Fast CPU/GPU Pattern Evaluation of Irregular Arrays

A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, and P. Vinetti

Dipartimento di Ingegneria Biomedica, Elettronica e delle Telecomunicazioni
Università di Napoli Federico II, Via Claudio, 21 Naples, Italy
a.capozzoli@unina.it

Abstract- An approach for the fast analysis of "irregular", i.e., of conformal, periodic or aperiodic, 2D arrays, based on the use of the *p*-series approach and Non-Uniform FFT (NUFFT) routines is proposed. The approach allows for modulating the computational burden depending on the array curvature and, thanks to the use of the NUFFT, the asymptotic growth of the computing time reduces to that of a few, standard FFTs. A sub-array partition strategy is also sketched and shown to further unburden the procedure and control the accuracy. The approach has been implemented in both sequential and parallel codes enabling its execution on CPUs and on cost-effective, massively parallel computing platforms like Graphic Processing Units (GPUs). Its performance in terms of computational efficiency and accuracy has been assessed by an extensive numerical analysis and also against benchmarks provided by algorithms based on fast Matrix-Vector Multiplication routines.

Index Terms- Aperiodic array antennas, conformal array antennas, fast antenna analysis, GPU computing, CUDA.

I. INTRODUCTION

Array pattern synthesis is a computationally challenging problem since it requires demanding iterative algorithms for the (local or global) optimization of a properly defined objective functional [1-3]. The computational bottleneck of such algorithms is essentially related to the repeated calculation of the far field pattern (FFP) and possibly of the functional gradient (FG) (as long as gradient-based optimization approaches are adopted).

Different kinds of arrays have been subject in the literature of synthesis procedures. Many of the developed synthesis algorithms refer to "regular arrays" (RAs), for which the elements are arranged on a periodic grid of a portion of a line or plane (see Fig. 1). In the last decade, "irregular arrays" (IAs), namely, arrays for which the elements lay on an "aperiodic" grid and/or on conformal lines or surfaces have been proposed (see Figs. 2-4) to overcome the typical issues of RAs [4-8]. Indeed, "aperiodic" structures allow, as compared to "periodic" ones, a more efficient power handling, if uniformly excited in amplitude [5], and permit improving the bandwidth performance [9], while also reducing the overall number of elements and mitigating the effects of the grating lobes [10]. Furthermore, "conformal" structures, as compared to linear or planar ones, satisfy aerodynamic and low-scattering requirements in aircraft antennas [4], permit space deployability [11] and considerably reduce the feed path length, thus improving the bandwidth behavior, of reflectarray antennas [7]. However, IA synthesis appears computationally more demanding than RAs synthesis, since the FFP or FG evaluations become more burdened.

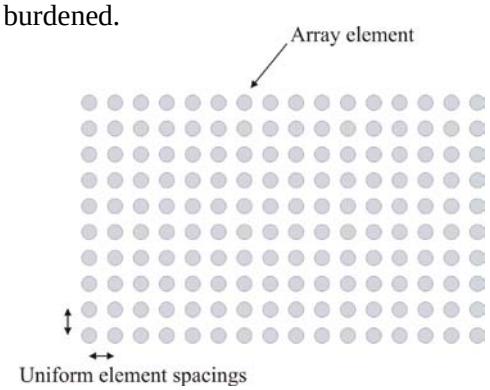

Fig. 1. Example of planar, periodic array.

For RAs, when the array factor can be employed [12] and the far field pattern is evaluated on a regular spectral grid, the excitation coefficients and the array factor are related by a "standard" Discrete Fourier Transform (DFT) link, i.e., a DFT defined on Cartesian, regular grids, as

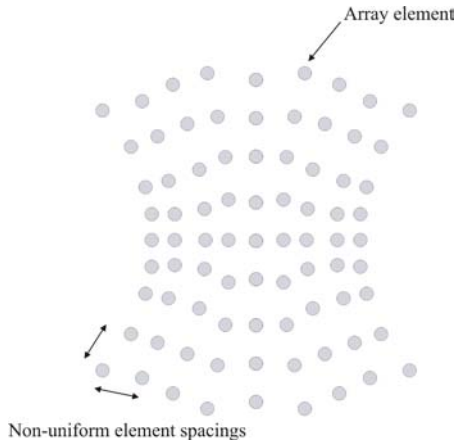

Fig. 2. Example of planar, aperiodic array.

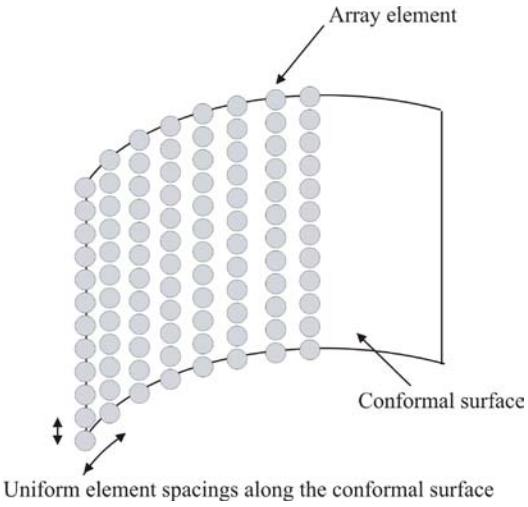

Fig. 3. Example of conformal, periodic array.

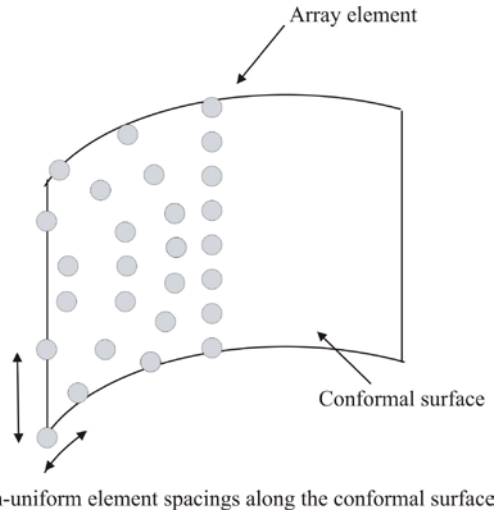

Fig. 4. Example of conformal, aperiodic array.

discussed in [13]. In this case, the speedup is achieved by means of standard Fast Fourier Transform (FFT) routines [14]. Indeed, taking for example arrays of M elements, the use of FFTs changes the $O(M^2)$ computational complexity of each DFT to $O(M\log M)$.

On the other hand, for IAs, the possibility of a direct use of FFTs [6-8] breaks down. Indeed, for planar IAs, evaluating the FFP as well as the FG requires the DFT to be computed on irregular grids (Cartesian non-uniform or non-Cartesian), so that the requirement of standard FFTs is not met anymore. Moreover, for IAs whose elements are arranged on non-linear or non-planar domains, a standard DFT link between array factor and excitation coefficients is lost [7].

The aim of the paper is to show how the asymptotic growth of the computational burden when dealing with IAs can be reduced to $O(M\log M)$ as long as the computation of the FFP can be recast as that of a few FFTs. To this end, three different tools are exploited in the following, namely the *p*-series approach [7,15,16], the Non-Uniform FFT (NUFFT) algorithms [17,18] and a sub-array partitioning strategy. In particular, the *p*-series approach enables, for conformal surfaces with mild curvature, recasting the link between the array excitation coefficients and the FFP as the sum of a few, possibly non-standard, DFTs. On the other hand, NUFFT algorithms quickly evaluate non-standard DFTs as the sum of a few FFTs. And so, the two approaches together are able to restore the yearned $O(M\log M)$ computational complexity. Finally, the sub-array partitioning strategy is capable to additionally improve the method in terms of computational burden and accuracy. It is also shown that the computational approach herein proposed can be even more fruitfully exploited if implemented on innovative, intrinsically parallel, off-the-shelf hardware provided by Graphical Processor Units (GPUs) [19]. GPUs represent, in fact, inexpensive, highly-parallel hardware, significantly mitigating the requirements in terms of space, management, cost and user access, when compared to more complex CPU grid/cluster systems [20]. In addition, while programming on GPUs remains more involved than standard sequential programming, the recent interest in GPUs for scientific computing has promoted the development of effective programming

frameworks [21,22], which in return simplified implementations on these platforms [23]. Finally, it is further shown how the proposed strategy fruitfully modulates, depending on the array curvature, the computational burden without impairing the accuracy of the FFP evaluation.

