The Journal of Supercomputing (2023) 79:9371–9382
https://doi.org/10.1007/s11227-022-05018-w

Hybrid CPU–GPU implementation of the transformed spatial domain channel estimation algorithm for mmWave MIMO systems

Diego Lloria[1] · Pablo M. Aviles[2] · Jose A. Belloch[2] · Sandra Roger[1] · Carmen Botella-Mascarell[1] · Almudena Lindoso[2]

Accepted: 22 December 2022 / Published online: 13 January 2023

Abstract
Hybrid platforms combining multicore central processing units (CPU) with many-core hardware accelerators such as graphic processing units (GPU) can be smartly exploited to provide efficient parallel implementations of wireless communication algorithms for Fifth Generation (5G) and beyond systems. Massive multiple-input multiple-output (MIMO) systems are a key element of the 5G standard, involving several tens or hundreds of antenna elements for communication. Such a high number of antennas has a direct impact on the computational complexity of some MIMO signal processing algorithms. In this work, we focus on the channel estimation stage. In particular, we develop a parallel implementation of a recently proposed MIMO channel estimation algorithm. Its performance in terms of execution time is evaluated both in a multicore CPU and in a GPU. The results show that some computation blocks of the algorithm are more suitable for multicore implementation, whereas other parts are more efficiently implemented in the GPU, indicating that a hybrid CPU–GPU implementation would achieve the best performance in practical applications based on the tested platform.

Keywords Graphic processing units · Multicore CPU · MIMO communication systems · Channel estimation

✉ Jose A. Belloch
jbelloc@ing.uc3m.es

Extended author information available on the last page of the article

1 Introduction

High performance computers with parallel computing capabilities are used in multiple segments of the industry [1, 2]. Most of the current fastest supercomputers are built from nodes containing several multicore central processing units (CPU) and one or multiple graphic processing units (GPU).[1] Many computationally expensive problems can be tackled by appropriately leveraging both kinds of processing elements, or a combination of them through hybrid CPU–GPU implementations. This requires an application-based customized analysis to identify possible bottlenecks and link properly each computational resource with the appropriate task, so that overall computational performance is maximized. In particular, digital signal processing for wireless communications is one of the fields which has largely benefited from these devices, as it is the case for efficient multiple-input multiple-output (MIMO) algorithm's implementations [3, 4].

In current 5G and future 6G systems, the use of millimetre Wave (mmWave) frequencies is one of the key technology drivers towards achieving enhanced Mobile Broadband services (eMBB) [5, 6]. In mmWave, the use of beamforming techniques with highly directional beams is mandatory to increase the gain of the communication link between Transmitter (Tx) and Receiver (Rx). This is achieved in practice by using massive MIMO systems, with several tens or hundreds of antenna elements. One of the drawbacks of massive MIMO communications is that they often require more complex signal processing techniques, that could benefit from efficient implementations. In this work, we focus on the channel estimation stage of a massive MIMO system. Recently, a Transformed Spatial Domain Channel Estimation method (TSDCE) for analog mmWave MIMO systems has been proposed in Ref. [7], which has been shown to outperform other approaches based, for instance, on orthogonal matching pursuit (OMP) [8] or Discrete Fourier Transform (DFT) processing [9]. The core operations of the method are mainly based on the Fast Fourier Transform (FFT) and Singular Value Decomposition (SVD), which are suitable for parallel processing. Although reference [7] shows that TSDCE's complexity is below well-known channel estimation schemes, it could still benefit from a parallel implementation of its main blocks.

A first approach to efficiently implement the TSDCE method was carried out by the authors in Ref. [10], through exploring the parallelism of the quad-core ARM Cortex-A53 processor contained in the embedded Xilinx Zynq UltraScale+ EG Heterogeneous MPSoC system [11]. Results of such multicore implementation allowed to identify some constraints and bottlenecks in the implementation. For instance, the multicore implementations of the block based on FFT were in some cases less efficient than the sequential implementation, suggesting that a GPU-based implementation could be better suited for such block. Furthermore, cellular base stations could in practice accommodate non-embedded devices with larger general purpose GPUs, which could run channel estimation methods such as the TSDCE.

