Benchmarking Parallel Computing Solutions for Rough Surface Scattering Canonical Problems

Pasquale Imperatore, *Senior Member, IEEE*, Francesco Gregoretti, Nicolas Pinel, Mehwish Nisar, Christophe Bourlier, Diego Romano, and Antonio Iodice, *Senior Member, IEEE*

***Abstract*—Accurate modeling of electromagnetic (EM) wave scattering from large-scale ground profiles is essential in remote sensing applications. However, the computational burden associated with the underlying numerical methods—particularly those based on full-wave solvers—can become prohibitive. To address the simulation of scattering from 1-D random rough surfaces, we have developed specialized parallel implementations of a method of moments (MoM)-based solver, optimized for various heterogeneous computing platforms, including shared-memory multicore central processing units (CPUs), manycore graphics processing units (GPUs), and distributed-memory multinode systems. For solving the resulting dense linear systems in parallel via LU factorization across different architectures, we leverage cutting-edge numerical libraries developed in the high-performance computing (HPC) research community, such as parallel linear algebra software for multicore architecture (PLASMA), matrix algebra on GPU and multicore architecture (MAGMA), and scalable linear algebra package (ScaLAPACK). These libraries serve as parallelized counterparts to the well-known LAPACK suite, each tailored to exploit distinct levels of hardware parallelism. Representative case studies are presented to validate the numerical behavior of the implemented solvers across diverse computational architectures. Parallel performance is thoroughly assessed through empirical benchmarks, demonstrating significant speedup, scalability, and computational efficiency. The developed parallel prototypes effectively harness the different degrees of parallelism available, enabling faster EM simulations, evaluation of scattering phenomena at higher resolutions, and higher fidelity modeling of electrically large surfaces. This work paves the way for the efficient computation of EM scattering from complex and/or multilayered structures, facilitating large-scale simulations on current supercomputing infrastructures.**

Received 17 July 2025; revised 10 February 2026; accepted 2 March 2026. This work was supported in part by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data, and Quantum Computing, Funded by European Union in the Next Generation EU Plan Framework; and in part by European Union through Italian National Recovery and Resilience Plan (NRRP) of Next Generation EU, Partnership on "Telecommunications of the Future" (Program RESTART), under Grant PE0000001. *(Corresponding author: Pasquale Imperatore.)*

Pasquale Imperatore and Mehwish Nisar are with the Institute for Electromagnetic Sensing of the Environment (IREA), National Research Council (CNR), 80124 Naples, Italy (e-mail: imperatore.p@irea.cnr.it; nisar.m@irea.cnr.it).

Francesco Gregoretti and Diego Romano are with the Institute for High Performance Computing and Networking (ICAR) of CNR, 80131 Naples, Italy, also with the INdAM Research Group GNCS, 00185 Rome, Italy, and also with the ICAR-CNR INdAM Research Unit, 80131 Naples, Italy (e-mail: francesco.gregoretti@icar.cnr.it; diego.romano@cnr.it).

Nicolas Pinel is with Icam Ouest School of Engineering—Nantes Campus, 44470 Carquefou, France, and also with IETR, Polytech Nantes, Nantes Université, 44306 Nantes, France (e-mail: nicolas.pinel@icam.fr).

Christophe Bourlier is with IETR, Polytech Nantes, Nantes Université, 44306 Nantes, France (e-mail: christophe.bourlier@univ-nantes.fr).

Antonio Iodice is with the Department of Electrical Engineering and Information Technology, University of Naples Federico II, 80125 Naples, Italy (e-mail: iodice@unina.it).

Digital Object Identifier 10.1109/TAP.2026.3675744

***Index Terms*—Dense linear algebra (DLA), electromagnetic (EM) scattering, high-performance computing (HPC), method of moment (MoM), parallel algorithms, radar cross section, rough surface.**

I. Introduction

ELECTROMAGNETIC scattering by rough surfaces is nowadays a problem of considerable interest, both theoretically and practically, in various fields of physics and engineering, such as remote sensing, radio wave propagation, and optics [1], [2].

Galileo's empirical observations paved the way for our modern understanding of celestial phenomena, delving into the phenomenon of scattering from the lunar surface [3]. He arrived at the conclusion that the lunar surface was rough through observations made with his telescope, although he did not possess the modern tools of electromagnetic (EM) theory.

Lord Rayleigh conducted one of the first comprehensive studies on rough surface scattering in the late 19th century [4]. More recently, the EM scattering problem has been addressed using both analytical and numerical asymptotic methods, and various techniques exist for single or layered rough surfaces [1], [2], [5], [6]. In particular, the different available analytical solutions are valid for different classes of rough interfaces. For instance, solutions according to the small perturbations method (SPM) [7] and the Kirchhoff approximation (KA) [8] are available for single or layered rough interfaces.

Moreover, numerical methods for evaluating scattering from rough surfaces are widely used, as they allow for accurate simulation of complex geometries and material properties, making them valuable tools for studying rough surface scattering in diverse applications. In particular, the method of moments (MoM) is a well-known numerical technique among integral equation-based methods, particularly advantageous for analyzing scattering from rough surfaces with arbitrary shapes and roughness profiles [9], [10], [11].

For the purposes of this investigation, we focus on the numerical scattering evaluation from 1-D, perfectly conducting, randomly rough surfaces obtained using MoM, which is assumed here as the reference canonical problem. Nonetheless, according to MoM, solving scattering from electrically large random rough surfaces might become a time-consuming task, with computer physical memory space typically imposing an upper bound on the feasible problem size.

In the framework of EM scattering from rough surfaces, numerical accelerations are implemented by introducing relevant physical approximations to reduce the computing time and/or memory space requirement. Among these methods, the

stationary iterative forward–backward (FB) method [6], [12], [13], [14], [15], [16] combined with the spectral acceleration (SA) [6], [17], [18], [19], [20] is the most efficient one for the scattering from highly conducting rough sea surfaces. For slightly rough surfaces, the banded matrix iterative approach/canonical grid (BMIA-CAG) [6], [21], [22], [23] is also efficient.

Another approach, the one followed in this study, aims to achieve high performance without sacrificing the accuracy of the numerical algorithms. For this purpose, the computational power of high-performance computing (HPC) systems is indispensable for effectively addressing a wide range of applications across key scientific and engineering fields. These applications include, for instance, weather forecasting [24], synthetic aperture radar (SAR) [25], and medical image [26] processing.

As a matter of fact, current HPC systems typically consist of clusters comprising many compute nodes, each holding a certain number of multicore central processing units (CPUs) and graphics processing units (GPUs) connected through a high-speed network. GPUs are manycore accelerators commonly used as parallel processors to offload computationally intense tasks from CPUs. Although parallelism has become the dominant paradigm in computer architecture, the heterogeneity in modern computing systems poses significant challenges to the inherent design of dedicated parallel algorithms for scientific applications and specific computing problems. To fully take advantage of the hierarchical level of parallelism offered by the pervasive contemporary HPC infrastructures, a redesign of the well-established numerical algorithms is typically required.

In particular, despite the high degree of parallelism offered by currently available computational platforms, no attempt has been made to comprehensively explore and design appropriate parallelization strategies for the aforementioned canonical scattering problem. This gap hinders our understanding of the potential performance gains achievable through employing HPC methodologies in this particular scenario. To the best of our knowledge, within the current literature, only a few very specific parallel approaches have been proposed [27], [28], [29], [30], [31], [32]. For instance, in [27], the parallelized conjugate gradient method (CGM) is investigated to evaluate the composite EM scattering for a 2-D target above a 1-D large-scale rough surface. A GPU-based finite-difference time-domain (FDTD) algorithm for rough sea surface scattering is presented in [28]. In [29], a GPU-based computation is adopted to accelerate the small slope approximation (SSA) method to evaluate scattering from a large rough surface. A parallel algorithm for airplane scattering with MoM using thousands of CPUs is presented in [30]. An investigation into GPU-based acceleration for MoM applied to scattering from conducting objects was conducted in [31].

To overcome the aforementioned issues and challenges, this article systematically presents, analyzes, and compares three distinct parallel versions of the solver for the considered canonical scattering problem. These versions are targeted to different components of contemporary HPC systems, including (shared-memory) multicore CPUs, manycore GPUs, and (distributed-memory) multinode architectures.

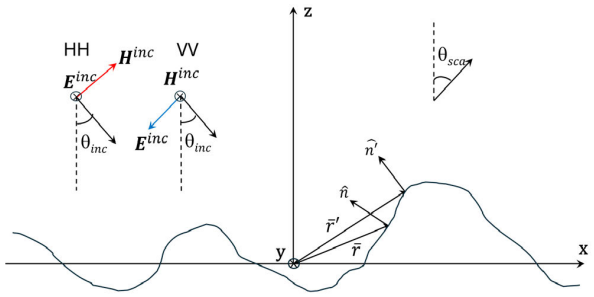


Fig. 1. Schematic of the EM scattering problem.

This article aims to develop portable and efficient implementations of a solver across a wide range of currently available heterogeneous architectures, thereby elucidating the advantages and limitations of the diverse parallel approaches. Consequently, the methodology for developing a high-performance solver leveraging the degrees of parallelism offered by the different architectures addresses two primary objectives: 1) to solve the canonical problem faster and 2) to tackle larger-size problems, where combined processing time and memory requirements pose challenges.

Furthermore, assessing and comparing the overall performance of various implementations of the solver across different parallel architectures can provide valuable insight into their relative performance and scalability for specific workloads, and can indeed be considered a benchmarking process.

To support our experimental investigation, we use two powerful HPC platforms located in Italy: the Ben cluster, hosted by CNR at the Naples ICAR headquarters, and the EuroHPC pre-exascale supercomputer Leonardo, which ranks 9th in the TOP500 global list [33], hosted by CINECA at the Bologna Technopole. Some very preliminary results of the present work were provided in [32], but here we describe in detail the adopted methodology, the performed study, and illustrate all the obtained results.

This article is organized as follows. In Section II, the theoretical principles underlying the canonical scattering problem are reviewed. The adopted HPC methodology is introduced in Section III. Prototype implementations and inherent performance analysis are discussed in Sections IV and V, respectively. Finally, conclusions are drawn in Section VI.

II. Canonical Scattering Problem

This section introduces the canonical EM scattering problem and its numerical solution using the MoM. Inherent computational requirements and limitations are also addressed.

A. Problem EM Formulation

The geometry of the problem is depicted in Fig. 1. We consider a plane wave impinging on a perfectly conducting rough surface, and we assume that the EM field and the surface height profile $z(x)$ are constant along the y direction. Throughout the article, in the field expressions, a time factor $e^{-i\omega t}$ is understood and suppressed. If the incident wave is horizontally polarized so that the incident electric field is

$$\mathbf{E}^{inc}(\mathbf{r}) = E^{inc}(\mathbf{r})\,\hat{\mathbf{y}} = e^{ik(x\sin\theta_{inc} - z\cos\theta_{inc})}\,\hat{\mathbf{y}} \tag{1}$$

then the surface current density can be evaluated by solving the following integral scalar equation:

$$E^{inc}(\mathbf{r}) = -i\omega\mu_0 \int_l \phi(\mathbf{r},\mathbf{r}')\, J_S(\mathbf{r}')\, dl' \tag{2}$$

where k is the free-space propagation constant, θ_{inc} is the incidence angle, μ_0 is the magnetic permeability of vacuum, $\mathbf{J}_s = J_S\hat{\mathbf{y}} = \hat{\mathbf{n}} \times \mathbf{H}$ is the surface current density, $\hat{\mathbf{n}}$ is the surface normal unit vector, $\mathbf{H}$ is the surface magnetic field

$$\phi(\mathbf{r},\mathbf{r}') = \frac{i}{4} H_0^{(1)}\left(k\left\|\mathbf{r}-\mathbf{r}'\right\|\right) \tag{3}$$

is the Green function in the 2-D space, $H_0^{(1)}(\cdot)$ is the Hankel function of the first kind and order zero, l is the surface profile, and the points $\mathbf{r} = x\hat{\mathbf{x}} + z(x)\hat{\mathbf{z}}$ and $\mathbf{r}' = x'\hat{\mathbf{x}} + z(x')\hat{\mathbf{z}}$ both belong to the surface profile, see Fig. 1, and $dl' = \sqrt{1 + ((dz)/(dx'))^2}dx'$.