The performance of the approach is tested by an implementation in C language on a standard, single-core (sequential) CPU and by an implementation in NVIDIA CUDA (Compute Device Unified Architecture) language [24] on a (multithread) NVIDIA GTX 260 GPU. More in detail, C language implementations of the proposed strategy and of an approach based on sequential Optimized Matrix-Vector Multiplication (OMVM) [25,26], generally having an $O(M^2)$ asymptotic complexity, but performing better than a brute-force (i.e., a matrix vector multiplication based on the use of "for loops" [27]) one, have been setup. The OMVM approach has been purposely developed in this paper to be used as a reference for assessing the performance of the proposed strategy. Speedups of more than 10 times for arrays of 10^4 elements obtained by our method as compared to OMVM, FFP evaluation are highlighted. Similarly, CUDA language implementations of the proposed strategy and of an approach based on parallel OMVM routines have been realized. Speedups of more than 8 times for arrays of 10^4 elements, when comparing the GPU version of our approach against that of employing parallel OMVMs are pointed out. Moreover, speedups of more than 40 times, when comparing the GPU and CPU versions of the developed algorithm, are indicated.

Finally, the accuracy of the procedure is discussed.

The paper is organized as follows. In Section II, the problem of radiation is formulated and the strategy exploited for the NUFFT-based evaluation of the FFP, relying on the use of the *p*-series approach, is presented. The benchmarking, OMVM-based method is also sketched. Section III briefly enlightens some details of the sequential (CPU) and parallel (GPU) implementations for both considered approaches (i.e. NUFFT and OMVM). Sections IV and V illustrate and compare the computational performance and accuracy of the NUFFT-based method, as compared to the OMVM-based one. Finally, in Section VI, conclusions are drawn and future developments are foreseen. In the Appendices, the NUFFT algorithm is shortly recalled, C-like and CUDA-like listings of the developed NUFFT routines are reported and ancillary calculations concerning the convenience of adopting a sub-array partitioning are presented.

II. RADIATION BY 2D IRREGULAR ARRAYS

In the following, the approach to the fast analysis of IAs is presented by referring to a general 2D geometry.

Let us consider an antenna array made of M elements, non-uniformly distributed on a 2D arbitrary surface, S, of equation $z=f(x,y)$, $(x,y)\in D$, with D a planar, auxiliary domain, so that the radiating elements are located at the points (x_m,y_m,z_m) with $z_m=f(x_m,y_m)$ and $m = 0, 1, \ldots, M-1$ (see Fig. 5). The complex excitation coefficients are denoted with a_m, $m = 0, 1, \ldots, M-1$.

Generally speaking, the FFP of an IA can be written as [4,6-8]

$$\underline{F}_r(u,v) = \sum_{m=0}^{M-1} a_m \underline{h}_r(u,v,x_m,y_m) e^{j[ux_m+vy_m]} \quad (1)$$

where

$$\begin{cases} u = \beta \sin\theta \cos\varphi \\ v = \beta \sin\theta \sin\varphi \end{cases}, \quad (2)$$

$\beta=2\pi/\lambda$, λ being the wavelength, and $\underline{h}_r(u,v,x_m,y_m)$ accounts for the radiation characteristics and position of the m-th element (see also Subsection C).

Henceforth, the vector aspects of the problem are dismissed. In other words, we assume that $\underline{h}$ can be factored out as

$$\underline{h}_r(u,v,x_m,y_m) = h(u,v,x_m,y_m)\underline{p}(u,v), \quad (3)$$

that is, all the array elements share a common polarization behavior described by $\underline{p}$. Accordingly, $\underline{F}_r(u,v)=F(u,v)\underline{p}(u,v)$, where

$$F(u,v) = \sum_{m=0}^{M-1} a_m h(u,v,x_m,y_m) e^{j[ux_m+vy_m]}. \quad (4)$$

We notice that

- for mild conformal geometries (the elements have approximately the same orientation), vector correction terms to eq. (3) are often negligible;
- for non-mild conformal geometries, the sub-array partitioning strategy helps to mitigate the assumptions needed for the validity of eq. (3) (see Subsection II.e).

In practice, the FFP is required at a number H of spectral positions (u_h,v_h), so that the corresponding discrete values F_h of F can be written, following eq. (4), as

$$F_h = \sum_{m=0}^{M-1} a_m h(u_h,v_h,x_m,y_m) e^{j[u_h x_m + v_h y_m]}. \qquad (5)$$

Thus, even in the case when the spatial and spectral points (x_m,y_m) and (u_h,v_h), respectively, form a Cartesian grids, the samples of the FFP cannot be evaluated by a standard FFT since eq. (5) is not in the form of a DFT [28].

A. Far Field Pattern Computation by Optimized Matrix-Vector Multiplications

Whenever it is not possible to conveniently express the function $h(u,v,x_m,y_m)$, an effective way to evaluate the samples F_h of the FFP is employing OMVM routines.

Indeed, the kernel $h(u_h,v_h,x_m,y_m)\exp j[u_h x_m + v_h y_m]$ of eq. (5) can be arranged as a matrix $\underline{\underline{B}}$ whose generic element is

$$B_{hm} = h(u_h,v_h,x_m,y_m) e^{j[u_h x_m + v_h y_m]} \qquad (6)$$

so that eq. (5) can be recast as a matrix-vector multiplication

$$F_h = \sum_{m=0}^{M-1} B_{hm} a_m. \qquad (7)$$

Eq. (7) is amenable to be evaluated by OMVM routines, which in general perform as $O(M^2)$ or, in the case of particular symmetries of $\underline{\underline{B}}$, as $O(M\log^5 M)$ [25,26].

In the following, we illustrate a strategy capable of reducing the computational complexity needed to calculate F_h's in cases when the function $h(x_m,y_m,u,v)$ can be factored out.

B. Factorization of the Function $h(u,v,x_m,y_m)$

As long as $h(u,v,x_m,y_m)$ can be written (in an exact or approximate way) as

$$h(x_m,y_m,u,v) = \sum_{p=0}^{P-1} \varphi_p(u,v)\psi_p(x_m,y_m), \qquad (8)$$

then the FFP samples F_h can be calculated as

$$F_h = \sum_{p=0}^{P-1} \varphi_p(u_h,v_h) \left[\sum_{m=0}^{M-1} a_m \psi_p(x_m,y_m) e^{j[u_h x_m + v_h y_m]} \right]. \qquad (9)$$

Now, each inner summation of eq. (9) is in the form of a (possibly non-uniform, depending on the values of x_m, y_m, u_h, and v_h) DFT which can be computed, in the general case, by a NUFFT routine call, performing, as already stressed in the Introduction, as $O(M\log M)$. Consequently, the FFP samples F_h can be evaluated by P NUFFT calls, for an overall $O(PM\log M)$ complexity.

Generally speaking, a simple way to obtain a factorization of h is to regard it as the kernel of a linear operator $\mathcal{A}$ so that the singular functions of $\mathcal{A}$, can be employed [29] as functions φ_p and ψ_p which then provide, when the summation in eq. (9) involves infinite terms, an exact representation of h. However, when truncating, such summation requires a high number of terms for an accurate representation, then the expansion of h can be obtained by selecting proper basis functions φ_p and ψ_p, depending on the features of h. In the following, we present a simple example, of relevant practical interest, concerning the factorization (8), for a proper choice of φ_p and ψ_p.

C. *p*-Series Factorization

In order to focus the attention on a case of practical interest, we consider an IA for which

$$h(u,v,x_m,y_m) = f(u,v) e^{jwz_m}, \qquad (10)$$

where $w = \sqrt{\beta^2 - (u^2+v^2)}$ and $f(u,v)$ is the element factor [12]. As prefigured at the beginning of Section II, h depends on the radiation characteristics of the m-th element through the

element factor p, and on its position through the quota z_m. For the sake of simplicity, in the following formulas we will skip the element factor, unessential in the discussion, and we will nevertheless deal with it throughout the numerical analysis.

Under this hypothesis, following the approach in [7, 15, 16] and on denoting by w_0 the value of w related to the main beam direction, eq. (4) can be rewritten as

$$F(u,v) = \sum_{m=0}^{M-1} a'_m e^{j[ux_m + vy_m + w'z_m]} \qquad (11)$$

with $w'=w-w_0$ and $a'_m = a_m \exp[jw_0 z_m]$. For mild curvatures of S, the exponential $\exp[jw'z_m]$ can be expanded by a truncated Taylor series up to the (P-1)-th order (p-series), so that

$$F(u,v) \cong \sum_{p=0}^{P-1} \frac{(jw')^p}{p!} \sum_{m=0}^{M-1} z_m^p a'_m e^{j[ux_m + vy_m]} \qquad (12)$$

and the discrete values F_h of F can be expressed as

$$F_h = \sum_{p=0}^{P-1} \frac{(jw'_h)^p}{p!} \sum_{m=0}^{M-1} z_m^p a'_m e^{j(u_h x_m + v_h y_m)}. \qquad (13)$$

Obviously, the smaller the curvature of S, the smaller the value of P required for a given accuracy. In practice, a proper value for P can be chosen to trade off the computational burden and the accuracy of the approach, as it will be clearer in Subsection II.e and in the numerical analysis presented in Section IV. In general, the number of p-series terms is chosen in a way to ensure that the argument of $\exp[jw'z_m]$ is less than a "small" value all over the spectral (u,v) region of interest. Such a value is typically assumed equal to $\pi/8$, or even lower if more accurate results are desired. Moreover, the chosen number of p-series terms depends also on the coverage, so that, once the array and the coverage are given, the number of p-series terms can be consequently assigned. We stress that the inner summation in eq. (13) is not in the form of a standard DFT [28], so that, to recover the desired computational complexity, a NUFFT structure should be employed.

