Postprint of: Topa T., Noga A., Stefański T., FPGA Acceleration of Matrix-Assembly Phase of RWG-Based MoM, IEEE Antennas and Wireless Propagation Letters Vol. 21, iss. 9 (2022), pp. 1847-1851, DOI: 10.1109/LAWP.2022.3183168

FPGA Acceleration of Matrix-Assembly Phase of RWG-based MoM

Tomasz Topa, *Member, IEEE*, Artur Noga, *Member, IEEE*, Tomasz P. Stefański, *Senior Member, IEEE*

***Abstract*—In this letter, the field-programmable-gate-array accelerated implementation of matrix-assembly phase of the method of moments (MoM) is presented. The solution is based on a discretization of the frequency-domain mixed potential integral equation using the Rao-Wilton-Glisson basis functions and their extension to wire-to-surface junctions. To take advantage of the given hardware resources (i.e., Xilinx Alveo U200 accelerator card), nine independent processing paths/runtime efficient compute units are developed and synthesized. Numerical results provided for a quadrifilar spiral antenna mounted on a conductive handset box show that the proposed parallelization scheme performs 9.53× faster than a traditional (i.e., serial) central processing unit (CPU) MoM implementation, and about 1.67× faster than a parallel six-core CPU MoM implementation.**

***Index Terms*—Method of moments, Field programmable gate arrays, Hardware acceleration.**

I. Introduction

THE method of moments (MoM) based on the electric field integral equation (EFIE) is one of the flagship computational tools, which has proven its usefulness and accuracy in solving a vast variety of real-world electromagnetics problems [1]. The potential of MoM, however, is impaired by its well-known high demands of computer resources in terms of central processing unit (CPU) time and memory storage needed to perform computations. Fortunately, many of these computations can be carried out independently, providing an opportunity for parallelization and acceleration of MoM solvers. A parallel implementation of EFIE-based MoM typically involves splitting the most computationally intensive parts of workload, i.e., solving the system of linear equations and assembling the impedance matrix, into tasks of various granularities, targeting architectural strengths of given computational resources, and minimizing data transfers between hardware components. Although much development work has been done to speed up MoM computations, perfect performance across multiple parallel processing platforms still remains a challenging problem. It stems from the complexity of MoM parallelization schemes due to a high level of concurrency and heterogeneity for a wide variety of accelerators and co-processors available currently on the market.

The first hardware-accelerated implementation of MoM on graphics processing unit (GPU) is presented in [2], where the GeForce 7600GT graphics card is employed to assembly the impedance matrix and solve the system of linear equations. Following this idea, CUDA-enabled graphics cards are employed to evaluate the current distribution on a short linear dipole antenna [3] and accelerate the electromagnetic scattering analysis of a square conductive plate [4]. Since then, various GPU accelerations have been proposed for MoM, resulting in large savings in computation times [5]–[7]. However, all the proposed hybridization schemes suffer from poor load balancing, making the computations less intensive, and thus, more time consuming. To address this issue, Kolundzija and Zoric [8] split the assembly phase of MoM into two different tasks, namely evaluation of self and non-self terms, and execute them respectively on CPU and GPU. The same task scheduling strategy is applied in [9]–[11], however, in order to reduce CPU stalls, the assembly of impedance matrix is partially overlapped with solving the MoM matrix equation. Additionally, in order to balance the workload between CPU and GPU, the processed data (i.e., the impedance matrix) is divided into small sets (i.e., rectangular tiles) that are reshaped during computations. At the same time, the research focused on the development of accelerated MoM codes on reconfigurable computing devices have been carried out. In [12] and [13], the FPGA-based implementations of `getrf()` and `gmres()` LAPACK routines are respectively presented for solving complex-valued systems of MoM equations. These designs (i.e., the MoM implementations) focus on data integrity, optimal resource utilization, scalability and load balancing. In [14], the PCI-hosted Nallatech 385A board featuring a single Intel Arria 10 GX 1150 FPGA is employed to accelerate the assembly phase of wire-grid MoM framework. Given the capabilities of underlying device architecture and its performance potential, only a single processing path (PP) is synthesized for handling both computations and data transfers.

