Page 1:

Isotropic radiators may be utilized of the array factor to simplify the algebra.

In the far field of the array

R1 = R

RN = R – (N-1)d \* cos(theta(

Magnitudes

Smart Antennas

A collection or swarm of particles is distributed in the problem space, where each particle is assigned

**4.3 Parallel PSO**

**CUDA Implementation:**

There are two parallel variants using CUDA/GPU for PSO algorithm [5] or for any evolution algorithm one global and other local or as paper [2] called embedded.

Concerning the global approach, Global variant was programmed on the GPU:

**1‐ Global:** Where all the mathematical calculations are parallelized, computing all fitness function, velocity, and position which Executes Fitness Evaluation and Position and Velocity Update for all particles in parallel using two different kernels on GPU.

**2- Local:** Where the Whole entire algorithm executed on the Local Memory of GPU except the initialization of the particles.

We implement the global one due to our limited memory of our GPU and to be able to use as many particles as we need without limitation of memory.

The sequential PSO Algorithm was implemented as reference, to Compare the performance of parallel variants to it. In all parallel implementations, the programming strategy involved the creation of one thread for each PSO 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. Thus, the sequential PSO algorithm and its parallel implementations have essentially the same structure. In the loops of the sequential code, each loop’s iteration is independent from all others [5].

CUDA programs launch parallel kernels with the following function‐call syntax:

Kernel<<<dimGrid,dimBlock>>>(…parameter\_list…);

Where dimGrid and dimBlock are specialized parameters that specify the dimensions of the parallel processing grid of blocks and the dimensions of the blocks of threads, respectively. Figure 3 depicts the structure of the sequential PSO algorithm where the following functional blocks can be observed [3]:

‐ **Population initialization**. It initializes each particle of the population in a random form.

‐ **Fitness function evaluation**.

‐ **Comparison**. It determines if an individual has better fitness than the best registered.

**‐ Imitation (updating)**. Every individual updates its position influenced by its own experience, and by the social environment.

The main idea is to create one thread for each PSO particle, as we will see below. Note that in the sequential PSO version (see Figure 3) all the functional modules are executed on the host processor. In the first parallel variant, the Global one, any arithmetic calculation is distributed to the GPU, replacing both the fitness function evaluation and position update modules by the associated kernel calls (see Figure 4) Use one thread for each particle in each kernel as we use 2 kernels: Kernel1 for (update the position of all particles) and Kernel2 for (evaluate the fitness of all particles).

In the second parallel variant, the Local one, only the initialization module remains running on the host processor, and there are a single kernel call associated to the evaluation, comparison and imitation modules, that run on the GPU until a termination condition is reached.

There are some practical considerations that must be taken into account to achieve a functional implementation of the parallel PSO algorithm on a GPU as stated in [5]:

**Overhead**. The GPU presents an overhead due to memory transferences between the host and the GPU device which are necessary during the information exchange. Because these transferences are relatively slow, any parallel implementation on a GPU must minimize their employment. Considering the overhead, it is understandable that the global variant (Global) are slower than the Local one, due to the information exchange between the host and the GPU during the algorithm execution. In the global variants, the information exchange is necessary since the host processor needs to know the information originated at both the evaluation and update modules in order to take any decision [5].

**Synchronization**. Before any decision branch, for example during the comparison process, all the running threads must be synchronized at the points where it is necessary to obtain unambiguous information. This point is particularly important when the threads have to communicate among themselves to share information [5].

Pseudo code:

*<initialize positions/velocities of all particles>*

*<perform a first evaluation of the fitness functions> kernel2*

*<set initial personal/iteration/global bests>*

**for(i = 0; i < Iterations\_Number; i++)**

**{**

*<update the position of all particles> kernel1*

*<re−evaluate the fitness of all particles> kernel2*

**}**

*<retrieve global best information to be returned as final result>*

Fig 2: Pseudo Code of the Parallel PSO algorithm using CUDA