Once the surface current density has been evaluated by solving the integral (2), the far-zone scattered field can be computed as follows:

$$E^s(\mathbf{r}) = \frac{\eta k e^{i(kr-\pi/4)}}{\sqrt{8\pi kr}} \int_l J_S(\mathbf{r}')\, e^{-i\hat{\mathbf{k}}_S\cdot\mathbf{r}'}\, dl' \tag{4}$$

where $\mathbf{r} = x\hat{\mathbf{x}} + z\hat{\mathbf{z}}$ is the point in the far zone, $\mathbf{r}' = x'\hat{\mathbf{x}} + z(x')\hat{\mathbf{z}}$ belongs to the surface profile, $\hat{\mathbf{k}}_S$ is the unit vector indicating the scattering direction, and η is the free-space intrinsic impedance.

Similarly, if the incident wave is vertically polarized, so that the incident magnetic field is

$$\mathbf{H}^{inc}(\mathbf{r}) = H^{inc}(\mathbf{r})\,\hat{\mathbf{y}} = e^{ik(x\sin\theta_{inc} - z\cos\theta_{inc})}\,\hat{\mathbf{y}} \tag{5}$$

then the surface magnetic field can be evaluated by solving the following integral scalar equation:

$$H^{inc}(\mathbf{r}) = \frac{1}{2}H(\mathbf{r}) + \int_l \left[\hat{\mathbf{n}}'\cdot\nabla\phi(\mathbf{r},\mathbf{r}')\right] H(\mathbf{r}')\, dl' \tag{6}$$

where $\mathbf{H} = H\hat{\mathbf{y}}$ is the surface magnetic field. Once the latter has been evaluated by solving the integral (6), the far-zone scattered field can be computed as follows:

$$H^s(\mathbf{r}) = \frac{k e^{i(kr-\pi/4)}}{\sqrt{8\pi kr}} \int_l \left[-\hat{\mathbf{n}}'\cdot\hat{\mathbf{k}}_S\right] H(\mathbf{r}')\, e^{-i\hat{\mathbf{k}}_S\cdot\mathbf{r}'}\, dl'. \tag{7}$$

B. Brief Review of the MoM

The MoM is a numerical method that has been used extensively to solve EM wave scattering problems, and several excellent textbooks [9], [10], [11] have been written on this active subject. This method leads to a full matrix equation, which can be solved via matrix inversion. In this article, the MoM is applied with the point-matching method and by using pulse basis functions. The main advantage of this discretization is that the elements of the impedance matrix are easy to calculate and efficient for scattering from rough surfaces.

Equations (2) and (6) can be cast as $\mathcal{L}(f) = g$, where $\mathcal{L}$ is an integral operator or integral-differential operator, f is the unknown function, and g is a given function related to the incident field. The MoM converts this boundary integral equation into a linear system defined as

$$\bar{\boldsymbol{Z}}\boldsymbol{X} = \boldsymbol{b} \tag{8}$$

where $\bar{\boldsymbol{Z}}$ is the so-called impedance matrix (which depends on the surface shape); $\boldsymbol{X}$ is the unknown vector to be resolved; and $\boldsymbol{b}$ is a given vector, which is related to the discretization of the incident field on the surface. The mathematical expression of the matrix elements can be found in [10]; for the reader's convenience, they are also reported in Appendix A. If the matrix $\bar{\boldsymbol{Z}}$ is not singular, the solution to the problem consists mainly of inverting the impedance matrix as

$$\boldsymbol{X} = \bar{\boldsymbol{Z}}^{-1}\boldsymbol{b}. \tag{9}$$

C. Computational Complexity, Memory Requirement, and Limitations

The calculation of the impedance matrix needs to store N^2 complex numbers, where N is the surface sampling number (i.e., number of unknowns). In addition, from a conventional LU decomposition, the number of multiplications to calculate $\bar{\boldsymbol{L}}$ (lower triangular matrix) and $\bar{\boldsymbol{U}}$ (upper triangular matrix) is $\mathcal{O}(N^3)$. For a huge problem, the LU decomposition can be time-consuming, and the memory requirement significantly increases. To overcome this issue, accelerations based on the introduction of physical assumptions, such as FB [6], [12], [13], [14], [15], [16] combined with SA [6], [17], [18], [19], [20] and banded matrix iterative approach/canonical grid (BMIA-CAG) [6], [21], [22], [23], can be implemented. This article explores an alternative approach that does not require the introduction of simplifying physical assumptions.

III. Parallel Computing Design Methodology

This section presents the rationale behind the adopted parallel strategies for the computational problem under consideration. The aim is to fully exploit the available degree of parallelism offered by various modern HPC platforms, including multicore and manycore (shared-memory) architectures, as well as multinode (distributed-memory) architectures [34], [35], [36]. Algorithmic details relevant to the developed efficient parallel solutions are provided in Section IV.

According to Section II, MoM-based computations involve solving a large and dense linear system of equations (LSE) arising from discretizing integral equations in the form (8), where $\bar{\boldsymbol{Z}} \in \mathbb{C}^{N\times N}$ is a dense, nonsymmetric, and complex-valued matrix, $\mathbf{b} \in \mathbb{C}^N$ is the input vector, and $\boldsymbol{X} \in \mathbb{C}^N$ is the unknown solution vector, with N denoting the number of unknowns. Therefore, the computational problem considered includes three main different tasks: the first step involves calculating the elements of the impedance matrix $\bar{\boldsymbol{Z}}$, while the second step consists of the solution of the matrix (8), which is indeed the computationally dominating operation for the canonical solver under consideration (Section II-C). Finally, the scattered field (observed quantities) can be obtained with relatively negligible additional computations.

In general, achieving scalability of parallel algorithms across multiple processing units requires that the granularity of the computation be adjustable [34], [35], [36]. Accordingly, for the specific problem at hand, adopting an approach that relies on some form of partitioning with adjustable block sizes might be appropriate to exploit the available parallelism.

We first discuss the parallelization of impedance matrix computation (Section III-A), followed by a detailed focus on

specific strategies for solving the inherent LSE on different parallel architectures (Section III-B). The case in which the impedance matrix $\bar{\boldsymbol{Z}}$ exceeds the main physical memory of the system is also addressed, as discussed in the following.

A. Impedance Matrix Parallel Computation

Parallelism is achieved by assigning each computing unit the task of calculating a subset of elements in the impedance matrix. To achieve the performance, redundant computations must be minimized, and the workload must be evenly distributed across all processing units. In particular, different parallel schemes can be used for impedance matrix computation on the selected architectures.

On shared-memory architectures, the computational workload is distributed across multiple cores within a single computing node, ensuring optimal performance by adjusting the block size of the data layout.

On distributed-memory architectures, the computation of the impedance matrix is partitioned into blocks assigned to all the engaged computing nodes and stored in their local memories, allowing the creation of a distributed version of the matrix $\bar{\boldsymbol{Z}}$. In this case, optimal performance in impedance matrix computation entails controlling the block size and the communication pattern of computing units, as further discussed in Section IV.

It is worth highlighting that, compared to a shared-memory system, a distributed-memory platform (in which each processor can access only its own local memory) offers the advantage of a significantly larger potential memory pool. This enables large-scale MoM simulations requiring extensive memory and computational resources.

B. Dense LSE Parallel Solution

To leverage the parallelism offered by modern HPC platforms, we strategically rely on established, open-source, high-performance dense linear algebra (DLA) libraries instead of developing new, specialized parallel LSE solvers from scratch. Specifically, we prioritize currently available libraries that offer efficient and accurate solvers for dense LSEs on different architectures. This approach has proven effective, enabling high performance, scalability, and portability across a wide range of computing platforms while also alleviating the need for laborious parallelization efforts.

This article focuses on one of the most efficient and numerically stable methods for solving dense LSEs, which involves LU decomposition (also known as LU factorization) followed by forward and backward substitutions. Partial (row) pivoting can also be incorporated to ensure both numerical stability and performance, resulting in a factorization $\bar{\boldsymbol{P}}\bar{\boldsymbol{Z}} = \bar{\boldsymbol{L}}\bar{\boldsymbol{U}}$, where $\bar{\boldsymbol{P}}$ is a permutation matrix that applies a number of row interchanges to $\bar{\boldsymbol{Z}}$, $\bar{\boldsymbol{U}}$ is the upper triangular, and $\bar{\boldsymbol{L}}$ is the lower triangular [37]. Parallel implementations of LU decomposition typically rely on block-based strategies to achieve computational efficiency. The underlying logic of block LU decomposition is reviewed in Appendix B.

Accordingly, we have identified state-of-the-art libraries that include LU-based LSE numerical computation functions on diverse heterogeneous architectures. The three selected libraries, which can somehow be considered parallel counterparts of the popular linear algebra PACKage (LAPACK) library [38], [39], are introduced in the following.

1) Parallel Linear Algebra Software for Multicore Architectures: It is a numerical library for solving problems in DLA, specifically designed to take advantage of shared-memory architectures based on multicore processors [40].

Parallel linear algebra software for multicore architecture (PLASMA) is optimized to achieve the highest possible performance through *thread-level* parallelism with fine granularity. To achieve efficient multithreaded execution, PLASMA is built around the following concepts: tile matrix layout, tile algorithms, and task-based scheduling. It operates on matrices in the tile layout, in which the matrix is subdivided into square blocks called tiles. In tile algorithms, square tiles are relatively small, so multiple cores can operate independently on different tiles. This approach provides fine-grained parallelism while allowing tiles to be cached and fully processed before being evicted from the cache, minimizing cache misses. PLASMA supports a substantial subset of LAPACK's functionality and offers a collection of routines for solving linear systems of equations using the open multiprocessing (OpenMP) standard [41], [42]. In particular, its recent version offers an extensive collection of optimized routines and specific routines for solving general systems of linear equations based on LU factorization with partial (row) pivoting [43].

2) Matrix Algebra on GPU and Multicore Architectures: It is a DLA library specifically developed to fully exploit heterogeneous CPU-GPU architectures [44], [45]. It targets large matrices and features LAPACK-compliant hybrid DLA routines for multicore CPUs enhanced with throughput-oriented accelerators (such as a GPU). Matrix algebra on GPU and multicore architecture (MAGMA) is based on compute unified device architecture (CUDA), which leverages the parallel processing power of GPUs for computationally intensive simulations. It offers functionalities for various types of problems, including dense, sparse, native, and hybrid.

In particular, MAGMA offers operations to accurately solve (8) using LU decomposition with partial (row) pivoting. The required memory spaces on the main-memory (host) and GPU (device) sides are preliminarily allocated to address the computation. This implies that the MAGMA solver requires that the entire impedance matrix be transferred from the host to the device memory before solving. Once (8) is solved using the factored form of $\bar{\boldsymbol{Z}}$, the solution vector is retrieved from the GPU global memory.

Finally, we emphasize that to take advantage of GPUs' superior performance in single precision, which is significantly faster than double-precision complex arithmetic (approximately ten times faster in theory), MAGMA introduces a second set of solvers that utilize mixed-precision iterative refinement techniques [46]. It is worth noting that, even if mixed-precision iterative techniques stem from the GPU context, the advantage of reduced execution time is also evident on other architectures, reporting negligible accuracy error (see Section V-C). For this reason, we have adopted mixed-precision iterative techniques when available (i.e., for PLASMA).

3) Scalable LAPACK: It is a high-performance linear algebra software library to run scalably on distributed-memory

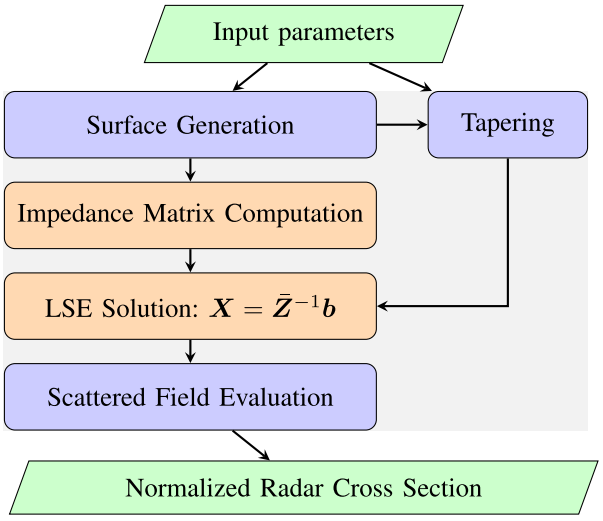


Fig. 2. Block diagram including the main steps of the sequential algorithm. The blocks in orange are amenable to parallelization.

(message-passing) architectures through *process-level* parallelism [47], [48], [49].

