

# CFO Parallel Implementation on GPU for Adaptive Beam-forming Applications

Eman Ahmed

Faculty of Computer  
and Information  
Sciences, Ain Shams  
University  
Abbassia 11566,  
Cairo, Egypt

K. R. Mahmoud

Faculty of  
Engineering, Helwan  
University  
Helwan,  
Egypt

Safwat Hamad

Faculty of Computer  
and Information  
Sciences, Ain Shams  
University  
Abbassia 11566,  
Cairo, Egypt

Z. T. Fayed

Faculty of Computer  
and Information  
Sciences, Ain Shams  
University  
Abbassia 11566,  
Cairo, Egypt

## ABSTRACT

The scientific community is still interested in heuristic techniques and optimization algorithms that could be applied in complex problems such as the antenna adaptive beam forming problem. This paper presents an empirical study of solving the problem of antenna adaptive beam forming using Central Force Optimization (CFO) algorithm. The algorithm implemented using Compute Unified Device Architecture (CUDA) then applied on a graphics processing unit (GPU). CFO is well known alternatives for global optimization based on a nature-inspired Heuristic. Extensive experimentations were applied to compare their performance through a number of case studies. CFO has a higher computational complexity but it gives good results. The experimentations showed that the resulting beam-pattern optimized by the CFO 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 CFO is proposed and implemented using (CUDA) then applied on a (GPU). The comparison is presented to show how the parallel version of the CFO outperforms the sequential one, thus an online procedure is available for time-critical applications of the adaptive beam-forming.

## Keywords

CFO, global optimization algorithm, evolutionary algorithm, CUDA, GPU.

## 1. INTRODUCTION

We ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download the template, and replace the content with your own material.

Modern optimization techniques are able to solve problems with a non-linear and non-convex dependence of design parameters. So, it has provoked great interest among the scientific and technical community in a wide variety of fields recently. Some of these algorithms have been used successfully in many electromagnetism and antenna problems. Recently there are many electromagnetism and antenna optimization techniques are used such as the new nature inspired algorithm: Central Force Optimization (CFO). CFO algorithm takes much time which lead us to use parallel computing to make it take less time. Recently researchers have great interest in using low cost GPUs for applications that require intensive parallel computing due to the ability of these devices to solve parallelizable problems much faster than traditional sequential processors.

In the work presented here, the problem of antenna adaptive beam forming is solved using sequential and parallel implementations of the CFO algorithm. This paper presents an approach for the implementation of CFO algorithm on GPU using the NVIDIA CUDA environment in the adaptive beam-forming applications. Furthermore, a comparative result is included to evaluate the performance of CFO algorithm using a set of case studies. The rest of the paper is structured as follows: Section 2 presents the problem formulation. In Section 3, CFO algorithm and some modification on it are presented. At the end of this section, the proposed modification in CFO is presented with its parallel implementation. Finally, section 4 outlines the conclusions.

## 2. PROBLEM FORMULATION

Adaptive antennas refer to a group of antenna technologies that increase the system capacity by reducing the co-channel interference and increase the quality by reducing the fading effects. A smart antenna array containing  $m$  identical elements can steer a directional beam to maximize the signal from desired users, signals of interest (soi), while nullifying the signals from other directions, signals not of interest.

Different techniques of placing nulls in the antenna patterns to suppress interference and maximizing their gain in the direction of desired signal and minimizing their gain in the direction of undesired signal have received considerable attention in the past and still have great interests recently such as Genetic Algorithm (GA) . In addition, various versions of CFO algorithm have been successfully used in linear and circular antenna array synthesis problems [2, 6].

Antennas may be arranged in a (line, circle, plane, etc.) to yield a different radiation pattern. In this work, the complex excitations, amplitudes and phases of the adaptive antenna array elements are calculated for a given 24-antenna elements uniform circular array (UCA). 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  $d_c = 0.5 \lambda$  where  $\lambda$  is the wavelength.

