IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 May 2022, accepted 2 June 2022, date of publication 29 June 2022, date of current version 11 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3187040

GPU-Accelerated Partially Linear Multiuser Detection for 5G and Beyond URLLC Systems

MATTHIAS MEHLHOSE[1], GUILLERMO MARCUS[2], DANIEL SCHÄUFELE[1], DANIYAL AMIR AWAN[1], NIKOLAUS BINDER[2], MARTIN KASPARICK[1], RENATO L. G. CAVALCANTE[1], (Member, IEEE), SŁAWOMIR STAŃCZAK[1,3], AND ALEXANDER KELLER[2]

[1]Department of Wireless Communications and Networks, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), 10587 Berlin, Germany
[2]NVIDIA, 10623 Berlin, Germany
[3]Department of Telecommunication Systems, Faculty IV-Electrical Engineering and Computer Science, Technical University of Berlin, 10587 Berlin, Germany

Corresponding author: Matthias Mehlhose (matthias.mehlhose@hhi.fraunhofer.de)

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF, Germany) in the project Open Testbed Berlin - 5G and Beyond (OTB-5G+) under Grant 16KIS0980 and supported as part of the 6G Research and Innovation Cluster 6G-RIC under Grant 16KISK020K.

ABSTRACT We have implemented a recently proposed partially linear multiuser detection algorithm in reproducing kernel Hilbert spaces (RKHSs) on a GPU-accelerated platform. Our proof of concept combines the robustness of linear detection and non-linear detection for the non-orthogonal multiple access (NOMA) based massive connectivity scenario. Mastering the computation of the vast number of inner products (which involve kernel evaluations) is a challenge in ultra-low latency (ULL) applications due to the sub-millisecond latency requirement. To address the issue, we propose a massively parallel implementation of the detection of user data in a received orthogonal frequency-division multiplexing (OFDM) radio frame. The result is a GPU-accelerated real-time OFDM receiver that enables detection latency of less than one millisecond that complies with the requirements of 5th generation (5G) and beyond ultra-reliable and low latency communications (URLLC) systems. Moreover, the parallelization and acceleration techniques explored and demonstrated in this study can be extended to many signal processing algorithms in Hilbert spaces, such as projection onto convex sets (POCS) and adaptive projected subgradient method (APSM) based algorithms. Results and comparisons with the state-of-the-art confirm the effectiveness of our approach.

INDEX TERMS Machine learning, wireless communication, multiuser detection, NOMA, MIMO, ultra-reliable low latency communication, massively parallel architectures, GPU, CUDA.

I. INTRODUCTION

Recently, a large body of research has been devoted to nonorthogonal multiple access (NOMA) [1]–[4] because the requirements of massive connectivity beyond 5G mobile networks necessitate the efficient use of time-frequency resources. In contrast to traditional orthogonal multiple access (OMA), such as orthogonal frequency-division multiple access (OFDMA), NOMA allocates the same time-frequency resource to multiple users in the same cell, which may result in strong multiuser interference. To deal with the strong interference, partially linear multiuser detection in reproducing kernel Hilbert spaces (RKHSs) has been proposed (e. g., see [5]–[10]). The reason for considering partially linear receivers is that, while nonlinear detectors can outperform linear detectors significantly in scenarios with strong multiuser interference, they may be highly sensitive even to small changes in a wireless environment (e. g., those caused by intermittent interference and multipath scattering in massive machine-type communication (mMTC) scenarios). In fact, theoretical studies [11] have shown that linear detectors can achieve a comparable spectral

The associate editor coordinating the review of this manuscript and approving it for publication was Turgay Celik.

efficiency to nonlinear methods in massive multiple-input multiple-output (MIMO) systems. Therefore, in massive MIMO NOMA systems, purely nonlinear detectors may be inefficient. Based on these facts, studies in [8]–[10] have designed a hybrid multiuser detector in RKHSs that combines the strengths of nonlinear and linear filters. The proposed supervised projection-based learning algorithm, which is a special case of the adaptive projected subgradient method (APSM) [12], has some very desirable features. For example, the algorithm learns to detect modulation symbols of a user directly without any intermediate parameter estimation which is often prone to errors. Furthermore, if the channel information is available, the algorithm can be initialized by a conventional linear filter, and the performance can be further improved in scenarios for which linear filters are insufficient when dealing with strong interference.

Basically, the APSM-based algorithm tracks the intersection of time-varying closed-convex sets to which the desired solution (often a vector or a function) belongs. In this regard, it resembles a classic projection-based set-membership estimation[1] (e. g., see [13], [14]). In general, such algorithms consist of a sequence of projections on the constructed closed convex subsets of an appropriately defined Hilbert space. The closed convex subsets are often chosen such that the projections (which require computing inner products) admit a closed-form solution; for example, projections onto hyperplanes, closed balls, closed halfspaces, etc. It is known that the computational effort of projection-based algorithms is dominated by the cost of computing projections. In ultra-low latency (ULL) applications, however, straightforward implementations of a large number of seemingly simple inner products in an orthogonal frequency-division multiplexing (OFDM) radio frame may consume too much time to satisfy the temporal and latency constraints. Luckily, many algorithms that involve projections and inner products in Hilbert spaces lend themselves very well to massive parallelism. In order to realize our goal of real-time GPU-accelerated multiuser detection, we exploit the intrinsic parallelism of APSM and carefully schedule operations across the memory hierarchy of the graphics processing unit (GPU).

Before we move on to our contributions, we mention the fact that in the future many Radio Access Network (RAN) functions will be virtualized on general-purpose commercial off-the-shelf (COTS) hardware. Recent industry trials have shown that virtualization of RAN functions in the Distributed Units (DUs), consisting of computationally intensive baseband operations, may not be cost-effective (compared to traditional Application-Specific Integrated Circuit (ASIC) hardware) and viable without using modern acceleration platforms. Therefore, baseband algorithms that are suited to such platforms are the subject of current interest in the wireless industry.

[1]It is important to mention that the classical set-membership estimation considers a finite number of sets whereas in our APSM-based algorithm an infinite number of sets is considered.

A. CONTRIBUTIONS

Our main contribution is a massively parallel GPU-accelerated NOMA system that can fulfill the real-time constraints of ULL in an OFDM-based system. Below, we summarize our contributions.

- Our focus in this study is to accelerate the detection of an OFDM radio frame which may involve computations of a large number of inner products. The acceleration is achieved by state-of-art parallelization and memory management techniques.
- The performance of the developed GPU-accelerated system is then demonstrated in a setup with 6 transmit and 16 receive antennas. Compared to the existing proof of concept (PoC) MATLAB implementation [8] we demonstrate that using massively parallel processor structures on a GPU reduces the processing time to milliseconds.
- GPU-accelerated signal processing is an important part of 5G and beyond communication systems [15]. Such platforms are particularly suited to signal processing algorithms consisting of a large number of projections requiring inner products that can be executed in parallel. This means that the techniques that we have developed in this work can also be used in other inner product (and projection) based algorithms, such as projection onto convex sets (POCS) signal processing in Hilbert spaces.

B. STRUCTURE

The paper is organized as follows. In Section II, we review the algorithm for multiuser detection proposed in [9]. In particular, some practical aspects of the algorithm are explained along with its potential for acceleration by parallel signal processing. Section III discusses the implementation of the algorithm and the techniques to optimize parallel processing and memory access on the targeted graphics processing unit (GPU) platform. Section IV describes the hardware equipment and the experimental setup deployed in the proof-of-concept. Finally, the results are presented in Section V.

II. MACHINE LEARNING BASED MULTIUSER DETECTION

In this section, we describe the APSM-based algorithm for multiuser detection that has been studied in [8]–[10]. Here, we focus on the implementation and computational aspects of the algorithm, and we avoid repeating unnecessary details.