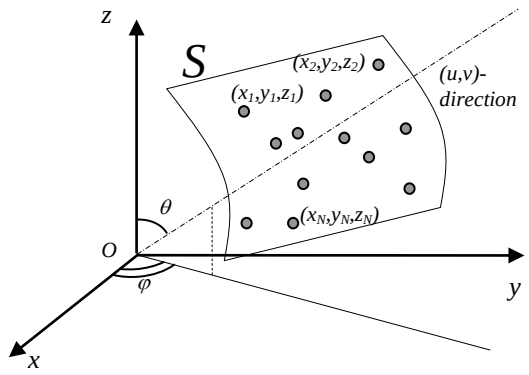

Fig. 5. Geometry of the problem.

D. Use of the NUFFT

Concerning the fast evaluation of each inner summation in eq. (13), a different NUFFT algorithm should be considered depending on the spatial and spectral grids at hand. More in detail:

a) for an arbitrary spatial grid (x_m,y_m) (aperiodic conformal array) and for a regular, Cartesian spectral lattice (u_h,v_h), a Non-Equispaced Data (NED), or "type-1", NUFFT is of interest [17];
b) for a regular, Cartesian spatial grid (periodic conformal array) and for an arbitrary spectral one, a Non-Equispaced Result (NER), or "type-2", NUFFT should be employed [17];
c) for arbitrary spatial and spectral grids, a "type-3" NUFFT should be adopted [18].

Since cases b) and c) are extensions of case a) which do not add any conceptual difficulty, in this paper we assume to evaluate the FFP on a regular, Cartesian spectral lattice, so that NED-NUFFTs are of interest (case a)). This is the most frequently occurring case, since, in antenna synthesis, the design specifications are usually given on a regular, Cartesian spectral lattice (u_h,v_h), leading indeed to the use of NED NUFFTs. Accordingly, for the sake of brevity, cases b) and c) will not be dealt with in the details.

Different approaches have been proposed in the literature for evaluating NUFFTs [17,18,30-33].

The main idea underlying many NUFFT algorithms is to approximate the non-uniform exponential function $\exp(jp\Delta u x_m)$ (having assumed that the h-th spectral point (u_h,v_h) corresponds to the $(p\Delta u,q\Delta v)$ uniform spectral grid point), by interpolating L, "oversampled", properly chosen and windowed uniform exponentials $\exp(jp\Delta u l\Delta x)$, $l=1,..,L$. Accordingly, non-

uniformly sampled exponentials can be approximated by properly weighted sums of uniform exponentials, enabling to exploit a finite number of standard FFT routine calls. It is worth noting that this strategy is not equivalent to a "brute-force thinning" of an array which, on the contrary, requires significantly denser element grids [34]. In this paper, we use the approach in [17], based on an "exact" representation of the exponentials $\exp(jp\Delta u x_m)$. For the reader's convenience, we quote Appendix A for a brief mathematical description of the employed NUFFT algorithm.

E. Sub-Array Partitioning

It should be mentioned that the array can be also partitioned into N sub-arrays, each one made up of m_n elements, such that $m_0+ m_1+... m_{N-1}=M$. Accordingly, eq. (13) can be rewritten by explicitly describing the radiation by each array portion, thus leading to

$$F_h = \sum_{n=0}^{N-1}\sum_{p=0}^{P-1} \frac{(jw_h')^p}{p!} \sum_{m=m_n}^{m_{n+1}} z_m^p a_m' e^{j(u_h x_m + v_h y_m)}. \quad (14)$$

As an advantage, the number P of terms involved to represent the exponential in (10) by a Taylor series expansion associated to each subarray is expected to reduce, for a fixed accuracy. Accordingly, the strategy can be applied to non-mild shapes, also to reduce model errors related to the vector aspects (i.e., model errors in the assumption (3)). To better enlighten this advantage, we mention the borderline case of a faceted array (or a faceted reflectarray [35-37]). In this case, across the junctions between the facets, the curvature is singular. Nevertheless, a partitioning into subarrays enables accurate computations with a number of P=1 terms for each facet (see also Subsection IV.c). Moreover, the sub-array partitioning strategy is further facilitated by the use of type-2 (for non-aperiodic arrays) or type-3 (for aperiodic arrays) NUFFT routines. Indeed, even when dealing with a uniform array, the field radiated by the various facets should be computed onto the common (u,v) grid associated to the overall antenna, a procedure requiring in general time-consuming interpolation stages. From this point of view, the opportunity of employing (type-2 or type-3) NUFFT routines, enabling arbitrary (u,v) output grids, offers the possibility of performing such an interpolation with $O(M\log M)$ complexity. In Section IV, we discuss how much convenient such a strategy can be.

Finally, the sub-array approach is amenable to a multi-level implementation [38], but, for the sake of simplicity, in this paper we will deal with a single-level one.

III. IMPLEMENTATION OF THE ALGORITHMS

The approach proposed in Subsections B-E has been implemented in both, a sequential code, running on conventional computing architectures (single-core CPU), and in a parallel code, taking advantage of GPU acceleration. Moreover, sequential and parallel implementations of an approach based on OMVM routines according to eq. (7) have been also setup to serve as a benchmark for the performance of the proposed approach.

For both, the sequential and parallel codes, particular care has been devoted to

- selecting high performance FFT routines, as required by the proposed, NUFFT-based approach;
- choosing high performance Matrix-Vector multiplication routines, as required by the OMVM-based scheme.

In the following, some implementation details concerning the developed sequential and parallel codes will be discussed. We remark that symbols in the following are defined in the Appendices A, B and C.

A. Sequential Implementations

All the sequential codes have been developed in ANSI C language. Such a choice is due to the use of the CUDA environment to develop the parallel counterpart. Indeed, a CUDA program consists of "phases" that are executed on the host (CPU) or the device (GPU) and of data structures that can be allocated on the host or the device, as well (see [24]). The host code is straight ANSI C code. The device code is ANSI C code, extended with special keywords for calling data-parallel functions (*kernels*), and managing the associated data structures. Accordingly, the development of a parallel code can be performed by starting with the sequential ANSI C code, spotting the phases that should be parallelized, and extending the corresponding instruction and data structures with

the special keywords for parallel executions provided by CUDA.

In particular, concerning the sequential code:

- the NUFFT algorithm has been implemented according to [17] (see also Appendix A); a particularly fast implementation of the FFT, based on the same philosophy of FFTW [39], and contained in the Intel Math Kernel Library (MKL) [40], has been exploited to speedup the required FFT calculations; a C-style listing of the algorithm is reported in Appendix B; the critical point of the algorithm is represented by the U matrix filling operations, performed within three, nested (m, l_1 and l_2) *for loops*; such a filling is "pseudo-random" (i.e., it does not obey to a "row-major" filling criterion [41]) since the indices i_x and i_y "jump" between non-consecutive values as long as m, l_1 and l_2 are swept; as known, this severely affects the memory latency when accessing the elements of U [41];
- the implementation of the OMVM-based approach relies on BLAS routines;

For both the cases, Intel Math Kernel Libraries (MKL) (v.1.0.02), including BLAS and FFT routines, have been exploited.

B. Parallel Implementations

As already stressed in the Introduction, all the parallel codes have been developed by means of the CUDA language [24].

For both, the NUFFT-based and OMVM-based algorithms, the GPU is exploited as an accelerating device, executing portions of the code in parallel [19]. More in detail:

- for the proposed approach, in correspondence to each NUFFT call, the execution is delivered to the GPU and the evaluation of each NUFFT performed by a proper, parallel implementation of the scheme detailed in Appendix A;
- for the OMVM-based approach, the evaluation of the matrix multiplication required by eq. (7) is performed, again through a parallel implementation on the GPU;

In the sequel, some details concerning the parallel implementations of the two considered approaches will be reported.