Now, the radiation pattern of the antenna array could be computed according to the *pattern multiplication theorem* as follows: **Array Pattern = Array Element Pattern x Array Factor (AF)**

Where the Array element pattern is the pattern of the individual array element and Array factor is a function dependent only on the geometry of the array and the excitation (amplitude, phase) of the elements. The array factor AF is independent of the antenna type assuming all of the

antenna elements are identical. Assuming that the elements of the array are uniformly-spaced with a separation distance  $d$ ; then the equation of the array factor that need to be maximized in specific direction using Evolutionary algorithms will be

$$AF(\theta, \varphi) = \sum_{n=1}^N I_n e^{j[\beta * r * \sin(\theta) * \cos(\varphi - \text{pos}_n) - \alpha_n]} \quad (1)$$

Where  $I_n$  is the current of array element  $n$ ,  $\alpha_n$  is the phase of array element  $n$ ,  $\text{pos}_n$  is the position of array element  $n$  in the circular array,  $r$  is the radius of the circular array,  $\theta$  was fixed to 90 degree and  $\beta$  is the phase shift which is here equal  $2\pi$ ,  $AF(\theta, \varphi)$  (Array factor for a uniformly-spaced  $N$ -element circular array) in  $\varphi$  direction.

### 3. CENTRAL FORCE OPTIMIZATION (CFO)

First, a brief description of classical CFO algorithm, introduced by R. A. Formato, will be illustrated as well as some modifications that are done on it. These modifications will be analysed and finally the improved CFO algorithm is proposed with its sequential and parallel implementation.

#### 3.1 Classical CFO Algorithm

Central Force Optimization (CFO) is a nature-inspired gravity-based meta-heuristic for a multidimensional search [4]. CFO is an optimization evolutionary algorithm (EA) that locates the extreme of an objective function. This objective function is defined on a decision space (DS) of unknown topology that is searched by the EA. Then the value of the objective function to be maximized is computed step-by-step at each particle's location. After that, it is taken as an input to a user-defined function that becomes CFO's mass.

CFO finds the maxima of an objective function  $f(x_1, \dots, x_{Nd})$  by flying a set of particles through the decision space (DS). In an  $N_d$  dimension, each particle  $p$  with position vector  $\vec{R}_{j-1}^p \in R^{N_d}$  experiences an acceleration  $\vec{A}_{j-1}^p$  at the discrete time step  $(j-1)$  given by

$$\vec{A}_{j-1}^p = G \sum_{\substack{k=1 \\ k \neq p}}^{N_p} U(M_{j-1}^k - M_{j-1}^p) (M_{j-1}^k - M_{j-1}^p)^\alpha \frac{(\vec{R}_{j-1}^k - \vec{R}_{j-1}^p)}{\|\vec{R}_{j-1}^k - \vec{R}_{j-1}^p\|^\beta} \quad (2)$$

where  $N_p$  is the total number of particles, the particle number  $p = 1, 2, \dots, N_p$ , the time step  $j = 0, 1, \dots, N_t$ ,  $N_t$  is the total number of iterations,  $G$  is the gravitational constant,  $\vec{R}_{j-1}^p$  is the position vector of particle  $p$  at step  $j-1$ ,  $M_{j-1}^p = f(\vec{R}_{j-1}^p)$  is the fitness value at particle  $p$  and time step  $j-1$ ,  $U(\cdot)$  is the Unit Step function and finally  $\beta$  and  $\alpha$  are the CFO exponents [5]. CFO mass is defined as the difference of fitness raised to the power  $\alpha$  multiplied by the Unit Step function. In addition, the Unit Step  $U(\cdot)$  is essential because it creates positive mass. Thus it insures that CFO's gravity is attractive. CFO starts with a user-specified initial particles positions and acceleration distributions. The initial acceleration vectors are usually set to zero. Then each particle's position vector at step  $j$  is updated according to the following equation:

$$\vec{R}_j^p = \vec{R}_{j-1}^p + \frac{1}{2} \vec{A}_{j-1}^p \Delta t^2, j \geq 1 \quad (3)$$

Where  $\Delta t$  is the increment in the time step. Particles may fly outside the decision space and should be retrieved to it. There are many possible particle retrieval methods. A useful one is the reposition factor  $F_{rep}$  ( $0 \leq F_{rep} \leq 1$ ) which plays an important role in CFO's convergence. It is shown in Figure 1 that  $F_{rep}$  is usually set to 0.5 or 0.9, or it may be variable [5]. Where  $R_k^{min}$  and  $R_k^{max}$  are the minimum and maximum

values of the  $k^{th}$  spatial dimension corresponding to the optimization problem constraints.