In this letter, the FPGA acceleration for the matrix-assembly phase of MoM using the RWG basis functions is presented. That is, the mixed potential integral equation (MPIE), which is a modification of EFIE [1], [15], is solved on CPU whereas the assembly phase is accelerated on FPGA. The implementation includes surface and line current densities and provides, like the approach described in [5], the possibility for handling arbitrarily-shaped conducting objects including wires (W), bodies (B) and wire-body junctions (J). The developed so-

Manuscript received Month 00, 2022; revised Month 00, 2022, accepted Month 00, 2022. This work was supported by the Polish Ministry of Science and Higher Education funding for statutory activities (BK-246/RAu-11/2022).

T. Topa and A. Noga are with the Department of Electronics, Electrical Engineering and Microelectronics, Silesian University of Technology, Gliwice 44-100, Poland (e-mails: tomasz.topa@polsl.pl, artur.noga@polsl.pl).

T. P. Stefański is with the Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk 80-233, Poland (e-mail: tomasz.stefanski@pg.edu.pl).

0000–0000/00$00.00 © 0000 IEEE

lution can be useful for the acceleration of design processes (i.e., optimizations) of microwave circuits and antennas. The application of FPGA not only speeds up computations but also keeps the power consumption at a low level, which allows one to reduce the environmental effects of computer simulations. The machines used in this study consist of hexa-core Intel i7-8700K CPU with NVIDIA Titan Xp GPU (machine M1), and quad-core Intel i7-3820 CPU with Xilinx Alveo U200 PCIe accelerator card (machine M2). We employ two different machines (M1 and M2) in our investigations because the FPGA card is passively cooled and requires special computer case with large air flow for its proper operation.

II. MoM overview

The MoM framework considered in this letter converts MPIE into the matrix equation

$$\mathbf{Z} \cdot \mathbf{I} = \mathbf{V} \tag{1}$$

where $\mathbf{Z}$ denotes the square complex-valued impedance matrix, $\mathbf{I}$ is the column vector of unknown current expansion coefficients, and $\mathbf{V}$ represents the voltage-excitation vector. Because the framework is aimed at handling the electromagnetic analysis of conductive wire-body objects, the impedance matrix can be written as

$$\mathbf{Z} = \begin{bmatrix} \mathbf{Z}^{\mathrm{BB}} & \mathbf{Z}^{\mathrm{BW}} & \mathbf{Z}^{\mathrm{BJ}} \\ \mathbf{Z}^{\mathrm{WB}} & \mathbf{Z}^{\mathrm{WW}} & \mathbf{Z}^{\mathrm{WJ}} \\ \mathbf{Z}^{\mathrm{JB}} & \mathbf{Z}^{\mathrm{JW}} & \mathbf{Z}^{\mathrm{JJ}} \end{bmatrix}. \tag{2}$$

The submatrices $\mathbf{Z}^{\gamma\beta}$, where $\gamma, \beta = B, W$ or J, represent mutual electromagnetic couplings between the current modes belonging to the subsets indicated by corresponding superscripts. Once the matrix entries Z_{mn} and the column vector entries V_m are evaluated, the dense linear complex-valued matrix equation (1) is solved by standard methods of linear algebra. However, in the context of MoM simulations, one of the most widely used techniques for solving (1) is the lower and upper (LU) decomposition offering many advantages over other possible approaches [3], [4].