1. NUFFT-based approach

All the stages of Appendix A have been carefully examined and parallelized, according to the key rules of GPU programming [24]. A CUDA-style listing of the algorithm is reported in Appendix C. More in detail:

Stage 1

The calculations of the samples of the spatial and spectral windows Φ and $\hat{\Phi}$, respectively, as well as of the indices $\mu_{x,m}$ and $\mu_{y,m}$ (see Appendix B) are fully independent from each other and are evaluated in parallel, rather than by *for loops* as in the sequential case.

The computation of the U matrix is also parallel, but requires some more care, since different approaches could be envisaged to this end and the best performing one should be selected. Indeed, due to the already remarked "pseudo-random" access to U required by the sequential implementation, devising an efficient filling procedure in the parallel case is not straightforward and represents the main difficulty to be solved throughout the parallelization of the whole code described in Appendix B.

A first possible parallelization strategy would be to commit a thread to compute a single matrix element of U. However, in this way, the generic thread should perform, due to the "pseudo-random" filling, a time-consuming browsing of the input elements to establish whether they contribute to the committed element of U or not.

As an alternative, the implemented parallel code employs a 1D block grid of length M, each block allocating $(2K+1)\times(2K+1)$ threads. In this way, the above mentioned browsing is avoided since each thread is assigned to a different input element and updates the corresponding element of the U matrix.

Generally speaking, the number of allocable blocks in a 1D grid depends on the computing capability of the employed GPU [24]. For the GPU employed in this paper, the number of allocatable blocks is 65535, which is large enough for all the considered numerical tests. For arrays with M>65535, the algorithm should foresee a sequential allocation of 1D block grids. Since the maximum number of allocatable blocks depends on the employed GPU, the actual performance of

the algorithm depends on the hardware performance of the available graphic card.

However, it should be noticed that, by this solution and regardless to M, more than one thread may need to simultaneously update (namely, read, compute and store a new value) the same element $U(i_x,i_y)$. Unfortunately, when this happens, a conflict such as Writing After Writing (WAW) and Writing After Reading (RAW) [41] can occur, affecting the results. To preserve the integrity of the data, atomic operations have been exploited [41,42], which basically ensure the semantic correctness of the algorithm through a serialization of the updating operations. We finally observe that, the parallel implementation is such that two threads belonging to the same block never update the same element $U(i_x,i_y)$ (although threads belonging to different blocks can do). Moreover, since each block is $(2K+1)\times(2K+1)$ sized and, generally speaking, K is usually "small" (typical values range from 6, for single precision arithmetic, to 12, for double precision [17]), in order to speed-up the memory access, the updating operations are performed first on a temporary $(2K+1)\times(2K+1)$ matrix, allocated in the shared memory (and then shared by threads belonging to the same block), and subsequently on the global memory by the mentioned atomic operations.

Stage 2

The computation of the required FFT has been parallelized by means of the latest release of the cuFFT library (cuFFT v2.3) [43], implementing several, optimized parallel FFT algorithms, and choosing the one to be used depending on the shared memory occupation of the input array and on the possibility of reducing its size to a power of an integer factor.

Stage 3

Extracting the elements of the $\hat{U}$ matrix, their scaling with the elements Φ_{pq} and the subsequent memory updates are independent, and then easily parallelizable, operations.

We finally remark that, throughout the parallel implementation of the NUFFT routine, the typical suggested guidelines in programming GPUs [24] and concerning, for example,

- avoiding divergencies due, f.i., to conditional statements or non-coalesced memory accesses;
- balancing the computational load among the available resources;

have been applied.

Furthermore, data padding [24] has been adopted to manage a generic input data size. It should be noticed that, for the considered case, data padding does not significantly affect the algorithm performance since the amount of employed padding is always less or equal to the block size (which, as above discussed, contains $(2K+1)\times(2K+1)$ only threads) and, as such, negligible for a large input data size M.

2. OMVM-based approach

The implementation of the OMVM-based approach relies on the latest release of the cuBLAS (cuBLAS v.2.3) routines [44].

Also for this case, data padding has been applied.

3. Multilevel parallelization

It is worth noting that, the particular expression in eq. (13) is amenable to a further level of parallelization, since the different terms of the p-series summation can be simultaneously computed and, in turn, each NUFFT can be parallelized according to the guidelines above. Unfortunately, a single GPU cannot handle more kernels simultaneously and hence cannot effectively manage a multi-level, parallel computation.

Nevertheless, with a multi-GPU system [45], the computation of each term of the p-series can be executed by a different GPU accomplishing, in turn, the computation of a parallel NUFFT. Afterwards, all the terms can be added together by means of a reduction operation on a "master" processing unit. This strategy allows operating a two-level parallelism, one to compute the p-series and one to compute the NUFFTs.

Similar considerations apply also to the sub-array partitioning approach (see eq. (14)). Indeed, also in this case the computations for each sub-array are independent from all the others and a two-level parallelism can be obtained. Obviously, in this case, a three-level parallelism can even be achieved, by exploiting the independence of the sub-arrays and of the p-terms.

In this paper, only results concerning a single-level parallelization are shown. The multi-level case is left to future developments since it does not introduce any conceptual difficulty, but for communication protocols between the GPUs [45].

It should be finally observed that, with reference to the parallel implementation of the proposed NUFFT-based approach, the order of computation among the different *p*-series terms, or, in other words, the order of computation of the different required NUFFT routine calls, could be rearranged to simultaneously execute more than one NUFFT on the same GPU. In this way, apparently a multi-level parallelism could be achieved on a single GPU. However, if ever such a solution would be more effective, it should not be considered of practical interest since it would not comply with the typically required *transparent scalability* on a multi-scale architecture [24].

IV. COMPUTATIONAL PERFORMANCE

In this Section, we present a numerical analysis showing the computational performance of all the developed algorithms. More in detail, after having illustrated the hardware setup employed for the tests, a comparison between the computational performance of the CPU and the GPU implementations of the NUFFT-based and OMVM-based algorithms are reported. Finally, the trade-off between computational performance and accuracy of the *p*-series and sub-array partitioning approaches is discussed.

A. Hardware Setup

Sequential implementations have been run on a personal computer with a single-core, Intel Pentium IV processor, with 3 GHz of clock frequency, and equipped with a RAM, 2.0 GBytes sized.

Parallel CUDA codes have been executed on the same personal computer used for the sequential tests, but powered by a GeForce GTX 260 GPU, having 24 multi-processors working at 800 MHz and equipped with a memory, 872 MByte sized.

B. Computational Performance of the Implemented Algorithms

Fig. 6 reports a comparison of computing times for the FFP evaluation by all the implementations discussed in Section III versus the size M of the IA. More in detail, the computing time has been normalized, for all the algorithms, to the corresponding one concerning the case M=80. As it can be seen, the two GPU implementations outperform the corresponding CPU ones, and, in particular, the NUFFT-based implementation on GPU ensures the lowest growth rate. It is worth noting that, the considered GPU computing times (here and in the following) include transfers from/to host (PC memory) to/from device (GPU global memory), so that they represent the effective speed-ups that the GPU can provide against the CPU architecture. In Fig. 6, the M^2 and $M\log M$ trends, agreeing with those for the two considered CPU-based algorithms, are also depicted for higher values of M. Finally, some relevant speed-ups are summarized in Table 1.

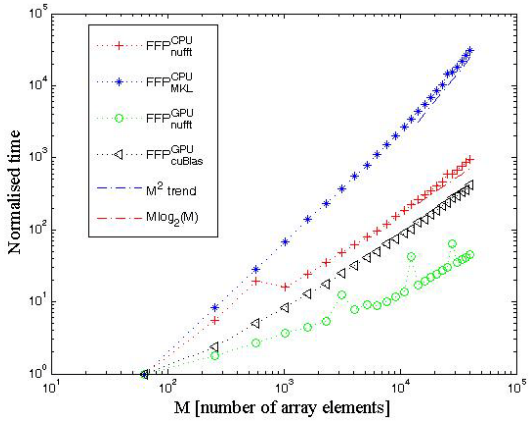

Fig. 6. Growth rates of the computing times for the FFP evaluation by all the implementations illustrated in Section III.

Table 1: Speed-ups among different implementations for an array with M=10^4.

Implementations	Speed-up
CPU NUFFT vs. OMVM	>10
GPU NUFFT vs. OMVM	8
NUFFT GPU vs. CPU	>40

In Fig. 7, the speed-up of the GPU implementation as compared to the CPU one (CPU computing time / GPU computing time) for the NUFFT-based approach against the size *M* of the input data is depicted. As it can be seen, the improvement in the performance for the GPU computation is significant already for small sized arrays (less than 100 elements) and the speed-up factor grows dramatically with the increasing IA dimension *M*.