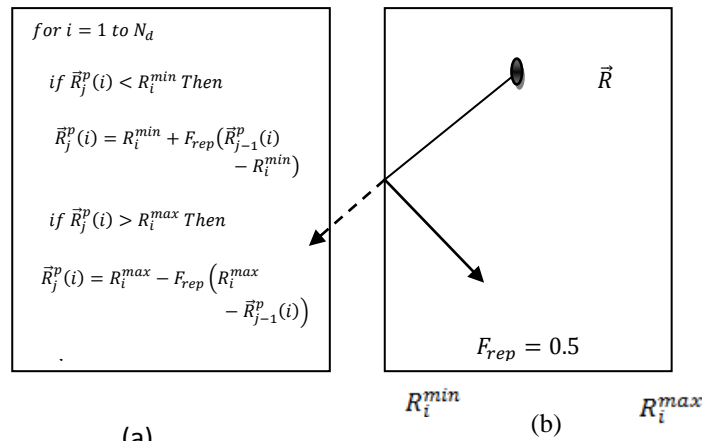


Fig 1: (a) Errant particle reposition factor retrieval.

(b) Illustration of particle repositioning in 2-D DS.

#### 3.2 Modifications Done on CFO Algorithm

CFO contains several user-specified parameters. The most important parameters (determined empirically) are the initial particle distribution (total number of particles  $N_p$  and their deployment in the DS) and the  $F_{rep}$ . The initial particle distribution determines how the decision space topology is sampled at the beginning of a run, while  $F_{rep}$  is important in avoiding local trapping. Here the modifications done on those important parameters and some analysis on it will be presented.

The first modification is Gamma Modification (Variable Initial Particle Distribution). Formato R. A presents this modification in details in [4].

In Brief this modification uses the following equation:

$$\vec{D} = \vec{X}_{min} + \gamma(\vec{X}_{max} - \vec{X}_{min}) \quad (4)$$

This equation is used to initialize particles on axes parallel to the original axes whose intersection point is marked by position vector  $\vec{D}$  where  $\vec{X}_{min}$  and  $\vec{X}_{max}$  are the diagonal's endpoint vectors of the original axes. Parameter  $0 \leq \gamma \leq 1$  determines where along the diagonal the orthogonal particle array is placed [4].

So this modification is to use a parameter called gamma ( $\gamma$ ) in initializing the particles using empirical values from 0 to 1 for the gamma parameter and then observe the best value for gamma that gives best particle distribution which gets the best results. After analyzing this modification, the experimental results showed that however using the best value for gamma (the best particle distribution) for the problem gives excellent results, this modification is not practical. Because it was found that there is a need to try all values of gamma with each problem to know the best value for gamma for this problem. This means that there is no standard or general value for gamma as it depends on the problem fitness function. So it cannot be considered as a modification to the algorithm, instead it can be considered as a way of particle initialization.

The second modification is Shrink Decision Space (DS). Formato R. A also presents this modification in details in [4]. The modification is to shrink the decision space (DS) around the best particle position as the iteration number increases. DS

size is adaptively reduced every 20th step around the particle's location that have the best fitness,  $\vec{R}_{best}$ . DS's boundary coordinates are reduced by one-half the distance from the best particle's position  $\vec{R}_{best}$  to the boundary of the DS on coordinate-by-coordinate basis. Thus,

$$x_i^{\min} = x_i^{\min} + \frac{\vec{R}_{best} \cdot \hat{e}_i - x_i^{\min}}{2} \quad (5)$$

And

$$x_i^{\max} = x_i^{\max} + \frac{x_i^{\max} - \vec{R}_{best} \cdot \hat{e}_i}{2} \quad (6)$$

are the equations used in shrinking where the primed coordinate is the new decision space boundary and the dot denotes vector inner product. After analyzing this modification the experimental results showed that this modification is very effective with all types of problems (unimodal /multimodal) (low Dimension/ high Dimension). This shrinking needs to have some sort of reactive adaptation as will be explained in the proposed CFO.

The third modification is Acceleration Clipping. Acceleration clipping is a new modification introduced in [5] to damp the particles' motion and to prevent particles from flying out of the decision space. The acceleration clipping scheme is introduced to limit the maximum acceleration of the particles to refine the particles motions and decreases the number of outside flying particles. After analyzing this modification, the experimental results showed that this modification was not effective enough so it does not included in the proposed CFO.

