



# Performance analysis of a millimeter wave MIMO channel estimation method in an embedded multi-core processor

Pablo M. Aviles<sup>1</sup> · Diego Lloria<sup>2</sup> · Jose A. Belloch<sup>1</sup> · Sandra Roger<sup>2</sup> · Almudena Lindoso<sup>1</sup> · Maximo Cobos<sup>2</sup>

Accepted: 22 March 2022 / Published online: 8 April 2022  
© The Author(s) 2022

## Abstract

The emerging Multi-Processor System-on-Chip (MPSoC) technology, which combines heterogeneous computing with the high performance of field programmable gate arrays (FPGA), is a promising platform for a large number of applications, including wireless communications and vehicular technology. In this specific application context, when multiple-input multiple-output (MIMO) scenarios are considered, the system usually has to manage a large number of communication links among sensors and antennas involving different vehicles and users. Millimeter wave (mmWave) communications are one of the key technology enablers toward achieving high data rates in beyond 5G systems (B5G). Communication at these frequency bands usually involves the use of large antenna arrays, often requiring high computational resources. One of the candidate platforms able to manage a huge number of communications is the Xilinx Zynq UltraScale+ EG Heterogeneous MPSoC, which is composed of a dual-core Cortex-R5, a quad-core ARM Cortex-A53, a graphics processing unit (GPU) and a high-end FPGA. This work analyzes the computational performance that requires a recent mmWave MIMO channel estimation algorithm in a platform of this kind. As a first approach, we will focus our work on the performance that can be achieved via the quad-core ARM Cortex-A53. To this end, we will use the libraries for numerical algebra (BLAS and LAPACK). The results show that our reference implementation is able to manage a large MIMO communication system with 256 antennas without exhausting platform resources.

**Keywords** MIMO Communication systems · Channel estimation · System-on-chip (SoC)

---

✉ Pablo M. Aviles  
paviles@ing.uc3m.es

Extended author information available on the last page of the article

## 1 Introduction

In the last decade, parallel systems are being employed in all segments of the industry in the form of multicore processors and many-core hardware accelerators [1–3]. Digital signal processing for wireless communications is one of the fields which has been largely benefited from these devices, for instance, to efficiently implement algorithms enabling multiple-input multiple-output (MIMO) communications [4, 5].

Millimeter wave (mmWave) communications are one of the key technology enablers toward achieving high data rates in 5G [6] and future 6G systems [7]. The use of these frequency bands considers beamforming techniques with highly directional beams that increase the gain of the communication link between transmitter (Tx) and receiver (Rx). This gain is achieved in practice by using massive MIMO systems, with a high number of antenna elements (in the order of several tens or hundreds). The use of MIMO systems with a large number of antennas increases further the complexity of many signal processing algorithms such as channel estimation, which could benefit from computationally efficient implementations.

The Xilinx Zynq Ultrascale+ MPSoC platform offers high levels of heterogeneity and parallelism since it is composed by four different processing elements: a dual-core Cortex-R5, a quad-core ARM Cortex-A53, a low-end Graphics Processing Unit (GPU) and a high-end FPGA. Therefore, this MPSoC offers multiple parallelism levels, although leveraging properly its computational resources becomes a challenging task that is being currently studied with basic operations such as matrix multiplication [8]. This work goes a step further and aims to implement and evaluate a MIMO mmWave channel estimation algorithm in a platform of the above kind, analyzing the device capability to handle the high computational demands of large communication systems. Specifically, we will tackle the novel implementation of the transformed spatial domain channel estimation method (TSDCE) for analog mmWave MIMO systems proposed in [9]. It is an iterative algorithm which performs, as core operations, singular value decompositions (SVD) and discrete Fourier transforms (DFTs) to estimate the channel from the available observations. Since both SVD and DFT can be computationally demanding as the number of antennas in the system grows, the aim of this work is to find a TSDCE implementation exhibiting a proper trade-off between performance and complexity.