III. Implementation Issues

The Alveo U200 accelerator card [16] used in this study is built on the Xilinx 16 nm UltraScale FPGA chip. Its advantages stem from the implementation of computations on reconfigurable hardware rather than their execution as a sequential code on a processor with fixed architecture. Therefore, the Alveo accelerator is adaptable to changing algorithms, arithmetics, and data structures, thus, it is capable to accelerate almost any computations without changing the hardware. Available resources of the Alveo card are given in Table I. Like many modern accelerators, the card offers fully IEEE-754 compliant single and double-precision floating point operations, and possesses the capability for overlapping data transfers with computations. A simplified version of the developed OpenCL code for assembling the impedance matrix of MoM is illustrated in Fig. 1. Based on the approach described in [14], the singularities arising from (1) are evaluated on the CPU host (routine `Z_self_terms()`)

```
#include "CL/xcl2.hpp"

// load the geometry
1: call geoinput(x, y, z, rad, ...);
...
// create command queue to perform the computations
2: cl_command_queue queue = clCreateCommandQueue(context, device, ...,
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE, ...);
...
// create the event list
3: cl_event events[32];
...
// allocate memory on the device for storing the results/output data
4: for (int i = 0; i < 16; i++)
5:  cl_mem cz_d[i] = clCreateBuffer(context, CL_MEM_WRITE_ONLY, ...);
...
// copy input data to the device memory
6: call clEnqueueWriteBuffer(queue, x_d, CL_FALSE, 0, ..., x, ...);
...
// launch the kernel/computation on the device
7: call Z_non_self_terms(queue, device, x_d, ..., cz_d , ..., &events);
// calculate the submatrices Zjw, Zwj, Zbj, Zjb, Zjj and the self-terms on the host
8: call Z_self_terms(cz, x, y, x, rad, ...);
9: call cpu_Z_junc(cz, x, y, x, rad, ...);
...
// release openCL objects
10: call clReleaseMemObject(cz_d[i], x_d, y_d, z_d, rad_d, ...);

//  ----------------- routine Z_non_self_terms -----------------
11: void Z_non_self_terms(cl_command_queue q, ..., float* x_d, ...,
clDoubleComplex* cz_d, ..., cl_event ev){
...
// calculate wire-wire, wire-body, body-wire and body-body interactions simultaneously
12: call clEnqueueNDRangeKernel(q, „Z_ww_kernel", cz_d[0], ..., ev[0]);
13: call clEnqueueReadBuffer(q, cz_d[0],CL_FALSE, ..., cz[off1],0, ev[1]);
14: call clEnqueueNDRangeKernel(q, „Z_wb_kernel", cz_d[1], ..., ev[2]);
15: call clEnqueueReadBuffer(q, cz_d[1],CL_FALSE, ..., cz[off2],0, ev[3]);
16: call clEnqueueNDRangeKernel(q, „Z_bw_kernel", cz_d[2], ..., ev[4]);
17: call clEnqueueReadBuffer(q, cz_d[2],CL_FALSE, ..., cz[off3],0, ev[5]);
18: for (int i = 0; i < num_iter; i++){
19:  call clEnqueueNDRangeKernel(q, „Z_bb_1_kernel", cz_d[3+6*i], i, ...,
ev[6+12*i]);
20:  call clEnqueueReadBuffer(q, cz_d[(3+6*i)],CL_FALSE, ...,
cz[6*i*off4],0, ev[7+12*i]);
...
21:  call clEnqueueNDRangeKernel(q, „Z_bb_6_kernel", cz_d[8+6*i], i, ...,
ev[16+12*i]);
22:  call clEnqueueReadBuffer(q, cz_d[(8+6*i)],CL_FALSE, ...,
cz[(5+6*i)*off4],0, ev[17+12*i]);
23: }
24: clFlush();
25: }
```

Fig. 1. Simplified FPGA-oriented OpenCL code for assembling impedance matrix of RWG-based MoM.

whilst the non-singular integrals on the FPGA device (routine `Z_non_self_terms()`). CPU is also employed to calculate the submatrices $\mathbf{Z}^{\mathrm{J}\beta}$ and $\mathbf{Z}^{\gamma\mathrm{J}}$ ($\gamma, \beta = B, W$ or J). This is mainly due to rather small number of wire-body junctions present in the investigated structure (results of numerical experiments show that computations of the submatrices $\mathbf{Z}^{\mathrm{J}\beta}$ and $\mathbf{Z}^{\gamma\mathrm{J}}$ should be offloaded to the device only when the analyzed structure contains more than 250 body-wire junctions, otherwise the CPU-based approach is more efficient). In order to fully exploit the processing power of device, nine concurrently executed tasks are performed – two for handling the wire-body interactions, one for evaluating the electromagnetic coupling between the wires, and six – between the bodies. The hardware implementation (i.e., synthesis) of PPs occupies resources as given in Table I. As one can notice, the utilization of LUTs and DSP slices is a bottleneck for the proposed FPGA implementation. It turned out that our attempts to attach an additional PP were unsuccessful because FPGA design tools reported problems at the level of interconnection routing. In general, designed digital circuits are not placed in isolation into FPGA chips, i.e., designed circuits are a part of a valid FPGA design which includes various additional components in order to run target applications. Since a single `clCreateBuffer()` call allocates a limited amount of

TABLE I
RESOURCES REQUIRED FOR ACCELERATED IMPLEMENTATION OF MATRIX-ASSEMBLY PHASE OF MOM.