A. PRELIMINARIES

In the following, the sets of natural numbers, non-negative integers, real numbers, and complex numbers are denoted by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}$, and $\mathbb{C}$, respectively. We define the range $\overline{N_1, N_2} := \{N_1, N_1 + 1, \ldots, N_2\}$, where $N_1, N_2 \in \mathbb{N}_0$ and $N_1 \leq N_2$.

This study deals with adaptive filtering in special (real) Hilbert spaces known as RKHSs. Briefly, given an arbitrary (linear) subspace $\mathcal{U} \subseteq \mathbb{R}^l$, an RKHS $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ is a Hilbert space with the inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$, uniquely associated with a positive definite function known as the reproducing kernel $\kappa : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ of $\mathcal{H}$. In this study, κ is

either the linear kernel denoted by κ_{L} and defined as $(\forall \mathbf{u}, \mathbf{v} \in \mathcal{U})\ \kappa_{\mathrm{L}}(\mathbf{u}, \mathbf{v}) := \mathbf{u}^T \mathbf{v}$ or the Gaussian kernel denoted by κ_{G} and defined as $(\forall \mathbf{u}, \mathbf{v} \in \mathcal{U})\ \kappa_{\mathrm{G}}(\mathbf{u}, \mathbf{v}) := \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2_{\mathbb{R}^l}}{2\sigma^2}\right)$, where $\|\cdot\|_{\mathbb{R}^l}$ is the Euclidean norm in $\mathbb{R}^l$ and the variance $\sigma^2 > 0$ is a design parameter for the kernel width. The RKHSs associated with κ_{L} and κ_{G} are denoted by $(\mathcal{H}_{\mathrm{L}}, \langle\cdot,\cdot\rangle_{\mathcal{H}_{\mathrm{L}}})$ and $(\mathcal{H}_{\mathrm{G}}, \langle\cdot,\cdot\rangle_{\mathcal{H}_{\mathrm{G}}})$, respectively.

Now consider either of the above-mentioned RKHSs and the well-known property that $(\forall \mathbf{u} \in \mathcal{U})\ \kappa(\mathbf{u}, \cdot) \in \mathcal{H}$. In this study, we deal with functions of the type $f := \sum_{n=1}^{N} a_n \kappa(\mathbf{u}_n, \cdot) \in \mathcal{H}$, where $a_n \in \mathbb{R}$ and $\mathbf{u}_n \in \mathcal{U}$.

For two functions $f := \sum_{n=1}^{N} a_n \kappa(\mathbf{u}_n, \cdot) \in \mathcal{H}$ and $g := \sum_{m=1}^{M} b_m \kappa(\mathbf{v}_m, \cdot) \in \mathcal{H}$, where $a_m, b_m \in \mathbb{R}$ and $\mathbf{u}_n, \mathbf{v}_m \in \mathcal{U}$, we define the inner product

$$\langle f, g\rangle_{\mathcal{H}} := \sum_{n=1}^{N}\sum_{m=1}^{M} a_n b_m \kappa(\mathbf{u}_n, \mathbf{v}_m), \tag{1}$$

inducing the norm

$$\|f\|^2_{\mathcal{H}} = \langle f, f\rangle_{\mathcal{H}}. \tag{2}$$

B. PARTIALLY LINEAR FILTER DESIGN

The studies in [8]–[10] combine the robustness of linear beamforming filters with a higher spatial resolution of nonlinear beamforming filters by designing a partially linear filter in RKHSs. In more detail, the RKHS $\mathcal{H}_{\mathrm{G}}$ associated with the Gaussian kernel κ_{G} is combined (i. e., summed) with the RKHS $\mathcal{H}_{\mathrm{L}}$ associated with the linear kernel κ_{L} to obtain a sum RKHS of partially linear filters. To be more precise, a partially linear filter f is defined as an element of the real RKHS $\mathcal{H} := \mathcal{H}_{\mathrm{L}} + \mathcal{H}_{\mathrm{G}} := \{w_{\mathrm{L}} f_{\mathrm{L}} + w_{\mathrm{G}} f_{\mathrm{G}} : f_{\mathrm{L}} \in \mathcal{H}_{\mathrm{L}}, f_{\mathrm{G}} \in \mathcal{H}_{\mathrm{G}}\}$, where $w_{\mathrm{L}}, w_{\mathrm{G}} \geq 0$ are fixed weights for the linear and the Gaussian part, respectively. Fact 1 shows how the kernel and inner products are computed in $\mathcal{H}$:

Fact 1 (Reproducing kernel of the weighted sum space [16]): Assume that the input space $\mathcal{U} \subseteq \mathbb{R}^l$ has a nonempty interior. Then, given any $w_{\mathrm{L}}, w_{\mathrm{G}} > 0$ and $\mathbf{u}, \mathbf{v} \in \mathcal{U}$, $\kappa(\mathbf{u}, \mathbf{v}) := w_{\mathrm{L}}\kappa_{\mathrm{L}}(\mathbf{u}, \mathbf{v}) + w_{\mathrm{G}}\kappa_{\mathrm{G}}(\mathbf{u}, \mathbf{v})$ is the reproducing kernel of the sum space $\mathcal{H}$ equipped with the inner product

$$\langle f, g\rangle_{\mathcal{H}} := w_{\mathrm{L}}^{-1} \langle f_{\mathrm{L}}, g_{\mathrm{L}}\rangle_{\mathcal{H}_{\mathrm{L}}} + w_{\mathrm{G}}^{-1} \langle f_{\mathrm{G}}, g_{\mathrm{G}}\rangle_{\mathcal{H}_{\mathrm{G}}}. \tag{3}$$

The inner products in the two component RKHSs in (3) are simple to compute because, according to (1), they consist of (mainly) kernel evaluations. Note that the closed form in (3) is in general not valid for arbitrary choices of the two kernels.

C. MULTIUSER DETECTION

Consider a multiuser uplink with K users and M receive antennas. We assume a non-dispersive channel so that the received signal (sampled at a fixed symbol rate) at the time $t \in \mathbb{N}_0$ is given by

$$\mathbf{r} : \mathbb{N}_0 \to \mathbb{C}^M : t \mapsto [r_1(t), r_2(t), \ldots, r_M(t)]^{\mathsf{T}} \tag{4}$$

$$= \sum_{k=1}^{K} \sqrt{p_k} b_k(t) \mathbf{h}_k + \mathbf{n}(t), \tag{5}$$

where $p_k \in \mathbb{R}$ is the transmit power of user $k \in \overline{1, K}$, and $b_k(t) \in \mathbb{C}$ is the modulation symbol. The vectors $\mathbf{h}_k \in \mathbb{C}^M$ and $\mathbf{n}(t) \in \mathbb{C}^M$ stand for the channel signature of user k and additive noise, respectively. Note that we do not assume any distribution and structure of the noise and the receive antenna array, respectively. The objective of multiuser detection considered in this study is to design a filter $g^k : \mathbb{C}^M \to \mathbb{C}$ for a selected user k, such that $(\forall t \in \mathbb{N}_0)\ \left|g^k(\mathbf{r}(t)) - b_k(t)\right| \leq \epsilon$, where $\epsilon > 0$ is a small predefined noise tolerance. In other words, the goal is to detect the desired modulation symbols directly without any intermediate parameter estimation.