This paper is structured as follows. Section 2 offers a brief overview of channel estimation theory for millimeter wave MIMO communications. In Sect. 3, we briefly describe the computational resources of the MPSoC Xilinx Zynq Ultrascale+. Next, in Sect. 4, we detail the implementation process and analyze the influence of the quad-core ARM Cortex-A53 on the performance. Finally, Sect. 5 provides some concluding remarks.

## 2 Channel estimation for millimeter wave MIMO communications

### 2.1 Channel model

Let us consider an mmWave MIMO channel characterized by  $L$  paths as in [10] [11], where the angle-of-arrival (AoD) and angle-of-departure (AoD) of path  $l$  are denoted by  $\psi_l$  and  $\phi_l$ , respectively. Each channel path is also affected by a complex gain coefficient  $\alpha_l$ . The full parametric channel model can be characterized as:

$$\mathbf{H}(\boldsymbol{\theta}) = \sqrt{n_t n_r} \sum_{l=1}^L \alpha_l \mathbf{a}_r(\psi_l) \mathbf{a}_t^H(\phi_l), \quad (1)$$

where  $\boldsymbol{\theta} \triangleq [|\alpha_1|, \angle\alpha_1, \phi_1, \psi_1, \dots, |\alpha_L|, \angle\alpha_L, \phi_L, \psi_L]^T$  is the parameter vector.

We consider a system where the Tx and Rx use both a uniform linear array of antennas for beamforming and combining with  $n_t$  and  $n_r$  antennas each, respectively. The antenna array responses at the Tx and Rx, assuming half-wavelength antenna separation, can be expressed as

$$\mathbf{a}_t(\phi_l) = \frac{1}{\sqrt{n_t}} [1, e^{-j\pi \cos \phi_l}, \dots, e^{-j\pi(n_t-1) \cos \phi_l}]^T, \quad (2)$$

$$\mathbf{a}_r(\psi_l) = \frac{1}{\sqrt{n_r}} [1, e^{-j\pi \cos \psi_l}, \dots, e^{-j\pi(n_r-1) \cos \psi_l}]^T. \quad (3)$$

In the channel model considered in this work, the  $\alpha_l$ -s are independent identically distributed (i.i.d.) random variables with distribution  $\alpha_l \sim \mathcal{CN}(0, \sigma_\alpha^2/L)$ , while the AoA ( $\psi_l$ ) and AoD ( $\phi_l$ ) are drawn from a uniform distribution  $\in [0, 2\pi]$ .

Hence, the problem of estimating the channel  $\mathbf{H}(\boldsymbol{\theta})$  can be addressed by finding out all the parameters in  $\boldsymbol{\theta}$ . Considering that measurements at mmWave have demonstrated that the channel at these frequencies is highly sparse [12], in practice the  $L$  paths are likely to be separated from each other, which simplifies the channel estimation task.

### 2.2 Pilot-based training phase

In order to estimate the parameters of the channel paths, a pilot-based training phase is carried out first. A pilot symbol  $s$ , known by Tx and Rx, is transmitted and received through a subset of  $P \leq N_{max}$  and  $Q \leq N_{max}$  spatial directions, respectively.  $N_{max}$  stands for the maximum number of angle quantization levels, which are limited due to assuming realistic phase shifters with limited angle resolution.

As in [9], the Tx and Rx have only one radio-frequency chain, and hence, the beamforming and combining operations are carried out in the analog domain. As in previous works [11], we particularize the beamforming/combining vectors to

match the channel response, i.e.,  $\mathbf{f} = \mathbf{a}_t(\bar{\phi}_p)$  for  $p = 0, 1, \dots, P - 1$  and  $\mathbf{w} = \mathbf{a}_r(\bar{\psi}_q)$  for  $q = 0, 1, \dots, Q - 1$ . For each  $\{q, p\}$  direction pair, the received signal is

$$y_{q,p} = \sqrt{\rho} \mathbf{w}_q^H \mathbf{H} \mathbf{f}_p s + \mathbf{w}_q^H \mathbf{n}, \tag{4}$$