ScaLAPACK routines are based on block-partitioned algorithms to minimize data movement between different memory levels. Load balance and scalability on distributed-memory systems heavily depend on how matrices are distributed among processes. ScaLAPACK routines utilize a message-passing paradigm, where each subroutine directly accesses only local data. Specifically, ScaLAPACK employs a method of dividing the matrix into blocks and distributing them across processors in a 2-D, repeating pattern (2-D block-cyclic decomposition scheme) to achieve well-balanced computations and scalable implementations [50]. It is important to note that ScaLAPACK does not provide a direct tool to correctly distribute a global matrix before using its routines. Hence, a custom data distribution mapping needs to be implemented to meet our specific requirements, as discussed in [51] p.108. Further discussion on this issue is provided in Section IV-B.

Finally, efficiency within a computing node is achieved through the use of basic linear algebra subroutines (BLAS). ScaLAPACK relies on distributed-memory versions of the Level-2 and Level-3 BLAS, along with a set of basic linear algebra communication subprograms (BLACSs) for communication tasks typical of parallel linear algebra computations [52].

IV. Prototype Implementation

Fig. 2 shows the block diagram of all the steps involved in the numerical computation of EM scattering from 1-D randomly rough surfaces, using MoM. The first step is to generate a random rough surface profile with known statistical properties. To reduce the computational error caused by the truncation effect of the finite-length surface, we consider a Gaussian beam wave as an incident field (tapering), according to Thorsos [53]. The next step is to evaluate the elements of the impedance matrix $\bar{\boldsymbol{Z}}$; subsequently, the linear system (8) is solved. Finally, the scattered field can be evaluated, thus obtaining the relevant normalized radar cross section (NRCS).

Historically, Fortran has always been the language for HPC, and it remains one of the most widely used high-level languages in various scientific fields, including computational EMs (CEMs). A rich legacy of existing, well-tested Fortran code can be leveraged to avoid the cost and effort associated with rewriting software. Extensive scientific libraries, including those for linear algebra, offer optimized routines that streamline development and enhance code performance. Indeed, Fortran is a compiled language familiar to many scientists and engineers, thereby facilitating code maintenance and development. Accordingly, all the different parallel implementations presented in this article have been written using the modern Fortran language.

Additionally, the surface profile generation requires the discrete Fourier transform (DFT), and we used the fastest Fourier transform in the west (FFTW) library [54] to implement it.

The following provides significant implementation details on parallel solutions developed for the three different computational architectures.

A. Shared-Memory Architecture

We have developed two different (shared-memory) implementations oriented to multicore CPU and hybrid CPU-GPU architectures. In both implemented solvers, multithreading is used for the computation of the impedance matrix. To perform the LSE solution step using multicore processors, we have developed a PLASMA-based implementation. In contrast, the MAGMA-based implementation offloads the LSE solution step to the GPU. These two approaches are effective for the case in which the impedance matrix $\bar{\boldsymbol{Z}}$ can fit comfortably in the memory of a single machine, and they are further discussed hereinafter.

1) CPU Multicore Architecture: A single processing unit computes the impedance matrix in parallel using `OpenMP` [42] for-loop parallelization for computing elements of the matrix. For each row, the calculations are independent of each other, implying that the computation can be parallelized, significantly improving performance. The data are stored in the main memory and accessed by all threads. We utilize the routine `plasma_zcgesv` from the PLASMA library to solve the LSE on multiple CPU cores in mixed-precision complex through LU decomposition with partial (row) pivoting. Specifically, mixed-precision algorithms initially adopt a lower precision (single-precision 32-bit floating-point arithmetic) to compute the expensive operations (for instance, LU $\mathcal{O}(N^3)$), and then carry out an iterative refinement process in the IEEE standard double-precision 64-bit floating-point arithmetic to achieve the solution. This strategy allows solving dense LSE in higher precision but at a speed characteristic of the much faster single-precision computations.

2) GPU + CPU Multicore Architecture: We use a hybrid (CPU + GPU) algorithm. In particular, we calculate the impedance matrix, as above, on the multicore CPU. Then, we allocate memory on the GPU for matrix $\bar{\boldsymbol{Z}}$ and vectors $\boldsymbol{X}$ and $\boldsymbol{b}$ using suitable CUDA routines. We also use the same library to transfer the data between the CPU and GPU memories and back to retrieve the solution. For the computations, we utilize the routine `magmaf_zcgesv_gpu` from the MAGMA library on the GPU, which solves the linear system using LU decomposition with partial (row) pivoting and

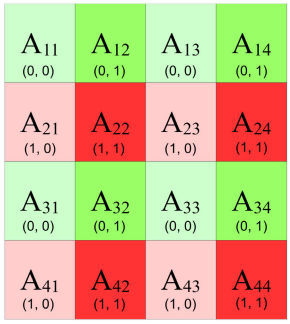


Fig. 3. Block-cyclic distribution on a 2×2 processor grid of a matrix consisting of 4×4 blocks: in brackets, coordinates of each processor owning the corresponding blocks.

leveraging mixed-precision iterative refinement techniques. In this case, the mixed-precision (double-complex and single-complex) algorithm for LU decomposition is very convenient because, in GPU architectures, the number of single-precision units outnumbers the double-precision ones.

B. Distributed-Memory Architecture

We implemented the data distribution mapping using a block-cyclic distribution across multiple processors. For this purpose, we utilized BLACS, part of ScaLAPACK, as a communication layer to distribute data and communicate between processors while filling the impedance matrix, distributing the input vector, and gathering the solution. For this purpose, the $N \times N$ impedance matrix is partitioned into blocks of a suitably prescribed size, which is stored on a different process in a block-cyclic distributed-memory environment. Specifically, the processors are organized into a logical 2-D grid, and the blocks are distributed across the processor grid (an illustrative example of a matrix consisting of 4×4 blocks is shown in Fig. 3). An integer array (array descriptor) is used to encapsulate information about the logical grid of processors. Then, the mapping of the matrix elements onto (distributed) local processors' memories is uniquely linked to the processor IDs. The matrix blocks are distributed cyclically across the grid processors, ensuring an even data distribution. With this strategy, elements needed for calculations on a specific processor are likely to be allocated in the local memory of that processor reducing communication overhead. This strategy helps balance the computational load. To solve the system of linear equations in double-precision complex, we use the `pzgesv` ScaLAPACK routine based on LU decomposition with partial (row) pivoting. See Appendix C for an overview of the parallel block LU decomposition algorithm implemented by ScaLAPACK.

V. Experiments

In this section, extensive benchmarking results across two different HPC platforms are presented. To this end, a detailed description of the employed computational infrastructures is first provided. A set of canonical case studies is then introduced to provide representative and reproducible workloads. Functional validation and numerical accuracy of the parallel prototypes are then assessed. Subsequently, well-established parallel performance metrics are introduced, and a detailed benchmarking analysis is conducted to compare the performance of the three developed parallel solutions quantitatively.

TABLE I
Specification of a Node of the Ben Computational Platform

Ben Platform Node	CPUs	GPU
Architecture	**2 × IBM POWER9™**	**Nvidia V100**
Core Clock	**2.7 GHz**	**1530 MHz**
Number of Cores	**2 × 16**	**5120 CUDA**
Memory size	**1024 GB**	**32 GB**
Memory Bandwidth	**108 GB/s**	**897 GB/s**
Interconnection	**Mellanox connectx-5**	**PCIe 3.0 x16**

TABLE II
Specification of a Node of the Leonardo Computational Platform

Leonardo Platform Node	CPUs	GPU
Architecture	**Intel®Xeon®Platinum 835**	**Nvidia A100 (custom)**
Core Clock	**2.6 GHz**	**1395 MHz**
Number of Cores	**32**	**7936 CUDA**
Memory size	**512 GB**	**64 GB**
Memory Bandwidth	**200 GB/s**	**1640 GB/s**
Interconnection	**Mellanox connectx-6**	**PCIe 4.0 x16**

TABLE III
Software Environment

Software	Ben Version	Leonardo Version
GNU compiler	**11.2.0**	**12.2.0**
PLASMA	**20.9.20**	**20.9.20**
MAGMA	**2.7.1**	**2.7.1**
CUDA toolkit	**11.3.1**	**12.1.0**
OpenBLAS	**0.3.18**	**0.3.24**
BLAS	**3.11.1**	**3.11.0**
CBLAS	**20110120**	**20110120**
OpenMPI	**4.1.2**	**4.1.6**
ScaLAPACK	**2.1.0**	**2.1.0**
FFTW	**3.3.10**	**3.3.10**

A. Experimental Setting

We conducted the experiments on two HPC clusters: Ben, hosted by ICAR CNR, and Leonardo Booster partition, hosted by CINECA. Each node of Ben housed two 16-core 2.7-GHz POWER9 processors, four NVIDIA Tesla V100 GPUs with 32 GB of memory each, and 1 TB of RAM, while each node of the Leonardo Booster partition housed one 32-core Intel Xeon Platinum 8358 CPU, 2.60 GHz, four NVIDIA Ampere A100 GPUs with 64 GB, and 512 GB of RAM. Table I summarizes the node specification for the Ben platform, while Table II summarizes the node specification for the Leonardo Booster partition.

Table III summarizes the software environments needed to run our experiments for both HPC platforms.

B. Case Studies

In this article, three main scenarios implying EM scattering from perfectly conducting random rough surfaces will be

TABLE IV
CASE STUDIES

Case	1.1	1.2	2.1	2.2	3.1	3.2	3.3	3.4
Polarization	TM	TM	TM	TM	TE	TE	TM	TM
Incidence Angle	30°	85°	30°	85°	30°	85°	30°	85°
EM wavelength [m]	1.0	1.0	1.0	1.0	0.057	0.057	0.057	0.057
Reference Surface	Surface 1	Surface 1	Surface 2	Surface 2	Surface 3	Surface 3	Surface 3	Surface 3

studied. For all three cases, the surface height pdf (probability density function) is assumed to be Gaussian, and the main parameter that characterizes each case is the shape of the surface autocorrelation function (ACF) as follows.

1) Gaussian ACF.
2) Exponential ACF.
3) Sea surface ACF, described by Elfouhaily et al. [55] spectrum model.

For the first two cases, the main simulation parameters are as follows.

1) *EM Wavelength in Vacuum*: $\lambda_0 = 1$ m.
2) *rms Height*: $\sigma_h = \lambda_0$.
3) *Surface Correlation Length*: $L_c = \lambda_0$.
4) *Spatial Sampling Step*: $\Delta x = 0.1$ and λ_0.
5) *Incidence Angle(s) (With Respect to Vertical Direction)*: $\theta_{inc} = \{30°; 85°\}$.
6) *Polarization(s)*: Both TM and TE.

The medium of incidence is the air, which is assimilated to the vacuum.

For the third case, we take a realistic radar frequency f, and the roughness is defined by the wind speed at 10 m above the sea surface, u_{10} (considering a fully developed sea). We will consider it as follows.

1) *Radar Frequency*: $f = 5.3$ GHz, corresponding to EM wavelength in vacuum: $\lambda_0 \simeq 0.057$ m.
2) *Wind Speed at 10 m Above the Sea Surface*: $u_{10} = 10$ m/s.

Spatial sampling step, incidence angles, and polarizations are the same as for cases 1 and 2.

As will be discussed in Sections V-E and V-F, we study the effect of increasing the number of surface samples N—which also corresponds to the number of unknowns here for perfectly conducting surfaces—on parallel performance using different HPC architectures. Due to hardware (memory) constraints, not all architectures can accommodate the full range of N, which in our investigations spans from 10^4 to 6×10^5. This corresponds to a generated surface length: $L = N\Delta x$ (from 10^3 to $6 \times 10^4 \lambda_0$).

To perform the experimental analysis, we selected a set of representative case studies, which are summarized in Table IV. Hereafter, we show the generated surface and its associated NRCS for cases 1.1, 2.1, and 3.1 (Figs. 4–6). They correspond to Gaussian, exponential, and sea surfaces. For the first two cases, the EM wavelength in vacuum is $\lambda_0 = 1$ m, the polarization is TM, and $N = 3 \times 10^4$, while for the third case, the polarization is TE and $N = 10^5$. For all three cases, the incidence angle is 30°. The plotted NRCSs correspond to the results obtained with a parallel solver, which are fully numerically consistent with the sequential (LAPACK-based) implementation, as discussed in Section V-C. The Gaussian and exponential surface plots have been zoomed in (from 300 m length to 100 m) to better see their shape.

