GPU-accelerated synthesis of echo generators

A. Capozzoli, C. Curcio, A. Liseno
Università di Napoli Federico II
Dipartimento di Ingegneria Biomedica, Elettronica e delle Telecomunicazioni
via Claudio 21, I 80125 Napoli (Italy)
Tel.: +39 081 7683358; Email: a.capozzoli@unina.it.

***Abstract*—We deal with an approach to the design of array-based, 2D echo generators.**
The radiating elements are located on non-uniform grids and the Quiet Zone (QZ) design specifications are enforced at non-uniformly spaced sampling locations. The radiating element positions and QZ design specification matching points are determined by a singular values optimization process.
In order to properly manage the computational burden, a proper parallel programming strategy is employed, in particular, by exploiting the potentialities recently offered by Graphics Processing Units (GPUs) through the Jacket Matlab toolbox.
Numerical results asses the performance of the technique in terms of computational requirements and QZ field behavior.

I. Introduction

Many applications, as antenna characterization [1] or the system testing of radars under their actual working conditions [2], require the generation of "canonical" waves in prescribed regions of space known as Quiet-Zones (QZs).
When the canonical wave of interest is a plane wave, Compact Antenna Test Ranges (CATRs) [3] represent a commonly employed solution, with well known drawbacks [3].
Most recently, Plane Wave Synthesizers (PWS'), i.e., arrays purposely designed to generate prescribed waves in their near-field regions, have been proposed [1], [4], [5]. Although exhibiting more hardware complexity, PWS' should permit to significantly improve the flexibility as compared to standard CATRs [7]. Furthermore, they are very helpful whenever the testing of large Objects Under Test (OUTs) is required, by avoiding likewise large QZs [1], [7]. Finally, by a similar concept, echo generators, i.e., array systems for which the canonical waves of interest is not only a plane wave but, more generally, radar echoes, can be realized [5], [6].
Due to the higher complexity as compared to CATRs, the design of a PWS or of an echo generator is not an easy task from the computational point of view.
Commercial CPUs have now reached a clock-speed growth standstill, as thermal dissipation prevents higher velocities due to the required increasingly higher transistor densities [8], [9]. This has led to a burst in the development of multi-core processors, capable of very high processing throughputs. Particularly, the use of off-the-shelf, Graphics Processing Units (GPUs) for general purpose processing has recently become increasingly popular, thanks to the virtual availability on all desktop computers and to the release of development tools to write algorithms for execution on GPUs.
Among such tools, the Compute Unified Device Architecture (CUDA), released by NVIDIA in 2007, provides an extension to ANSI C supported by several keywords and constructs [10], enabling an easier environment to program general-purpose applications onto the GPU. On the other side, AccelerEyes Jacket [11] is a commercial product being developed by AccelerEyes, allowing a standard Matlab code to be run on the GPU, so enabling the inexperienced programmer to benefit of GPU code acceleration, allowing to extend existing Matlab scripts to parallel processing or quickly writing new parallel ones.
In some applications, the ease in programming is traded-off in terms of a slightly slower overall execution time as compared to low level, e.g. CUDA, applications [12]. In other cases, it has been verified that Jacket and CUDA have comparable performance [13].
Following [5], [6], the objective of this paper is to deal with the synthesis of 2D echo generators and determine how much the algorithm can be improved by a very simple implementation on a GPU+CPU platform with Matlab and the Jacket toolbox.

II. The problem

Let us consider a time harmonic echo generator [5], [6] composed by an array of N radiators, arranged over a panel $D_x \times D_y$ sized, located in the $z = 0$ plane, and with the n-th element having coordinates $Q_n = (x'_n, y'_n, 0)$ and excitation coefficient w_n (see Fig. 1).
The target of the array is radiating, in a prescribed region of the $z > 0$ half space, termed the QZ, a prescribed wave. Throughout the paper we suppose the QZ being a box, $L_x \times L_y \times L_z$ sized, starting at $z = d$. We require deviations from the desired field with tolerances of $\pm 0.5dB$ for the amplitude and $\pm 5^\circ$ for the phase.
For a fixed QZ, the aim of the design procedure is to choose the panel size, the number of radiators, their locations and their excitation coefficients to meet the design specifications.