where  $\rho \in \mathbb{R}^+$  is the transmit power. The noise term  $\mathbf{n} \sim \mathcal{CN}(0, \Sigma_n)$  is a complex additive white Gaussian noise  $1 \times n_r$  vector with covariance  $\Sigma_n = \sigma_n^2 \mathbf{I}_{n_r}$ , where  $\mathbf{I}_{n_r}$  denotes the  $n_r \times n_r$  identity matrix. Then, the system signal-to-noise ratio is given by  $\rho/\sigma_n^2$ . We assume, for simplicity, that the symbol  $s$  is set to 1 during training.

After the pilot transmission through the  $Q \times P$  directions, an observation matrix is obtained

$$\mathbf{Y} = \begin{bmatrix} y_{0,0} & y_{0,1} & \dots & y_{0,P-1} \\ y_{1,0} & y_{1,1} & \dots & y_{1,P-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{Q-1,0} & y_{Q-1,1} & \dots & y_{Q-1,P-1} \end{bmatrix} = \sqrt{\rho} \mathbf{G}(\boldsymbol{\theta}) + \mathbf{N}. \tag{5}$$

The noise matrix  $\mathbf{N} \in \mathbb{C}^{Q \times P}$  contains i.i.d.  $\sim \mathcal{CN}(0, \sigma_n^2)$  elements, and  $\mathbf{G} \in \mathbb{C}^{Q \times P}$  encodes the channel parameter vector  $\boldsymbol{\theta}$ . The effect of the different path components can be separated to write the observation matrix as a sum of path contributions  $\mathbf{G}^{(l)}(\boldsymbol{\theta}_l) \in \mathbb{C}^{Q \times P}$ , each one being dependent on a parameter vector  $\boldsymbol{\theta}_l = [|\alpha_l|, \angle \alpha_l, \phi_l, \psi_l]^T$  as

$$\mathbf{Y} = \sqrt{\rho} \sum_{l=1}^L \mathbf{G}^{(l)}(\boldsymbol{\theta}_l) + \mathbf{N}. \tag{6}$$

### 2.3 TSDCE method

TSDCE is an iterative channel estimation algorithm whose main steps are summarized in Fig. 1. Starting from the most powerful path component of the channel ( $l = 1$ ), the first step applies the two-dimensional inverse DFT (2D-IDFT) to the observation matrix  $\mathbf{Y}$  in (6), resulting in a new matrix  $\mathbf{D}$ . Next, a cropping step is performed to extract the upper-left submatrix containing the informative part of matrix  $\mathbf{D}$ . Then, an estimate of the contribution corresponding to the most powerful path component is obtained after performing the SVD of the cropped matrix, to achieve a rank-one approximation through the dominant singular-value.

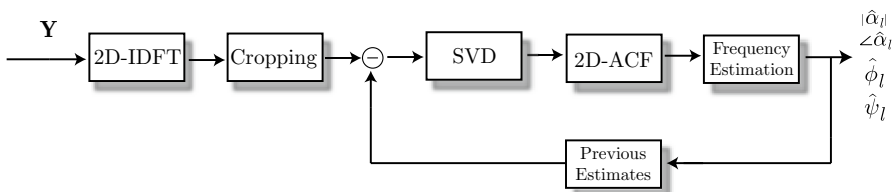


Fig. 1 Steps of the TSDCE method

The following step is based on applying the 2D sample autocorrelation function (ACF) to exploit its denoising properties. The phase angles of the elements of the resulting matrix contain the necessary information for the estimation of  $\psi_l$  and  $\phi_l$ .

In a final stage, spatial frequency estimation is performed. Following the discussion in [9], the frequency estimation can be obtained through a four-step process: unwrapping the phases, estimating their slopes by weighted least squares (WLS), designing the weights for the WLS optimization problem and estimating the path complex coefficient (both  $|\hat{\alpha}_1|$  and  $\angle \hat{\alpha}_1$ ).