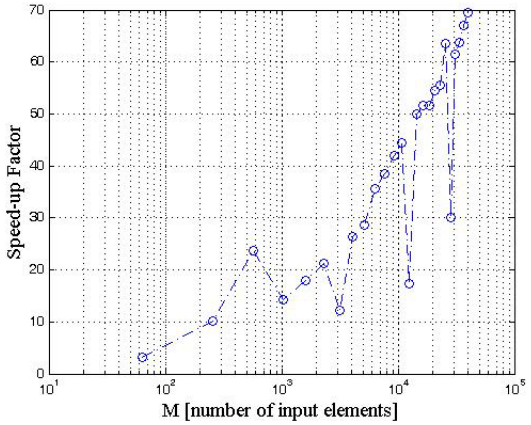

Fig. 7. Speed-up of the GPU vs. CPU implementations for the NUFFT-based approach.

In order to explain such a speed-up, a code profiling has been performed, enlightening that the improved performance is essentially due to the critical filling of the matrix *U* of the sequential case and to the employed effective solution in its parallelization.

We finally note that, the relative drops of GPU/CPU performance are due to particular sizes of the input elements whose data structure does not effectively fit the characteristics (number of shared registers, constant memory size, number of allocatable blocks, number of processors) of the employed hardware. As a consequence, for such particular input dimensions, the code execution is not as massively parallel as it occurs for the others. Nevertheless, the GPU still guarantees a significant speedup as compared to the CPU.

C. Computational Burden of the *p*-Series and of the Sub-Array Partitioning Strategy

We now aim at briefly clarifying, for the sequential implementation, the conditions under which the sub-array partitioning strategy (eq. (14)) becomes computationally convenient with respect to the only *p*-series approach (eq. (13)).

The computational burdens of eqs. (11) and (14) are reported and compared in Appendix D. As it can be expected, it turns out that (eq. (D3)), for sequential implementations, the sub-array partitioning becomes convenient as long as it "favorably" exchanges *p*-series terms with sub-arrays.

Obviously, these conclusions do not hold true when a multi-level parallelism is employed since, in this case, the computation time can be reduced by a *p*-series/sub-array partitioning approach despite the higher number of operations.

A remarkable case when the sub-array partitioning becomes convenient is that (already mentioned) of faceted arrays [35-37], i.e., when the surface *S* is made up by contiguous planar portions (facets) with different relative inclinations. In this case, each facet can be associated to a sub-array which, being flat, requires just 1 *p*-series term. To enlighten this point, we have considered the case of a faceted array having 3 facets and *M*=19600 elements. Tab. 2 summarizes the speed-ups obtained by the sub-array partitioning strategy as compared to *p*-series only evaluations of the FFP, as a function of *P*.

Table 2: Comparing the computational performance of sub-array partitioning vs. *p*-series.

# *p*-series terms	Speed-up
3	1
4	1.34
5	1.67
6	2
7	2.34

V. ACCURACY

In this Section, we present a numerical analysis illustrating the accuracy of the proposed, NUFFT-based strategy, by focusing the attention on two examples: a linear, aperiodic and parabolic, aperiodic arrays.

A. Linear, Aperiodic Array

Let us begin with a linear (non-conformal), aperiodic array. More in detail, we consider an equivalently tapered Chebyshev, 1D array [6,8], made of 2048 elements having uniform

excitations. The elements positions (see Fig. 8) have been properly determined, according to the approach in [6,8], in order to synthesize the same pattern of a Chebyshev array of 1024 uniformly spaced elements having a side lobe level of -26dB. The resulting inter-element spacing of the array elements varies from a minimum of 0.33λ to a maximum of 1.8λ, while the overall array size is 1000λ.

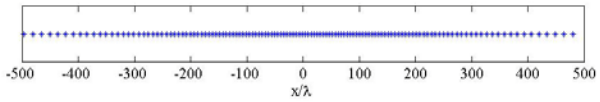

Fig. 8. Elements positions for the Chebyshev array. For the sake of clarity, only the position of one element every 16 are shown.

The adopted synthesis algorithm is based on the optimization of a proper objective functional and requires direct evaluations of the FFP, which have been performed by means of the proposed approach. Obviously, the array being linear, only 1 *p*-series term is required.

Figure 9 shows the synthesized FFP of the equivalently tapered Chebyshev array. The computations have been sequential and the reported "exact evaluation" has been performed by the OMVM approach. The computing times for the proposed and OMVM approaches have been 35ms and 62ms, respectively.

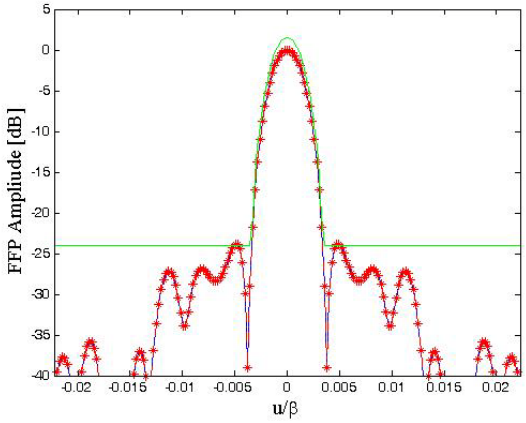

Fig. 9. *u* cut of the FFP of the synthesized equivalently tapered, 1D Chebyshev array. Red stars: proposed approach. Blue dashed line: exact evaluation.

B. Parabolic, Aperiodic Array: Accuracy of the *p*-Series and Sub-Array Partitioning Strategies

In this Subsection, we highlight, with reference to the case of a parabolic, aperiodic array, the accuracy of the *p*-series approach versus the array surface curvature, and the improvements in the accuracy provided by the sub-array partitioning strategy.

To this end, we consider two IAs having the same number (i.e., 65388) of elements lying on two parabolic surfaces having the same diameter (D) but different focal length (f). In particular, the first IA, say IA_1, has a focal/diameter ratio (f/D) equal to 1, while the other, say IA_2, has f/D equal to 1.5. Under these hypotheses, the curvature of IA_2 is smoother than that of IA_1 (see Fig. 10).

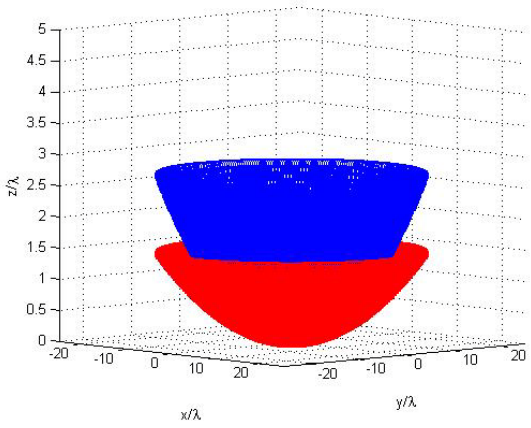

Fig. 10. The two considered parabolic IAs for the analysis of the *p*-series and sub-array partitioning accuracies. Blue: IA_1. Red: IA_2.

The histogram in Fig. 11 indicates the Root Means Square (RMS) error between the exact evaluations of the FFPs for IA_1 and IA_2 (eq. (7)) and their computations with the proposed approach (eq. (14)), against the number of considered *p*-series terms and sub-arrays. Assuming, as acceptable accuracy, the one corresponding to a RMS equal to 1%, Fig. 11 shows that if no partitioning is adopted for IA_1, even six *p*-series terms are not enough to attain the desired precision. On the contrary, partitioning IA_1 into 16 sub-arrays ensures the desired accuracy already with 5 terms and splitting up further the array into 64 or 256 sub-arrays reduces the number of required *p*-series terms to 4 or 3, respectively. Concerning now IA_2, Fig. 11 shows that its milder

curvature gives rise to a faster *p*-series convergence with respect to IA_1. Indeed, 4 *p*-series terms now ensure 1% of RMS error even without sub-array partitioning. Introducing the partitioning in this case allows reducing the *p*-series terms to 3 or 2 with 16 or 256 sub-arrays, respectively.

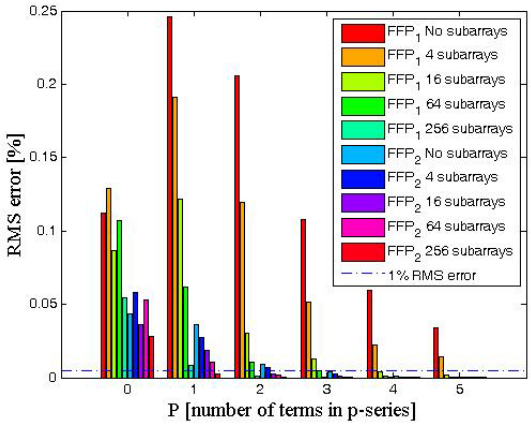

Fig. 11. RMS errors when computing the FFPs of IA_1 and IA_2 against number of *p*-series terms for different numbers of sub-arrays. FFP_1 refers to IA_1, while FFP_2 refers to IA_2.