		2 wire-body PPs		wire-wire PP		6 body-body PPs	
Resources	**Available**	**Used**	**Utilization (%)**	**Used**	**Utilization (%)**	**Used**	**Utilization (%)**
Digital signal processing (DSP) blocks/slices	6840	1580	23	399	6	3126	46
Look-up tables (LUTs)	1182 K	264 K	22	93 K	8	567 K	48
Flip-flops (FFs)	2364 K	304 K	13	103 K	4	636 K	27
36 Kb RAM blocks (BRAMs)	6480	254	4	15	<1	672	10
Registers	2364 K	48 K	2	15 K	<1	156 K	7

global memory on the card, four sets of four output data buffers are created (one set per global memory bank) to utilize all available device main-memory resources (line 5). Following the current implementation strategies, the data transfers between the host and the device are overlapped with the FPGA kernel execution (line 12 – 22). Additionally, global memory reads and writes are pipelined and locally cached through 36 Kb BRAM-based memory banks. In our implementation of MoM computations, the estimated frequency of card is equal to 312 MHz whereas the target frequency for this card is equal to 300 MHz. In general, an estimated frequency higher than the one associated with the device target indicates that designed compute units can run in hardware. Once the assembly phase is finished, the multithreaded `zgesv()` routine from MKL library [17] is invoked to solve the MoM matrix equation. Using this routine here serves two major benefits. Firstly, it simplifies the kernel development process (i.e., matrix assembly), and secondly – it improves its performance (i.e., more hardware resources are utilized to implement the kernel). Furthermore, as presented and concluded in [12], [18] and [19], solving a complex-valued system of linear equation on FPGA may be less effective than the relevant multicore approach especially, when the LU decomposition scheme is employed.

IV. RESULTS

To evaluate the performance potential provided by the described parallelization scheme, a quadrifilar spiral antenna (QSA) mounted on a conductive handset box of a dimension of $0.2 \times 0.2 \times 0.2$ m^3 (height $\times$ width $\times$ depth) is considered [20]. The antenna consists of four separate spiral arms placed at 90° to each other, as presented in Fig. 2. Each arm of the total length of 256 mm (the spiral constant is equal to $a = 1.42$ mm/rad) is connected to the top surface of handset box by a short (0.03 m) wire of the radius of 0.5 mm. The arms are fed with delta-gap voltage sources with 90° phase shift between feeding signals, i.e., 0°, 90°, 180°, and 270°. For the purpose of numerical modeling, the structure is subdivided into 4800 triangular patches (the handset) and 148 linear segments (the QSA antenna). The total number of unknowns (i.e., degrees of freedom) associated with the model is equal to 7684, which corresponds to a memory footprint of 900 MB when complex double-precision arithmetic is employed.

Fig. 3 shows the input impedance of the OSA antenna in Fig. 2, computed at increments of 10 MHz in the frequency range from 1.4 GHz to 3.1 GHz. The results are in very good agreement with the output data from CONCEPT-II taken as the reference [21]. Since the spiral arms are located 0.03 m above the top surface of handset box (see Fig. 2), the input impedance of antenna becomes smooth enough to meet the impedance matching requirements for the intended operating frequency band of 2.4835–2.5 GHz (i.e., operating frequency band of the big low earth orbit satellite communication system). The total execution time taken to solve the problem on a single CPU core (the reference approach on M1) is equal to 15425 s (90.2 s per single frequency point). This time is reduced to 2726 s (5.66×) when all available CPU cores are used on the machine M1, and to 2253 s (6.85×) when the described CPU/FPGA parallelization scheme is employed on the machine M2. The assembly phase takes 8841 s on M1 for the sequential CPU implementation, 1546 s for the six-core implementation, and 928 s for the proposed hybridization

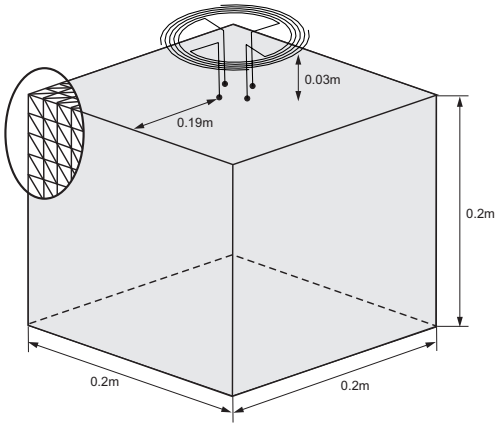

Fig. 2. Quadrifilar spiral antenna mounted on a conductive cubic handset box.