As a general comment, as the Gaussian and exponential surfaces are very rough and have very high slopes (owing to $\sigma_h = \lambda_0$ and $L_c = \lambda_0$), the NRCS is spread over all observation angles, such that the specular direction peak is hardly stronger than the levels in other directions. In fact, for the exponential case, the maximum peak occurs in a slightly different direction (about 37°) for this generated surface. In contrast, for the sea surface case, even if it is a case of relatively high winds ($u_{10} =$ 10 m/s), it corresponds to less high slopes ($\sigma_s \simeq 0.19$ here). The strongest constraint for the sea surface is that it implies generating very long surfaces in order to fairly represent the full sea surface spectrum. Indeed, here for $u_{10} = 10$ m/s and for a fully developed sea, we need to have a minimum surface length of $L_{min} \simeq 254$ m. This makes, for a radar frequency $f =$ 5.3 GHz, a minimum number of samples of about 4.5×10^4 (for a sampling step $\Delta x = 0.1\,\lambda_0$).

C. Numerical Results and Validation

As a preliminary step, we validate the numerical results obtained with the implemented parallel prototypes.

Parallel execution can involve arithmetic operations being performed in a different order from the sequential case. As operations in floating-point arithmetic are not associative, the operation reordering inherent in parallel execution can lead to tiny discrepancies in the intermediate results. Therefore, we compare the numerical results obtained with the parallel solvers, using the sequential solver as a reference. This investigation is crucial to fully demonstrating the functional equivalence of the developed solvers and quantitatively assessing their relative numerical accuracy.

In particular, the numerical results—expressed in terms of NRCS for discrete scattering directions (output vector $\boldsymbol{\sigma}^0 \in \mathbb{R}^M$, where M denotes the number of considered scattering angles)—obtained with three developed parallel solvers are compared to those produced by the sequential solver (namely $\hat{\boldsymbol{\sigma}}^0 \in \mathbb{R}^M$), across the different case studies presented in Section V-B. For the purpose of accuracy assessment, a fixed problem size of $N = 3 \times 10^4$ is used, allowing for a numerically consistent comparison across the three architectures within the limits of their maximum commonly available memory. The comparison is conducted under identical input conditions.

For illustrative purposes, Fig. 7 shows the NRCS difference, $(\boldsymbol{\sigma}^0 - \hat{\boldsymbol{\sigma}}^0)$, between the parallel and sequential solvers for case study 3.1, plotted against the scattering angle for each parallel solver. The graphs show that, for case study 3.1, the absolute errors are mainly localized around the specular direction, thus reflecting the angular dependence of the scattered field.

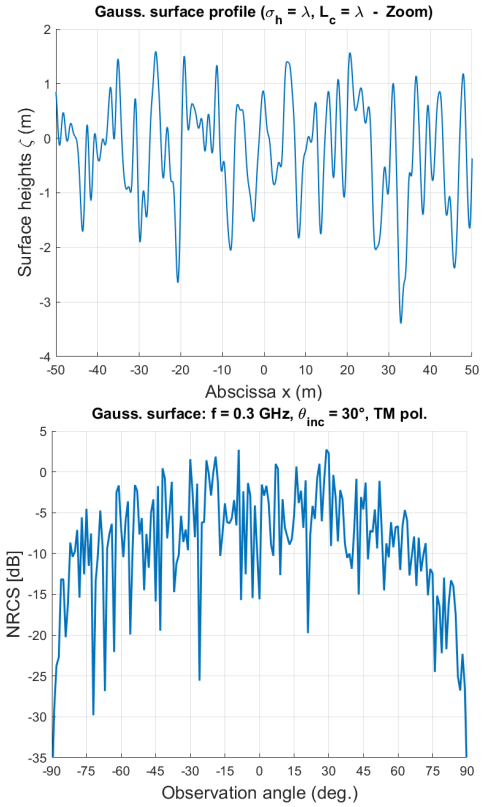


Fig. 4. Generated Gaussian surface (top—zoomed in) and its associated NRCS (bottom) for case 1.1 (λ_0 = 1 m and θ_{inc} = 30°, TM pol.)

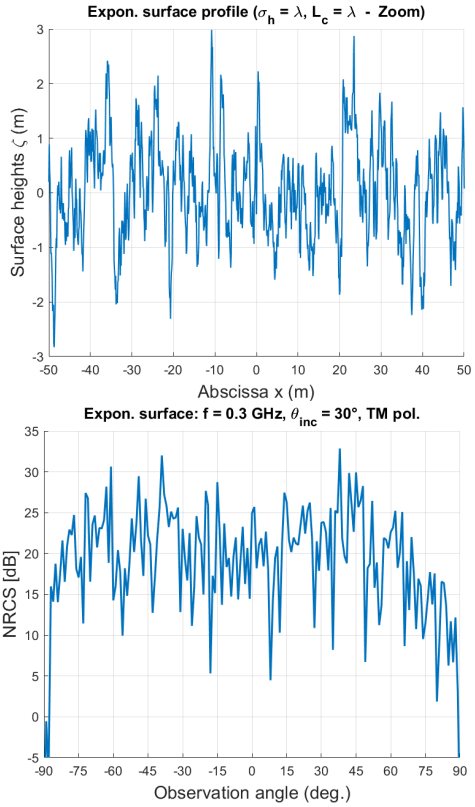


Fig. 5. Generated exponential surface (top—zoomed in) and its associated NRCS (bottom) for case 2.1 (λ_0 = 1 m and θ_{inc} = 30°, TM pol.)

TABLE V
ERROR METRICS

Error Metric	Expression	Abbr.
Relative L_2 norm	$\frac{\|\sigma^0 - \hat{\sigma}^0\|_2}{\|\hat{\sigma}^0\|_2}$	**Rel.** L_2
Root Mean Squared Error	$\sqrt{\frac{1}{M}\sum_{m=1}^{M}(\sigma_m^0 - \hat{\sigma}_m^0)^2}$	**RMSE**
Infinity norm	$\|\sigma^0 - \hat{\sigma}^0\|_\infty = \max_m \|\sigma_m^0 - \sigma_m^0\|$	∞
Relative infinity norm	$\frac{\|\sigma^0 - \hat{\sigma}^0\|_\infty}{\|\hat{\sigma}^0\|_\infty}$	**Rel.** ∞

Absolute errors in all cases consistently remain below 10^{-12} across the entire angular range, thus demonstrating the overall numerical validity of the developed parallel implementations.

The quantitative comparison is also carried out using different error metrics (see Table V), as detailed in Table VI for all the considered case studies and for all parallel solvers. It is worth noting that, for surfaces with exponential correlation (cases 2.1 and 2.2), the increased errors may be primarily ascribed to high-frequency scattering (i.e., short wavelengths), which is more sensitive to the sharper surface features.

Based on the observed differences, we can conclude that the different parallel solvers developed for different computational architectures produce numerically consistent results.

D. Parallel Performance Metrics

Well-known performance metrics are typically adopted to assess the parallel performance of an algorithm or system quantitatively. In general, it is important to evaluate how efficiently the developed prototypes handle increasing numbers of processing units or workloads.

TABLE VI
ERROR METRICS FOR PLASMA, MAGMA, AND SCALAPACK-BASED PARALLEL SOLVERS WITH RESPECT TO LAPACK-BASED SEQUENTIAL IMPLEMENTATION

Case Study	Used Library	Error Metrics			
		Rel. L_2	**RMSE**	∞	**Rel.** ∞
1.1	PLASMA	3.38e-15	1.64e-15	7.77e-15	4.15e-15
	MAGMA	4.75e-15	2.31e-15	9.55e-15	5.09e-15
	ScaLAPACK	4.22e-13	2.05e-13	7.58e-13	4.04e-13
1.2	PLASMA	1.56e-15	2.37e-15	1.07e-14	5.41e-16
	MAGMA	1.66e-15	2.52e-15	2.13e-14	1.08e-15
	ScaLAPACK	1.79e-13	2.72e-13	1.64e-12	8.36e-14
2.1	PLASMA	1.12e-10	3.57e-08	1.58e-07	8.16e-11
	MAGMA	2.46e-11	7.83e-09	3.58e-08	1.85e-11
	ScaLAPACK	1.78e-11	5.69e-09	2.98e-08	1.54e-11
2.2	PLASMA	1.18e-10	2.05e-07	6.62e-07	7.93e-11
	MAGMA	4.57e-11	7.91e-08	3.65e-07	4.37e-11
	ScaLAPACK	3.60e-11	6.23e-08	2.79e-07	3.34e-11
3.1	PLASMA	5.82e-15	5.37e-15	4.35e-14	6.73e-15
	MAGMA	4.49e-15	4.15e-15	2.98e-14	4.60e-15
	ScaLAPACK	7.15e-14	6.60e-14	4.13e-13	6.38e-14
3.2	PLASMA	5.45e-15	5.79e-15	5.28e-14	5.76e-15
	MAGMA	5.11e-15	5.43e-15	3.64e-14	3.97e-15
	ScaLAPACK	2.43e-14	2.58e-14	2.34e-13	2.55e-14
3.3	PLASMA	1.53e-15	1.39e-15	1.33e-14	2.10e-15
	MAGMA	1.20e-15	1.10e-15	5.99e-15	9.43e-16
	ScaLAPACK	2.62e-14	2.39e-14	1.35e-13	2.13e-14
3.4	PLASMA	1.23e-15	1.02e-15	5.33e-15	1.02e-15
	MAGMA	1.32e-15	1.10e-15	9.77e-15	1.88e-15
	ScaLAPACK	4.33e-14	3.58e-14	2.27e-13	4.36e-14

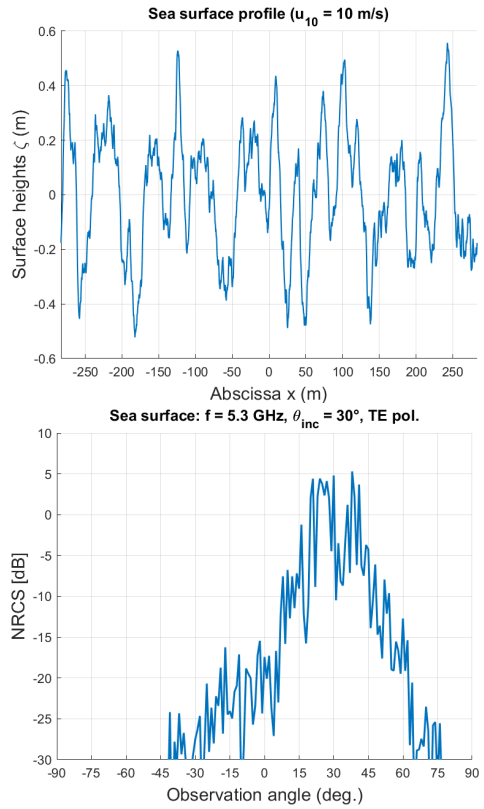


Fig. 6. Generated sea surface (top) and its associated NRCS (bottom) for case 3.1 ($\lambda_0 \simeq 0.057$ m and $\theta_{inc} = 30°$, TE pol.)

Specifically, to quantify the reduction in execution time, for a prescribed problem size, as the number of processing units is increased, the speedup, denoted as S_p, is formally defined as

$$S_p = \frac{T(W,1)}{T(W,p)} \tag{10}$$

where $T(W,p)$ represents the execution time for solving a problem of size W utilizing p processing units. The numerator, $T(W,1)$, thus means the sequential execution time of a single processing unit. Ideally, S_p would equal p, indicating perfect linear speedup. However, in practice, experimentally observed speedup typically exhibits sublinear scaling ($S_p < p$) as a consequence of various parallelization costs (e.g., communication overhead between processors and synchronization costs). Specifically, the overhead $O_h = pT(W,p) - T(W,1)$ can lead to a decrease in efficiency at larger scales. Memory bandwidth limitations can also contribute to O_h and become a bottleneck [34], [35], [36]. Moreover, the theoretical limits of speedup for a fixed problem size are often described by *Amdahl*'s Law, which states that the speedup is asymptotically limited (as $p \to \infty$) by the fraction of the computation that is inherently sequential [34], [35], [36], [56]. Additionally, the parallel efficiency, ε_p, can be quantified as follows:

$$\varepsilon_p = \frac{S_p}{p}. \tag{11}$$

The metric in (11) provides a normalized measure of the effective utilization of the added processing resources, with the ideal efficiency being unitary. In this article, we also use the term computational gain to refer to the ratio between the execution time of the sequential version and that of the parallel version, assuming that all computational units of the considered architecture are utilized.

