Using Parallel Computing for Adaptive Beamforming Applications

Eman Ahmed[1], **K. R. Mahmoud**[2], **Safwat Hamad**[1], **and Z. T. Fayed**[1]

[1]Faculty of Computer and Information Sciences, Ain Shams University, Abbassia 11566, Cairo, Egypt
[2]Faculty of Engineering, Helwan University, Helwan, Egypt

Abstract— Recently, smart antenna systems have been widely considered to provide interference reduction and improve the capacity, data rates, and performance of wireless mobile communication. Smart antenna arrays with adaptive beamforming capability are very effective in the suppression of interference and multipath signals. The techniques of placing nulls in the antenna patterns to suppress interference and maximizing their gain in the direction of desired signal have received considerable attention in the past and are still of great interest using evolutionary algorithms such as genetic algorithms (GA) and particle swarm optimization (PSO) algorithm. In this paper, for adaptive arrays using space division multiple access (SDMA), the optimal radiation pattern design of smart antennas is developed based on the particle swarm optimization (PSO) technique. The PSO is applied to a 24-element uniform circular array (UCA) to calculate the complex excitations, amplitudes and phases of the adaptive array elements. The antenna elements consist of vertical (z-directed) half-wave dipole elements equally spaced in the x-y plane along a circular ring, where the distance between adjacent elements is $d_c = 0.5\lambda$. It is found that the resulting beampattern optimized by the PSO required a large processing time which is not acceptable for an on line applications. Hence, the demand for a parallel solution that accelerates these computations is considered. Therefore, a parallel version of PSO is proposed and implemented using Compute Unified Device Architecture (CUDA) then applied on a graphics processing unit (GPU). The comparison is presented to show how the parallel version of the PSO outperforms the sequential one, thus an online procedure is available for time-critical applications of the adaptive beamforming.

1. INTRODUCTION

Over the last decade, wireless technology has grown at a formidable rate, thereby creating new and improved services at lower costs. This has resulted in an increase in airtime usage and in the number of subscribers. The most practical solution to this problem is to use spatial processing. Andrew Viterbi, founder of Qualcomm Inc., clearly stated: "*Spatial processing remains as the most promising, if not the last frontier, in the evolution of multiple access systems*".

Spatial processing is the central idea of adaptive antennas or smart-antenna systems. Although it might seem that adaptive antennas have been recently discovered, they date back to World War II with the conventional Bartlett beamformer. It is only of today's advancement in powerful low-cost digital signal processors, general purpose processors (and ASICs — Application-Specific Integrated Circuits), as well as innovative software-based signal-processing techniques (algorithms), that smart antenna systems have received enormous interest worldwide. In fact, many overviews and tutorials have emerged, and a great deal of research is being done on the adaptive and direction-of-arrival (DOA) algorithms for smart-antenna systems. As the number of users and the demand for wireless services increases at an exponential rate, the need for wider coverage area and higher transmission quality rises. Smart-antenna systems provide a solution to this problem [3].

The increasing interest of researchers in using low cost GPUs for applications requiring intensive parallel computing is due to the ability of these devices to solve parallelizable problems much faster than traditional sequential processors. The first applications of evolutionary algorithms (EAs) on GPUs have been developed to solve specific Adaptive beamforming problems; This paper presents an approach for the implementation of PSO algorithms on GPUs which, using the NVIDIA CUDA environment in the adaptive beamforming applications [1]. For each running swarm a thread block is scheduled with a number of threads equal to the number of particles in the swarm The rest of the paper is structured as follows: Section 2 presents the problem formulation. Section 3 describes the experimental results. Finally, conclusions and remarks are presented in Section 4.

2. PROBLEM FORMULATION

The techniques of placing nulls in the antenna patterns to suppress interference and maximizing their gain in the direction of desired signal have received considerable attention in the past and are still of great interest using evolutionary algorithms such as genetic algorithms (GA) or the

sequential quadratic programming (SQP) algorithm. It is recognized that the PSO algorithm is a practical and powerful optimization tool for a variety of electromagnetic and antenna design problems. Compared with other evolutionary algorithms such as the GA and simulated annealing (SA), the PSO algorithm is much easier to understand and implement and requires minimum mathematical processing. In recent years, various versions of the PSO algorithm have been successfully used in linear and circular antenna array synthesis problems. Many of the attempts on antenna array synthesis assume that the elements of the array are represented by isotropic point sensors isolated from each other or the element pattern may be modeled by a cosine function. However, in practice, the elements of antenna arrays have finite physical dimensions and specific radiation characteristics [2].