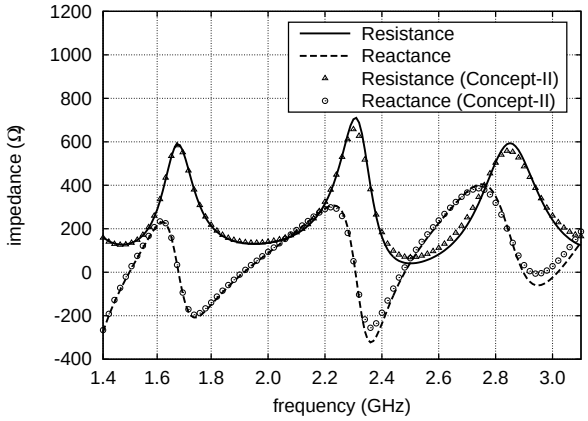

Fig. 3. Input impedance of QSA antenna in Fig. 2 as function of frequency.

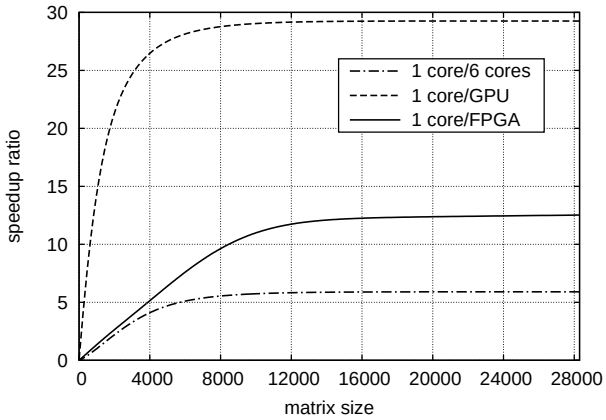

Fig. 4. Measured speedup of matrix-assembly phase of wire-body MoM as function of matrix size N (global memory installed on GPU device is of size 12 GB, the speedup analysis is limited to problem of size 28300 unknowns).

framework executed on the machine M2. As it can be seen, the FPGA-based assembly scheme performs 9.53× faster than the single core approach on M1, and 1.67× faster than the six-core implementation. Note that the last result (928 s) includes the time (75 s) required for transferring the data between CPU and the PCI-hosted FPGA device. The solution of MoM matrix equation (the four-core approach performed on M2) takes 1325 s (4.87× faster than the serial, i.e., a single core, implementation on M1) and for a given problem size of 7684 unknowns contributes more than 58% to the total execution time (2253 s) of the proposed hybridization scheme. To better illustrate the computing throughput provided by the considered heterogeneous processing platform, the measured speedup of matrix-assembly phase as a function of the problem/matrix size is presented in Fig. 4. The speedup is evaluated by changing both the discretization of antenna and the handset box depicted in Fig. 2, while keeping the ratio of numbers of the body and wire basis functions associated with the handset and antenna model reasonably constant over the range of N. The total power consumption during the computations on FPGA, that is computing the non-self terms, equals to around 56 W (62 W less comparing to the six-core approach). Most of this power (37 W) is consumed for calculating the body-body interactions, some (18 W) for handling the wire-body and wire-wire interactions, and the remaining (1 W) for transferring the output data between the host and the device.

To make the hardware acceleration analysis more complete, computations of the non-singular integrals have also been offloaded to Titan Xp GPU within the machine M1 and carried out concurrently with the computations of on-diagonal elements (singularities) performed on the host. In this case, the assembly of impedance matrix (7684×7684 elements) and the total execution take 310 s and 1490 s, respectively. Then, it corresponds to the speedup factors of around 28.52× and 10.35×, respectively, when compared to the reference single-core approach. However, obtained performance improvements of the matrix-assembly phase (4.98× over the six-core approach and 2.99× over the FPGA-based implementation) come at the price of the total power consumed by the Titan Xp device (230 W).

V. Conclusion

In this letter, the acceleration of matrix-assembly phase of wire-body MoM framework, employing the heterogeneous CPU/FPGA computing platform, is demonstrated. Timing and speedup results show that the proposed hybrid processing scheme performs 9.53× faster when compared to the reference single-core CPU implementation, and about 1.67× faster when compared to the six-core approach for the problem size of 7684 unknowns. It is also worth pointing out that the given FPGA-based approach consumes 2.11× less power than the multicore implementation and 4.11× less power than the GPU-based implementation, making it a suitable candidate for energy-efficient acceleration of MoM simulations.

Acknowledgment

The Titan Xp GPU used for the research presented in this paper was donated by the NVIDIA Corporation.