III. The design procedure

We suppose to have fixed the panel size according to the procedure in [5], [6]. Following the results in [14], The purpose now is to satisfying the specifications on the 3D QZ with the least number of radiators by enforcing constraints only on the first edge of the QZ itself, namely, only on a finite, 2D region, $L_x \times L_y$ sized of the plane $z = d$, henceforth termed the Quiet Plane (QP).
The radiated field $\underline{E}$ at P_m can be written as

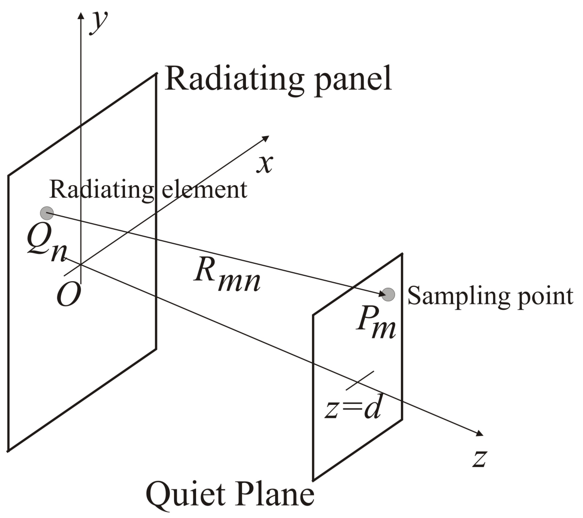

Fig. 1. Geometry of the problem.

$$\underline{E}(P_m) = \sum_{n=1}^{N} w_n \underline{f}(\theta_{mn}, \phi_{mn}) \frac{e^{-jk_0 R_{mn}}}{R_{mn}}, \qquad (1)$$

where $\underline{f}$ is the element factor, k_0 is the wavenumber and $(R_{mn}, \theta_{mn}, \phi_{mn})$ are the spherical coordinates of P_m in a reference system with center at Q_n (see Fig. 1). Eq. (1) holds true if P_m is located in the far-zone region of each of the involved radiators.

For the sake of simplicity, a scalar problem is addressed so that the specifications concern the y-component E of the field given by

$$E(P_m) = \sum_{n=1}^{N} w_n f(\theta_{mn}, \phi_{mn}) \frac{e^{-jk_0 R_{mn}}}{R_{mn}}, \qquad (2)$$

where f is the homologous component of E. As the element factor, that of an open-ended waveguide (OEWG) is here considered, so that f is the y-component of $\underline{f}$.

Eq. (2) holds true if P_m is located in the far-zone region of each of the involved radiators. However, it has been verified that this approximation does not affect the accuracy of the synthesized echo generation [6].

The constraints are expressed through a point matching procedure with the desired field on a set of M sampling points of the QP at the coordinates $P_m = (x_m, y_m, d)$ (see Fig. 1). The number M and the locations P_m of such points are also unknowns of the design approach. Point matching is here preferred since, even by using more general testing functions, the problem of their discretization cannot be evaded. It should be also mentioned that, the discretization procedure of the QP that will be described below outperforms the sampling approach in [15], as shown in [16].

To enforce the design specifications at the M sampling points P_m, the excitation coefficients w_n should satisfy the following linear system

$$\underline{\underline{A}} \cdot \underline{w} = \underline{q}, \qquad (3)$$

where $\underline{w}$ is the vector containing all the element excitations, $\underline{q}$ is the vector containing the samples of the prescribed field on the QP and

$$\underline{\underline{A}} = \begin{bmatrix} f_{11}\frac{e^{-jk_0 R_{11}}}{R_{11}} & \cdots & f_{1N}\frac{e^{-jk_0 R_{1N}}}{R_{1N}} \\ \vdots & \ddots & \vdots \\ f_{M1}\frac{e^{-jk_0 R_{M1}}}{R_{M1}} & \cdots & f_{MN}\frac{e^{-jk_0 R_{MN}}}{R_{MN}} \end{bmatrix}, \qquad (4)$$

where $f_{mn} = f(\theta_{mn}, \phi_{mn})$.