### 3 Exploring the Xilinx Zynq Ultrascale+ MPSoC

Nowadays, high end FPGAs contain not only programmable logic but a variety of components that make possible an outstanding computational capacity. This work is based in the Xilinx Zynq UltraScale+ MPSoC family [13] which is subdivided into three subfamilies intended for different application fields. The CG subfamily is intended for optimized industrial applications, covering among others: IoT, motor control, sensors, etc. The EG subfamily is intended for aerospace and defense applications, covering 5G communications and cloud computing, and the EV subfamily is intended for high definition video applications. All elements of all subfamilies contain a dual-core ARM Cortex R5 [14] and a quad core ARM cortex A53 [15]. The EV and EG subfamilies have also a low end Mali GPU [16] and only subfamily EV adds an H.264/H.265 video codec. In all MPSoC devices from Zynq UltraScale family the system is subdivided into two main parts:

- PS (Processing System), which contains all the microprocessors. On the one hand, the quad-core ARM Cortex-A53 that implements the ARM v8-A 64-bit instruction set with frequency up to 1.5 GHz. We are accessing to these cores in two different ways : (1) by using the Xilinx Software Development Kit (SDK) [17] and (2) by using the operating system Petalinux [18] which allows us to customize, build and deploy embedded Linux solutions on Xilinx processing systems. On the other hand, the dual-core ARM Cortex-R5 that belongs to the family of 32-bit RISC ARM v7-R processors can be used in *lockstep* mode (micro-synchronized dual execution), and in *split* mode (each core can work in parallel and executing different tasks). Besides those microprocessors, the PS also contains a Mali 400 MP2 GPU where OpenGL ES 2.0 can be used to perform general purpose computations, as shown in [19–21].
- PL (Programmable Logic), which contains the re-configurable logic. This is the part that can fit different hardware designs according to the needs of the application and the available resources. For this work, we have selected the development board Ultra96 [22] from Avnet which contains a Xilinx Zynq UltraScale+ MPSoC ZU3EG A484 FPGA of the EG subfamily. The selected device contains: 154K System Logic Cells, 141K CLB Flip-Flops, 71K CLB LUTS, 360 DSP slices and block RAM that makes a total of 7.6 MBytes. In order to design and optimize the hardware that is located in the Programmable Logic, we use the tool SDSoC [23], available in the Xilinx Environment.

Inside the device, multiple interconnection options between PL and PS are available, making possible high-speed data transfer suitable for the most demanding applications. Figure 2 shows a general scheme of the Xilinx Zynq UltraScale+ EG subfamily including its main components.

Since the platforms of this kind will share its computational resources among different applications running at the same time, this work will focus on the main resource, the quad-core ARM Cortex-A53, which can be easily configured for exploiting parallelism among different operations regardless of the type of operations or instructions to be executed.

#### 4 Multi-core-based implementation

The main goal of this paper is to evaluate the computational time that requires the execution of the TSDCE method. To this end, we leverage versions of software libraries for numerical linear algebra, such as LAPACK (*Linear Algebra Package*) [24] and BLAS (*Basic Linear Algebra Subprograms*) [25] which are used for carrying out SVD decompositions and Matrix multiplications. Other computational expensive block of the algorithm is the 2D-ACF block, see Fig. 1, which is based on FFT and 2D-IDFT. These computations are performed in our code via the implementation available in the FFTW library for ARM architectures [26]. All of the previous libraries allow multi-thread support. The data type used, both in the FFT and in the SVD, as well as in the rest of the blocks that require complex numbers, is `double complex`. For the rest of the data, `double` or `int` have been used, depending on the specific case. Table 1 shows the routines that have been used for implementing the TSDCE method [9].