References

[1] R. F. Harrington, *Field Computation by Moment Methods*, New York, NY, USA: Macmillan, 1968.
[2] 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.
[3] M. J. Inman, A. Z. Elsherbeni, and C. J. Reddy, "CUDA based GPU solver for method of moments simulations," presented at the *26th Int. Annu. Rev. Prog. ACES*, Tampere, Finland, Apr. 25-29, 2010.
[4] E. Lezar, and D. Davidson, "GPU-accelerated method of moments by example: monostatic scattering," *IEEE Antennas Propag. Mag.*, vol. 52, no. 6, pp. 120–135, Dec. 2010.
[5] T. Topa, A. Noga, and A. Karwowski, "Adapting MoM with RWG basis functions to GPU technology using CUDA," *IEEE Antennas Wireless Propag. Lett.*, vol. 10, pp. 480–483, May 2011.
[6] D. De Donno, A. Esposito, G. Monti, and L. Tarricone, "MPIE/MoM acceleration with a general-purpose graphics processing unit," *IEEE Trans. Microw. Theory Techn.*, vol. 60, no. 9, pp. 2693–2701, Sep. 2012.
[7] X. Mu, H.-X. Zhou, K. Chen, and W. Hong, "Higher order method of moments with a parallel out-of-core LU solver on GPU/CPU platform," *IEEE Trans. Antennas Propag.*, vol. 62, no. 11, pp. 5634–5646, Nov. 2014.
[8] B. Kolundzija, and D. Zoric, "Efficient evaluation of MoM matrix elements using CPU and/or GPU," in *Proc. 6th Eur. Conf. on Antennas and Propag.*, Prague, Czech Republic, Mar. 2012, pp. 702–706.
[9] T. Topa, "Load-balanced Fortran-based out-of-GPU memory implementation of the method of moments," *IEEE Antennas Wireless Propag. Lett.*, vol. 16, pp. 813–816, Aug. 2016.
[10] A. Karwowski, A. Noga, and T. Topa "An efficient framework for analysis of wire-grid shielding structures over a road frequency range," *Radioengineering*, vol. 25, no. 4, pp. 629–636, Dec. 2016.
[11] A. Karwowski, A. Noga, and T. Topa "Computationally efficient technique for wide-band analysis of grid-like spatial shields for protection against LEMP effects," *Appl. Comput. Electromagn. Soc. J.*, vol. 32, no. 1, pp. 87–92, Jan. 2017.
[12] T. Hauser, A. Dasu, A. Sudarsanam, and S. Young, "Performance of a LU (lower and upper) decomposition on a multi-FPGA system compared to a low power commodity microprocessor system," *J. Scalable Comput.: Pract. Experience*, vol. 8, no. 4, pp. 373–385, Dec. 2007.
[13] A. Devi, M. Gandhi, K. Varghese, and D. Gope "Accelerating method of moments based package-board 3D parasitic extraction using FPGA," *Microw. Opt. Technol. Lett.*, vol. 58, no. 4, pp. 776–782, Apr. 2016.
[14] T. Topa, "Porting wire-grid MoM framework to reconfigurable computing technology," *IEEE Antennas Wireless Propag. Lett.*, vol. 19, no. 9, pp. 1630–1633, Sept. 2020.
[15] J. R. Mosig, "Arbitrarily shaped microstrip structures and their analysis with a mixed potential integral equation," *IEEE Trans. Microw. Theory Tech.*, vol. 36, no. 2, pp. 314–323, Feb. 1988.
[16] Xilinx, *Alveo Data Center Accelerator Card Platforms. User Guide*, Jul. 30, 2021. [Online]. Available: http://www.xilinx.com
[17] Intel, *Intel OneAPI Math Kernel Library. Developer Reference*, Jun. 28, 2021. [Online]. Available: http://www.intel.com

[18] T. de Matties, J. de Fine Licht, and T. Hoefler, "FBLAS: Streaming Linear Algebra Kernels on FPGA," presented at the *Int. Conf. HPC Netw. Stor. Anal. (SC'19)*, Denver, CO, USA, Nov. 17–22, 2019.

[19] S. Kestur, J. D. Devids, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Lixouri, Greece, Jul. 5–7, 2010, pp. 288–293.

[20] R. A. Abd-Alhameed, M. Mangoud, P. S. Excell, and K. Khalil, "Investigations of polarization purity and specific absorption rate for two dual-band antennas for satellite-mobile handsets," *IEEE Trans. Antennas Propag.*, vol. 53, no. 6, pp. 2108–2110, Jun. 2005.

[21] Hamburg University of Technology, *The CONCEPT-II*. [Online]. Available: http://www.tet.tuhh.de/en/concept-2/