C. Accuracy of the *p*-Series Approach Versus the Degree of Aperiodicity

We finally consider the case of a 2D IA, whose 16384 elements are aperiodically distributed on a paraboloid with focal length/diameter ratio equal to 0.8, a typical value in the applications. The inter-element spacing varies from 0.35λ to 0.9λ, while the excitation coefficients have been chosen according to:

$$a_m = e^{\alpha\cdot(F/d_m)-1} e^{-i*\beta d_m}, \quad m = 0,1,...,M-1 \qquad (15)$$

where d_m is the distance of the *m*-th array element from the foci, • is the wave-number and α has been properly determined to obtain an amplitude tapering of -4dB at the edge. 3 *p*-series terms have been considered, ensuring a negligible RMS error (~10^{-5}) in the visible region [*0.4,0.4*]x[*0.4,0.4*] $\bullet^2$ of the (*u*,*v*) plane.

Figure 12 compares the FFP evaluation of the considered IA by means of the proposed approach to the exact evaluation (eq. (7)).

The robustness of the proposed approach versus the "degree of aperiodicity" of the array is illustrated in Table 3 which reports the RMS error of the FFP evaluation when an increasing random fraction of array elements is erased, as compared to the setting of Fig. 12, thus increasing the degree of aperiodicity. It should be mentioned that, as long as an increasing random fraction of array elements is erased, the sidelobe intensity rises up, which leads to the higher RMS in Tab. 3. This could be however mitigated by an increasing number of *p*-series terms.

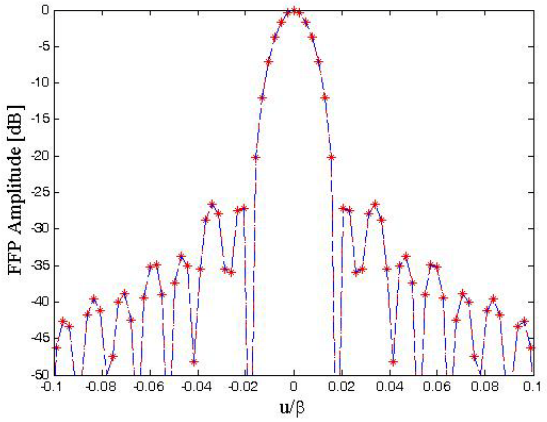

Fig. 12. *u* cut of the FFP of the parabolic, aperiodic array. Red stars: proposed approach with 3 *p*-series terms. Blue dashed line: exact evaluation.

Table 3: RMS vs degree of aperiodicity.

RMS error	Erased elements [%]
10^{-8}	0
2.9^{-3}	1
1.4^{-1}	5
0.27	10

VI. CONCLUSIONS

An approach for the fast analysis of IAs based on the use of the *p*-series expansion and NUFFT routines has been proposed and implemented in both, sequential (CPU) and parallel (GPU) codes.

The performance of the algorithms has been analyzed both in terms of computational efficiency and of achievable accuracy.

In particular:

- both, the sequential and parallel, NUFFT-based approaches are capable of improved performance as compared to (sequential and parallel) algorithms based on OMVMs;

- a sub-array partitioning approach can further reduce the computational burden by speeding-up the convergence of the *p*-series;
- a proper parallel code implementation enables GPU computing to significantly speed-up the execution as compared to that on CPU.

We finally remark that, some of these results can be extended to the FG computation in synthesis algorithm and to the case of volumetric (3D) IAs. Concerning array synthesis, it should be mentioned that often multi-stage synthesis approaches are employed as in [46] and that the most computationally demanding stages can strongly benefit of calculating FFP and FG by the approach here above proposed, committing the computation according to more sophisticated vector models just to the last synthesis steps.

APPENDIX A: THE NUFFT ALGORITHM

According to [17], the "exact" representation of the exponential function $\exp(jp\Delta ux_m)$ is the following:

$$e^{jp\Delta ux_m} = \frac{(2\pi)^{-1/2}}{\Phi(p\Delta u/c)} \sum_{l_1 \in Z} \hat{\Phi}(cx_m - l_1) e^{jp\Delta ul_1/c} \quad \text{(A1)}$$

where $c > 1$ is an "oversampling factor", Φ is a C_0^∞ function with support in $[-\pi,\pi]$ and strictly positive in $[-\pi/c,\pi/c]$, and $\hat{\Phi}$ is its Fourier transform.

Following eq. (A1), any of the NUFFTs in eqs. (13) or (14) can be rewritten as

$$\tilde{b}_{pq} = \sum_{m=0}^{M-1} b_m e^{j(x_m p\Delta u + y_m q\Delta v)} = \underbrace{\frac{(2\pi)^{-1}}{\Phi(p\Delta u/c)\Phi(q\Delta v/c)}}_{\text{Step 3}} \underbrace{\underbrace{\sum_{l_1,l_2 \in Z} \underbrace{\left\{ \sum_{m=0}^{M-1} b_m \hat{\Phi}(cx_m - l_1)\hat{\Phi}(cy_m - l_2) \right\}}_{U(l_1,l_2)\,(\text{Step 1})} e^{jp\Delta ul_1/c} e^{jq\Delta ul_2/c}}_{\text{Evaluate by a standard 2D FFT (Step 2)}}}_{} \quad \text{(A2)}$$

where $b_m = z_m^p a_m'$. Henceforth, we assume to be interested in the values of $\tilde{b}_{pq}$ for p=-N_1/2,…, N_1/2 and q=-N_2/2,…, N_2/2.

The sums over l_1 and l_2 in eq. (A2) require the computation of standard 2D, FFTs. Furthermore, they can be effectively evaluated provided that $\hat{\Phi}$ is small outside some interval [-K,K], so that it is required that Φ has compact support in [-α,α] and $\hat{\Phi}$ is concentrated, as much as possible, in [-K,K]. To this end, a Kaiser-Bessel window Φ is used [17].

The computation of eq. (A2) can be divided into three stages (see also [17]).

Stage 1

For each (x_m,y_m), the nearest equispaced spatial frequencies l_1/c and l_2/c are determined. The samples of the windowing/interpolating functions Φ and $\hat{\Phi}$, respectively, are computed. The inner m summation in eq. (A2), that is, the $U(l_1,l_2)$ function, is calculated.

Stage 2

A standard, 2D FFT routine is performed on U. The output matrix $\hat{U}$ has size $cM \times cM$.

Stage 3

$\hat{U}$ is reduced to an $N_1 \times N_2$ matrix and then scaled with the windowing function $(2\pi)^{-1}/\Phi(p\Delta u/c)\Phi(q\Delta v/c)$.

APPENDIX B: C-STYLE LISTING OF THE SEQUENTIAL NUFFT ALGORITHM

```
//*********
//* STEP 1 *
//*********

for(p=-N1/2;p<N1/2;p++) {
   for(q=-N2/2;q<N2/2;q++) {

   Φpq=Φ(2πp/(cN1))Φ(2πq/(cN2));
                              }
                           }

   for (m=0;m<M;m++) {

      μx,m=round(c*xm);
      μy,m=round(c*ym);

      for (l1=-K;l1<=K;l1++) {

         ix=mod(μx,m +l1+c*N1/2,c*N1);
```

```
        w_xm=Φ̂(c*x_m-(μ_x,m+l_1));

        for(l_2=-K;l_2<=K;l_2++){

           i_y=mod(μ_y,m+l_2+c*N_2/2,c*N_2);
           w_ym=Φ̂(c* y_m-(μ_y,m+l_2));
           U[i_x,i_y]=U[i_x,i_y]+w_xm*w_ym*b_m;
                                }
                            }
                    }

//*********
//* STEP 2 *
//*********

Û =fft(U); // 2D fft routine provided by MKL


//*********
//* STEP 3 *
//*********

for (p=0;p<N_1;p++){
   for(q=0;q<N_2;q++) {

      nufft[p,q]=div(u[(c-
1)*N_1/2*c*N_2+p*c*N_2+(c-1)*N_2/2+q],Φ_pq);
                        }
                    }
```