The fourth modification and last modification is Particle Initialization (Random, Uniform on Axis, Uniform Grid). This modification presented in [6] in details. It presents three ways in particle initialization: initialize them in uniform grid, initialize them uniform on axis, or use random initialization. After analyzing this modification and trying the three different ways to initialize particles in CFO; the experimental results showed that it depends on the problem. Some problems get its best results when initialize its particles uniform on Axis and other when initialize them on Grid and other when initialize them random. But in most of the test functions and in the problem of adaptive beam-forming, the initialization of the particles uniform on Axis is much better than other initializations.

### 3.3 The Proposed Modification of CFO Algorithm

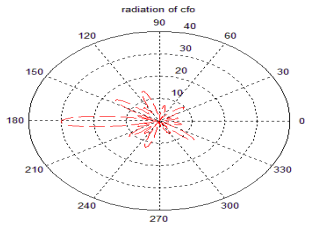
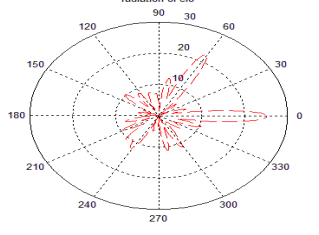
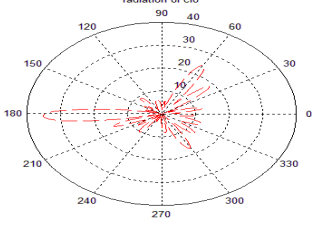
As explained above the modifications done on CFO; now will present what was taken from these modifications and the contributions done on it. One of the best modifications done was shrinking the Decision Space which means to limit the space you search in. The presented modification was to shrink DS every 20 iterations by half the distance between the best particle's position and the boundary of DS. To improve the results of this modification there is a need to make it based on performance measures such as convergence speed and fitness saturation. The results showed that as iterations increasing CFO converge more to the optimum so shrinking DS can be done every dynamic number of iterations and by dynamic ratio of the distance between the best particle's position and the boundary of the DS.

So first shrinking the DS every dynamic number of iterations will be introduced. This can be done by shrinking DS every 50 iterations and then every 45 iterations and so on until every

5 iterations. This means that after 50 iterations the first Shrinking to the Decision Space will be made. Then as CFO converge more to the optimum, the second shrink will be made after 45 iterations and then after 40 iterations and so on; minus 5 each time until reach to shrink after 5 iterations and continue shrinking after 5 iterations till the end. So the DS size is adaptively reduced every 50 to 5 steps around the particle's location that have the best fitness.

Second shrinking the DS by dynamic ratio of the distance between the best particle's position and the boundary of the DS will be introduced. This can be done as following: first shrink the DS by small value because CFO still not converge enough to the optimum and then increase the shrinking value as iterations increases. Thus Increase the Shrinking DS Ratio (shDSratio) as iterations increases. Thus DS's boundary coordinates are reduced by shDSratio multiplied by the distance from the best particle's position to the boundary of the DS on a coordinate-by-coordinate basis. Thus,  $x_i^{\min} = x_i^{\min} + \frac{\vec{R}_{best} \cdot \hat{e}_i - x_i^{\min}}{shDSratio}$  and  $x_i^{\max} = x_i^{\max} - \frac{x_i^{\max} - \vec{R}_{best} \cdot \hat{e}_i}{shDSratio}$ , are the equations that used in the shrinking where shDSratio changed linearly from 0.1 to 0.5. The proposed CFO algorithm is as the classical CFO but adds shrinking DS step as explained above and use On-Axis particle initialization as illustrated in the proposed CFO algorithm flow chart in figure 5. The sequential result of CFO is illustrated in table1 for three test cases in adaptive beam-forming application.

Table 1. CFO Sequential results

Pattern	CFO
Desired: 180 Undesired :60,240	
Desired:0,60 Undesired :180,30	
Desired: 180,60 Undesired :240,30	

### 3.4 Parallel CFO

After analyzing sequential CFO to see which step that take most time, it was found that update acceleration is the step that take the most time as shown in figure 2 as it takes 99.63% from the total time of the CFO algorithm.