D. ADAPTIVE DETECTION IN SUM RKHS

In this section, we describe the APSM-based detection algorithm of [8]–[10]. We convert the complex vector $\mathbf{r}(t) \in \mathbb{C}^M$ into two real vectors $\mathbf{r}_1(t) := [\Re(\mathbf{r}(t))^{\mathsf{T}}\ \Im(\mathbf{r}(t))^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{2M}$ and $\mathbf{r}_2(t) := [\Im(\mathbf{r}(t))^{\mathsf{T}}\ -\Re(\mathbf{r}(t))^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{2M}$ which enables processing in real Hilbert spaces as considered in [12], [17]. Similarly, the training modulation symbols are converted to $[b_1(t)\, b_2(t)]^{\mathsf{T}} := [\Re(b(t))\ \Im(b(t))]^{\mathsf{T}} \in \mathbb{R}^2$. The proposed filter $f : \mathbb{R}^{2M} \to \mathbb{R}$ operates on $\mathbf{r}_1(t)$ and $\mathbf{r}_2(t)$ separately. The relation between f and the complex-valued filter g described in Section II-C is given by $(\forall t \in \mathbb{N}_0)\ (\mathbb{C} \ni) g(\mathbf{r}(t)) = f(\mathbf{r}_1(t)) + i f(\mathbf{r}_2(t))$, where i is the solution to the equation $i^2 = -1$. To simplify indexing, we define a new time index: $(\forall t \in \mathbb{N}_0)\ (\forall l \in \overline{1, 2})\ n := 2t + l - 1$, $\mathbf{r}_n = \mathbf{r}_{2t+l-1} := \mathbf{r}_l(t)$ and $b_n = b_{2t+l-1} := b_l(t)$. Henceforth, we denote the input space of received signals by $\mathcal{U} := \left\{\mathbf{r}_n \in \mathbb{R}^{2M} : n \in \mathbb{N}_0\right\}$. We now turn our attention to the design of an adaptive filter f such that $(\forall n \in \mathbb{N}_0)\ |f(\mathbf{r}_n) - b_n| \leq \epsilon$, where the precision is controlled by the design parameter $\epsilon > 0$. We assume that $f \in \mathcal{H}$ and a training sample $(\mathbf{r}_n, b_n) \in \mathcal{U} \times \mathbb{R}$ is available $\forall n \in \mathbb{N}_0$. Then, a closed and convex set of functions in $\mathcal{H}$ consistent with the training sample at time n is given by

$$C_n := \left\{f \in \mathcal{H} : |\langle f, \kappa(\mathbf{r}_n, \cdot)\rangle_{\mathcal{H}} - b_n| \leq \epsilon\right\}. \tag{6}$$

In the online learning setting considered here, the training samples arrive as a sequence and each sample defines a set of the form in (6). Ideally, the objective is to find a filter $f^* \in \mathcal{H}$ such that f^* is a member of all these sets, i. e., $f^* \in \bigcap_{n\in\mathbb{N}_0} C_n$. However, since it is challenging to find a low-complexity algorithm to solve this problem, we allow a finite number of sets not to share a common intersection and consider the simplified problem:

$$\text{find}\, f^* \in \bigcap_{n \geq n_o} C_n, \tag{7}$$

for some $n_o \in \mathbb{N}_0$, under the assumption that $\bigcap_{n\geq n_o} C_n \neq \emptyset$. The advantage of the above formulation is that we can find an $f \in \mathcal{H}$ that is close to the intersection in (7) utilizing an APSM-based [12], [18] algorithm which we describe below.

As in [17], [18], given an index set $\mathcal{J}_n$ of size W, and starting from an arbitrary $f_0 \in \mathcal{H}$, we construct a sequence of filter estimates in $\mathcal{H}$ given as

$$(\forall n \in \mathbb{N}_0) f_{n+1} = \sum_{j \in \mathcal{J}_n} q_j^n \mathbf{P}_{C_j}(f_n), \tag{8}$$

where $\mathbf{P}_{C_j}(f_n) = f_n + \beta_j^n \kappa(\mathbf{r}_j, \cdot) = f_n + \beta_j^n (w_{\mathrm{L}}\, \kappa_{\mathrm{L}}(\mathbf{r}_j, \cdot) + w_{\mathrm{G}}\, \kappa_{\mathrm{G}}(\mathbf{r}_j, \cdot))$ is the projection of f_n onto the set C_j, with β_j^n given by

$$\beta_j^n := \begin{cases} \frac{b_j - \langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}} - \epsilon}{\kappa(\mathbf{r}_j, \mathbf{r}_j)}, & \text{if } \langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}} - b_j < -\epsilon, \\ 0, & \text{if } |\langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}} - b_j| \le \epsilon, \\ \frac{b_j - \langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}} + \epsilon}{\kappa(\mathbf{r}_j, \mathbf{r}_j)}, & \text{if } \langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}} - b_j > \epsilon, \end{cases} \tag{9}$$

and where $(q_j^n)_{j \in \mathcal{J}_n}$ are non-negative weights satisfying $\sum_{j \in \mathcal{J}_n} q_j^n = 1$.

The index set $\mathcal{J}_n$ defined as $\mathcal{J}_n := \overline{n - W + 1, n}$ if $n \ge W - 1$, otherwise $\mathcal{J}_n := \overline{0, n}$, allows for a subset of sets $C_1, C_2, \ldots, C_n$ to be processed concurrently to accelerate convergence, and the weights q_j^n can be used to adaptively prioritize the sets. The computational advantage of this algorithm is that the projection $\mathbf{P}_{C_j}(f_n)$ only requires simple inner products, and the overall algorithm is amenable to parallelization resulting in significant acceleration as discussed in Section II-E.

Before we move onto the next section, it can be verified that the filter estimate generated by (8) can be decomposed as $f_n := \sum_{i=1}^{n-1} \gamma_i^{(n)} \kappa(\mathbf{r}_i, \cdot) = \sum_{i=1}^{n-1} \gamma_i^{(n)} w_{\mathrm{L}} \kappa_{\mathrm{L}}(\mathbf{r}_i, \cdot) + \sum_{i=1}^{n-1} \gamma_i^{(n)} w_{\mathrm{G}} \kappa_{\mathrm{G}}(\mathbf{r}_i, \cdot) =: f_{\mathrm{L},n} + f_{\mathrm{G},n}$, where $f_{\mathrm{L},n} \in \mathcal{H}_{\mathrm{L}}$ and $f_{\mathrm{G},n} \in \mathcal{H}_{\mathrm{G}}$ [9]. Since $\mathcal{H}_{\mathrm{L}}$ is nothing but the Euclidean space $\mathbb{R}^{2M}$, it is spanned by the Euclidean basis (which we refer to as the *linear dictionary* in the following) $\mathcal{D}_{\mathrm{L}} := \{w_{\mathrm{L}}\kappa_{\mathrm{L}}(\mathbf{e}_1, \cdot), w_{\mathrm{L}}\kappa_{\mathrm{L}}(\mathbf{e}_2, \cdot), \ldots, w_{\mathrm{L}}\kappa_{\mathrm{L}}(\mathbf{e}_{2M}, \cdot)\}$, where $\mathbf{e}_m \in \mathbb{R}^{2M}$ is the canonical basis vector having a one at the m-th index and zeros elsewhere. So, every $\kappa_{\mathrm{L}}(\mathbf{r}_n, \cdot)$ can be expressed as $\sum_{m=1}^{2M} [\mathbf{r}_n]_m \kappa_{\mathrm{L}}(\mathbf{e}_m, \cdot)$, with $[\mathbf{r}_n]_m$ the m-th entry of $\mathbf{r}_n$. As a result, the linear component $(\forall n \in \mathbb{N})\, f_{\mathrm{L},n} = w_{\mathrm{L}} \sum_{i=1}^{n-1} \gamma_i^{(n)} \kappa_{\mathrm{L}}(\mathbf{r}_i, \cdot) = w_{\mathrm{L}} \sum_{m=1}^{2M} \gamma_m^{(\mathrm{L},n)} \kappa_{\mathrm{L}}(\mathbf{e}_m, \cdot)$ consists of $2M$ components with their coefficients $\gamma_m^{(\mathrm{L},n)}$ updated by projections $\mathbf{P}_{C_j}(f_n)$ in (8). In contrast to the linear component, the *Gaussian dictionary* given by $\mathcal{D}_{\mathrm{G},n} := \{w_{\mathrm{G}}\kappa_{\mathrm{G}}(\mathbf{r}_1, \cdot)\,, w_{\mathrm{G}}\kappa_{\mathrm{G}}(\mathbf{r}_2, \cdot), \ldots, w_{\mathrm{G}}\kappa_{\mathrm{G}}(\mathbf{r}_{n-1}, \cdot)\}$ grows with time n as the iterations progress.