APPENDIX C: CUDA-STYLE LISTING OF THE PARALLEL NUFFT ALGORITHM

```
//*********
//* STEP 1 *
//*********

/* Generates a 1D grid of threads to evaluate
μ_x,m and μ_y,m. NUM_THREADS = # threads per block
*/
dim3 dimGrid_mu(M/NUM_THREADS,1);
dim3 dimBlock_mu(NUM_THREADS,1);

// Parallel evaluation of μ_x,m and μ_y,m
data_round<<<dimGrid_mu,dimBlock_mu>>>(x_m,y_m,μ_x,
m,μ_y,m);

// Generates a 2D grid of threads to evaluate
Φ_pq
dim3 dimGrid_phi(N_1/BLOCK_SIZE, N_2/BLOCK_SIZE);

// Parallel evaluation of Φ_pq

dim3 dimBlock_phi(BLOCK_SIZE,BLOCK_SIZE);
Φ<<<dimGrid_phi,dimBlock_phi>>>(Φ_pq,N_1,N_2);

/* Generates a 2D grid of threads to evaluate
w_xm and w_ym and evaluates those quantities */

dim3 dimGrid_phi_hat(M,1);
dim3 dimBlock_phi_hat(2*K+1,1);
Φ̂ <<<dimGrid_phi_hat,dimBlock_phi_hat>>>(w_xm,x_m
,μ_x,m,M);
Φ̂ <<<dimGrid_phi_hat,dimBlock_phi_hat>>>(w_ym,y_m
,μ_y,m,M);

// Generates a 1D grid of threads and
evaluates U
dim3 dimBlock_u(2*K+1,2*K+1);
dim3 dimGrid_u(M,1);
U_matrix_evaluation<<<dimGrid_u,
dimBlock_u>>>(b_m,M,μ_x,m,μ_y,m,U,w_xm,w_ym,N_1,N_2);

//*********
//* STEP 2 *
//*********
Û =cuFFT(U);

//*********
//* STEP 3 *
//*********

dim3           dimGrid_scaling(N_1/BLOCK_SIZE,N_2
/BLOCK_SIZE);
dim3 dimBlock_scaling(BLOCK_SIZE,BLOCK_SIZE);
scaling<<<dimGrid_scaling,dimBlock_scaling>>>(
Φ_pq);
```

APPENDIX D: COMPUTATIONAL BURDENS OF THE *p*-SERIES AND SUB-ARRAY PARTITIONING APPROACHES

In the un-partitioned case, according to eq. (13), the number of operations needed to determine the FFP, say N_o, is (neglecting summation operations as compared to multiplications)

$$N_o = M\left\{P\left[4.5c^2\log_2(c^2M)+20K^2+3\right]+2\right\}, \quad \text{(D1)}$$

where K and c are the NUFFT oversampling factor and interpolation length, respectively, and the complexity for the evaluation of a single NUFFT has been determined according to [17].

When the surface is partitioned into N_{sub}• M sub-arrays, the computational complexity becomes (neglecting again summation operations as compared to multiplications)

$$N_o^{sub} = M\cdot\left\{P'\cdot\left[N_{sub}\cdot(4.5c^2\log_2(c^2M)+20K^2+2)+1\right]+N_{sub}+1\right\} \quad \text{(D2)}$$

where P'• P is the number of p-series terms needed in eq. (17) to achieve the same accuracy as for the un-partitioned case. Dividing (D2) by (D1) and enforcing that the ratio is less than one, we have:

$$N_o^{sub} \Big/ N_o \le 1 \Rightarrow$$
$$\Rightarrow N_{sub} \le \frac{9c^2 P \log_2 c + 20K^2 P + \Delta P + 1}{1 + (20K^2 + 9c^2 \log_2 c)(P - \Delta P)} \quad \text{(D3)}$$

where $P'=P-\bullet P$. Eq. (D3) provides a necessary (but not sufficient) condition, in terms of number of sub-arrays N_{sub}, for the sub-array partitioning approach to be computationally convenient as compared to the un-partitioned case, for a fixed accuracy. Obviously, in eq. (D3), $\bullet\theta \quad P<P$ since the partitioning can reduce the number of *p*-series terms at most to $P'=1$. Generally speaking, $\bullet P$ is a function of N_{sub} and it increases with the number of sub-array partitions.

To be more specific we observe that, typically, the values of the NUFFT oversampling factor and interpolation length are 2 and 6, respectively. Substituting these values in eq. (D3), we get:

$$N_{sub} \le \frac{756P + \Delta P + 1}{1 + 756(P - \Delta P)} \cong \frac{P}{(P - \Delta P)} \le P \,. \quad \text{(D4)}$$

REFERENCES

[1] O.M. Bucci, G. D'Elia, G. Mazzarella, and G. Panariello, "Antenna pattern synthesis: a new general approach", *Proc. of the IEEE*, vol. 82, no. 3, pp. 358-371, Mar. 1994.

[2] A. Capozzoli and G. D'Elia, "Global optimization and antennas synthesis and diagnostics, part one: concepts, tools, strategies and performances", *Progr. Electromagn. Res. PIER*, vol. 56, pp. 195-232, 2006.

[3] A. Capozzoli and G. D'Elia, "Global optimization and antennas synthesis and diagnosis, part two: applications to advanced reflector antennas synthesis and diagnosis techniques", *Progr. Electromagn. Res. PIER*, vol. 56, pp. 233-261, 2006.

[4] L. Josefsson and P. Persson, *Conformal array antenna theory and design*, J. Wiley & Sons., New York, 2006.

[5] G. Caille, I. Lager, L.P. Ligthart, C. Mangenot, A.G. Roederer, G. Toso, and M.C. Viganò, "Aperiodic arrays for multiple beam satellite applications," *Proc. of the 11th Int. Symp. on Microw. Opt. Tech.*, pp. 419-422, Dec. 2007.

[6] A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, and P. Vinetti, "FFT & aperiodic arrays with phase-only control and constraints due to super-directivity, mutual coupling and overall size", *Proc. of the 30th ESA Antenna Workshop on Antennas for Earth Observ., Science, Telecomm. and Navig. Space Missions*, May 2008.

[7] A. Capozzoli, C. Curcio, G. D'Elia, and A. Liseno, "Fast power pattern synthesis of conformal reflectarrays", *Proc. of the IEEE Antennas Prop. Symp.*, pp. 1-4, July 2008.

[8] A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, and P. Vinetti, "FFT & equivalently tapered arrays", *Proc. of the XXIX URSI General Assembly*, Aug. 2008.

[9] J.H. Doles III and F.D. Benedict, "Broad-band array design using the asymptotic theory of unequally spaced arrays," *IEEE Trans. Antennas Prop.*, vol. 36, no. 1, pp. 27-33, Jan. 1988.

[10] A. Akdagli and K. Guney, "Shaped-beam pattern synthesis of equally and unequally spaced linear antenna arrays using a modified tabu search algorithm," *Microw. Optical Tech. Lett.*, vol. 36, no. 1, pp. 16-20, Jan. 2003.

[11] J. Huang, M. Lou, A. Feria, and Y. Kim, "An inflatable L-band microstrip SAR array", *Proc. of the IEEE Antennas Prop. Int. Symp.*, pp. 2100-2103, Jun. 1998.

[12] R.E. Collin, *Antennas and radiowave propagation*, McGraw-Hill, New York, 1985.

[13] A.W. Rudge, K. Milne, A.D. Olver, and P. Knight, *The Handbook of Antenna Design Vol. 2*, London, Peter Peregrinus, 1983.

[14] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Math. Comput.*, vol. 19, no. 90, pp. 297-301, Apr. 1965.

[15] V. Galindo-Israel and R. Mittra, "A new series representation of the radiation integral with application to reflector antennas", *IEEE Trans. Antennas Prop.*, vol. AP-25, no. 5, pp. 631-641, Sept. 1977.

[16] O.M. Bucci, G. Franceschetti, and G. D'Elia, "Fast analysis of large antennas – a new computational philosophy", *IEEE Trans. Antennas Prop.*, vol. AP-28, no. 3, pp. 306-310, May 1980.

[17] K. Fourmont, "Non-equispaced fast Fourier transforms with applications to tomography,"

J. Fourier Anal. Appl., vol. 9, no. 5, pp. 431-450, Sept. 2003.

[18] J. Y. Lee and L. Greengard, "The type 3 nonuniform FFT and its applications", *J. Comput. Phys.*, vol. 206, n. 1, pp. 1-5, Jun. 2005.

[19] J.D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing", *Proc. of the IEEE*, vol. 96, no. 5, pp. 879-899, May 2008.

[20] www.top500.org .

[21] T. R. Halfhill, "Parallel processing with CUDA", *Microproc. Rep.*, http://www.nvidia.com/docs/IO/55972/220401_Reprint.pdf, Jan. 28, 2008.

[22] S. S. Stone, J. P. Haldar, S. C. Tsao, W. M. Hwu, B.P. Sutton, and Z.P. Liang, "Accelerating advanced MRI reconstructions on GPUs", *J. Parallel Distr. Comp.*, vol. 68, no. 10, pp. 1307-1318, Oct. 2008.