The excitation coefficients w_n are determined by means of a Singular Value Decomposition (SVD) approach [5], [6].

The criterion leading the choice of the number and locations of the radiators and QP sampling points is the optimization of $cond(\underline{\underline{A}})$, the condition number of $\underline{\underline{A}}$. Since $cond(\underline{\underline{A}})$ is the ratio between the maximum and minimum singular values of $\underline{\underline{A}}$, this task is accomplished by making the singular values dynamics of $\underline{\underline{A}}$ flatter. Accordingly, on denoting by $T = \min\{M, N\}$ and without addressing issues related to the norm of the matrix $\underline{\underline{A}}$, the "optimal" QP and radiator grids can be defined as those maximizing the functional

$$\Phi(Q_1, \ldots, Q_N, P_1, \ldots, P_M) = \sum_{t=2}^{T} \frac{\sigma_t}{\sigma_1}, \qquad (5)$$

evaluating the "area" subtended by the normalized singular values σ_t/σ_1.

Concerning the choice of M, it is fixed according to saturation criterion in [16]. Concerning the number N, in order to keep $cond(\underline{\underline{A}})$ low, it is chosen so that $N = M$ [6].

To reduce the computational burden and strengthen the optimization of Φ against the traps, proper mapping functions transforming uniform 2D lattices into non-uniform ones are employed to represent Q_n and P_m by a few parameters [5], [6]. In this way, the locations (x_k, y_k) of both the radiators and QP points are represented as

$$(x_k, y_k) = (h(\xi_k, \eta_k), g(\xi_k, \eta_k)), \qquad (6)$$

with

$$h(\xi_k, \eta_k) = \sum_{r=1}^{R} \sum_{s=1}^{S} \alpha_{rs} L_r(\xi_k) L_s(\eta_k), \qquad (7)$$

and

$$g(\xi_k, \eta_k) = \sum_{r=1}^{R} \sum_{s=1}^{S} \beta_{rs} L_r(\xi_k) L_s(\eta_k), \qquad (8)$$

where (ξ_k, η_k) defines a uniform lattice of $(-1, 1) \times (-1, 1)$, L_j is a properly defined set of basis functions (e.g., Legendre polynomials, as employed in the numerical analysis) and α_{rs} and β_{rs} are proper expansion coefficients.

The number of searched for parameters can be further reduced by accounting for the reflection symmetry of the radiator locations and QP grid around the origin. In particular,

- h can be chosen as odd with respect to ξ and even with respect to η;
- g can be chosen as even with respect to ξ and odd with respect to η.

IV. GPU ACCELERATION

Although recently cast to weak forms of multi-core architecturs, CPUs are essentially designed to compute scalars, aiming at optimizing the system control with low latency instead of high throughputs.
In the opposite, GPUs follow a Single Instruction Multiple Data (SIMD) [10] computing pattern. In other words, thanks to their advanced multi-core architectures and high-bandwidth data access, GPUs executes the same instruction on data elements concurrently, enabling the simultaneous processing of thousands of light-weight threads and thus achieving high computational throughputs across a large quantity of data.
AccelerEyes Jacket [11] is a Matlab toolbox running CUDA in the background and designed to work on NVIDIA GPUs which automatically wraps Matlab code to high performance graphics card primitives. At runtime, the memory transfers are optimized, the code performance are tuned and the GPU kernels are efficiently launched.
The interpretive nature of the Matlab language is maintained by providing transparent access to the GPU compiler, so that all GPU-specific programming details are handled by Jacket, freeing the user to deal with low-level (e.g., C++, CUDA) languages. The programmer is enabled to write and run code on the GPU in the Matlab native language.
Jacket's variables belong to the Matlab workspace as any other CPU Matlab variable, and all standard data types are supported, including double precision complex type. When operations or functions are performed on Jacket's variables, the execution is automatically performed on the GPU instead of the CPU.
To illustrate the simplicity of the use of Jacket, Algorithms 1 and 2 report the Matlab and Jacket pseudo-codes, respectively, for the evaluation of functional Φ. As it can be seen, Algorithm 2 represents a minor modification of Algorithm 1, mainly concerning the casting of the input variables by means of the **gdouble** function (required since the Matlab **fminunc** function only accepts inputs of data type **double**) and leading to their transfer from the CPU to the GPU, or the definition of new GPU variables by the **gzeros** function.
It should be noticed that in the pseudo-codes 1 and 2, data have been packed into large scale matrices to exploit the Matlab "vectorization" feature, namely, a feature of the compiler allowing the same line of code to apply to scalar, vectors or matrices (*polymorphism*). Indeed, Matlab performs these vectorized operations much more efficiently than loops and automatically multithreads some of them, also thanks to "multicore-aware" functions [17]. In this way, the most benefits of the availability of multi-core CPUs or GPU can be achieved.

