for a given design to run at a given clock rate is not always possible for a given FPGA device. This is a fundamental limitation of FPGA development. Often the remedy is only implementing parts of the algorithm on FPGA, and running the rest on the CPU.

REFERENCES

[1] P. Lysaght; P. A. Subrahmanyam; "Guest Editors' Introduction: Advances in Configurable Computing," IEEE CS and IEEE CASS, pp. 85-89, March-April 2005Macdonald, V. H.

[2] "FPGA versus DSP, Design, Reliabilty and Maintenance," Altera White Paper 01023, www.altera.com/literature/wp/wp-01023.pdf

[3] Jahyun J. Koo, David Fern´andez, Ashraf Haddad and Warren J. Gross; "Evaluation of a High-Level-Language Methodology for High-Performance Reconfigurable Computers," Proc. IEEE Int. Conf. ASAP, pp. 30-35, July 2007.

[4] O. Kilic, M. S. Mirotznik, J. P. Durbano, "Application of FPGA Based FDTD Simulators to Rotman Lenses," Proc. 2006 ACES Conference, Miami, FL.

[5] C. He, W. Zhao, and M. Lu, "Time Domain Numerical Simulation for Transient Waves on Reconfigurable Coprocessor Platform," Proc. of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05), 2005.

[6] M. Dorigo, V. Maniezzo and A. Colorni, "The Ant System: Optimization by a Colony of Cooperating Agents," IEEE Trans. Systems, Man, and Cybernetics, Part B, Vol:26,No. 1, 1996, pp. 1-13

[7] T. Hiroyasu, M. Miki, Y. Ono and Y. Minami, "Ant Colony for Continuous Functions," *The Science and Engineering,* Vol. XX, No.Y, Doshisha University, Japan, 2000.

[8] D. Corne, M. Dorigo, & F. Glover. 1999. The ant colony optimization meta-heuristic. In *New ideas in optimization*, 11–32. M. Dorigo and G. D. Caro, eds. New York: McGraw-Hill.

[9] W. Lei, and W. Qudi. 2002. Further example study on ant system algorithm based continuous space optimization. *4th World Congress on Intelligent Control and Automation*, Shanghai, China, pp. 2541–2545.

[10] K. Socha, 2004 ACO for continuous and mixed-variable optimization. *Proc. of 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS'2004)*, Brussels, Belgium.

[11] O. Kilic, "Comparison of Nature Based Optimization Methods for Multi-beam Satellite Antennas," Proc. 2008 ACES Conference, Niagara Falls, Canada

[12] W. A. Stutzman and G. A. Thiele, "Antenna Theory and Design," Artech House, 1997.

[13] B. Scheuermann, K. Sob, M. Guntsch, M. Middendorf, O. Diessel, H. ElGindy and H. Schmeck, "FPGA implementation of population-based ant colony optimization", Applied Soft Computing, Vol. 4, Issue 3, August 2004, pp 303-322.

[14] Chia-Feng Juang, Chun-Ming Lu, Chiang Lo, and Chi-Yen Wang, "Ant Colony Optimization Algorithm for Fuzzy Controller Design and Its FPGA Implementation", IEEE Transaction on Industrial Electronics, Vol. 55, No. 3, March 2008, pp 1453-1462.

[15] R. Wain. I. Bush, M. Guest, M. Deegan, I. Kozin, C. Kitchen, "An overview of FPGAs and FPGA programming," http://www.cse.scitech.ac.uk/disco/publications/FPGA_overview.pdf

Ozlem Kilic graduated from The George Washington University (1996) with a D.Sc. in Electrical Engineering. She is presently a professor with the Catholic University of America. Before joining CUA, she worked at the U.S. Army Research Laboratories, Adlephi, 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.