[23] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications", *IEEE Antennas Prop. Mag.*, vol. 47, no. 6, pp. 71-78, Dec. 2005.

[24] D. Kirk and H. M. Hwu, *CUDA Textbook*, in press.

[25] M. Bläser, "Lower bounds for the multiplicative complexity of matrix multiplication", *Comput. Complex.*, vol. 8, no. 3, pp. 203-226, Dec. 1999.

[26] K. Atkinson and D. D. K. Chien, "A fast matrix-vector multiplication method for solving the radiosity equation", *Adv. in Comput. Math.*, vol. 12, no. 2-3, Feb. 2000, pp. 151-174.

[27] http://matrixprogramming.com/MatrixMultiply/

[28] D. Sundararajan, *The Discrete Fourier Transform: Theory, Algorithms and Applications*, Singapore, Word Scientific, 2001.

[29] F. Smithies, *Integral equations*, Cambridge, Cambridge University Press, 1958.

[30] J. P. Boyd, "A fast algorithm for Chebyshev, Fourier and sinc interpolation onto an irregular grid", *J. Comput. Phys.*, vol. 103, no. 2, pp. 243-257, Dec. 1992.

[31] A. Dutt and V. Rokhlin, "Fast Fourier transforms for nonequispaced data", *SIAM J. Sci. Comput.*, vol. 14, no. 6, pp. 1368-1393, Nov. 1993.

[32] Q. H. Liu and N. Nguyen, "An accurate algorithm for nonuniform fast Fourier transforms (NUFFT's)", *IEEE Microw. Guided Wave Lett.*, vol. 8, no. 1, pp. 18-20, Jan. 1998.

[33] J. A. Fessler, B. P. Sutton, "Nonuniform fast Fourier transforms using min-max interpolation", *IEEE Trans. Signal Proc.*, vol. 51, no. 2, pp. 560-574, Feb. 2003.

[34] W. P. M. N. Keizer, "Large planar array thinning using iterative FFT techniques", *IEEE Trans. Antennas Prop.*, vol. 57, no. 10, pp. 3359-3362, Oct. 2009.

[35] H. Legay, B. Salome, E. Labiole, M. A. Milon, D. Cadoret, R. Gillard, R. Chaharmir,and J. Shaker, "Reflectarrays for satellite telecommunication antennas", *Proc. of the 2nd Europ. Conf. on Antennas Prop.*, Nov. 11-16, 2007.

[36] A. G. Roederer, "Reflectarray antennas", *Proc. of the 3rd Europ. Conf. on Antennas Prof.*, pp. 18-22, Mar. 2009.

[37] A. Capozzoli, C. Curcio, G. D'Elia, A. Liseno, D. Bresciani, and H. Legay, "Fast phase-only synthesis of faceted reflectarrays", *Proc. of the 3rd Europ. Conf. on Antennas Prop.*, pp. 1329-1333, Mar. 23-27, 2009,

[38] N. Yuan, T. S. Yeo, X. C. Nie, and L. W. Li, "A fast analysis of scattering and radiation of large microstrip antenna arrays", *IEEE Trans. Antennas Prop.*, vol. 51, no .9, pp. 2218-2226, Sept. 2003.

[39] M. Frigo and S.G. Johnson, "The design and implementation of FFTW3", *Proc. of the IEEE*, vol. 93, no. 2, pp. 216-231, Feb. 2005.

[40] The Numerical Algorithms Group, *Intel Math Kernel Library Reference Manual*, Intel Corporation, 2001.

[41] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a quantitative approach*, San Francisco, USA, Morgan Kauffman Publisher, 2007.

[42] NVIDIA CUDA Reference Manual, v. 2.0, Jun. 2008.

[43] CUDA cuFFT Library, Oct. 2007.

[44] CUDA cuBLAS Library, Sept. 2007.

[45] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S.H.M. Buijssen, M. Grajewski, and S. Turek, "Exploring weak scalability for

FEM calculations on a GPU-enhanced cluster", *Parallel Comp.*, vol. 33, no. 10-11, pp. 685-699, Nov. 2007.

[46] A. Capozzoli, C. Curcio, G. D'Elia, and A. Liseno, "Power pattern synthesis of multifeed reconfigurable reflectarrays", *Proc. of the 29th ESA Antenna Workshop on Multiple Beams Reconfig. Antennas*, Apr. 2007.

Amedeo Capozzoli graduated (summa cum laude) in Electronic Engineering and received the PhD degree in Electronic Engineering and Computer Science, both from the University of Naples Federico II in 1994 and 2000 respectively.
In 1996**,** he was awarded the Telecom Italia Prize for the best degree thesis in Electronic Engineering discussed at the University of Naples Federico II in the Academic Year 1994-1995**.** In November 1999**,** he won the open competition for the post of Researcher at the University of Naples Federico II. In September 2002, he was awarded the Barzilai Prize for young scientists at the XIV Riunione Nazionale di Elettromagnetismo. In April 2003**,** he won the open competition for the post of Associate Professor at Politecnico di Milano. Since January 2005**,** he has been Associate Professor of Electromagnetic Fields at the University of Naples Federico II. His research interests include synthesis and diagnosis of radiating systems, inverse-scattering techniques, advanced measurement techniques, and the restoring of aberrations due to propagation through random media.

Claudio Curcio received the Laurea degree (summa cum laude) in Electronic Engineering and the PhD degree in Electronic and Telecommunication Engineering, both from the Università di Napoli Federico II, Naples, Italy, in 2002 and 2005**,** respectively. In 2006-2007**,** he held a post-doctoral position at the University of Naples Federico II. He is currently a Researcher at Università di Napoli Federico II**.** His main fields of interest are antenna measurements, phaseless near-field/far-field transformation techniques, optical beamforming techniques for array antennas, and reflectarray synthesis.
In February 2002, he was the recipient of the Optimus Award at the SIMAGINE 2002 "Worldwide GSM & Java Card Developer Contest."

Giuseppe D'Elia was born in Italy in 1950. He received the EE degree (summa cum laude) from the Università di Napoli, Naples, Italy. From 1983 to 1987, he was with the IRECE Institute of the National Research Council (CNR). From 1987 to 1990, he was an Associate Professor of Antennas and Propagation at the Università di Salerno, Salerno, Italy. From 1990 to 2001, he was an Associate Professor of Antennas at the Dipartimento di Ingegneria Elettronica e delle Telecomunicazioni, Università di Napoli Federico II, Naples, Italy, where, since 2001, he has been a full Professor of Electromagnetic Fields.
Prof. D'Elia has been a visiting scientist at many microwave labs, such as the Electrical Engineering Research Laboratory, University of Texas at Austin; the Popov Society, Moscow, Russia; the Institute of Electrical Engineering of the Queen Mary College at the University of London; the Microwave Laboratory of the Marconi GEG, Great Britain; and JPL, Pasadena, California. His main fields of interest include the transient behavior of antennas in dispersive media, efficient and non-redundant techniques for analysis and synthesis of reflector and array antennas, phase-less near-field antenna characterization, wavefront reconstruction from amplitude data by blind deconvolution, NF-FF transformation techniques, non-redundant representation of radiated or scattered fields, and inverse scattering and remote sensing from polarimetric data. Prof. D'Elia was awarded the 1999 Honorable Mention for the H. A. Wheeler Applications Prize Paper Award of the IEEE Antennas and Propagation Society.

Angelo Liseno was born in Italy in 1974. He received the Laurea degree (summa cum laude) and the PhD degree in 1998 and 2001, respectively, both in Electrical Engineering, from the Seconda Università di Napoli, Italy. In 2001-2002, he held a postdoctoral position at the Seconda Università di Napoli. In 2003-2004, he was a research scientist with the Institut flir Hochfrequenztechnik und Radarsysteme of the Deutsches; Zentrum für Luft- und Raumfahrt (DLR), Oberpfaffenhofen, Germany. Since 2005**,** he has been a Researcher with the Università di Napoli Federico II**,** Dipartimento di Ingegneria Elettronica e delle Telecomunicazioni, Naples, Italy. His main fields of interest are phaseless

near-field/far-field transformation techniques, remote sensing, and inverse scattering.

Pietro Vinetti was born in Naples in 1978**.** He received the Laurea degree in Telecommunication Engineering from the University of Naples Federico II**,** Naples, Italy, in 2003**.** Since 2005**,** he has been a PhD student in Electronic and Telecommunication Engineering at University of Naples Federico II, with electromagnetic as his area of interest. His research activity is mainly focused on the development of innovative near-field antenna-characterization systems, based on non-invasive dielectric probes and phaseless near-field/far- field transformation techniques.