Algorithm 1 Multi-core CPU Matlab pseudo-code for evaluating functional Φ.

```
function $\Phi(\underline{\alpha}^R,\underline{\gamma}^R,\underline{\alpha}^{QZ},\underline{\gamma}^{QZ})$

global $N,M,\underline{L}_x^R,\underline{L}_y^R,\underline{L}_x^{QZ},\underline{L}_y^{QZ},k_0,d$

X_PRIME = zeros(1,N);
Y_PRIME = zeros(1,N);
for $n \leq N$
    X_PRIME(n) = sum(sum($\underline{\alpha}^R$.*$\underline{L}_x^R$(n)));
    Y_PRIME(n) = sum(sum($\underline{\gamma}^R$.*$\underline{L}_y^R$(n)));
end

X = zeros(1,M);
Y = zeros(1,M);
for $m \leq M$
    X(n) = sum(sum($\underline{\alpha}^{QZ}$.*$\underline{L}_x^{QZ}$(m)));
    Y(n) = sum(sum($\underline{\gamma}^{QZ}$.*$\underline{L}_y^{QZ}$(n)));
end

[XX,XX_PRIME]=meshgrid(X,X_PRIME);
[YY,YY_PRIME]=meshgrid(Y,Y_PRIME);
Rmn=(d.^2+(XX-XX_PRIME).^2+(YY-YY_PRIME).^2);
Kx = $k_0$*(XX-XX_PRIME)./Rmn;
Ky = $k_0$*(YY-YY_PRIME)./ Rmn;

A = f(Kx, Ky) .*exp(-j*$k_0$ * Rmn);

S=svd(A);

$\Phi$=sum(S)
```

V. NUMERICAL RESULTS

In this Section, we report a performance analysis of the approach concerning the two CPU and GPU implementations and results concerning the synthesis of the echo produced by three perfectly conducting cylinders.
The processing has been performed on a Genesis Tesla I-7950 workstation, with a 8-core Intel CPU i7-950, working at $3.06GHz$ and with $6Gb$ of RAM. The workstation is equipped with an Nvidia Tesla C2050, benefitting of the state-of-the-art Fermi GPU architecture and consisting of 14 streaming multiprocessors (SMs), each containing 32 streaming processors (SPs), or processor cores, running at $1.15GHz$. The C2050 is further equipped with a $2.8GB$, off-chip, global memory and supports double precision arithmetics.

A. Computational performance analysis

A test case consisting of a $35\lambda \times 24\lambda$ sized array, generating a $20\lambda \times 14\lambda$ sized QP region at $d = 35\lambda$ has been considered. The size of both, the array panel and the QP region have been progressively enlarged of a factor $1 \leq \alpha \leq 2$ introduced to study the computational performance of the algorithm. The optimization of Φ is performed by the Matlab **fminunc**