[1] http://top500.org.

In this work, we assess the potential benefits of a hybrid CPU–GPU implementation of the TSDCE method. We now consider a general purpose platform composed of an eight-core CPU and a high-end CUDA-based GPU which allow, not only to combine CPU and GPU code, but also to exploit the potential of the numerical algebra libraries in both versions, namely, CPU (BLAS [12] and LAPACK [13]) and GPU (cuBLAS and cuSOLVERS).[2] The execution times on CPU and GPU of the three main computing blocks of the algorithm (two of them based on FFT and the third based on SVD) are assessed and compared to the sequential implementation time (in a single CPU core). The reason to change the considered platform from the one used in Ref. [10] is that the latter is of embedded nature and presents a low-end Mali GPU [14] with limited computational performance regarding floating point precision [15, 16]. The results indicate that each main computing block benefits from a different implementation, supporting the need to complement the multicore-based analysis in Ref. [10] with a hybrid CPU–GPU overall implementation assessment.

The remainder of the paper is structured as follows. Section 2 presents the basis of the TSDCE channel estimation procedure. In Sect. 3, we describe the hybrid CPU–GPU implementation of the TSDCE. Next, in Sect. 4, we detail the experimental evaluation. Finally, Sect. 5 provides some concluding remarks.

2 TSDCE algorithm: fast channel estimation for mmWave channels

2.1 Channel model

Let us assume a single-user mmWave MIMO geometric channel [17, 18], where both the Tx and Rx are equipped with a uniform linear array (ULA) of n_t and n_r antenna elements, respectively. L denotes the number of scatterers, each one contributing with a single Tx-Rx propagation path. The complex channel coefficient for each path is defined by α_l, $l = 1, \ldots, L$, while ψ_l and ϕ_l stand for the Angle-of-Arrival (AoA) and Angle-of-Departure (AoD), respectively. Using the full parametric model, the channel is expressed as:

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

where $\boldsymbol{\theta} \triangleq \left[|\alpha_1|, \angle\alpha_1, \phi_1, \psi_1, \ldots, |\alpha_L|, \angle\alpha_L, \phi_L, \psi_L\right]^T$ is the parameter vector. Note that $|\alpha_l|$ and $\angle\alpha_l$ stand for the magnitude and phase of each channel coefficient. α_l-s are independent identically distributed (i.i.d.) random variables with distribution $\alpha_l \sim \mathcal{CN}(0, \sigma_\alpha^2)$. AoAs ($\psi_l$) and AoDs ($\phi_l$) are drawn from a uniform distribution $\in [0, 2\pi]$.

The antenna array responses at the Tx and Rx, under a half-wavelength antenna separation assumption, can be expressed as, respectively:

[2] https://docs.nvidia.com/cuda/index.html.

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

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

The problem of estimating the channel $\mathbf{H}(\boldsymbol{\theta})$ becomes a problem of estimating the parameters in $\boldsymbol{\theta}$. MmWave channels are highly sparse, as demonstrated by measurements [19]. Hence, the L paths are likely to be separated from each other, which simplifies the channel estimation task.

2.2 Pilot-based training phase

A previous step to the estimation of the channel using the TSDCE or similar methods is the open-loop pilot-based training phase, where a set of possible Tx and Rx directional beams is tested. More specifically, the beam search space is given by a codebook comprising P and Q codewords/directions at the Tx and Rx side, respectively. An observation matrix is formed after transmitting the pilot symbol through the $Q \times P$ direction combinations.

The process is as follows: 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. $N_{\max}$ denotes the maximum number of angle quantization levels. Note that this parameter is limited due to assuming realistic phase shifters with limited angle resolution. In the analog beamforming case, the Tx and Rx have only one radio-frequency chain, so the beamforming and combining operations are carried out in the analog domain [7].