Remark 1 (ML: Computation and Communication): It is common in literature to assume that the channels between the users and the base stations remain constant only for a certain time period known as the channel coherence time [19]. Therefore, the training time should be much smaller than the coherence time such that a large portion of the coherence time can be used for detection/communication. From a processing and computation point-of-view, the training time is the total time it takes to collect/sample the training set and learn a good detection filter. This means that learning algorithms that have low complexity and can be accelerated are highly desired. Furthermore, in high-data rate communication fast detection is also highly desirable, so that as much data as possible can be detected during the coherence time.

E. ACCELERATION VIA PARALLEL PROCESSING

As mentioned above, the concurrent projections in (8) accelerate the convergence. Unfortunately, large values of W result in significant computational and memory burden and this may result in intolerable latency in real-time applications. However, note that the W projections in (8) are independent, which means that they can be computed in parallel on platforms equipped with GPUs optimized for such applications. In addition, the computation of β_j^n in (9) involves the inner product $\langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}}$. Since $f_n := \sum_{i=1}^{n-1} \gamma_i^{(n)} \kappa(\mathbf{r}_i, \cdot)$, this inner product is a sum of n independent inner products (equivalently kernel evaluations due to (1)), so these can also be computed in parallel. Furthermore, each linear kernel evaluation requires the dot-product $(\forall \mathbf{u}, \mathbf{v} \in \mathcal{U})\ \mathbf{u}^{\mathsf{T}}\mathbf{v} = \sum_{i=1}^{2M} [\mathbf{u}]_i [\mathbf{v}]_i$, while the Gaussian kernel evaluation requires the calculation of the Euclidean-norm $\sqrt{(\mathbf{u} - \mathbf{v})^{\mathsf{T}}(\mathbf{u} - \mathbf{v})}$ followed by further operations. This means that both kernel evaluations are independent component-by-component computations for any two input vectors followed by further operations on the sum or accumulation of these computations. The complexity of the above operations is linear in the dimension of the input vectors (the number of antennas M in this study), however the computation time can be reduced drastically if each component-by-component computation is executed in parallel. Finally, each computation requires access to data stored in various memory locations (discussed in Section III) on the computing platform which means that, in addition to the computational aspects, special attention should be paid to the memory allocation and access. Before we move on to the real-time implementation of the algorithm in (8), we remark how this APSM-based algorithm can be implemented in practical communication systems.

Remark 2 (Focus on Detection): APSM is an online algorithm that keeps track of the set of minima of an infinite number of time-changing objective functions. As mentioned in Remark 1, in a traditional communication system, generally an initial "training phase" is carried out during which the desired user sends a certain number of training pilots to the receiver, which allows the receiver to approximate a good initial receive filter using APSM. The training phase is then stopped, and it is followed by the communication or "detection phase" during which the response of the trained filter is used as an estimate of the desired user's modulation symbols. In principle, retraining is only required if the communication channels change abruptly resulting in large errors. In situations, such as mobile scenarios, where communication channels change gradually, we can, in principle, track these changes using APSM through the data symbols. Therefore, after the initial training phase, the latency is mainly dependent on the time incurred during detection of radio frames, which involves computation of a large number of inner products according to (1) *and* (3)*. Due*

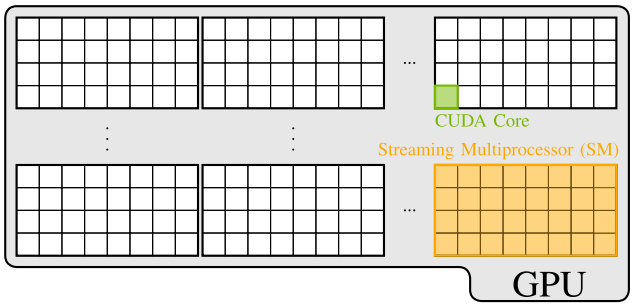


FIGURE 1. CUDA-device mapping. A GPU consists of thousands of CUDA cores grouped as SMs executing code in SIMT style. The number of SMs is determined by the size of the device. See Section III-A.

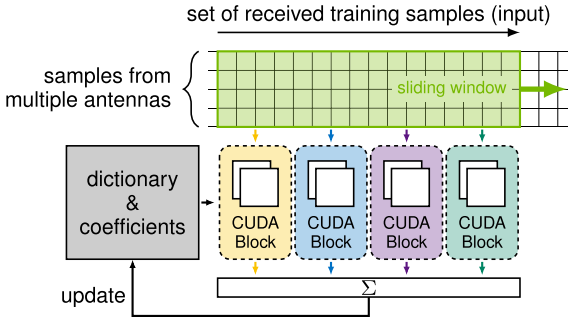


FIGURE 2. Training overview. The received vectors in the sliding window are processed by the CUDA blocks in parallel, computing projections of f_n onto the W convex sets defined by the W received vectors and the corresponding modulation symbols. The resulting estimate (bottom) is used to update the dictionary and the coefficients (left) then representing f_{n+1}.

to this reason, the acceleration of detection is the main focus of our work in the following.

III. REAL-TIME IMPLEMENTATION

In this section, we review the target platform, followed by a detailed description of the parallelization and optimization techniques used to achieve real-time performance.

A. PLATFORM OVERVIEW

GPUs are massively parallel processing devices that share a common architecture for general-purpose programming using the Compute Unified Device Architecture (CUDA) programming model. CUDA is a C++ extension that allows parallel code to be described as threads organized in a hierarchical structure, while also providing a framework for the communication between the host and the device. Each thread represents a logically independent sequence of instructions, similar to a thread in a classic programming model. Threads are grouped into blocks (groups of threads) and grids (groups of blocks). This hierarchical organization maps very well onto the device structure, with threads organized within the Streaming Multiprocessors (SMs) as warps on the hardware cores. A warp is a common grouping of cores executing in parallel as single instruction multiple threads (SIMT), and a SM can have one or more warps active at any given time, depending on the number of cores available per SM architecture. GPUs are composed of one or more SMs as depicted in Figure 1.

The above-mentioned hierarchy implies several types of memory, which differ in their sizes, bandwidths, and latencies. Global Memory is available to all threads and it is the largest (several Gigabytes off chip), but it also has the highest latency (from a hundred up to thousands of cycles). Shared Memory is available within SMs, and it is only available to threads belonging to the same block. It has lower latency (tens of cycles) than Global Memory and is smaller in size (a few kilobytes on chip). Finally, registers are local to every thread, small in number (a few per thread in a typical scenario) with the lowest latency.

Note that threads can cooperate with each other by collaborating to perform tasks more efficiently or operating in batches, for example, to reduce the perceived latency of the overall memory access (known as latency hiding). Finally, the architecture has limited coherence of both execution and memory consistency at different levels of the hierarchy. A detailed description of the platform is found in the CUDA programming guide [20].

B. BASIC IMPLEMENTATION

As discussed at the end of Section II-D, to access the filter estimate f_n in (8), we require access to the overall dictionary $\mathcal{D}_n := \{\mathcal{D}_{\mathrm{G},n} \cup \mathcal{D}_{\mathrm{L}}\}$ and the coefficients $\gamma_i^{(n)}$ at time n. The new filter estimate f_{n+1} is computed using projections on a sequence (sliding window) of W convex sets constructed from f_n, the training sample received at time n, and $\mathcal{D}_n$. The Gaussian dictionary is then extended with the newly arriving training sample and, along with the new coefficients $\gamma_i^{(n+1)}$, it is used to store f_{n+1}.