Algorithm 2 Multi-core CPU & GPU Matlab-Jacket pseudo-code for evaluating functional Φ.

```
function $\Phi(\underline{\alpha}^R,\underline{\gamma}^R,\underline{\alpha}^{QZ},\underline{\gamma}^{QZ})$

global $N$,$M$,$\underline{L}_x^R$_h,$\underline{L}_y^R$_h,$\underline{L}_x^{QZ}$_h,$\underline{L}_y^{QZ}$_h,$k_0$_h,$d$_h

$\underline{\alpha}^R$_h = gdouble($\underline{\alpha}^R$);
$\underline{\gamma}^R$_h = gdouble($\underline{\gamma}^R$);
$\underline{\alpha}^{QZ}$_h = gdouble($\underline{\alpha}^{QZ}$);
$\underline{\gamma}^{QZ}$_h = gdouble($\underline{\gamma}^{QZ}$);

X_PRIME_h = gzeros(1,N,'gdouble');
Y_PRIME_h = gzeros(1,N,'gdouble');
gfor $n \leq N$
    X_PRIME_h(n) = sum(sum($\underline{\alpha}^R$_h.*$\underline{L}_x^R$_h(n)));
    Y_PRIME_h(n) = sum(sum($\underline{\gamma}^R$_h.*$\underline{L}_y^R$_h(n)));
end

X_h = gzeros(1,M,'gdouble');
Y_h = gzeros(1,M,'gdouble');
gfor $m \leq M$
    X_h(n) = sum(sum($\underline{\alpha}^{QZ}$_h.*$\underline{L}_x^{QZ}$_h(m)));
    Y_h(n) = sum(sum($\underline{\gamma}^{QZ}$_h.*$\underline{L}_y^{QZ}$_h(n)));
end

[XX_h,XX_PRIME_h]=meshgrid(X_h,X_PRIME_h);
[YY_h,YY_PRIME_h]=meshgrid(Y_h,Y_PRIME_h);
Rmn_h=(d_h.^2+(XX_h-XX_PRIME_h).^2+(YY_h-YY_PRIME _h).^2);
Kx_h = $k_0$_h*(XX_h-XX_PRIME_h)./Rmn_h;
Ky_h = $k_0$_h*(YY_h-YY_PRIME_h)./ Rmn_h;

A_h = f(Kx_h, Ky_h) .*exp(-j*$k_0$_h * Rmn_h);

S_h=svd(A_h);

$\Phi$=sum(S_h)
```

function. Accordingly, the performance in the evaluation of functional Φ is only reported.

Fig. 2 depicts the times required for the initial assignments. Furthermore, Figs. 3 and 4 show the time required by the CPU and GPU implementations for the evaluations of $\underline{\underline{A}}$ and of its SVD. As it can be seen, the most time consuming part is the latter.

From these results, and considering the overall computing time, the CPU and GPU implementations approximately perform the same when α is close to 1. Opposite to this, i.e., for very large echo generators, the speedup of the GPU implementation as compared to the CPU ranges from 2 to 4.5, for α ranging between 1.5 and 2.

B. Design performance analysis

We now report on the results concerning the generation of the field scattered by three, infinitely long, perfectly conducting cylinders having circular cross section and radius equal to $a = 5\lambda$, with reciprocal center spacing of $d_{cyl} = 15\lambda$ (see Fig. 5). A test case consisting of a $35\lambda \times 35\lambda$ sized array of 253 radiating elements, generating a $22\lambda \times 22\lambda$ sized QP region at $d = 35\lambda$ has been dealt with. The cylinders' axes are assumed to be parallel to the y-axis, they are illuminated by an orthogonally impinging plane wave leading to a fully 2D scalar problem, while the reciprocal distance to the QP is $d = 300\lambda$. Under the these circumstances, the spatial spectrum of the field impinging on the QP contains plane waves with maximum impinging angles ψ of up to approximately 2.9°.

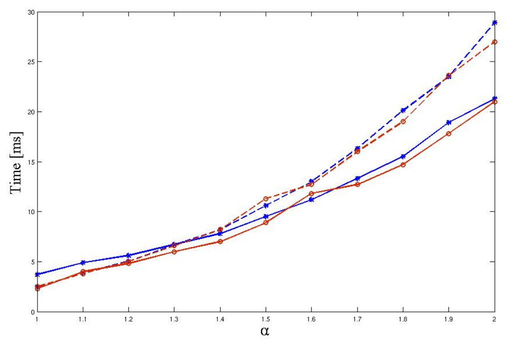

Fig. 2. *Initial assignments*. Blue solid line: X_PRIME_h+Y_PRIME_h. Blue dashed line: X_PRIME+Y_PRIME. Red solid line: X_h+Y_h. Red dashed line: X+Y.