Beamforming/combining vectors are calculated to match the channel response [20], so $\mathbf{f} = \mathbf{a}_t\left(\bar{\phi}_p\right)$ for $p = 0, 1, \ldots, P-1$, and $\mathbf{w} = \mathbf{a}_r\left(\bar{\psi}_q\right)$ for $q = 0, 1, \ldots, Q-1$. For each $\{q, p\}$ direction pair, the received signal is

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

where $\rho \in \mathbb{R}^+$ is the transmit power. The noise term $\mathbf{n} \sim \mathcal{CN}\left(0, \boldsymbol{\Sigma}_{\mathbf{n}}\right)$ is a complex additive white Gaussian noise $1 \times n_r$ vector with covariance $\boldsymbol{\Sigma}_{\mathbf{n}} = \sigma_n^2 \mathbf{I}_{n_r}$, where $\mathbf{I}_{n_r}$ stands for the $n_r \times n_r$ identity matrix. The symbol s is set to 1 during training for the sake of simplicity. Hence, the system signal-to-noise ratio is given by ρ/σ_n^2.

The observation matrix is obtained after transmitting the pilot symbols through the $Q \times P$ directions

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

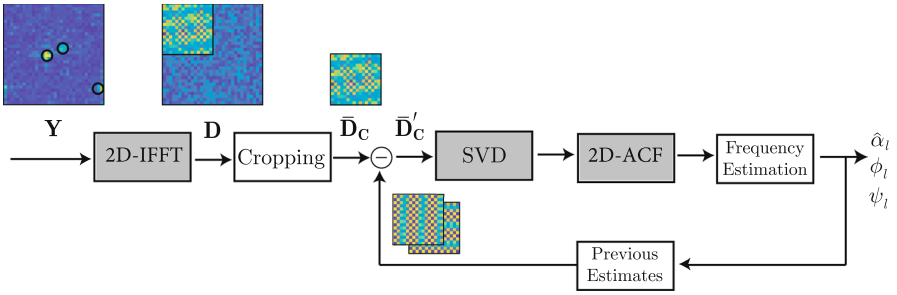

Fig. 1 Steps of the TSDCE method. Most computationally demanding blocks are highlighted in grey

The noise matrix $\mathbf{N} \in \mathbb{C}^{Q \times P}$ contains i.i.d. $\sim \mathcal{CN}\left(0, \sigma_n^2\right)$ elements, and $\mathbf{G} \in \mathbb{C}^{Q \times P}$ encodes the channel parameter vector $\boldsymbol{\theta}$.

Elements in the observation matrix can be rearranged to separate the effect of the different path components

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

In the above expression, the observation matrix is written 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 $\boldsymbol{\theta}_l = \left[|\alpha_l|, \angle\alpha_l, \phi_l, \psi_l\right]^T$. This formulation paves the way for the implementation of the TSDCE method.

2.3 TSDCE method

TSDCE is an iterative algorithm which works over the observation matrix $\mathbf{Y}$ to estimate the parameters of the channel. Fig. 1 shows the main steps or building blocks of the TSDCE. It also includes the representation of the input and output signals in the most relevant steps through an example, where the observation matrix shows the presence of three significant paths in the channel (i.e. $L = 3$).