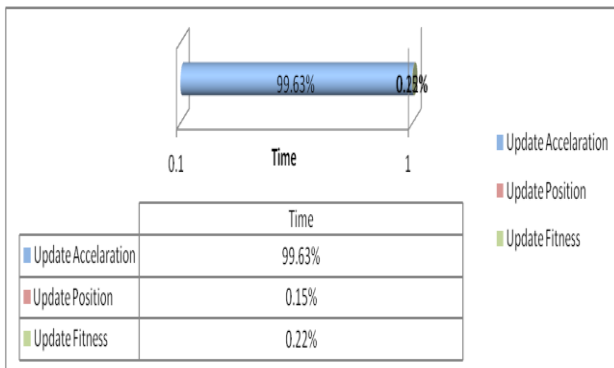


Fig 2: CFO time distribution

The update acceleration step is the main problem in implementing parallel CFO because the equation of update acceleration is dependent on last updated position and fitness for all particles.

The loop on particles that update acceleration, position and fitness cannot be parallelized because the update acceleration step for a specific particle is dependent on the calculated position and fitness of the previous particle in the loop. When trying to isolate update of acceleration for all particles then update position and fitness for all particles the results of the Adaptive beam-forming problem was affected badly as seen in experimental result of CUDA results section. The flow chart of parallel CFO implementation is illustrated in figure 6.

#### 3.4.1 Implementation of Parallel CFO On The GPU

As in PSO and in any EVs; CFO has two parallel variants one global and other local. Global: Where all the mathematical calculations are parallelized, computing fitness function, acceleration, and position for all particles in parallel using two different kernels on GPU. Local: Where the Whole entire algorithm executed on the Local Memory of GPU except the initialization of the particles which is executed on the CPU. The global one is chosen to be implemented due to the limited memory of the available GPU and to be able to use as many particles as needed without limitation of memory.

The sequential CFO algorithm was implemented as reference, in order to assess the performance of parallel variants as shown in table1 of CFO sequential results. In any parallel implementations, the programming strategy involved the creation of one thread for each CFO Particle. The rule was to replace all the sequential loops (specifically those where the iterations were in terms of the Particles number) by a single multithreading kernel call. The structure of the sequential CFO algorithm contains the following functional blocks [3]: - Population initialization which initializes each particle of the population on axis, Fitness function evaluation, and Update acceleration, and Update position.

The main idea is to create one thread for each CFO particle; Note that in the sequential CFO version all the functional modules are executed in one loop on the host processor.

And to be able to execute CFO parallel it was split to 3 independent loops (update fitness, update position, and update acceleration) which affects badly on the results as will be seen in the results section in addition that the used GPU is not support the double precision so it lacks for accuracy and if GPU that supports double precision is used this will affects on the time as it will take more time.

In the first parallel variant, the Global one, any arithmetic calculation is distributed to the GPU, replacing both the fitness function evaluation and update position and update acceleration modules by the associated kernel calls (see Figure 3) Use one thread for each particle in each kernel as there are three kernels are used: Kernel1 for (evaluate the fitness of all particles), Kernel2 for (update the position of all particles), Kernel3 for (update the acceleration of all particles).

```

Initialize CFO Parameters
Initialize Particles Position and acceleration
<perform a first evaluation of the fitness functions>      kernel1
For (i = 0; i < Number_of_Iterations; i++)
{
    <update the position of all particles>                 kernel2

    <re-evaluate the fitness of all particles>             kernel1

    update reposition factor                             CPU

    <update the Acceleration of all particles>            kernel3

    Shrink Decision Space                               CPU
}
retrieve global best information to be returned as final result
    
```

Fig 3: Pseudo Code of the Parallel CFO algorithm using CUDA

#### 3.4.2 CFO CUDA Results:

Here the first version of the adaptive beam forming application with CFO using CUDA is proposed and this is sample of the experimental results that show CPU time and GPU time of CFO for a three different test cases of adaptive beam-forming. Experiments were run on a PC equipped with an Intel Core (TM) 2Duo processor running at 2.80 GHz with a NVIDIA GeForce 9500GT 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 = 180, Number of Iterations = 250.

The sequential execution of the program took 1617729 ms while running the CFO algorithm on GPU NVIDIA (GeForce 9500) the execution time was only 298158 ms. In particular the achieved running speedup was of about 5.5 times as illustrated in figure 4. Some of selected results is shown in Table 2 where the first column displays images illustrating the optimum normalized radiation pattern resulted from the proposed CFO 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 three different test cases.