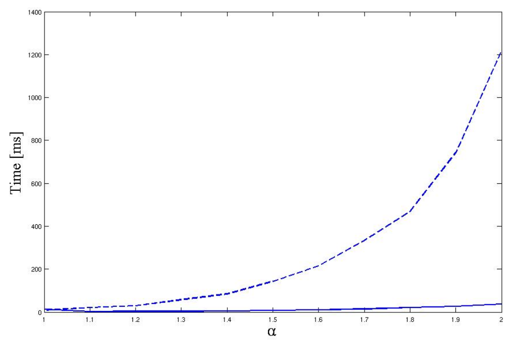

Fig. 3. *Evaluation of* $\underline{\underline{A}}$. Blue solid line: GPU. Blue dashed line: CPU.

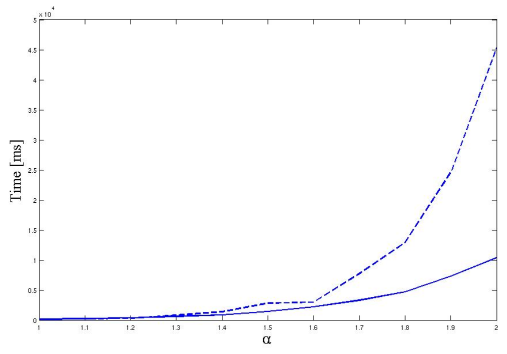

Fig. 4. *Evaluation of* $\underline{\underline{A}}$. Blue solid line: GPU. Blue dashed line: CPU.

It has been verified that the synthesized echo generator is capable of generating the scattered field within the required accuracy. Figs. 6 and 7 compare the behavior of the synthesized versus desired scattered field amplitude and phase respectively, for a cut along the x-axis. Of course, the phase inaccuracies correspond to the regions with the least field amplitude values. We remark that this approach has proved to be robust against uncertainties in the radiator locations and excitations, see [6].

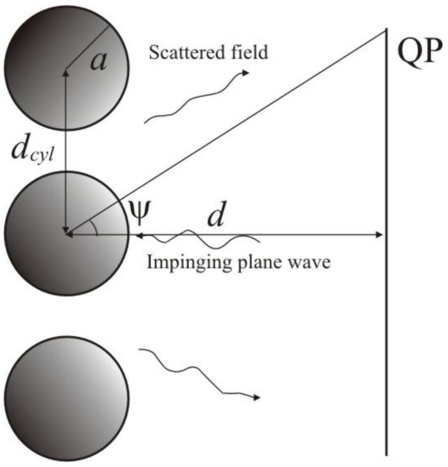

Fig. 5. Geometry relevant to the generation of the field scattered by three perfectly conducting cylinders.

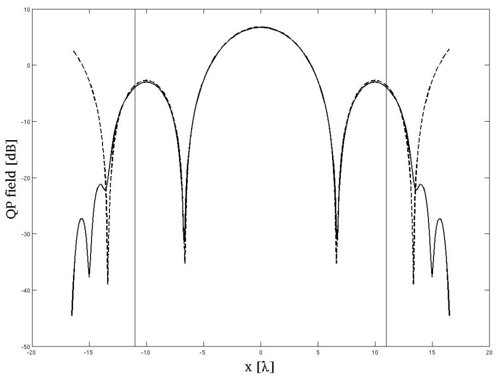

Fig. 6. Scattered field generation: cuts along the x-axis of the desired (dahsed line) and synthesized (solid line) field amplitudes over the QP. The vertical line denotes the QP boundary.

VI. Conclusions & future developments

We have presented an approach to the design of array-based, 2D echo generators.

The computational burden has been easily and properly managed by a parallel programming on GPUs through the Jacket Matlab toolbox. Numerical results have assessed the performance of the technique in terms of computational requirements and QZ field behavior.

Future activities will regard the development of a time-domain echo generator and dealing with the issue of the mutual coupling between the array elements.