Every training sample (the received vector (5)) consists of samples from multiple antennas at time n during the training phase. One CUDA block can be used for each vector in the sliding window, with multiple threads inside the block computing the inner products $\langle f_n, \kappa(\mathbf{r}_j, \cdot)\rangle_{\mathcal{H}}$ in parallel. The final summation is performed in a reduction step, which can also be executed in parallel. As the new training samples arrives, the sliding window advances from n to $n+1$ and the process is repeated. This implementation variant is depicted in Figure 2. Furthermore, as discussed in Section II-E, since both the inner products and the per-vector operations can be computed in parallel, we can use the GPU resources efficiently, as detailed later in this section.

The detection stage computes $f_n(\mathbf{r}_n)$ (according to the formulas in (1) and (3)) as an estimate of b_n, where f_n is the trained filter at detection time n and $\mathbf{r}_n$ is the received vector. It does not modify $\mathcal{D}_n$ or $\gamma_i^{(n)}$, instead using them to access $f_n := \sum_{i=1}^{n-1} \gamma_i^{(n)} \kappa(\mathbf{r}_i, \cdot)$. Because $f_n(\mathbf{r}_n)$ is essentially the inner product $\langle f_n, \kappa(\mathbf{r}_n, \cdot)\rangle_{\mathcal{H}}$, the estimation can also be parallelized by processing each input vector in parallel, one per block. The required $n-1$ inner products are computed one per thread within the block. Finally, the results from threads in a block are summed to produce an estimation from the input vector as $f_n(\mathbf{r}_n) := \sum_{i=1}^{n-1} \gamma_i^{(n)} \kappa(\mathbf{r}_i, \mathbf{r}_n)$. This is depicted in Figure 3.

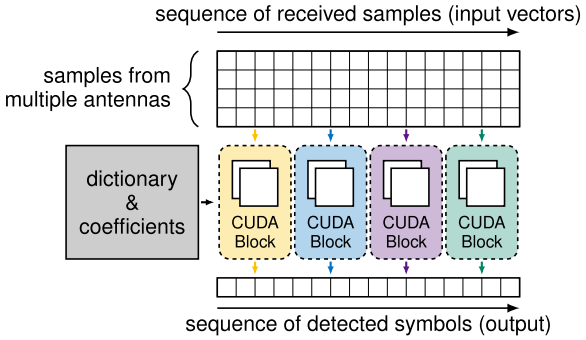


FIGURE 3. Detection overview. The top depicts the sequence of input vectors $\mathbf{r}_n$ for which we need to calculate $f_n(\mathbf{r}_n)$ as an estimate of b_n. Each input vector is processed by one CUDA block in parallel, and inside each CUDA block, the corresponding inner product is computed by multiple threads in parallel. The dictionary and the coefficients represent f_n. The result of each block is the approximation of the symbol b_n, which is stored in an array (bottom).

We point out that further parallelization is possible as the detection of each desired user can be performed independently. The number of users that can be detected in parallel is therefore only limited by the GPU resources available.

C. OPTIMIZATION OF INNER PRODUCTS

Both training and detection share the computation of the inner products $\langle f_n, \kappa(\mathbf{r}_n, \cdot)\rangle_{\mathcal{H}}$, which is the dominant operation of projection based algorithms. Implemented as a CUDA kernel to be executed on the GPU (not to be confused with the kernel functions κ), it is the primary target for optimization with respect to the latency of the system.

Parameters like the number of threads per group, the number of groups per block, and the total amount of vectors cached in shared memory all influence the performance. Device properties like type and speed grade of global memory, number of SMs, and amount of shared memory are important to consider. The optimal set of parameters is found by sweeping the parameter space combining experimentation and guidance by profiling tools.

While optimizing the parameters determining the execution of the CUDA kernel is part of performance optimization, algorithm transformations for massively parallel architectures are required beforehand. In the following sections, we explore such principle techniques that allow us to achieve the desired performance of a detection latency below 1 millisecond.

1) MULTIPLE SAMPLE VECTORS PER BLOCK

Our first technique aims at improving the *occupancy* of the device, which is a measure of the utilization of the resources available within the GPU. Utilization, which is a function of several factors, can be maximized by allocating enough work to keep the device busy within the constraints of the hardware. The most important constraints are the number of threads per SM, the number of registers used per thread, and the amount of shared memory utilized by a CUDA block.

To improve the occupancy of the CUDA kernel, we describe the algorithm using Cooperative Groups

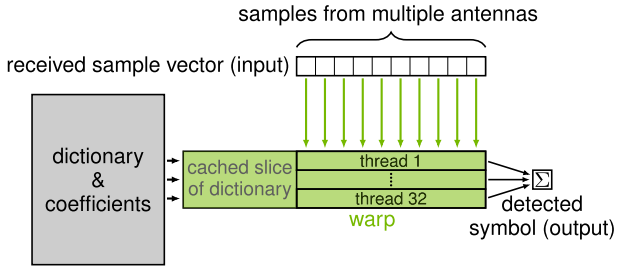


FIGURE 4. Shared Memory implementation. Each thread in the CUDA block (center) computes the inner product using one dictionary entry by performing the required computations over the components of the vector $\mathbf{r}_n$ (top). Multiple elements of the dictionary (left) are cached in shared memory (center, left) in batches, allowing for the efficient processing of a subset of the dictionary. The final result is accumulated and stored as the approximated modulation symbol (right).

(which are primitives for partitioning work inside a CUDA block, see [20]) and assign the processing of an input vector $\mathbf{r}_n$ in (5) to a single group. The threads within this group are responsible for performing the parallel computations for a particular input vector, and the result is reduced (accumulated) at the end in parallel to obtain the estimated symbol. We then assign multiple groups to one CUDA block, with each group processing a different received vector within the block. We can therefore use the number of groups in a block to maximize the occupancy of the SM.

2) USAGE OF SHARED MEMORY

Our second technique benefits from using the shared memory within the SMs of a device. As mentioned before, each block processes multiple input vectors and, for each input vector, computes inner products using the same dictionary entries. Hence each entry of the dictionary may be loaded once and reused multiple times. Shared memory is well suited for this because it is accessible to all groups within a block. As a consequence global memory traffic is reduced by lowering the amount of data requested, reducing the time to load the data.

Every thread in the block reads the vector $\mathbf{r}_n$ from memory and then calculates the inner products with one entry of the cached shared slice of $\mathcal{D}_n$. Once the inner products are calculated for this slice, a new slice of $\mathcal{D}_n$ is read into the cache and this process continues until the inner products for all members of $\mathcal{D}_n$ are computed. This is depicted in Figure 4 for the threads of one group; other groups (not depicted) in the same block operate over different input vectors but the same cached slice of the dictionary.

3) TUNING AND HIDING LATENCY

While calculating the inner products, the CUDA kernel spends most of its time accessing shared memory, and the computations that follow memory reads are faster than the memory accesses, which can be verified by using a profiler tool. Therefore, it is desirable to have a balance between memory accesses and computation, so neither will dominate (and possibly limit) the performance.