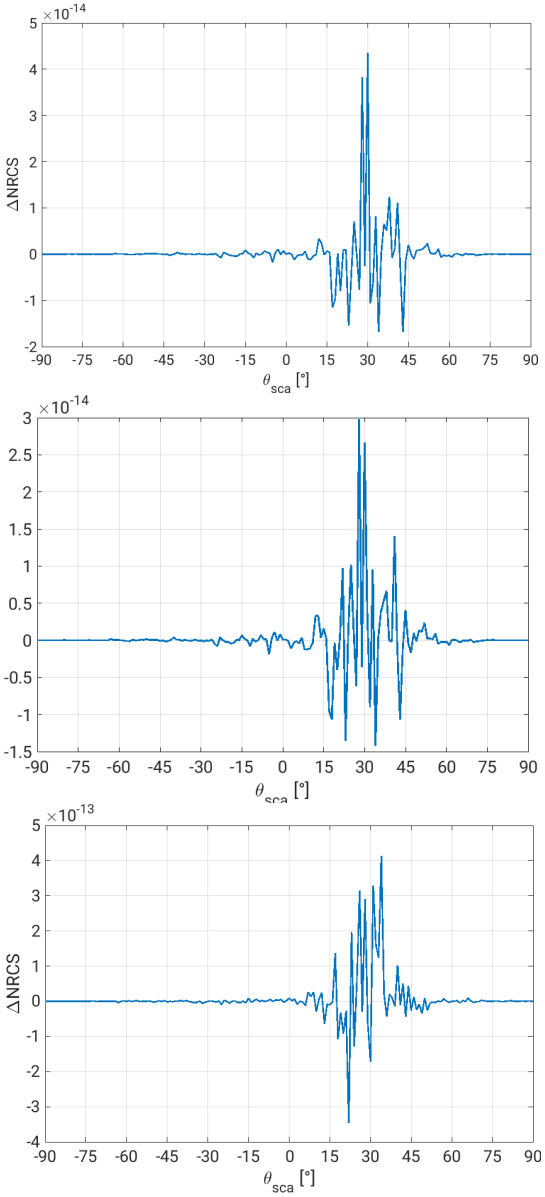


Fig. 7. NRCS numerical error (relative to the sequential solver) as a function of scattering angle (degrees) for case study 3.1, using parallel solvers based on PLASMA (top), MAGMA (center), and ScaLAPACK (bottom).

Analyzing the trends of these metrics provides valuable insight into the scalability characteristics of the developed parallel solvers and the associated hardware architectures.

E. Parallel Performance Analysis

This section is devoted to investigating the experimental performance achievable with the different parallel implementations. Specifically, we focus on case study 3.1 and its extension to cover cases with varying surface lengths. As a preliminary step, we represent in log-log scale the execution time required to solve the LSE using the sequential (LAPACK-based) implementation as a function of the number of unknowns N, for both used platforms (see Fig. 8).

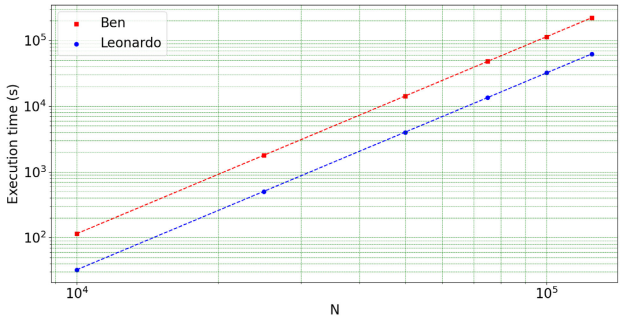


Fig. 8. Sequential implementation: execution time (seconds) for LSE solution as a function of the number of unknowns N, using Ben and Leonardo platforms.

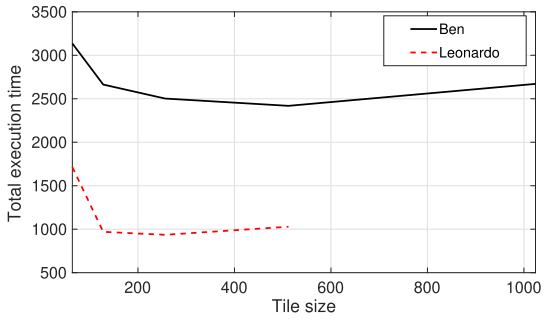


Fig. 9. PLASMA-based implementation: total execution time (in seconds) versus tile size using Ben and Leonardo multicore architectures, for $N = 1 \times 10^5$.

While the Ben platform performs well, the Leonarto platform demonstrates superior performance. In particular, to analyze the trend of the execution time for solving the LSE, we perform a linear fit of the experimental data using an N^α model, obtaining $\alpha = 2.995$ and $\alpha = 2.994$ for the Ben and Leonardo platforms, respectively. These results are fully consistent with the theoretical computational complexity of the LU decomposition, which is $O(N^3)$.

Subsequently, we proceed with a quantitative assessment of the performance results obtained using the aforementioned parallel solvers and architectures. Scalability is then discussed in relation to the computational resources used and the problem size. We begin by evaluating the exploitation of multicore CPU architectures in the context of tile-based DLA computations using the PLASMA library. As a preliminary step, we investigate how the square tile size affects performance, aiming to identify the optimal value—i.e., the one that best fits into the cache hierarchy and minimizes total execution time—for each platform under test. In particular, we vary the tile size in increments of 64 and measure the total execution time for solving a matrix problem of dimension $N = 1 \times 10^5$. The resulting performance curves are shown in Fig. 9, where execution time is plotted as a function of the tile size. We observe that the optimal tile size is 512 for the Ben platform and 256 for the Leonardo platform. These differences are likely attributable to architectural distinctions, such as cache size and memory hierarchy. Consequently, all subsequent experiments are conducted using these empirically determined optimal tile sizes for each platform.

TABLE VII

COMPARISON OF TOTAL EXECUTION TIMES (SECONDS) USING MULTICORE CPUS ACROSS BEN AND LEONARDO PLATFORMS, FOR AN INCREASING NUMBER OF UNKNOWNS N

N	Ben	Leonardo
1.0×10^3	7.55	3.73
2.5×10^4	61.06	29.05
5.0×10^4	340.99	145.08
7.5×10^4	1059.58	429.23
1.0×10^5	2418.63	935.54
1.25×10^5	5993.64	—
1.50×10^5	7877.60	—

TABLE VIII

COMPARISON OF LSE AND TOTAL EXECUTION TIMES (SECONDS) USING GPU + CPU ACROSS BEN AND LEONARDO PLATFORMS, FOR INCREASING NUMBER OF UNKNOWNS N

N	Ben		Leonardo	
	LSE Time	Tot. Time	LSE Time	Tot. Time
1.0×10^4	2.28	3.22	5.50	6.05
2.5×10^4	7.56	12.85	8.62	11.92
3.5×10^4	13.61	23.88	13.41	20.25
4.5×10^4	—	—	20.78	32.54

Table VII shows the execution times of the implementation (based on PLASMA) for the multicore CPU architecture on both platforms. It is notable that Leonardo, while having the same number of physical cores, performs better due to its different architecture. In Fig. 10(a), we plot the computational gain, which is defined as the ratio between the execution time of the sequential (LAPACK-based) solver and that of the multithreaded (PLASMA-based) solver, as a function of the number of unknowns N. It is evident that Ben, although it is generally performing worse, shows a constant gain for different N up to 1.5×10^5, which is better than that observed on Leonardo. It is worth noting that on multicore CPUs, memory utilization is closely tied to thread-level parallelism, cache utilization, and efficient data locality. Since total elements scale with N^2, the availability of 1-TB RAM on Ben allows it to theoretically support very large matrices (over 2.0×10^5 elements per side) when the sequential approach is considered, while the actual size in multithreading implementations is more limited [see Fig. 11(a)]. Although the memory usage for the two platforms is generally comparable, the largest N values considered for Ben and Leonardo were 1.5×10^5 and 1.0×10^5, respectively.

In the context of a hybrid GPU + CPU execution paradigm, where the multicore CPUs handle matrix construction and the GPU performs LU decomposition, Table VIII compares the performance of the MAGMA-based implementation on Ben and Leonardo as the problem size increases. For small-scale problems ($N = 1 \times 10^4$), Ben outperforms Leonardo, likely because the V100 GPU hosted on Ben fits better with the MAGMA feature of exploiting mixed precision on tensor cores for smaller cases. Conversely, as the problem size grows, the Leonardo architecture becomes more competitive, having a higher number of GPU cores and a better CPU architecture.

For this case, the associated computational gain, calculated as the ratio between the sequential execution time and the parallel execution time (hybrid CPU + GPU), is depicted in

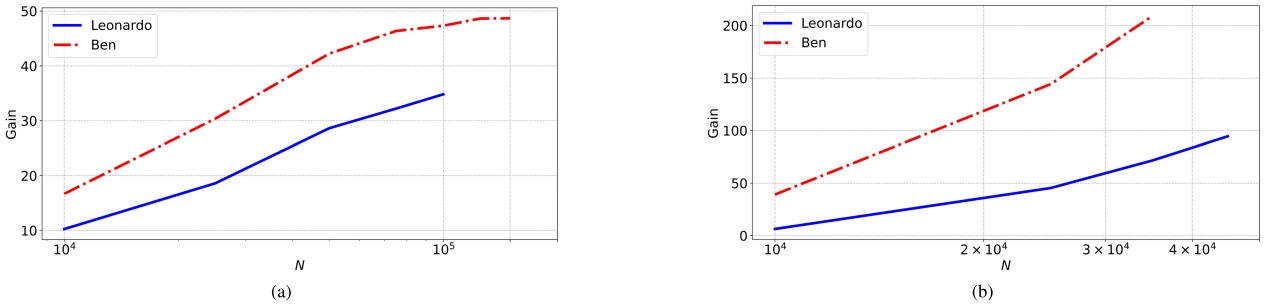


Fig. 10. Computational gain over sequential (LAPACK-based) implementation as a function of the number of unknowns N using Ben and Leonardo platforms. (a) Multicore CPU (PLASMA-based) implementation and (b) multicore CPU + manycore GPU (MAGMA-based) implementation.

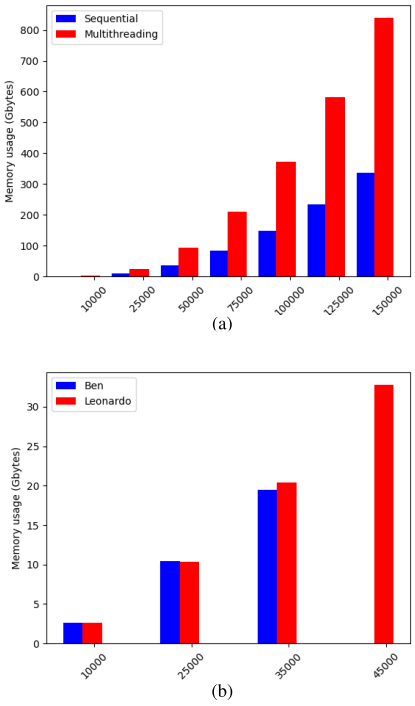


Fig. 11. Memory usage (GB) versus number of unknowns N. (a) Ben CPU RAM usage for sequential and multithreaded (PLASMA-based) implementations and (b) GPU device memory usage (MAGMA-based implementation) using Ben and Leonardo architectures.

Fig. 10(b) as a function of the number of unknowns N. The results reveal a clear higher gain for Ben than Leonardo, likely because the sequential implementation runs faster on Leonardo. Remarkably, Leonardo is the only platform able to complete the largest problem size tested ($N = 4.5 \times 10^4$) due to its higher memory capability. The MAGMA-based implementation is limited by the device memory's capacity to accommodate a large-scale matrix. For instance, the 32-GB GPU of the Ben platform allows handling matrices up to approximately $N = 4 \times 10^4$. In comparison, the 64-GB GPU of the Leonardo platform allows handling matrices up to approximately $N = 5 \times 10^4$ [see Fig. 11(b)]. Hence, for matrix sizes exceeding the GPU device memory, strategies such as matrix tiling, streaming data in chunks, or using multiple GPUs become necessary.

In the distributed-memory (multinode) setting, we define the total number of computing units as $p = p_s \cdot p_d$, where p_s denotes the number of CPU cores per node and p_d denotes the number of computing nodes employed in the computation. This choice reflects the need to distribute processes according to a 2-D grid, as adopted by ScaLAPACK. The grid can be organized across cores and nodes to minimize communication overhead. The algorithm used for constructing the impedance matrix is more efficient when the process grid dimensions satisfy $p_s = p_d$, hence the decision to use a square 2-D grid.