The particle swarm is used in Adaptive Beamforming application as the following:

The particles are manipulated according to the following equations:

$$v_{id}^{k+1} = \omega^k v_{id}^k + c_1^k rand_1(pbest_{id} - x_{id}^k) + c_2^k rand_2(gbest_d - x_{id}^k) \quad (1)$$

$$x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1}\Delta t \quad (2)$$

where, w^k is the current inertia weight introduced to balance between the global and local search abilities c_1^k, c_2^k are the current acceleration constants, which represent the weighting of stochastic acceleration terms that pull each particle towards *pbest, gbest*, positions. $rand_1$, $rand_2$, are two random numbers in the range $[0, 1]$. At each iteration k, the current weighting value and acceleration rates can be calculated from the following equation:

$$\omega^k = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{iter'_{\max}} \times k, \quad c_1^k = c_1^{\max} - \frac{c_1^{\max} - c_1^{\min}}{iter_{\max}} \times k, \quad \text{and} \quad c_2^k = c_2^{\min} + \frac{c_2^{\max} - c_2^{\min}}{iter_{\max}} \times k$$

where, $w_{\max}$ is the initial weight (0.9), $w_{\min}$ is the final weight (0.4), $iter'_{\max}$ is 0.75 of the maximum iteration number ($iter_{\max}$). The value of the inertia weight is fixed to $w_{\min}$ after $iter'_{\max}$. $c_1^{\max}$ and $c_1^{\min}$ are the maximum and minimum values of c_1^k, $c_2^{\max}$ and $c_2^{\min}$ are the maximum and minimum values of c_2^k. An improved optimum solution was observed when changing c_1^k from 2.5 to 0.5, changing c_2^k from 0.5 to 2.5,

$$\text{objective function} = \sum_{i=1}^{N} a_i |E(\varphi_i)| - \sum_{j=1}^{M} b_j |E(\varphi_j)| \quad (3)$$

where E is the electric field intensity and the constants a_i and b_j are the weights that control the contribution from each term to the overall objective function. The constant N represents the number of desired users, and M represents the number of interferers. In our analysis, The weights a_i and b_j are considered are considered to be the same.

The implementation is done using cuda where two independent kernels (functions) were used. One kernel is for updating particles position and velocity and the other for calculating fitness for each particle position. Both kernels run for 40 particles in parallel (one thread for each particle) which succeeded in achieving an outstanding performance as stated in the result section.

The Proposed Algorithm (Frame Work) works as following: First we Initialize Particles with Random Positions (X), Velocity (V) Vectors, *Initial Values of* ω, c_1, c_2 (*On CPU*). Then one Kernel calculates fitness function (Array Factor) for each particle in parallel one thread for each particle (On GPU). Next, we Calculate the iteration best and global best metrics On CPU. The loop is then executed for max number of iterations.

3. EXPERIMENTAL RESULTS

Here we propose the first version of the adaptive beamforming application with PSO using CUDA and this is sample of the experimental results.

Experiments were run on a PC equipped with an Intel Core$^{(TM)}$2Duo processor running at 2.80 GHz with a NVIDA GeForce 9600 GT video card from NVIDIA corporation. All of the simulation runs were performed under the following settings:

Number of antenna in antenna array = 24

Number of Particle = 40

Number of Iterations = 150

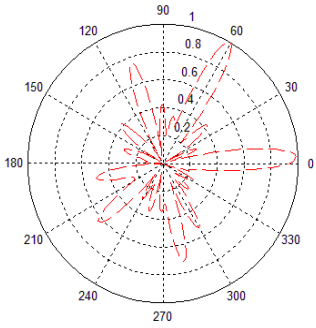
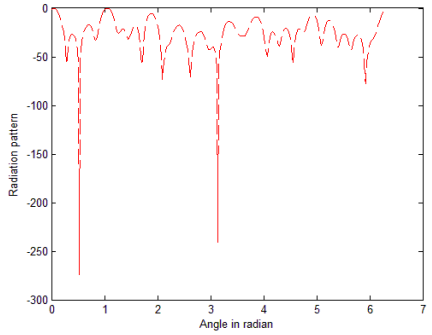
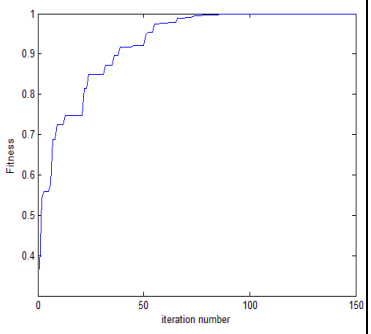
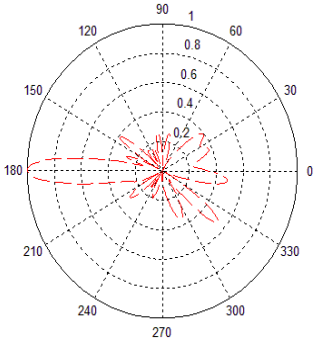
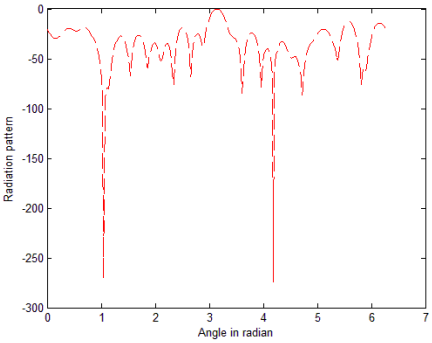
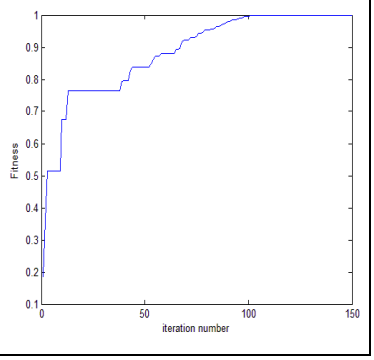
The sequential execution of the program took 4446 ms while running the proposed algorithm on GPU NVIDA (GeForce 9600) the execution time was only 265 ms. In particular the achieved running speedup was of about 16 times as illustrated in Table 1.

In Table 2, the first column displays images illustrating the optimum normalized radiation pattern resulted from the proposed PSO the second column shows figures that illustrate the optimum normalized radiation pattern measured in dB, and the last column illustrates the change of fitness value with iterations. The results were recorded for five different test cases.

Table 1: Sample runs and the average speed up.

Desired signal direction	**Undesired signal direction**	**Elapsed CPU time in (msecond)**	**Elapsed GPU time in (msecond)**	**Speed up rate**
0,90,180	30	4446	265	16.7774
0,60	180,30	4446	265	16.7774
180,60	240,30	4491.985	265	16.9509
180	60,240	4446	280	15.8786
180	300,120,30	4524	280	16.1571
			Average	= 16.50828

Table 2: Experimental results.

	(a) Radiation Pattern	**(b) y-axis is Radiation Pattern in dB and x-axis is angle in radian**	**(c) y-axis is fitness value and x-axis is iteration number**
Desired: 0, 60 Undesired :180,30			
Desired: 180 Undesired 60, 240			

4. CONCLUSIONS

In this paper, we have developed a parallel Adaptive Beamforming using PSO on GPU. The experimental results showed that the proposed algorithm is more efficient compared to a well coded serial version in terms of execution time. The possibility to efficiently run many swarms at the same time is another advantage that permits to improve the optimization success rate. Furthermore, the proposed design is modular and versatile: evolutionary steps are split into independent kernels, which makes it easy to adapt them to different variants of the PSO algorithm. For example, the neighborhood topology could be changed by changing the bests Update kernel. More simply, our

CUDAPSO can be adapted to the optimization of other goals by changing the fitness evaluation kernel. The performance still can be improved by using shared memory instead of global memory.

REFERENCES

1. Mussi, L. and S. Cagnoni, "Particle swarm optimization within the CUDA architecture," Viale G. Usberti 181a, I-43124 Parma, Italy.
2. Mahmoud, K. R., M. El-Adawy, S. M. M. Ibrahem, R. Bansal, and S. H. Zainud-Deen, "A comparison between circular and hexagonal array geometries for smart antenna systems using particle swarm optimization algorithm," *Progress In Electromagnetics Research*, PIER 72, 75–90, 2007.
3. Balanis, C. A., *Antenna Theory Analysis and Design*, John Wiley & Sons, Inc., Publication, Hoboken, New Jersey, 2005.