Both the inner products (i. e., the linear and the Gaussian kernel evaluations) require access to the received vector $\mathbf{r}_n$ and members of the dictionary $\mathcal{D}_n$. The simple approach

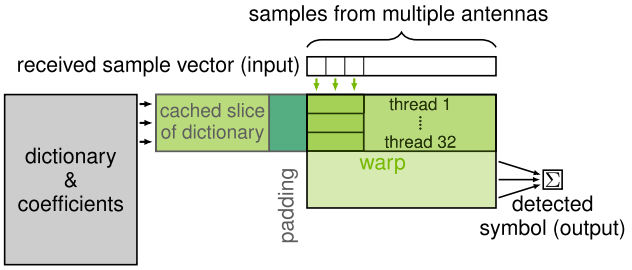


FIGURE 5. Balanced implementation. Threads in a CUDA block process subsets of both the input vectors and the dictionary entries in contrast to Figure 4. Since only a subset of the vector $\mathbf{r}_n$ is processed, the intermediate results (before the entire $\mathbf{r}_n$ is processed) have to be temporarily stored. The size of the cached dictionary elements is independent of the number of threads in the block (in contrast to Figure 4), and it is tuned to balance the time spent in computing and data load.

described in Section III-C2 can be optimized by changing the order of the computations and increasing the amount of data in the cached slice (see Figure 5). By fetching only a subset of $\mathbf{r}_n$ once and reusing it across a bigger section of the cache, without computing the full inner product in one go, the number of cache loads is reduced and the efficiency is increased. Note that this comes at a relatively small cost of requiring intermediate storage in shared memory until the entire $\mathbf{r}_n$ is processed. This "rearrangement" of the inner product computations is shown in Figure 5, and it achieves a better balance between memory access and computations, allowing for load and computation to overlap and decrease the overall inner product computation time by about half.

IV. EXPERIMENTAL SETUP

This section describes the hardware components of our real-time multiuser detection setup. The transmitter, receiver, and signal processing equipment are shown in Figure 6. Except for the Uniform Planar Array (UPA) shown in Figure 7, the components are COTS devices.

A. SIGNAL PROCESSING

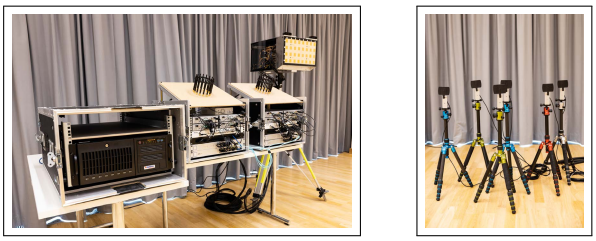
For development and testing, we use different locals servers and also several high-performance computing (HPC) units that are equipped with multiple central processing unit (CPU) cores and at least one GPU. Our local server is shown in Figure 6(a), this server is equipped with a Xeon W-3245 CPU and two RTX 2080 Ti consumer GPUs. This server also handles the signal processing part and the data transfer from and to the Software-Defined Radios (SDRs).

B. RADIO ACCESS

The basestation is composed of four Ettus USRP N310 SDRs. Furthermore, we use a single National Instruments (NI) OctoClock for a global positioning system (GPS) disciplined clock and timing source for our SDRs. With this setup, all four SDRs, each equipped with four ports on the transmitter (Tx) and receiver (Rx) path, behave like a single SDR system with up to sixteen synchronized physical antenna ports for both the Rx and Tx paths.

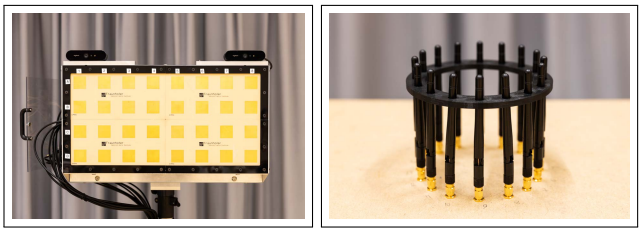
TABLE 1. NOMA system parameters.

	Value	Comment
Maximum system bandwidth	30.72 MHz	
UCA radius	6.5 cm	
UPA patch resonant frequency	2.442 GHz	$\lambda = 12.28$ cm
UPA patch element spacing	6.14 cm	$\lambda/2$
UPA patch element length and width	3.07 cm	$\lambda/4$
Used polarization	vertical	
Number of Tx SDR modules	1	NAMC-SDR
Number of Rx SDR modules	4	USRP N310
Number of Rx antenna ports	16	



(a) Base station (receiver) (b) Users (transmitter)

FIGURE 6. Real-time machine learning based multiuser detection setup.



(a) Uniform Planar Array (b) Uniform Circular Array

FIGURE 7. Antenna array configurations.

A subset of our system parameters is shown in Table 1 and is also given in the following Section V. For over the air signal transmission, we use an OFDM system mode with a sampling rate of 30.72 MHz. Note that we can switch between radio signals from our UPA shown in Figure 7(a) and our Uniform Circular Array (UCA) shown in Figure 7(b). The receive antenna arrays and user equipment (the transmitters) are described below.

1) USERS ANTENNAS

Each user antenna is installed on a tripod, allowing for conveniently adjusting location and height, relative to the receiving antenna array. Every single antenna shown in Figure 6(b) represents a single transmitting user, sending its radio signal, and is connected to one of the eight transmitter ports on our SDR module [21]. According to the datasheet, the antenna gain is 5 dBi at 2.5 GHz and up to 7 dBi at 5.7 GHz, and the antenna polarization is vertical.

2) UNIFORM PLANAR ARRAY

The UPA, shown in Figure 7(a), is equipped with 32 cross-polarized patch antenna elements. These elements are

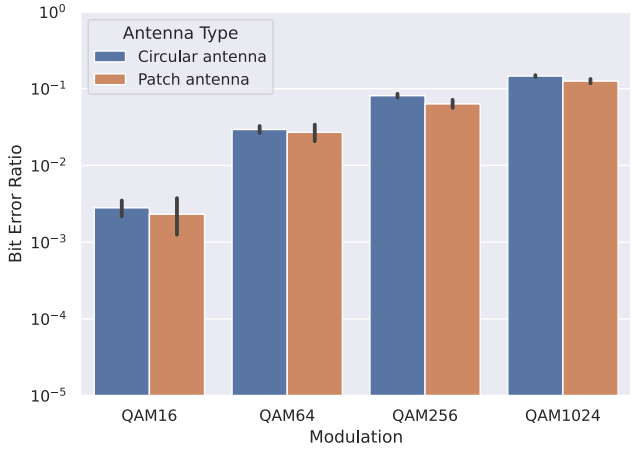


FIGURE 8. Average relative bit error rate for different modulation schemes. Black bars show the ±1SD confidence interval. BPSK and QPSK were omitted from the plot because the BER is negligible.

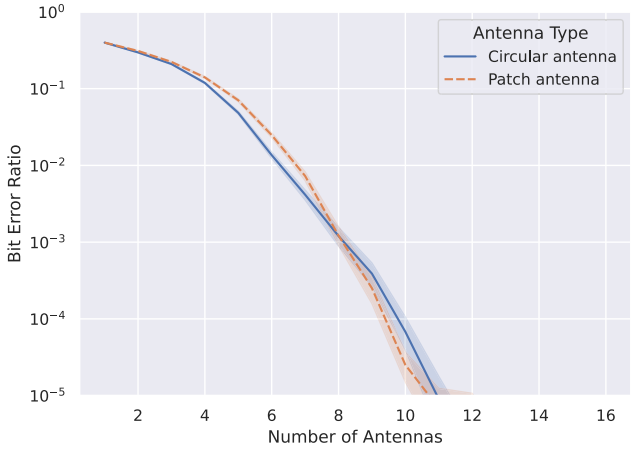


FIGURE 9. The average relative bit error rate for different numbers of antennas using QPSK. Shaded regions show the ±1SD confidence interval.

TABLE 2. Summary of the performance improvements of the detection CUDA kernel for various optimization steps as described in Section III-C.