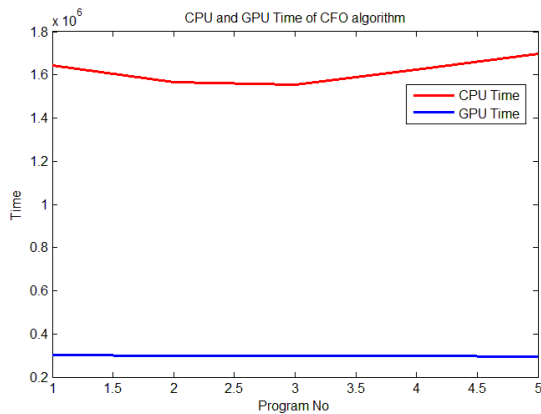


Fig 4: sample runs and average speed up =5.5 for CFO

#### 4. CONCLUSION

In this paper, a parallel version of CFO is proposed, implemented using (CUDA), and applied on a (GPU). Extensive experimentations showed that the parallel version of the CFO outperforms the sequential one, thus a real time adaptive beam-forming algorithm procedure can be used for time-critical applications.

Furthermore, a comparative study showed that; in the sequential mode, CFO algorithm produces more accurate results. The reason behind this is that the CFO relies on double precision computations which are not actually supported by the GPU used here.

#### 5. REFERENCES

- [1] Mohammad Shihab et. s.l. DESIGN OF NON-UNIFORM CIRCULAR ANTENNA ARRAYS USING PARTICLE SWARM OPTIMIZATION.: Journal of ELECTRICAL ENGINEERING, 2008, Vol. 59. 216-220.
- [2] BALANIS, C. A. Antenna Theory: Analysis and Design. New Jersey : JohnWiley & Sons, Inc., 2005.
- [3] Eman Ahmed, K. R. Mahmoud, Safwat Hamad, and Z. T. Fayed. Using Parallel Computing for Adaptive Beamforming Applications.. Cambridge, USA : PIERS Proceedings, July, 2010. 5-8.
- [4] Formato, R. A. IMPROVED CFO ALGORITHM FOR ANTENNA. USA : Progress In Electromagnetics Research B, 2010, Vol. 19.
- [5] Dib, G. M. Qubati and N. I. MICROSTRIP PATCH ANTENNA OPTIMIZATION USING MODIFIED CENTRAL FORCE OPTIMIZATION. Jordan : Progress In Electromagnetics Research B, 2010, Vol. 21.
- [6] Formato, Richard A. Central Force Optimization: A New Nature Inspired Computational Framework for Multidimensional Search and Optimization. USA : s.n.