The process starts with the estimation of the parameters of the largest gain path component ($l = 1$). First, a two-dimensional inverse FFT (2D-IFFT) is applied to the observation matrix, resulting in matrix $\mathbf{D}$. Then, a cropping procedure extracts the upper-left submatrix which contains the relevant information, leading to a new matrix $\bar{\mathbf{D}}_{\mathbf{C}}$. The rest of the paths are estimated using the same procedure, although the cropped matrix is updated by successive interference cancellation to remove the contribution of the estimated paths, leading to a new matrix $\bar{\mathbf{D}}'_{\mathbf{C}}$. By performing a SVD of $\bar{\mathbf{D}}'_{\mathbf{C}}$, an estimate of the contribution of the current path component is obtained, achieving a rank-one approximation by means of the dominant singular value. In the next step, a 2D sample Autocorrelation Function (2D-ACF) is applied due to its denoising properties. The resulting matrix (the phase angles of its elements) contains the information for estimating ψ_l and ϕ_l. Finally, as discussed in

Ref. [7], spatial frequency estimation is carried out in a four-stage process, which allows to estimate the path complex coefficient (both $|\hat{\alpha}_l|$ and $\angle\hat{\alpha}_l$).

Note that the most computationally demanding blocks of the TSDCE algorithm are highlighted in grey in Fig. 1.

3 Hybrid CPU–GPU TSDCE implementation

The implementation of the TSDCE method used a desktop computer with an NVIDIA GeForce RTX 3060 GPU and an Intel Core i7-9700F with 8 core processors running at 3.0 GHz and 4GB DDR4-2666 RAM, although only 4 out of the 8 cores have been used as in Ref. [10]. Each CPU core is equipped with a 256 KiB L1 data cache and a 256 KiB L1 instruction cache. The cores share a 2MB L2 unified cache. The maximum bandwidth is 41.6 GB/s. The GPU was composed of one of the newest NVIDIA Ampere architectures and gathers a total of 3584 CUDA cores spread in 28 GPU multiprocessors.[3]

We tackled the TSDCE implementation by focusing on the three most computationally demanding individual blocks: IFFT, SVD and 2D-ACF (see Fig. 1), with the aim of exploring an efficient hybrid solution. The GPU implementation used the cuBLAS and cuSOLVER libraries for linear algebra, and the FFT library cuFFT.[4] The 2D-ACF block has been implemented mainly through an IFFT, a matrix conjugate and a matrix product. The implementation in the multicore CPU has been done through the OpenMP programming model with the analogous linear algebra libraries, namely, BLAS [12] and LAPACK[13], and the FFTW library [21]. The specific routines used for the multicore CPU implementation where already reported in Ref. [10].

Table 1 shows the specific routines used for the GPU implementation and a description of their usage. We detail next which functions have been used for the implementation of each of the blocks under study:

- **IFFT block**: The implementation of the IFFT block requires only the usage of the `cuFFTPlanMany` and `cufftExecZ2Z` routines.
- **SVD block**: First, the `cublasZgeam` routine is called to obtain the transpose of matrix $\bar{\mathbf{D}}'_{\mathbf{C}}$. Then, the SVD of $\bar{\mathbf{D}}'_{\mathbf{C}}$ is calculated by using the `cusolverDnZgesvdbufferSize` and `cuSolverDnZgesvd` routines. Finally, to calculate the rank-one approximation after the SVD, the `cublasZgemm` routine is called.
- **2D-ACF block**: to implement this block, first an FFT is obtain through the use of the `cufftPlan2d` and `cufftExecZ2Z` routines. Next, as an intermediate step, an element-wise matrix product is performed. Finally, the inverse FFT is applied over the result of the previous step.

[3] https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf.
[4] https://docs.nvidia.com/cuda/index.html.

Table 1 Routines of cuFFT, cuBLAS and cuSOLVER used to implement the TSDCE method. $Op(\mathbf{X})$ stands for matrix identity, transpose or conjugate of $\mathbf{X}$