	RTX 2060 super	RTX 2080 Ti	Titan V
Baseline Implementation	7.82 ms	3.676 ms	2.23 ms
Multiple Vectors/Block	7.76 ms	2.180 ms	2.37 ms
Shared Memory	1.66 ms	1.615 ms	0.479 ms
Balanced	1.25 ms	0.906 ms	0.403 ms

arranged in 4 rows and 8 columns. Each of 32 equidistant patch elements operates at a center frequency of 2.442 GHz.

3) UNIFORM CIRCULAR ARRAY

The sixteen physical Rx antennas are arranged as a UCA, as shown in Figure 7(b). Uniform spacing is ensured by a ring retainer around the 16 antennas. The antenna operates in the 2.4 GHz and 5 GHz Wireless Local Area Network (WLAN) band with an omnidirectional radiation pattern and vertical polarisation.

V. PERFORMANCE RESULTS

In this section, we present performance results for the experiments based on the setup presented in Section IV. The objective is to demonstrate the real-time performance of the partially linear filtering algorithm presented in Section II-D in combination with the parallelization techniques and optimizations presented in Section III. In the following, we first show the performance of the partially linear filtering for various simulation settings and parameters. We then compare the detection latencies of multiple platforms and show the acceleration in processing using techniques proposed in this study.

For the algorithm in (8), unless noted otherwise, we use uniform Gaussian and linear weights, i. e., $w_{\mathrm{L}} = w_{\mathrm{G}} = 0.5$, a sliding window size of $W = 20$, and a Gaussian kernel variance of $\sigma^2 = 0.05$. We use 16 antennas, and 6 users with uniform transmit power. Each data point is an average of 100 transmissions, consisting of 685 training symbols and 3840 data symbols each. The symbols consist of Gray-coded constellation points. Modulated training and data symbols occupy up to 144 subcarriers on 5 OFDM symbols, and 27 OFDM symbols, respectively. We use 15 kHz subcarrier spacing, which results in a total signal bandwidth of 2.16 MHz.

Figure 8, shows the BER performance of standard modulation schemes. All experiments were performed with the above-mentioned parameters. The error rate is negligible for binary phase-shift keying (BPSK) and quadrature phase-shift keying (QPSK), while for higher-order modulations a significant number of bit errors occur. As shown in Figure 6(b), the angular separation of the users is very small, which renders the task of separating them difficult. Nevertheless, we observe that our algorithm can reach acceptable bit error rates even for high modulation schemes. Note that in a practical system our algorithm would be followed by forward error correction (FEC), which will correct the remaining bit errors.

In Figure 9, we demonstrate the BER performance as a function of an increasing number of antennas, where the specified number of antennas was chosen randomly among the 16 antennas available in our system. As mentioned in Section II-D, the algorithm does not assume any particular antenna array structure. The measurements are executed for the two types of antenna arrays presented in Section IV. We note that the results for both antenna types are very similar. We observe that the users become simpler to separate with an increasing number of antennas, and hence the bit error rate tends toward zero. Even for a relatively small number of antennas as compared to the number of users, the performance is acceptable because the Gaussian kernels provide the capability to separate users that cannot be separated linearly.

In Table 2, we present the detection latencies of the different implementations presented in Section III-C on different GPUs. The GPUs represent two different device families: The RTX 2060 super and RTX 2080 Ti are consumer-grade boards with Graphics Double Data Rate (GDDR) memory, while the Titan V is a data-center class board equipped with High Bandwidth Memory (HBM) and

with more computing resources. All used GPU cards execute the same code (neither device-specific optimizations nor tuning was carried out). The table does not show the time elapsed during training, which amounts to roughly 100 ms. As mentioned in Remark 2, the relatively long training time is not of concern because retraining is, in principle, only required if the environment changes abruptly. For comparison, a MATLAB implementation of our algorithm takes more than 40 s to execute on an i7-6700T CPU with simple parallelization (up to 8 Threads on 4 Cores) from the native Parallel Toolbox. The comparison with signal processing in MATLAB may not be fair, because we did not optimize the MATLAB code for speed. Nevertheless, one may conclude that the optimized CUDA implementation is several orders of magnitude faster than a native CPU-based implementation.

VI. CONCLUSION

Our proof-of-concept provides a practical and fast implementation of recently proposed partially linear adaptive filtering for multiuser detection on GPU-accelerated platforms. We exploit the parallelism intrinsic to the mathematical formulation and distribute the computations in an "optimal" way across the targeted GPU. The techniques developed for hiding latency and accelerating memory access are key to reaching real-time performance. As a result, we can perform multiuser detection with a detection latency of below 1 millisecond with our commercial off-the-shelf (COTS) laboratory setup. In future work, similar techniques will be applied to drastically reduce the total amount of processing time spent on training. Moreover, channel coding will be included in the future. Finally, we note that APSM shares many operations with similar projection-based algorithms in Hilbert spaces which have seen many applications in signal processing and machine learning. Therefore, our acceleration techniques may be generalized to such projection-based algorithms and algorithms based on reproducing kernel Hilbert spaces. The developed and used CUDA APSM library code is publicly available on the GitHub server of the Fraunhofer HHI [22].

REFERENCES

[1] W. Shin, M. Vaezi, B. Lee, D. J. Love, J. Lee, and H. V. Poor, "Non-orthogonal multiple access in multi-cell networks: Theory, performance, and practical challenges," *IEEE Commun. Mag.*, vol. 55, no. 10, pp. 176–183, Oct. 2017.

[2] Y. Wang, B. Ren, S. Sun, S. Kang, and X. Yue, "Analysis of non-orthogonal multiple access for 5G," *China Commun.*, vol. 13, no. 2, pp. 52–66, Feb. 2016.

[3] Z. Ding, X. Lei, G. K. Karagiannidis, R. Schober, J. Yuan, and V. K. Bhargava, "A survey on non-orthogonal multiple access for 5G networks: Research challenges and future trends," *CoRR*, vol. abs/1706.05347, pp. 1–32, Jun. 2017.

[4] H. Tabassum, M. S. Ali, E. Hossain, M. J. Hossain, and D. I. Kim, "Non-orthogonal multiple access (NOMA) in cellular uplink and downlink: Challenges and enabling techniques," 2016, *arXiv:1608.05783*.

[5] K. Higuchi and A. Benjebbour, "Non-orthogonal multiple access (NOMA) with successive interference cancellation for future radio access," *IEICE Trans. Commun.*, vol. E98.B, no. 3, pp. 403–414, 2015.

[6] X. Su, H. Yu, W. Kim, C. Choi, and D. Choi, "Interference cancellation for non-orthogonal multiple access used in future wireless mobile networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2016, no. 1, pp. 1–12, Dec. 2016.

[7] S. M. R. Islam, N. Avazov, O. A. Dobre, and K.-S. Kwak, "Power-domain non-orthogonal multiple access (NOMA) in 5G systems: Potentials and challenges," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 721–742, 2nd Quart., 2017.

[8] M. Mehlhose, D. A. Awan, R. L. G. Cavalcante, M. Kurras, and S. Stanczak, "Machine learning-based adaptive receive filtering: Proof-of-concept on an SDR platform," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–5.

[9] D. A. Awan, R. L. G. Cavalcante, M. Yukawa, and S. Stanczak, "Detection for 5G-NOMA: An online adaptive machine learning approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[10] D. A. Awan, "Robust learning in wireless networks: Efficacy of models and prior knowledge in learning from small sample sets," Ph.D. thesis, Netw. Inf. Theory (NetIT), Faculty IV, Tech. Univ. Berlin, Berlin, Germany, 2021, doi: 10.14279/depositonce-11266.

[11] E. Bjornson, J. Hoydis, and L. Sanguinetti, *Massive MIMO Networks: Spectral, Energy, and Hardware Efficiency*. Norwell, MA, USA: Now Foundations and Trends, 2017.