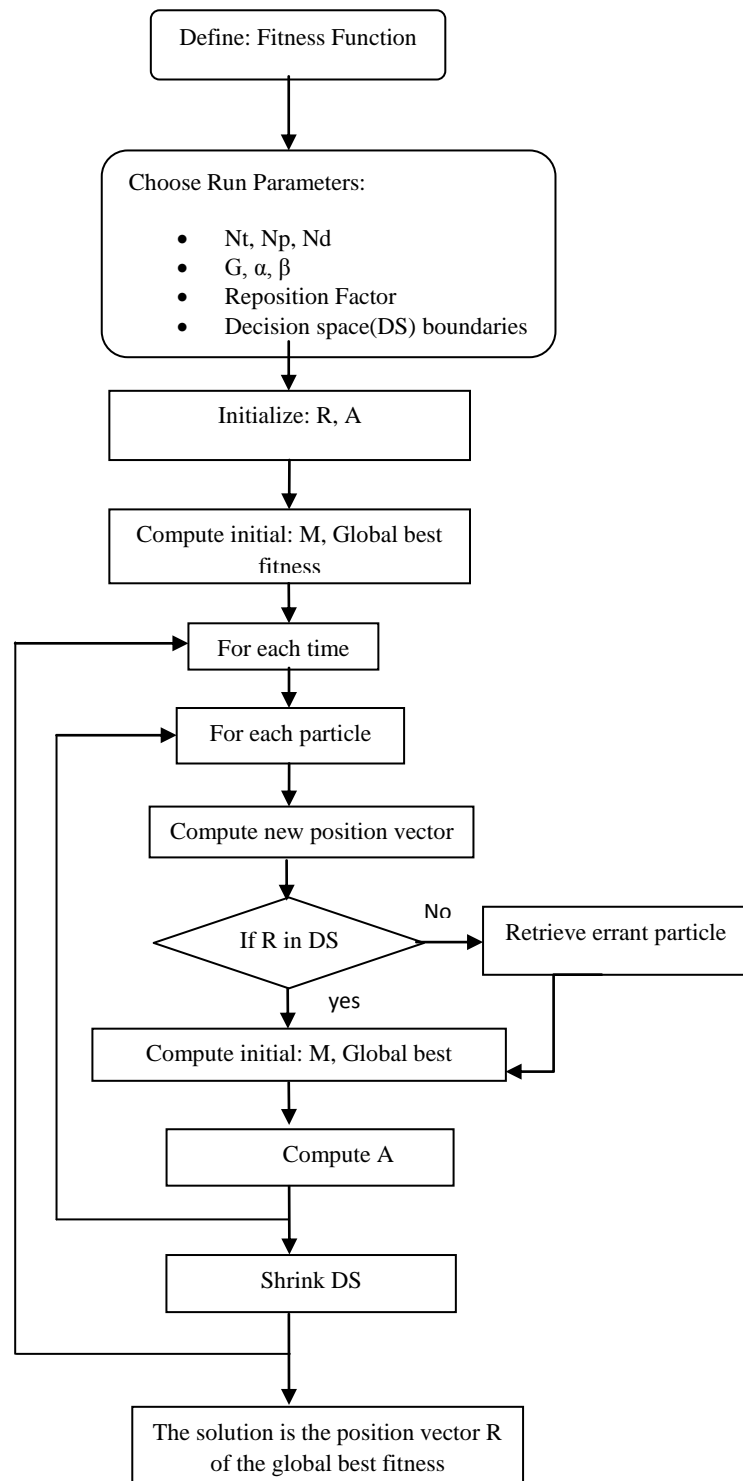


Fig 5: Flowchart of the main steps of the proposed CFO algorithm

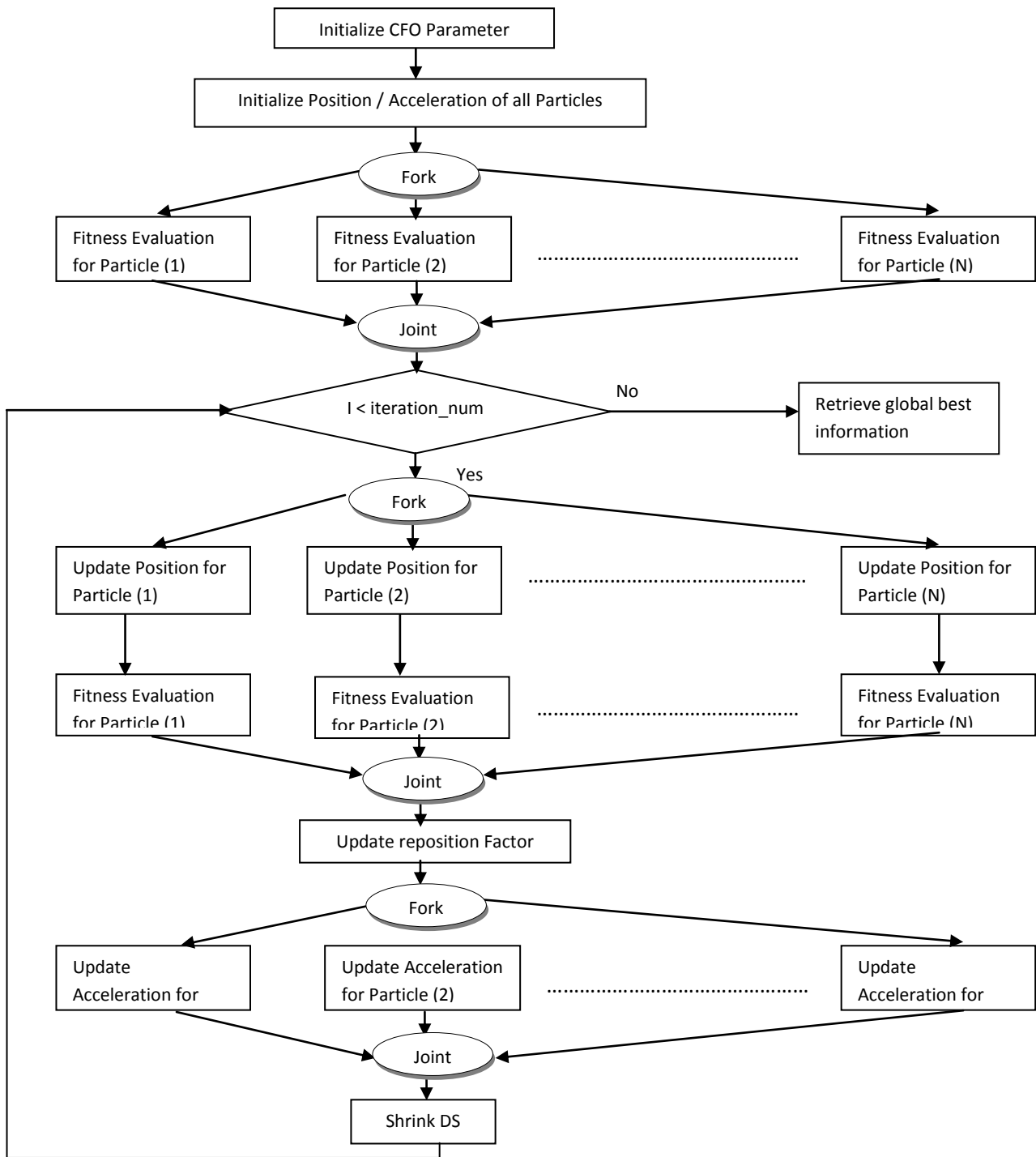
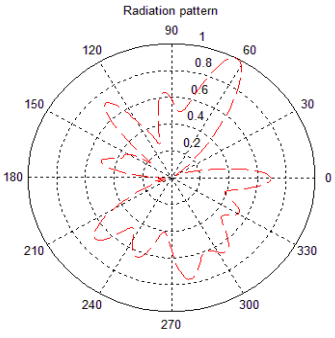
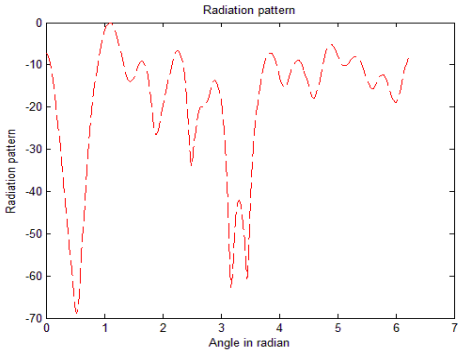
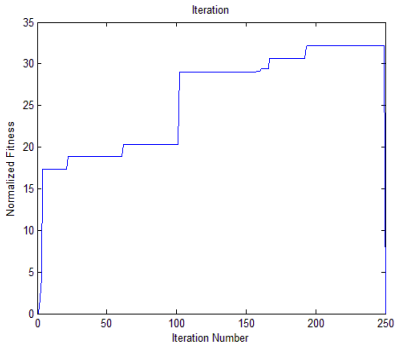
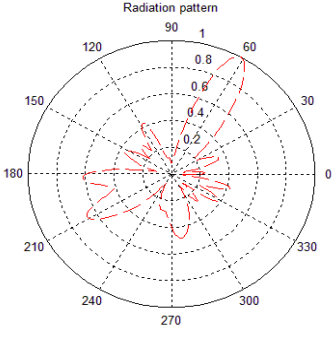
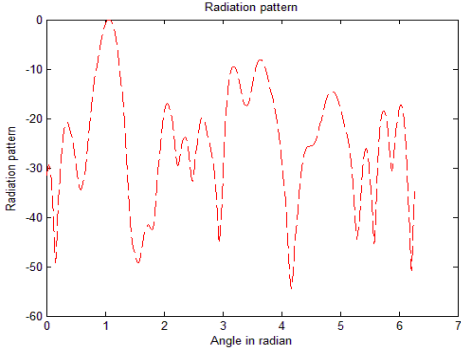
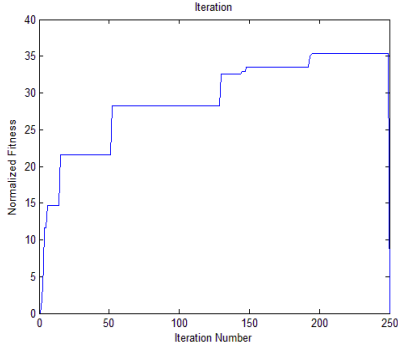


Fig 6: Flow Chart of the Parallel CFO algorithm

Table 2. Experimental results of CFO using CUDA

	(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, 60 Undesired: 240, 30			
Desired: 180 Undesired: : 60, 240	