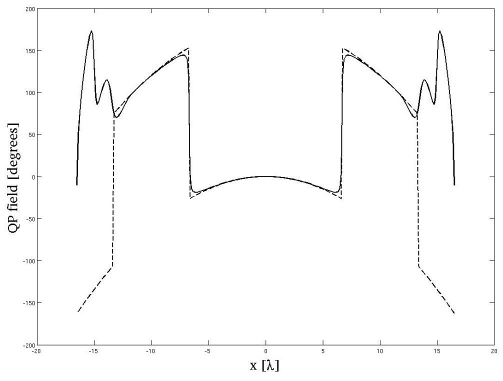

Fig. 7. Scattered field generation: cuts along the x-axis of the desired (dahsed line) and synthesized (solid line) field phases over the QP. The vertical line denotes the QP boundary.

References

[1] A.W. Rudge, K. Milne, A.D. Olver, P. Knight, *The handbook of antenna design*, London, Peter Peregrinus, 1982.

[2] M. Ciattaglia, A. De Luca, L. Infante, S. Mosca, M. Albani, "Near field techniques for radar synthetic environment simulator", *Proc. of the Int. Conf. on Electromagn. in Adv. Appl.*, Turin, Italy, Sept. 17-21, 2007, pp. 780-783.

[3] A.D. Olver, "Compact antenna test ranges", *Proc. of the 7th Int. Conf. on Antennas Prop. (ICAP)*, York, UK, Apr. 15-18, 1991, pp. 99-108.

[4] A. Capozzoli, G. D'Elia, "On the plane wave synthesis in the near-field zone", *Proc. of the Int. Conf. on Antenna Tech.*, Ahmedabad, India, Feb. 23-24, 2005, 273-277.

[5] A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, P. Vinetti, "A novel approach to the design of generalized plane-wave synthesizers", *Proc. of the 3rd Europ. Conf. on Antennas Prop.*, Berlin, Germany, Mar. 23-27, 2009, CD ROM.

[6] A. Capozzoli, C. Curcio, A. Liseno, "Time-harmonic echo generation", *accepted for publication on IEEE Trans. Antennas Prop.*.

[7] C.C. Courtney, D.E. Voss, R. Haupt, L. LeDuc, "The theory and architecture of a plane-wave generator", *Proc. of the 24th AMTA Symp.*, Cleveland, Ohio, Nov. 3-8, 2002, pp. 353-358.

[8] P. Wendykier, J.G. Nagy, "Parallel Colt: A high-Performance Java library for scientific computing and image processing", *ACM Trans. on Math. Soft.*, vol. 37, n. 3, pp. 31:1-31:22, Sept. 2010.

[9] www.accelereyes.com/content/collateral/GPUAcceleratedISAR_ver2.pdf

[10] D.B. Kirk, W.W. Hwu, *Programming massively parallel processors*, Morgan Kaufmann, Burlington, MA, 2010.

[11] www.accelereyes.com.

[12] N.A. Davis, A. Pandey, B.A. McKinney, "Real-world comparison of CPU and GPU implementations of SNPrank: a network analysis tool for GWAS", *Bioinformatics 2010*: doi: 10.1093/bioinformatics/btq638, Nov. 2010.

[13] Y. Lin, R.A. Renaut, "The application of projected conjugate gradient solvers on Graphical Processing Units", *submitted for publication to Parallel Computing*, submitted for publication.

[14] A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, "On the sampling of electromagnetic fields", *Proc. of the URSI Int. Symp. on Electromagnetic Theory*, Berlin, Germany, Aug. 16-19, 2010, pp. 185-188.

[15] O.M. Bucci, C. Gennarelli, C. Savarese, "Representation of electromagnetic fields over arbitrary surfaces by a finite and nonredundant number of samples", *IEEE Trans. Antennas Prop.*, vol. 46, n. 3, pp. 351-359, Mar. 1998.

[16] A. Capozzoli, C. Curcio, A. Liseno, P. Vinetti, "Field sampling and field reconstruction: a new perspective", *Radio Sci.*, vol. 45, RS6004, 31 pp., 2010, doi:10.1029/2009RS004298.

[17] G. Sharma, J. Martin, MATLAB$^{®}$: A language for parallel computing, *Int. J. Parallel Prog.*, vol. 37, n. 1, pp. 3-36, 2009.