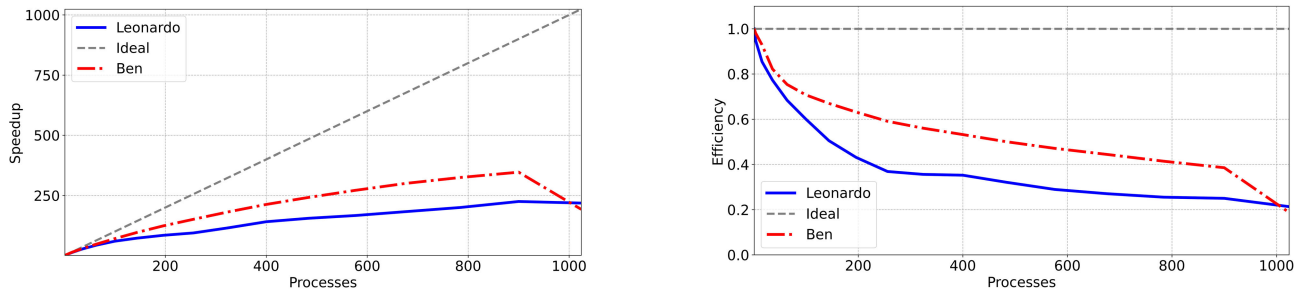
For the ScaLAPACK-based implementation, the performance metrics (see Section V-D) are experimentally evaluated for a fixed problem size of $N = 10^5$ and varying the process grid ($p_s \times p_d$ from 1×1 to 32×32) employed in the computation, on both the Ben and Leonardo platforms. In particular, Fig. 12(a) illustrates the measured speedup S_p as a function of the number of processes. The ideal (linear) speedup, which represents a theoretical upper bound for parallel performance, is also shown (dashed line) as a reference. We observe a speedup exhibiting an almost linear progression as the number of computing units increases, demonstrating reasonable scaling behavior. Correspondingly, the parallel efficiency ε_p, shown in Fig. 12(b), is high for moderate levels of parallelism but gradually declines as the number of computing units increases. This behavior is typical of fixed-size problems like LU decomposition, where the computational workload per processing unit decreases, while communication and synchronization costs increase, thereby reducing overall efficiency.

The scalable performance of the ScaLAPACK-based implementation exhibits distinct behavior across the Ben and Leonardo clusters. The Ben cluster consistently demonstrates superior speedup and efficiency compared to the Leonardo cluster. This disparity is also reflected in efficiency; Ben maintains a higher efficiency across various process counts, utilizing its computational resources better. However, Leonardo outperforms Ben in terms of the time-to-solution metric, as shown in Table IX, which presents a direct comparison of wall-clock times for both architectures, across a sample of all analyzed parallel configurations.

To relate the observed scalability to architectural constraints, we consider two additional metrics. First, we estimate a lower bound on the effective memory throughput during the LU factorization as

$$B_{\text{eff}}(p) = \frac{M_{\text{tot}}(p)}{T_{\text{LSE}}(p)} \tag{12}$$

where $T_{\text{LSE}}(p)$ is the measured time for solving the dense linear system and $M_{\text{tot}}(p)$ is the total resident memory footprint of the distributed impedance matrix. The resulting $B_{\text{eff}}(p)$ is



(a) (b)

Fig. 12. Parallel performance results. (a) Speedup. (b) Efficiency, as a function of the number of engaged processes p. Reference ideal speedup and ideal efficiency are also depicted (dashed lines).

TABLE IX

WALL-CLOCK TIMES (IN SECONDS) FOR A FIXED PROBLEM SIZE OF $N = 10^5$, WITH RESPECT TO THE PROCESS GRID (FROM 1×1 TO 32×32)

p_d	**Ben**	**Leonardo**
1	142944.375	35801.469
4	9617.836	2618.650
8	2963.987	818.110
12	1481.839	492.372
16	945.099	379.235
20	671.599	253.947
24	526.835	214.798
28	440.114	178.942
32	742.902	163.841

TABLE X

SUMMARY OF PARALLEL PERFORMANCE METRICS FOR THE SCALAPACK SOLVER (PROBLEM SIZE $N = 10^5$). HERE, p IS THE TOTAL NUMBER OF MPI PROCESSES, S_p IS THE SPEEDUP, $\varepsilon_p = S_p/p$ IS THE PARALLEL EFFICIENCY, e_p IS THE KARP–FLATT SERIAL-PLUS-OVERHEAD FRACTION, B_{eff} IS A LOWER BOUND ESTIMATE OF THE EFFECTIVE MEMORY THROUGHPUT (IN GB/S), AND $\rho(p) = B_{\text{eff}}/B_{\text{PEAK,TOT}}$ IS THE RATIO BETWEEN EFFECTIVE AND AGGREGATE PEAK MEMORY BANDWIDTH

Platform	p	S_p	ε_p	e_p	B_{eff}	$\rho(p)$
Ben	64	48.2	0.753	0.0052	0.065	0.008
	256	151.2	0.591	0.0027	0.281	0.016
	1024	192.4	0.188	0.0042	0.537	0.016
Leonardo	64	43.8	0.684	0.0073	0.282	0.018
	256	94.4	0.369	0.0067	0.995	0.031
	1024	218.5	0.213	0.0036	4.932	0.077

compared with the aggregate peak main-memory bandwidth $B_{\text{peak,tot}}(p) = p_d B_{\text{peak}}$, where B_{peak} is the per-node bandwidth reported in Tables I and II.

Second, we use the Karp–Flatt parameter [57]

$$e_p = \frac{\frac{1}{S_p} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{13}$$

to quantify the effective "serial-plus-overhead" fraction in the strong-scaling regime. Representative values of S_p, the corresponding efficiency ε_p, e_p, and $B_{\text{eff}}(p)$ for selected process grids are summarized in Table X.

The results in Table X complement the speedup and efficiency curves of Fig. 12. On both platforms, the efficiency remains high for moderate process counts (e.g., $\varepsilon_p \approx 0.75$ at $p_s \times p_d = 8 \times 8$, $p = 64$) but decreases at the largest grids, reaching $\varepsilon_p \approx 0.19$ on Ben and $\varepsilon_p \approx 0.21$ on Leonardo for $p = 1024$. At the same time, the Karp–Flatt parameter remains in the range $e_p = \mathcal{O}(10^{-3})$ for all configurations, indicating that the effective serial-plus-overhead fraction remains below 0.8% for the considered problem size. The estimated effective throughput during the LU factorization $B_{\text{eff}}(p)$ increases with p but remains several orders of magnitude below the aggregate peak bandwidth $B_{\text{peak,tot}}(p) = p_d B_{\text{peak}}$; even at $p = 1024$ we obtain $B_{\text{eff}}(p) \ll B_{\text{peak,tot}}(p)$ on both systems. The observed decrease in parallel efficiency at the maximum core utilization is thus mainly attributed to node-level resource contention (among MPI processes and system daemons for shared floating-point units and cache hierarchy) rather than to DRAM bandwidth saturation. Therefore, the conducted computational performance analysis elucidates the achievable parallel performance using two different platforms, also highlighting the scalability limitations of the proposed approach.

Finally, we emphasize that the adoption of a distributed solution is primarily motivated by physical memory constraints, which become critical for large-scale problem instances, such as electrically large surfaces. Indeed, to take advantage of the large number of computing nodes available on the Leonardo platform, we use the developed solver to compute the solution for the case $N = 6 \times 10^5$ using a process grid of $p_s \times p_d = 30 \times 120$, achieving the solution in 5577 s, demonstrating the feasibility of the approach. Specifically, the experiments conducted in a distributed environment show that memory usage grows in a quadratic to super-quadratic fashion with N, increasing from 1.72 GB ($N = 1 \times 10^4$) to 6.71 TB ($N = 6 \times 10^5$), consistent with the memory demands of storing and operating on dense $N \times N$ matrices. Notably, for $N = 6 \times 10^5$, memory usage significantly surpasses the typical node memory capacity on HPC clusters. In this context, the distributed implementation effectively circumvents the limitations imposed by per-node memory capacity, enabling the solution of problem sizes that would otherwise be intractable.

F. Final Remarks

This study has presented and comparatively evaluated three parallel implementations solving a canonical EM problem, each optimized for a different parallel architecture. The proposed approaches were systematically benchmarked on two distinct HPC platforms, enabling a detailed quantitative evaluation of their performance. This benchmarking effort provides valuable insights into the interplay between the considered

numerical method for EM scattering and the architectural features of modern HPC systems. In particular, it elucidates the advantages and limitations of each parallelization strategy when applied to the simulation of scattering from rough surfaces, thus highlighting the tradeoffs in terms of scalability, memory usage, and execution efficiency. Furthermore, the conducted analysis is particularly relevant in the context of electrically large surface scattering simulations, which are crucial in a wide range of applications—spanning from remote sensing and radar imaging to wireless channel modeling, nondestructive testing, and advanced materials characterization—where computational demands often push the limits of available HPC resources. A significant strength of the proposed implementations is their reliance on well-established, high-performance open-source libraries. This design choice not only ensures numerical robustness and portability across platforms but also enhances the reproducibility of results and facilitates the broader adoption of these methodologies within the CEMs community. Overall, the work offers a relevant and up-to-date benchmarking framework, shedding light on the practical performance implications of current HPC methodologies and technologies for MoM-based rough surface scattering applications.

Finally, we would like to highlight that the spirit of the present work is precisely to overcome the motivations that have historically led to a proliferation of iterative techniques in the context of EM scattering simulations. In particular, for conventional iterative solvers, such as the CGM and its more advanced variants, the typical computational complexity is $P \cdot \mathcal{O}(N^2)$, where P is the number of iterations to reach the prescribed convergence tolerance. A common strategy to reduce P is to employ advanced preconditioners. Similar to the direct LU-based approach, these solvers require the computation of the impedance matrix. For scattering from rough surfaces, techniques such as multilevel fast multipole algorithm (MLFMA) and FFT-based methods like BMIA/CAG allow us to accelerate the matrix-vector products within an iterative solver. In addition, these algorithms avoid calculating the full impedance matrix explicitly by decomposing the interactions into strong and weak contributions. Only the elements associated with strong interactions are computed. Therefore, the proposed LU-based approach provides a direct and robust alternative to iterative methods, thereby avoiding issues related to convergence, preconditioner tuning, or solver stability, which can be particularly challenging for electrically large or ill-conditioned problems.

VI. Conclusion

Effective MoM-based parallel solution strategies have been developed for the computationally intensive canonical problem of rough surface scattering. These strategies allow for the timely numerical analysis of electrically large rough surfaces across various computational architectures. Accordingly, the efficient use of the parallelism offered by heterogeneous platforms has been systematically addressed for the canonical problem considered in this article, by employing an effective and easily replicable approach that enables straightforward parallelization through well-established, high-performance, open-source libraries for dense algebra computation. Several representative case studies have been explored, and inherent numerical results carried out using the implemented solvers for diverse HPC architectures have been illustrated. The parallel performances achievable with the developed prototype solutions have also been experimentally assessed and comparatively discussed, thus demonstrating inherent speedup and scalability. The solutions and their performance provide a systematic framework for benchmark analysis.

As a result, the developed parallel prototypes efficiently use currently available HPC platforms to successfully accelerate the computation of the canonical problem, including large-scale cases previously considered practically unsolvable. Their adoption will be crucial to achieve faster solutions, as well as to explore scattering phenomena at higher resolutions or higher frequencies, or simulate scattering from large surfaces with greater fidelity. Moreover, they could also be useful for fully validating analytical scattering models [58].

The developed parallel solvers enable the accurate, efficient, and scalable simulation of the scattering from rough surfaces; however, our investigation has been conducted considering a single conducting surface in the context of 2-D geometry. Further studies will be devoted to demonstrating that the proposed computational approach is amenable to systematic extension for handling large-scale supercomputer simulations of scattering from electrically large targets in more complex scenarios (e.g., targets over a randomly rough surface or within randomly rough layered structures) [7], [8]. To overcome limitations associated with matrix sizes exceeding the GPU memory, a full multi-GPU extension based on cuSOLVER, which provides distributed DLA kernels specifically designed for modern multi-GPU architectures, will also be investigated [59].