Library	Routine	Operations
cuFFT	`cufftPlan2d`	Initialization required to use cuFFT
	`cufftExecZ2Z`	Computes the FFT of a double complex rectangular matrix
cuBLAS	`cublasCreate`	Initialization required to use cuBLAS
	`cublasZdscal`	Scales a double complex vector by a scalar
	`cublasSetMatrix`	Copies a submatrix from a matrix $\mathbf{A}$ in host memory to a matrix $\mathbf{B}$ in device memory
	`cublasGetMatrix`	Copies a submatrix from a matrix $\mathbf{A}$ in device memory to a matrix $\mathbf{B}$ in host memory
	`cublasZgeam`	Computes $\alpha Op(\mathbf{A}) + \beta Op(\mathbf{B})$
	`cublasZcopy`	Copies a double complex vector to another double complex vector
	`cublasZgemm`	Computes $\alpha Op(\mathbf{A})Op(\mathbf{B}) + \beta\mathbf{C}$
cuSOLVER	`cusolverDnCreate`	Initialization required to use cuSolver
	`cusolverDnZgesvd_bufferSize`	Calculate the sizes needed for preallocated buffer for the computation of the SVD
	`cusolverDnZgesvd`	Computes the SVD of a double complex rectangular matrix using a QR method

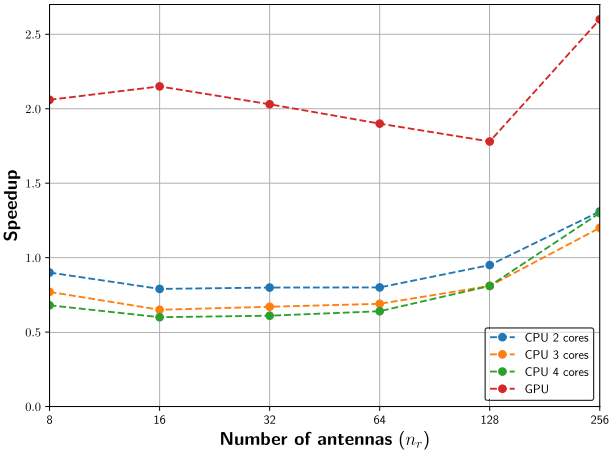

Fig. 2 Speedup for the IFFT block by using 2, 3, 4 CPU cores and GPU

Table 1 also describes the `cublasCreate`, `cublasZdscal`, `cublasSetMatrix`, `cublasGetMatrix` and `cublasZcopy` routines of cuBLAS library which perform auxiliary steps such as initialization, copy, and data transfer, which have been used also in the SVD and 2D-ACF blocks as needed.

4 Experiments

The experimental evaluation assumes equal values for the parameters P, Q, n_r and n_t for the sake of simplicity (note that in more realistic systems these parameters may differ), which implies that the size of the observation matrix $\mathbf{Y}$ is directly $n_r \times n_t$. The values that have been evaluated are 16, 32, 64, 128, and 256. Although the TSDCE algorithm also depends on the number of path components L, the evaluations in this work are focused on the execution time to estimate one of the paths. Thus, the results are independent of L. In order to better assess the performance enhancement of the GPU and multicore CPU implementations, the *Speedup* measure has been used, which is defined as the quotient between the execution time with a single CPU core over the execution time with the GPU or with several CPU cores (2, 3 or 4). For each block under analysis and each processing element, 30 time measurements have been recorded and sorted in ascending order. The final Speedup value has been calculated as the average of the 5 central values to avoid bias due to outliers.

The execution time of the IFFT block is assessed in Fig. 2, representing the Speedup for the IFFT block using 2, 3 or 4 CPU cores and when using the GPU as the number of antennas increases. Note that these results depend exclusively on P and Q. It can be observed that the GPU achieves the largest Speedup values in all cases, which is around 2. The maximum Speedup is 2.75 and appears for