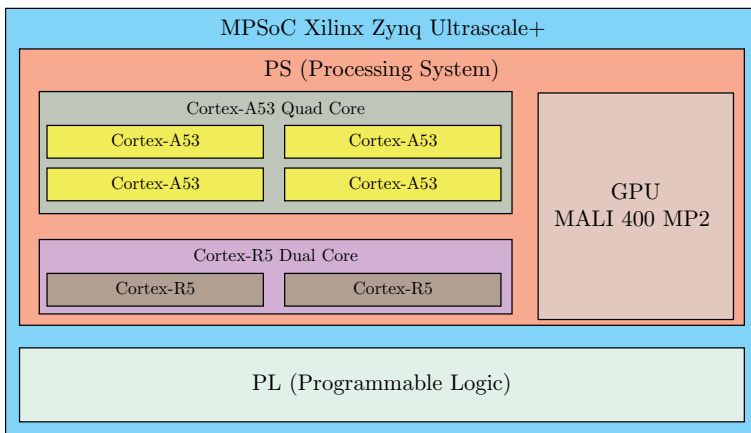


Fig. 2 Functional block diagram of the MPSoC Xilinx Zynq Ultrascale+ EG

**Table 1** Routines of LAPACK, BLAS and FFTW that have been used for implementing the TSDCE method

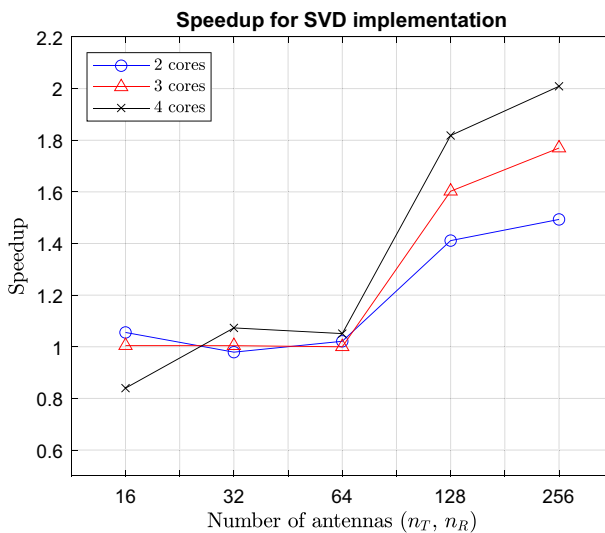
Library	Routine	Operations
FFTW	fftw_init_threads	Initialization required to use threads
	fftw_plan_with_nthreads	Set number of threads at FFTW
	fftw_plan_dft_2d	Create a fft plan
	fftw_execute	Execute a fft plan
	fftw_destroy_plan	Destroy a fft plan
	fftw_cleanup_threads	Free resources allocated internally
	openblas_set_num_threads	Set number of threads at OpenBLAS
	cblas_dgemm	Computes a matrix-matrix product with double general matrices
OpenBLAS	cblas_zdscal	Computes the product of a double complex vector by a double scalar
	cblas_zcopy	Copies a double complex vector to another double complex vector
	LAPACKE_zgesdd	Computes the singular value decomposition of a double complex general rectangular matrix using a divide and conquer method
	LAPACKE_zlaset	Initializes the off-diagonal elements and the diagonal elements of a double complex matrix to given values
	LAPACKE_zlange	Returns the value of the Frobenius norm of a double complex general rectangular matrix

## 5 Experiments

The algorithm has been tested for several values of the parameters involved. Although in practical systems the number of transmitting antennas can differ from the number of receiving antennas, for the sake of simplicity we considered a  $n_T = n_R$ , leading to square matrices for processing. In particular, the number of antennas was varied among 16, 32, 64, 128 and 256. Regarding the number of channel paths  $L$ , we considered a realistic mmWave where the number of paths is usually low and below  $L = 10$ . Regarding the number of quantized angles in transmission and reception,  $P$  and  $Q$ , respectively, we considered them equal to the number of antennas, as in [9].