Appendix A
Impedence Matrix

The full expressions for the impedance matrix (8) used in this study, which can be derived when MoM is applied along with the point-matching method and depend on the shape of the surface, are provided in this section. For a perfectly conducting surface, the elements of $\bar{\boldsymbol{Z}} = \{Z_{mn}\}_{m,n\in\{1,\ldots,N\}}$ are defined, for both the polarizations (TE and TM), as follows [10], [11]. For the TE polarization (Dirichlet boundary conditions), we have

$$Z_{mn} = \frac{i\Delta l_n}{4} \begin{cases} H_0^{(1)}\left(k\left\|\mathbf{r}_m - \mathbf{r}_n\right\|\right), & m \neq n \\ 1 + \dfrac{2i}{\pi} \ln\left(\dfrac{\gamma}{4e} k \Delta l_n\right), & m = n \end{cases} \tag{14}$$

where $\Delta l_m = \Delta x \sqrt{1 + (z'(x_m))^2}$ is the length of the segment of the surface that is centered at x_m, Δx is the width of the rectangular pulse basis function, k is the free-space propagation constant, $\mathbf{r}_n = (x_n, z(x_n))$ and $\mathbf{r}_m = (x_m, z(x_m))$ are two points on the surface profile, $\|\mathbf{r}_n - \mathbf{r}_m\| = \sqrt{(x_n - x_m)^2 + (z(x_n) - z(x_m))^2}$, $z'(x)$ represents the derivative of the scattering surface profile $z(x)$, $H_0^{(1)}(\cdot)$ is the Hankel function of the first kind and order zero, and $\gamma = 1.78107$ is the exponential of the Euler constant.

Moreover, N is the number of rectangular pulse basis functions used to expand the unknown functions, and the elements of $\boldsymbol{b} = \{b_m\}_{m\in\{1,\ldots,N\}}$ are defined as $b_m = E^{inc}(\mathbf{r}_m)$.

TABLE XI
BLOCK LU FACTORIZATION: TOP-LEFT STEP

Input: Matrix A partitioned into 2×2 blocks: $A = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \end{bmatrix}$
Step 1: Compute LU factorization of the leading block: $\bar{A}_{11} = \bar{L}_{11}\bar{U}_{11}$ **Step 2:** Solve triangular systems: $\bar{L}_{21} = \bar{A}_{21}\bar{U}_{11}^{-1}$, $\bar{U}_{12} = \bar{L}_{11}^{-1}\bar{A}_{12}$ **Step 3:** Update the trailing submatrix: $\bar{A}'_{22} = \bar{A}_{22} - \bar{L}_{21}\bar{U}_{12}$
Output: Blocks $\bar{L}_{11}, \bar{U}_{11}, \bar{L}_{21}, \bar{U}_{12}, \bar{A}'_{22}$

For the TM polarization (Neumann boundary conditions), matrix elements are defined as

$$Z_{mn} = \begin{cases} \dfrac{ik\Delta l_m}{4} H_1^{(1)}\left(k\left\|\mathbf{r}_n - \mathbf{r}_m\right\|\right) \dfrac{(\mathbf{r}_n - \mathbf{r}_m)}{\|\mathbf{r}_n - \mathbf{r}_m\|} \cdot \hat{n}_n, & m \neq n \\ \dfrac{1}{2} - \dfrac{\Delta x}{4\pi} \dfrac{z''(x_n)}{1 + (z'(x_n))^2}, & m = n \end{cases} \tag{15}$$

where $\hat{n}_n = \hat{n}(\mathbf{r}_n)$, $z''(x)$ represents the second derivative of the surface profile and $H_1^{(1)}(\cdot)$ is the Hankel function of the first kind and order one. Moreover, $b_m = E^{inc}(\mathbf{r}_m)$.

APPENDIX B
BLOCK LU DECOMPOSITION

Different algorithms exist for the block LU factorization. Here, we focus on the right-looking variant [37]. Let us consider the $N \times N$ dense matrix $\bar{A}$ to be partitioned

$$\bar{A} = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \end{bmatrix} \tag{16}$$

where $\bar{A}_{11}$ is an $R \times R$ submatrix, with R being the blocking parameter. The goal is to obtain the following decomposition:

$$\bar{A} = \begin{bmatrix} \bar{L}_{11} & 0 \\ \bar{L}_{21} & \bar{L}_{22} \end{bmatrix} \begin{bmatrix} \bar{U}_{11} & \bar{U}_{21} \\ 0 & \bar{U}_{22} \end{bmatrix}. \tag{17}$$

Therefore,

$$\bar{A} = \begin{bmatrix} \bar{L}_{11}\bar{U}_{11} & \bar{L}_{11}\bar{U}_{12} \\ \bar{L}_{21}\bar{U}_{11} & \bar{L}_{21}\bar{U}_{12} + \bar{L}_{22}\bar{U}_{22} \end{bmatrix}. \tag{18}$$

First, the LU factorization $\bar{L}_{11}\bar{U}_{11} = \bar{A}_{11}$ is performed. Then, the systems $\bar{A}_{21} = \bar{L}_{21}\bar{U}_{11}$ and $\bar{A}_{12} = \bar{L}_{11}\bar{U}_{12}$ are solved to obtain $\bar{L}_{21}$ and $\bar{U}_{12}$, respectively. Finally, $\bar{A}'_{22} = \bar{A}_{22} - \bar{L}_{21}\bar{U}_{12}$, which is the Schur complement of $\bar{A}_{11}$ with respect to $\bar{A}$, is evaluated (see Table XI). The decomposition is then continued by recursively applying the same steps to the updated $\bar{A}_{22}$ block. According to this scheme, matrix multiplication becomes the dominant operation.

APPENDIX C
PARALLEL BLOCK LU DECOMPOSITION

The parallel LU factorization algorithm proceeds by steps in a blocked Gaussian-elimination fashion. At each step, one column of blocks of the matrix $\bar{A}$ (Fig. 3) is factored. Then, a block of rows is computed. Finally, the update of the remaining matrix occurs. Supposing that the first step is completed (Fig. 13), the submatrices $\bar{L}_1$ and $\bar{U}_1$ have already been evaluated, and the remaining part of the matrix

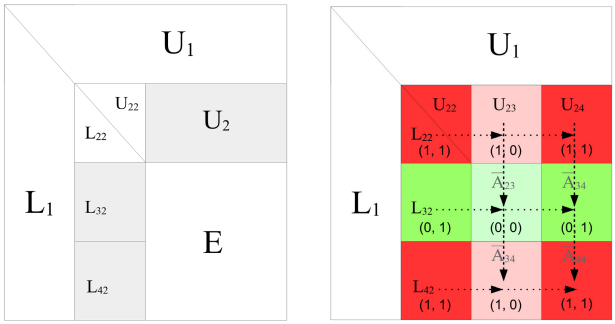


Fig. 13. Parallel block LU factorization algorithm. (a) Matrix update at the second step: trapezoidal submatrices $\bar{L}_1$ and $\bar{U}_1$ have already been evaluated in the first step. (b) Communications at the second step: dashed lines correspond to column broadcasts and dotted lines correspond to row broadcasts.

$\begin{bmatrix} \bar{A}'_{22} & \bar{A}'_{23} & \bar{A}'_{24} \\ \bar{A}'_{31} & \bar{A}'_{33} & \bar{A}'_{34} \\ \bar{A}'_{41} & \bar{A}'_{43} & \bar{A}'_{44} \end{bmatrix}$ to be factorized has already been updated at the first step.

We illustrate the second step as an example.

The processes belonging to the second grid column perform a local LU factorization on that block column. The result produces $\bar{L}_{22}$, $\bar{L}_{32}$, and $\bar{L}_{42}$ blocks of L and the block $\bar{U}_{22}$. Then, the row block $\bar{U}_2 = [\bar{U}_{23}\ \bar{U}_{24}]$ distributed among one row of the process grid is obtained through the solution of a set of lower triangular systems. To this purpose, $\bar{L}_{22}$ has to be sent to each process of the grid row that will solve the systems $\bar{A}'_{23} = \bar{L}_{22}\bar{U}_{23}$ and $\bar{A}'_{24} = \bar{L}_{22}\bar{U}_{24}$ independently. Communications needed to update the trailing submatrix $\bar{E}$ occur in two stages. First, each process that owns one of the parts $\bar{L}_{32}$ or $\bar{L}_{42}$ sends it to every process belonging to the same grid row. This can be done together with the broadcast of the block $\bar{L}_{22}$. Subsequently, each process that owns a block of $\bar{U}_2$ sends it to the other processes belonging to the same grid column. Now, each process can update the block of matrix E it owns independently

$$\bar{E} = \begin{bmatrix} \bar{A}''_{33} = \bar{A}'_{33} - \bar{L}_{32}\bar{U}_{23} & \bar{A}''_{34} = \bar{A}'_{34} - \bar{L}_{32}\bar{U}_{24} \\ \bar{A}''_{43} = \bar{A}'_{43} - \bar{L}_{42}\bar{U}_{23} & \bar{A}''_{44} = \bar{A}'_{44} - \bar{L}_{42}\bar{U}_{24} \end{bmatrix}.$$

REFERENCES

[1] F. T. Ulaby, R. K. Moore, and A. K. Fung, *Microwave Remote Sensing: Active and Passive. Volume 2-Radar Remote Sensing and Surface Scattering and Emission Theory*. Reading, MA, USA: Addison-Wesley, 1982.

[2] L. Tsang and J. A. Kong, *Scattering of Electromagnetic Waves: Advanced Topics*. Hoboken, NJ, USA: Wiley, 2004.

[3] G. Galilei, *Dialogue Concerning the Two Chief World Systems, Ptolemaic and Copernican*. Berkeley, CA, USA: Univ. California Press, 2023.

[4] L. Rayleigh, *Theory of Sound*, vol. 2. New York, NY, USA: Macmillan, 1920.

[5] P. Imperatore, A. Iodice, M. Pastorino, and N. Pinel, "Modelling scattering of electromagnetic waves in layered media: An up-to-date perspective," *Int. J. Antennas Propag.*, vol. 2017, no. 1, 2017, Art. no. 7513239.

[6] C. Bourlier, *Radar Propagation and Scattering in a Complex Maritime Environment*. London, U.K.: Elservier, 2018.

[7] P. Imperatore, A. Iodice, and D. Riccio, "Electromagnetic wave scattering from layered structures with an arbitrary number of rough interfaces," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 4, pp. 1056–1072, Apr. 2009.

[8] N. Pinel, N. Déchamps, C. Bourlier, and J. Saillard, "Bistatic scattering from one-dimensional random rough homogeneous layers in the high-frequency limit with shadowing effect," *Waves Random Complex Media*, vol. 17, no. 3, pp. 283–303, Jun. 2007.

[9] R. F. Harrington, *Field computation by moment methods*. Oxford, U.K.: Oxford Univ. Press, 1996.

[10] C. Bourlier, N. Pinel, and G. Kubické, *Method of Moments for 2D Scattering Problems: Basic Concepts and Applications*. Hoboken, NJ, USA: Wiley, 2013.

[11] L. Tsang, J. A. Kong, K.-H. Ding, and C. O. Ao, *Scattering of Electromagnetic Waves: Numerical Simulations*. Hoboken, NJ, USA: Wiley, 2004.

[12] D. A. Kapp and G. S. Brown, "A new numerical method for rough-surface scattering calculations," *IEEE Trans. Antennas Propag.*, vol. 44, no. 5, pp. 711–722, May 1996.

[13] D. Holliday, L. L. DeRaad, and G. J. St-Cyr, "Forward-backward method for scattering from imperfect conductors," *IEEE Trans. Antennas Propag.*, vol. 46, no. 1, pp. 101–107, Jan. 1998.

[14] R. J. Adams and G. S. Brown, "An iterative solution of one-dimensional rough surface scattering problems based on a factorization of the Helmholtz operator," *IEEE Trans. Antennas Propag.*, vol. 47, no. 4, pp. 765–767, Apr. 1999.

[15] M. Rodriguez Pino, L. Landesa, J. L. Rodriguez, F. Obelleiro, and R. J. Burkholder, "The generalized forward-backward method for analyzing the scattering from targets on ocean-like rough surfaces," *IEEE Trans. Antennas Propag.*, vol. 47, no. 6, pp. 961–969, Jun. 1999.

[16] A. Iodice, "Forward-backward method for scattering from dielectric rough surfaces," *IEEE Trans. Antennas Propag.*, vol. 50, no. 7, pp. 901–911, Jul. 2002.

[17] H.-T. Chou and J. T. Johnson, "A novel acceleration algorithm for the computation of scattering from rough surfaces with the forward-backward method," *Radio Sci.*, vol. 33, no. 5, pp. 1277–1287, Sep. 1998.

[18] H.-T. Chou and J. T. Johnson, "Formulation of forward-backward method using novel spectral acceleration for the modeling of scattering from impedance rough surfaces," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 1, pp. 605–607, Jan. 2000.