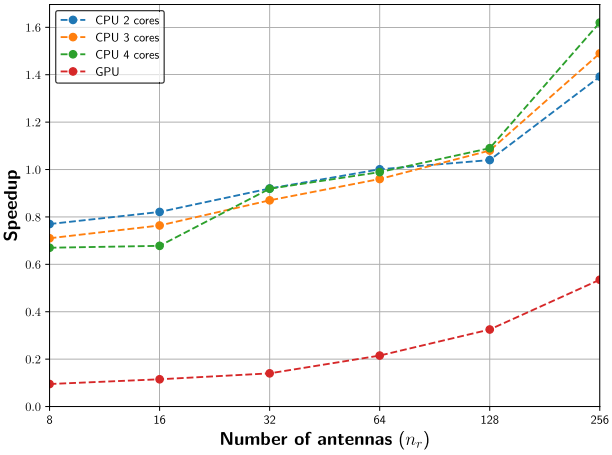

Fig. 3 Speedup for the SVD block by using 2, 3, 4 CPU cores and GPU

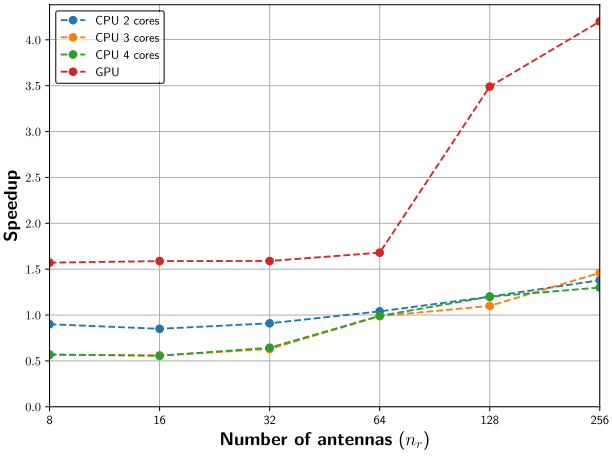

Fig. 4 Speedup for the 2D-ACF block by using 2, 3, 4 CPU cores and GPU

$n_r = 256$, indicating that in massive MIMO systems, the IFFT block largely benefits from a GPU implementation. In fact, the Speedup values for the multicore implementations are in several cases under 1, an effect that was already observed in Ref. [10] for the implementation of the 2D-ACF block, also based on FFT.

The Speedup results for the SVD block are shown in Fig. 3. Note that the SVD execution time depends mainly on n_r and n_t. As it can be seen, in this case the Speedup is larger in the multicore CPU options than in the GPU, regardless of the number of cores and with a maximum of 1.6, i.e. it is lower than the one of the IFFT block. It is worth noting that the GPU Speedup is below 1, which implies

that the execution time of the SVD on GPU is inefficient and worse than the single-core CPU implementation.

Finally, Fig. 4 shows the Speedup for the 2D-ACF block as a function of the number of antennas. The Speedup values are in all cases superior with the GPU, showing a remarkable increase with respect to the CPU Speedup values for $n_r \geq 64$. For instance, when $n_r = 256$, the GPU Speedup is larger than 4 in contrast to only around 1.5 on the CPU. Since the 2D-ACF is based on FFT processing, it is also interesting to compare its Speedup results with those for the IFFT block in Fig. 2. For $n_r \leq 64$, the GPU Speedups have similar trends for both blocks, with larger Speedup values for the IFFT case. The reason for this Speedup difference may come from the data transfers between GPU and CPU necessary for the additional operations in the 2D-ACF. However, when $n_r \geq 128$, the GPU Speedup for the 2D-ACF block is much higher than for the IFFT. As the number of antennas grows, the time spent for data transfers in the 2D-ACF is negligible within the overall execution time, which becomes dominated by the algebraic operations with large matrices.

5 Conclusion

In this paper, a hybrid CPU–GPU implementation of a transformed spatial domain MIMO channel estimation algorithm has been performed, with the aim of exploring the optimal link between computational resource and algorithm's computing block. The results have been obtained in terms of execution time for a single path channel estimation and are represented through the speedup over a single-core implementation. Under the assumption of equal antenna and beam search space dimensions, it was observed that the multicore resources are less efficient for FFT-related computational blocks, showing in several cases speedup values under 1, as already happened in a previous work focused only on multicore implementation. Regarding SVD-related computations, in this case the multicore options are more advantageous than the GPU processing, confirming that each of the three main algorithm's computing blocks could benefit from a different implementation type. In future works, the general multi-path case will be considered to explore the impact of the iterative nature of the algorithm on the optimal computational resource management.