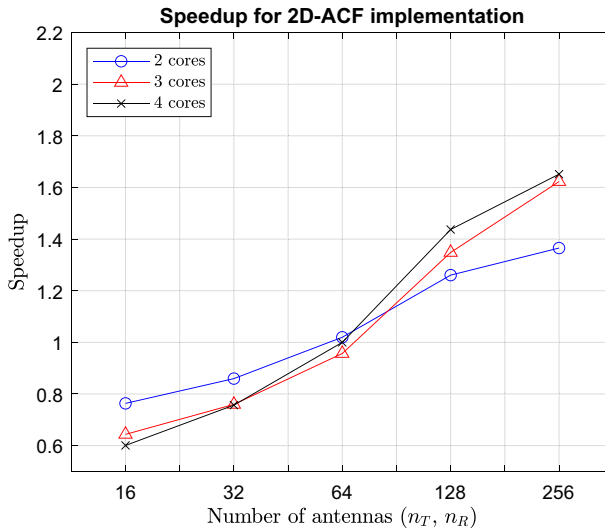
It can be observed that the TSDCE algorithm has two blocks that fundamentally influence the overall computational cost and that will be assessed independently: the SVD and the 2D-ACF (based on FFT). Starting from the SVD, each execution of the SVD depends mainly on the values of  $n_T$  and  $n_R$  and it does not depend on the number of paths, thus, the value of  $L$  does not affect the SVD runtime. Figure 3 shows the speedup of an iteration of the SVD block for various values of  $n_T = n_R$ . It can be seen that some improvement is obtained when increasing the number of cores for  $n_T$  values greater than 64. As expected, the speedup when using 2, 3 or 4 cores, defined as the ratio of execution time with many cores over the sequential execution time with a single core, is more noticeable as the values of  $n_T$  and  $n_R$  increase. The maximum achieved speedup is equal to 2, and it is obtained for the 256-antennas case.

Figure 4 shows the speedup of one iteration of the 2D-ACF block as a function of the number of transmitting/receiving antennas. The 2D-ACF curves, based on FFT processing, show a similar trend as the SVD implementation. The FFTW3 libraries offer a speedup improvement for values of  $n_T$  greater than 128, thus, an improvement



**Fig. 3** Speedup for the SVD implementation using 2, 3 or 4 cores for different numbers of antennas





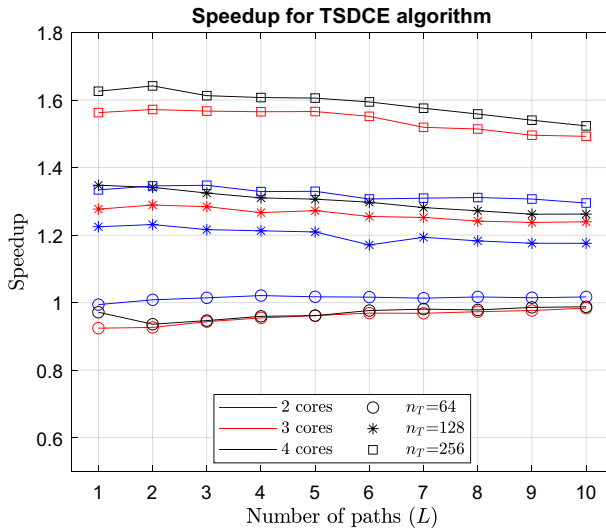
**Fig. 4** Speedup for the 2D-ACF implementation using 2, 3 or 4 cores for different numbers of antennas

is obtained by increasing the number of cores for those cases. However, it is important to note that, unlike for the SVD, the speedup using 4 cores with respect to using 3 cores is practically the same, so this block should be in practice parallelized with up to 3 cores. Furthermore, the maximum speedup achieved is 1.6, which is lower than for the SVD block.