[12] I. Yamada and N. Ogura, "Adaptive projected subgradient method for asymptotic minimization of sequence of nonnegative convex functions," *Numer. Funct. Anal. Optim.*, vol. 25, nos. 7–8, pp. 593–617, Jan. 2005, doi: 10.1081/NFA-200045806.

[13] P. L. Combettes, "The foundations of set theoretic estimation," *Proc. IEEE*, vol. 81, no. 2, pp. 182–208, Feb. 1993.

[14] H. Stark, Y. Yang, and Y. Yang, *Vector Space Projections: A Numerical Approach to Signal and Image Processing, Neural Nets, and Optics*. New York, NY, USA: Wiley, 1998.

[15] *NVIDIA Aerial: Build and Deploy GPU-Accelerated 5G Virtual Radio Access Networks (vRAN)*. Accessed: Jun. 24, 2022. [Online]. Available: https://developer.nvidia.com/aerial-sdk

[16] M. Yukawa, "Adaptive learning in Cartesian product of reproducing kernel Hilbert spaces," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 6037–6048, Nov. 2015.

[17] K. Slavakis, S. Theodoridis, and I. Yamada, "Adaptive constrained learning in reproducing kernel Hilbert spaces: The robust beamforming case," *IEEE Trans. Signal Process.*, vol. 57, no. 12, pp. 4744–4764, Dec. 2009.

[18] S. Theodoridis, K. Slavakis, and I. Yamada, "Adaptive learning in a world of projections," *IEEE Signal Process. Mag.*, vol. 28, no. 1, pp. 97–123, Jan. 2011.

[19] J. Du and R. A. Valenzuela, "How much spectrum is too much in millimeter wave wireless access," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 7, pp. 1444–1458, Jul. 2017.

[20] *CUDA Programming Guide*. Accessed: Jun. 24, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[21] *N.A.T. GmbH NAMC-SDR*. Accessed: Jun. 24, 2022. [Online]. Available: https://www.nateurope.com/products/NAMC-SDR.html

[22] *CUDA APSM Library*. Accessed: Jun. 24, 2022. [Online]. Available: https://github.com/fraunhoferhhi/libapsm

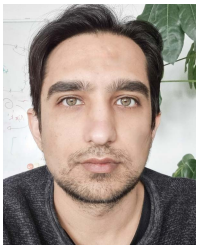
MATTHIAS MEHLHOSE received the Dipl.-Ing. (FH) degree in telecommunications and communications techniques from the University of Applied Sciences for Engineering and Economics (FHTW), Berlin, in 2008, and the Master of Science degree in electrical engineering from the Technical University of Berlin (TU Berlin), in 2011.

He started his employment at the Wireless Communications and Networks Department, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), in 2007, with an internship. Until 2012, he works there as a Student Assistant. Since 2012, he has been working as a Research Associate and an Engineer with the Wireless Communications and Networks Department, Fraunhofer Institute for Telecommunications, HHI. His research interest includes implementation of real-time signal processing algorithms on multi-core processors with high-speed data links to SDR systems equipped with massive MIMO antenna arrays.

GUILLERMO MARCUS received the degree in electronics engineering from Simón Bolívar University, Venezuela, the M.Sc. degree from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Mexico, and the Ph.D. degree *(magna cum laude)* from Heidelberg University, Germany. He joined NVIDIA, in 2013. He is currently a Senior Software Engineer at NVIDIA at the intersection of graphics, communications, and machine learning. He is interested in software engineering and applied scientific computing with a focus on accelerated computing and application-specific accelerators. He received the PRACE Award at ISC 2011.

DANIEL SCHÄUFELE received the B.Sc. degree in IT-systems engineering from the Hasso-Plattner-Institute, University of Potsdam, Germany, in 2012, and the B.Sc. and M.Sc. degrees in computer engineering from the Technical University of Berlin, Germany in 2015 and 2017, respectively.

He is currently a Research Associate with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, Germany. His research interest includes applying machine learning algorithms to wireless communication systems.



DANIYAL AMIR AWAN received the M.Sc. and Ph.D. *(summa cum laude)* degrees in electrical engineering from the Technical University of Berlin (TU Berlin), Germany, in 2013 and 2020, respectively. He is currently a Research Associate with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), Berlin, Germany. Previously, he worked as a Research Associate at the TU Berlin and as a Guest Researcher at the HHI. His research interests include application of optimization theory, function analysis, and machine learning to problems in wireless communication systems.

NIKOLAUS BINDER received the M.S. degree in computer science from the University of Ulm, Germany.

He is currently a Senior Research Scientist at NVIDIA. He joined NVIDIA, in 2011. He helped planning and implementing several government funded projects for two companies, maintained their IT infrastructure, and worked for the mental images GmbH as a Research Consultant. His research, publications, and presentations are focused on quasi-Monte Carlo methods, photorealistic image synthesis, ray tracing, and rending algorithms with a strong emphasis on the underlying mathematical and algorithmic structure.

MARTIN KASPARICK received the Dipl.-Ing. and Dr.-Ing. degrees from the Technische Universität Berlin (TU Berlin), Germany, in 2009 and 2015, respectively. From 2010 to 2013, he had been a Research Associate with the Heinrich Hertz Chair of Information Theory and Theoretical Information Technology, TU Berlin. Since 2013, he has been with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), Berlin. He currently leads the Signal and Information Processing Group, Fraunhofer HHI. His current research interests include network optimization, V2X communications, software-defined radio, and the application of machine learning to wireless communication systems.

RENATO L. G. CAVALCANTE (Member, IEEE) received the Electronics Engineering degree from the Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil, in 2002, and the M.E. and Ph.D. degrees in communications and integrated systems from the Tokyo Institute of Technology, Tokyo, Japan, in 2006 and 2008, respectively. He is currently a Research Fellow with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), Berlin, Germany, and a Lecturer with the Technical University of Berlin. Previously, he held appointments as a Research Fellow with the University of Southampton, U.K., and as a Research Associate with The University of Edinburgh, U.K. His current research interests include signal processing for distributed systems, multiagent systems, convex analysis, machine learning, and wireless communications. He received the Excellent Paper Award from the Institute of Electronics, Information and Communication Engineers in 2006 and the IEEE Signal Processing Society (Japan Chapter) Student Paper Award in 2008. He also coauthored a study that received a Best Student Paper Award at the 13th IEEE International Workshop on Signal Processing Advances in Wireless Communications in 2012. From April 2003 to April 2008, he was a recipient of the Japanese Government (MEXT) Scholarship.

SŁAWOMIR STAŃCZAK received the Dipl.-Ing. and Dr.-Ing. *(summa cum laude)* degrees in electrical engineering from the TU Berlin, in 1998 and 2003, respectively, and the Habilitation Degree (venialegendi), in 2006. He studied electrical engineering with specialization in control theory at the Wroclaw University of Technology and at the Technical University of Berlin (TU Berlin). Since 2015, he has been a Full Professor of network information theory with the TU Berlin and the Head of the Wireless Communications and Networks Department, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI). He is the coauthor of two books and more than 200 peer-reviewed journal articles and conference papers in the area of information theory, wireless communications, signal processing, and machine learning. From February 2018 to July 2020, he was the Chairperson of the ITU-T Focus Group on machine learning for future networks, including 5G. From 2012 to 2015, he was an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING.

ALEXANDER KELLER received the Dr.Rer.Nat. degree *(summa cum laude)* from the Technical University of Kaiserslautern.

He is currently the Director of Research at NVIDIA. Before, he had been the Chief Scientist of mental images. Prior to industry, he worked as a Full Professor of computer graphics and scientific computing at Ulm University. He has more than three decades of experience in ray tracing, pioneered quasi-Monte Carlo methods for light transport simulation that are used across industries, and connected the domains of machine learning and transport simulation. His research interests include wireless communication systems, machine learning, and computer graphics.