Author Contributions All authors contributed equally to this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Thanks to Grant PID2020-113785RB-100 funded by MCIN/AEI/10.13039/ 501100011033 and the Ramón y Cajal Grant RYC-2017-22101. Thanks to the Spanish Ministry of Science and Innovation under Grants PID2019-106455GB-C21 and PID2020-113656RB-C21, as well as the Regional Government of Madrid throughout the projects MIMACUHSPACE-CM-UC3M and PEJD-2019-PRE/TIC-16327.

Availability of data and materials No additional data or materials available.

Declarations

Conflict of interest The authors declare that they have no known competing financial interest or personal relation-

ships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable.

References

1. Czarnul P, Proficz J, Drypczewski K (2020) Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems. Sci Program 2020
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. 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 Veh Technol 61(8):3796–3800
4. 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
5. 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 Commun Mag 52(2):106–113
6. Giordani M, Polese M, Mezzavilla M, Rangan S, Zorzi M (2020) Toward 6G networks: use cases and technologies. IEEE Commun Mag 58(3):55–61
7. 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 Wirel Commun 20(9):5926–5941
8. Lee J, Gil G-T, Lee YH (2014) Exploiting spatial sparsity for estimating channels of hybrid MIMO systems in millimeter wave communications. in IEEE Global Communications Conference
9. Montagner S, Benvenuto N, Baracca P (2015) Channel estimation using a 2D DFT for millimeter-wave systems. in IEEE 81st Vehicular Technology Conference (VTC Spring)
10. Aviles PM, Lloria D, Belloch JA, Roger S, Lindoso A, Cobos M (2022) Performance analysis of a millimeter wave MIMO channel estimation method in an embedded multi-core processor. J Supercomput 78:14756–14767
11. Xilinx Inc, Zynq UltraScale+ MPSoC data sheet: overview. DS891 (v1.7) 2018
12. Dongarra J, Croz JD, Hammarling S, Hanson RJ (1985) A proposal for an extended set of Fortran basic linear algebra subprograms. in ACM Signum Newsletter, pp 2–18
13. 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
14. Olson T (2010) Mali-400 MP: a scalable GPU for mobile devices. in Hot3D Session. Proceedings International Conference on High Performance Graphics
15. Belloch JA, Leon G, Badia JM, Lindoso A, San Millan E (2021) Evaluating the computational performance of the Xilinx Ultrascale+ EG heterogeneous MPSoC. J Supercomput 77(1):2124–2137 (**January**)
16. 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
17. Alkhateeb A, El Ayach O, Leus G, Heath RW (2014) Channel estimation and hybrid precoding for millimeter wave cellular systems. IEEE J Select Top Signal Process 8(5):831–846
18. Ge X, Shen W, Xing C, Zhao L, An J (2022) Training beam design for channel estimation in hybrid mmWave MIMO systems. IEEE Trans Wirel Commun 21(9):7121–7134

19. 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 Sel Areas Commun 32(6):1164–1179
20. 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
21. Frigo M, Johnson SG (2005) The design and implementation of FFTW3. Proc IEEE 93(2):216–231

Authors and Affiliations

Diego Lloria[1] · Pablo M. Aviles[2] · Jose A. Belloch[2] · Sandra Roger[1] · Carmen Botella-Mascarell[1] · Almudena Lindoso[2]

Diego Lloria
diego.lloria@uv.es

Pablo M. Aviles
paviles@ing.uc3m.es

Sandra Roger
sandra.roger@uv.es

Carmen Botella-Mascarell
carmen.botella@uv.es

Almudena Lindoso
alindoso@ing.uc3m.es

[1] Computer Science Department, Universitat de València, Valencia, Spain

[2] Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Leganés, Spain