The speedup for the implementation of the complete TSDCE algorithm using 2, 3 or 4 cores for different numbers of antennas and channel paths is shown in Fig. 5. It is worth noting that the number of SVD and 2D-ACF executions for each algorithm run depend both on  $L$ . More specifically, for each execution of the TSDCE algorithm, the SVD block is run  $L - 1$  times, whereas the 2D-ACF block is run  $L^2$  times. It can be seen in Fig. 5 that speedups higher than 1 are only achieved for numbers of antennas greater or equal than 64, being the maximum speedup around 1.6 for the 4-cores implementation with 256 antennas. Since the system is still far from achieving the maximum speedup of 4, this result shows that there is still room for improving the overall algorithm implementation, for instance, by optimizing the used data structures to reduce the number of writing/reading accesses to memory.

## 6 Conclusions

In this paper, we have explored the capabilities of the quad-core ARM Cortex-A53 that is present at the Xilinx Zynq Ultrascale+, a candidate embedded system to carry out tasks in 5G systems. Therefore, we have used this system to tackle a novel implementation of the transformed spatial domain channel estimation method (TSDCE) inside a MIMO communication system and, in this sense, to assess how the proposed implementation is able to exploit the quad-core parallelism.



**Fig. 5** Speedup for the implementation of the TSDCE algorithm using 2, 3 or 4 cores for different numbers of antennas and channel paths

We evaluated the execution times of the two most computationally demanding blocks (SVD and 2D-ACF) by using different numbers of threads and, also, the performance of the complete TSDCE algorithm. We observed that our implementation can manage a MIMO system composed of 256 antennas without exhausting the embedded system, which indicates that this kind of platforms are promising for carrying out real-time communication tasks. However, we also detected that, although there exists an improvement in performance by using a high number of cores, there is still a bottleneck in the data structures that limits the achievement of the maximum speedup, especially for systems with less than 64 antennas. To cope with this, as a future work, it would be of interest to explore heterogeneous implementations of the method, combining the multi-core resources with other resources such as GPU or FPGA.

**Acknowledgements** Thanks to Grant PID2020-113785RB-100 funded by MCIN/AEI/10.13039/501100011033 and the Ramón y Cajal Grant RYC-2017-22101. The work has been also supported by the Spanish Ministry of Science and Innovation under Grants RTI2018-097045-B-C21, PID2019-106455GB-C21 and PID2020-113656RB-C21, as well as the Regional Government of Madrid throughout the projects MIMACUHSAPCE-CM-UC3M (2022/00024/001) and PEJD-2019-PRE/TIC-16327.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission

directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.