[19] D. Torrungrueng, H.-T. Chou, and J. T. Johnson, "A novel acceleration algorithm for the computation of scattering from two-dimensional large-scale perfectly conducting random rough surfaces with the forward-backward method," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 4, pp. 1656–1668, Jul. 2000.

[20] D. Torrungrueng, J. T. Johnson, and H.-T. Chou, "Some issues related to the novel spectral acceleration method for the fast computation of radiation/scattering from one-dimensional extremely large scale quasi-planar structures," *Radio Sci.*, vol. 37, no. 2, pp. 1–20, Apr. 2002.

[21] L. Tsang, C. H. Chang, and H. Sangani, "Banded matrix iterative approach to Monte-Carlo simulations of scattering of waves by large-scale random rough surface problems: TM case," *Electron. Lett.*, vol. 29, no. 2, pp. 166–167, 1993.

[22] L. Tsang, C. H. Chang, H. Sangani, A. Ishimaru, and P. Phu, "A banded matrix iterative approach to Monte Carlo simulations of large-scale random rough surface scattering: TE case," *J. Electromagn. Waves Appl.*, vol. 29, no. 9, pp. 1185–1200, 1993.

[23] L. Tsang, C. H. Chan, K. Pak, and H. Sangani, "Monte-Carlo simulations of large-scale problems of random rough surface scattering and applications to grazing incidence with the BMIA/canonical grid method," *IEEE Trans. Antennas Propag.*, vol. 43, no. 8, pp. 851–859, Aug. 1995.

[24] J. Michalakes, "HPC for weather forecasting," in *Parallel Algorithms in Computational Science and Engineering*. Cham, Switzerland: Springer, 2020, pp. 297–323.

[25] P. Imperatore, A. Pepe, and E. Sansosti, "High performance computing in satellite SAR interferometry: A critical perspective," *Remote Sens.*, vol. 13, no. 23, p. 4756, Nov. 2021.

[26] C. A. S. J. Gulo, A. C. Sementille, and J. M. R. S. Tavares, "Techniques of medical image processing and analysis accelerated by high-performance computing: A systematic literature review," *J. Real-Time Image Process.*, vol. 16, no. 6, pp. 1891–1908, Dec. 2019.

[27] L.-X. Guo, A.-Q. Wang, and J. Ma, "Study on EM scattering from 2-D target above 1-D large scale rough surface with low grazing incidence by parallel MoM based on PC clusters," *Prog. Electromagn. Res.*, vol. 89, pp. 149–166, 2009.

[28] C. Jia, L. Guo, and P. Yang, "EM scattering from a target above a 1-D randomly rough sea surface using GPU-based parallel FDTD," *IEEE Antennas Wireless Propag. Lett.*, vol. 14, pp. 217–220, 2015.

[29] W.-Q. Jiang, M. Zhang, P.-B. Wei, and X.-F. Yuan, "CUDA-based SSA method in application to calculating EM scattering from large two-dimensional rough surface," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 4, pp. 1372–1382, Apr. 2014.

[30] Y. Zhang, Z. Lin, X. Zhao, and T. K. Sarkar, "Performance of a massively parallel higher-order method of moments code using thousands of CPUs and its applications," *IEEE Trans. Antennas Propag.*, vol. 62, no. 12, pp. 6317–6324, Dec. 2014.

[31] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas Propag.*, vol. 56, no. 7, pp. 2130–2133, Jul. 2008.

[32] P. Imperatore et al., "Method of moments-based scattering from rough surfaces using high-performance parallel implementations for heterogeneous computing architectures," in *Proc. Photon. Electromagn. Res. Symp.-Spring (PIERS-Spring)*, May 2025, pp. 1–6.

[33] *TOP500 Global Rankings*. 2025. Accessed: May 15, 2025. [Online]. Available: https://top500.org/lists/top500/2024/11/

[34] T. G. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*. London, U.K.: Pearson, 2004.

[35] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*. Amsterdam, The Netherlands: Elsevier, 2012.

[36] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Reading, MA, USA: Addison-Wesley, 1995.

[37] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2013.

[38] E. Anderson et al., *LAPACK Users' Guide*. Philadelphia, PA, USA: SIAM, 1999.

[39] J. Demmel, "LAPACK: A portable linear algebra library for supercomputers," in *Proc. IEEE Control Syst. Soc. Workshop Comput.-Aided Control Syst. Design*, Dec. 1989, pp. 1–7.

[40] J. Dongarra et al., "PLASMA: Parallel linear algebra software for multicore using OpenMP," *ACM Trans. Math. Softw.*, vol. 45, no. 2, pp. 1–35, May 2019.

[41] M. Abalenkovs et al., "Plasma 17.1 functionality report," LAPACK Work. Note, Univ. Tennessee, Tech. Rep. 293, 2017.

[42] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, MA, USA: MIT Press, 2007.

[43] J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek, "Achieving numerical accuracy and high performance using recursive tile LU factorization with partial pivoting," *Concurrency Comput., Pract. Exper.*, vol. 26, no. 7, pp. 1408–1431, May 2014.

[44] E. Agullo et al., "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," in *Proc. J. Phys., Conf.*, vol. 180, Jul. 2009, Art. no. 012037.

[45] *Magma*. Accessed: Mar. 15, 2024. [Online]. Available: https://icl.utk.edu/magma/

[46] A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham, "Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 476, no. 2243, Nov. 2020, Art. no. 20200110.

[47] L. Blackford et al., *ScaLAPACK Users' Guide*. Philadelphia, PA, USA: SIAM, 1997.

[48] L. S. Blackford et al., "ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance," in *Proc. ACM/IEEE Conf. Supercomputing*, Jan. 1996, p. 5–es.

[49] *ScaLAPACK-Scalable Linear Algebra PACKage*. Accessed: Mar. 7, 2024. [Online]. Available: http://www.netlib.org/scalapack

[50] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, "ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers," in *Proc. 4th Symp. Frontiers Massively Parallel Comput.*, Los Alamitos, CA, USA, Oct. 1992, pp. 120–127.

[51] C. H. Koelbel, *The High Performance Fortran Handbook*. Cambridge, MA, USA: MIT Press, 1994.

[52] J. Dongarra et al., *Sourcebook of Parallel Computing*, vol. 3003. San Mateo, CA, USA: Morgan Kaufmann, 2003.

[53] E. I. Thorsos, "The validity of the Kirchhoff approximation for rough surface scattering using a Gaussian roughness spectrum," *J. Acoust. Soc. Amer.*, vol. 83, no. 1, pp. 78–92, Jan. 1988.

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

[55] T. Elfouhaily, B. Chapron, K. Katsaros, and D. Vandemark, "A unified directional spectrum for long and short wind-driven waves," *J. Geophys. Res., Oceans*, vol. 102, no. C7, pp. 15781–15796, 1997.

[56] L. D'Amore, V. Mele, D. Romano, and G. Laccetti, "Multilevel algebraic approach for performance analysis of parallel algorithms," *Comput. Informat.*, vol. 38, no. 4, pp. 817–850, 2019.
[57] A. H. Karp and H. P. Flatt, "Measuring parallel processor performance," *Commun. ACM*, vol. 33, no. 5, pp. 539–543, May 1990.
[58] P. Imperatore, A. Iodice, and D. Riccio, "Consistency and validity of perturbative formulations for scattering from rough multilayers," *IEEE Trans. Antennas Propag.*, vol. 60, no. 4, pp. 2019–2027, Apr. 2012.
[59] *NVIDIA Cusolver: A Collection of Dense and Sparse Direct Solvers*. Accessed: Nov. 21, 2025. [Online]. Available: https://developer.nvidia.com/cusolver

Pasquale Imperatore (Senior Member, IEEE) received the Laurea degree (summa cum laude) in electronic engineering and the Ph.D. degree in electronic and telecommunication engineering from the University of Naples Federico II, Naples, Italy, in 2001 and 2010, respectively.

He is currently a Senior Research Scientist with the National Research Council (CNR), Institute for Electromagnetic Sensing of the Environment (IREA), Naples. Previously, he worked with Wise S.p.A., Naples, and later with CREATE-NET, Trento, Italy. Subsequently, he joined the Department of Biomedical, Electronic, and Telecommunication Engineering, University of Naples Federico II. His research interests include microwave remote sensing and electromagnetic propagation and scattering, with particular emphasis on theoretical scattering models, waves in random layered media, SAR data modeling, processing, and calibration, SAR interferometry and tomography, parallel algorithms, and high-performance computing (HPC).

Dr. Imperatore is a member of the Topical Advisory Panel for Geomatics (MDPI) and the Earth and Planetary Science Scientific Advisory Board for IntechOpen. He served as a lead editor for several journal special issues focused on electromagnetics and remote sensing and a reviewer for numerous peer-reviewed journals. He is an Associate Editor for IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and an Associate Editor for *Frontiers in Imaging*. Moreover, he has coedited two books in the field of remote sensing and geospatial technology.

Francesco Gregoretti received the Ph.D. degree in applied mathematics and computer science from the University of Naples Federico II, Naples, Italy, in 2006.

Since 2012, he has had a permanent position as a Researcher with Italian National Research Council (ICAR-CNR). His research interests include scientific computing, parallel algorithms, and high-performance computing.

Nicolas Pinel received the Engineering degree and the M.S. degree in electronics and electrical engineering from Polytech Nantes, Nantes, France, in 2003, and the Ph.D. degree from the University of Nantes, Nantes, in 2006.

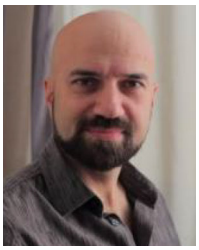
He was with the Institut d'Electronique et des Technologies du numéRique (IETR) Laboratory, Nantes, for seven years, and Alyotech, Rennes, France, for four years. In 2017, he joined Icam, Carquefou, France, where he works as an Associate Professor. In particular, he deals with asymptotic and numerical scattering models, waves in random and complex layered media, and processing. He has authored or co-authored more than 50 peer-reviewed journal articles, books, and book chapters. His research interests include the areas of radar and optical remote sensing, and electromagnetic scattering and propagation.

Mehwish Nisar received the Ph.D. degree in electronics engineering from the La Sapienza University of Rome, Rome, Italy, in 2022.

Currently, she is working as a Research Scientist with the Institute for Electromagnetic Sensing of the Environment (IREA), National Research Council of Italy (CNR), Naples, Italy. Her primary research interests include electromagnetic modeling, SAR processing, analytical and numerical scattering models, and efficient parallel algorithm development for large-scale simulations.

Christophe Bourlier received the M.S. degree in electronics from the University of Rennes, Rennes, France, in 1995, and the Ph.D. degree from the Systéme Electronique et Informatique (SEI) Laboratory, Nantes, France, in 1999.

Now, he is with the Institute of Electronics and Numerical Technologies (IETR) Laboratory, Polytech Nantes, Nantes University, Nantes, France. He works as a Full Researcher at the National Center for Scientific Research on electromagnetic wave scattering from rough surfaces (ocean-like surfaces) and manufactured objects for microwaves and infrared remote sensing applications and radar signatures. He is the author of more than 250 journals and conference papers. He is the author and co-author of four books entitled *Shadowing Function From Randomly Rough Surfaces: Derivation and Application*, *Radar Propagation and Scattering in a Complex Maritime Environment: Modeling and Simulation From MATLAB*, *Method of Moments for 2-D Scattering Problems: Basic Concepts and Applications*, and *Electromagnetic Wave Scattering From Random Rough Surfaces-Asymptotic Models*.



Diego Romano received the Ph.D. degree in computational and computer science from the University of Naples Federico II, Naples, Italy, in 2012.

He has been with Italian National Research Council (CNR), Naples, since 2008. He is currently a Senior Research Scientist with CNR. His work explores the application of parallel and distributed computing to solve complex problems, including developing parallel algorithms for edge computing, enhancing the performance of numerical algorithms on multicore CPUs and GPUs, and applying AI to hyperspectral image classification. His research focuses on high-performance computing (HPC), GPU computing, scientific computing, and computer graphics.

Antonio Iodice (Senior Member, IEEE) is currently a Full Professor of electromagnetic fields with the University of Naples Federico II, Naples, Italy. He is the Coordinator of the B.S. and M.S. degrees programs in telecommunications and digital media engineering with the University of Naples Federico II. He has been a principal investigator for research projects on remote sensing and wireless propagation and has authored two books and over 110 articles published in peer-reviewed international journals.

Dr. Iodice is a URSI Senior Member. He was a recipient of the 2009 Sergei A. Schelkunoff Transactions Prize Paper Award from the IEEE Antennas and Propagation Society.