## References

1. Eldredge M, Hughes TJ, Ferencz RM, Rifai SM, Raefsky A, Herndon B (1997) “High-performance parallel computing in industry,” *Parallel Computing*, vol. 23, no. 9, pp. 1217–1233, parallel computing methods in applied fluid mechanics. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167819197000495>
2. Belloch JA, Amor-Martin A, Garcia-Donoro D, Martínez-Zaldívar FJ, Garcia-Castillo LE (2019) On the use of many-core machines for the acceleration of a mesh truncation technique for fem. *J Supercomput* 75(1):1686–1696
3. Badia JM, Belloch JA, Cobos M, Igual F, Quintana-Orti ES (2019) Accelerating the srp-phat algorithm on multi- and many-core platforms using opencl. *J Supercomput* 75(1):1284–1297
4. Roger S, Ramiro C, Gonzalez A, Almenar V, Vidal AM (2012) Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector. *IEEE Trans Vehicular Technol* 61(8):3796–3800
5. Ramiro C, Roger S, Gonzalez A, Almenar V, Vidal AM (2013) Multicore implementation of a fixed-complexity tree-search detector for MIMO communications. *J Supercomput* 65(3):1010–1019
6. Roh W, Seol J, Park J, Lee B, Lee J, Kim Y, Cho J, Cheun K, Aryanfar F (2014) Millimeter-wave beamforming as an enabling technology for 5G cellular communications: theoretical feasibility and prototype results. *IEEE Communi Magaz* 52(2):106–113
7. Giordani M, Polese M, Mezzavilla M, Rangan S, Zorzi M (2020) Toward 6G networks: use cases and technologies. *IEEE Commun Magaz* 58(3):55–61
8. Belloch JA, Leon G, Badia JM, Lindoso A, San Millan E (2021) Evaluating the computational performance of the xilinx ultrascale+ eg heterogeneous mpso. *J Supercomput* 7(1):2124–2137
9. Roger S, Cobos M, Botella-Mascarell C, Fodor G (2021) Fast channel estimation in the transformed spatial domain for analog millimeter wave systems. *IEEE Trans Wireless Commun* 20(9):5926–5941
10. Alkhateeb A, El Ayach O, Leus G, Heath RW (2014) Channel estimation and hybrid precoding for millimeter wave cellular systems. *IEEE J Selected Topics Signal Process* 8(5):831–846
11. Zhang C, Guo D, Fan P (2016) “Tracking angles of departure and arrival in a mobile millimeter wave channel,” in *2016 IEEE International Conference on Communications (ICC)*, May pp. 1–6
12. Akdeniz MR, Liu Y, Samimi MK, Sun S, Rangan S, Rappaport TS, Erkip E (2014) Millimeter wave channel modeling and cellular capacity evaluation. *IEEE J Selected Areas Commun* 32(6):1164–1179
13. Xilinx Inc, “Zynq UltraScale+ MPSoC Data Sheet: Overview,” *DS891 (v1.7)*,(2018)
14. ARM, “Cortex-R5. Technical Reference Manual. Revision r1p2,”(2011)
15. ARM “ARM Cortex-A53 MPCore Processor. Technical Reference Manual. Revision r0p4,” (2016)
16. Olson T (2010)“Mali-400 MP: a scalable GPU for mobile devices,” in *Hot3D Session. Proc. International Conference on High Performance Graphics*
17. Xilinx Inc., “Embedded System Tools Reference Manual,” *UG1043 (v2019.1)*, (2019)
18. Xilinx Inc, “Petalinux Tools Documentation,” *UG1144 (v2018.3)*, (2018)
19. Trompouki MM, Kosmidis L (2016) “Towards general purpose computations on low-end mobile GPUs,” in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pp. 539–542
20. Trompouki MM, Kosmidis L (2017)“Optimisation opportunities and evaluation for GPGPU applications on low-end mobile GPUs,” in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pp. 950–953
21. Trompouki M.M, Kosmidis L (2018) “Brook auto: high-level certification-friendly programming for GPU-powered automotive systems,” in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*pp. 100:1–100:6
22. Avnet Inc, “Ultra96-V2 Hardware user guide. Version 1.0,” (2019)
23. Xilinx Inc, “SDSoC Environment User Guide,” *UG1027 (v2017.4)*, (2018)

24. Tomov S, Dongarra J, Baboulin M (2008) “Towards dense linear algebra for hybrid gpu accelerated manycore systems.” LAPACK Working Note, Tech. Rep. 210, Oct. [Online]. Available: <http://www.netlib.org/lapack/lawnspdf/lawn210.pdf>
25. Dongarra J, Croz JD, Hammarling S, Hanson RJ (1985) A proposal for an extended set of Fortran basic linear algebra subprograms. *ACM Signum Newsletter* 65:2–18
26. Frigo M, Johnson SG (2005) “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231 special issue on “Program Generation, Optimization, and Platform Adaptation”

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Authors and Affiliations

Pablo M. Aviles<sup>1</sup>  · Diego Lloria<sup>2</sup> · Jose A. Belloch<sup>1</sup> · Sandra Roger<sup>2</sup> · Almudena Lindoso<sup>1</sup> · Maximo Cobos<sup>2</sup>

Diego Lloria  
dielloal@uv.es

Jose A. Belloch  
jbelloc@ing.uc3m.es

Sandra Roger  
sandra.roger@uv.es

Almudena Lindoso  
alindoso@ing.uc3m.es

Maximo Cobos  
maximo.cobos@uv.es

<sup>1</sup> Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Leganés, Spain

<sup>2</sup> Computer Science Department, Universitat de València, Valencia, Spain