Savannah: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources

Zhenzhou Qi* Duke University
Chung-Hsuan Tung* Duke University
Anuj Kalia Microsoft
Tingjun Chen Duke University

ABSTRACT

5G new radio (NR) employs frequency range 2 (FR2) in the millimeter-wave (mmWave) bands, which employs a much shorter slot duration compared to FR1 (sub-7 GHz) systems and, therefore, poses significant challenges for softwarized baseband processing in virtualized radio access networks (vRANs). Existing systems supporting software baseband processing focus on enabling (massive) multiple-input and multiple-output (MIMO) using multi-core edge server(s). These solutions may fail to meet the more stringent processing deadline in FR2 or require more intensive computational resources. In this paper, we present Savannah, an efficient mmWave baseband processing framework using minimal and heterogeneous computing resources including CPU and eASIC. Savannah addresses the challenges associated with baseband processing in FR2 by applying techniques for vectorizing matrix operations and memory access patterns, supporting heterogeneous computation via offloading LDPC decoding to an eASIC, and enabling single-core operation. We show that Savannah, using a single CPU core and the ACC100 accelerator, can support a 2×2 MIMO link with 100 MHz bandwidth, yielding a data rate of up to 487 Mbps.

CCS CONCEPTS

• **Networks** → **Wireless access points, base stations and infrastructure**; **Network performance analysis**.

KEYWORDS

Millimeter-wave, Baseband processing, ACC100 accelerator

ACM Reference Format:
Zhenzhou Qi, Chung-Hsuan Tung, Anuj Kalia, and Tingjun Chen. 2024. **Savannah**: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3636534.3690707

*Both authors contributed equally to this work.

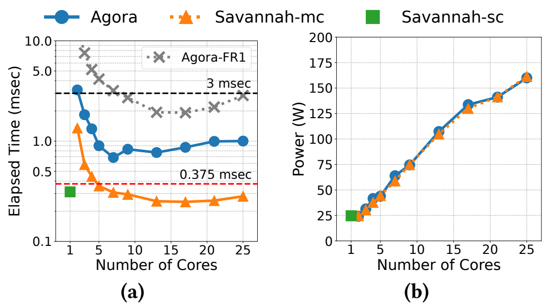


Figure 1: 99.9th percentile elapsed time and power consumption achieved by Agora [13] and Savannah (this work) in a 2×2 MIMO link with 100 MHz bandwidth in 5G FR2 with 0.125 ms slot. The gray dashed line indicates Agora processing the same amount of fronthaul traffic in an 8×8 MIMO link with 20 MHz bandwidth in FR1 with 1 ms slot. Power consumption excludes the system's standby power.

1 INTRODUCTION

5G networks are designed to support diverse applications that require high wireless data rates. 3GPP has defined frequency range 1 (FR1) and frequency range 2 (FR2) utilizing the sub-7 GHz and millimeter-wave (mmWave) bands, respectively [7]. Recent research targeting FR2 systems has received significant attention due to the widely available spectrum in the mmWave band [15, 17, 18, 24, 41, 42]. However, the higher data rate is accompanied by a tighter processing deadline due to the much shorter slot duration in FR2, posing significant challenges associated with real-time baseband processing [3]. Another feature of the 5G network is the virtualized Radio Access Network (vRAN), where the Physical layer (PHY) is centralized and assigned to distributed units (DUs) and centralized units (CUs). vRAN enables multiple radio units (RUs) to share compute resources, improving facility utilization and efficiency. 3GPP reports multiple split options [3] to offload the PHY processing stages to the DU/CU and option 8 only leaves radio frequency processing in RU, benefiting from vRAN architecture the most. Studies thrive in the context of split option 8 for both software baseband processing and sharing resource pool [13, 21, 25, 53, 61].

Previous research [13, 21] has focused on improving the efficiency of 5G vRANs via the softwarization of baseband processing. For example, Agora [13] is a real-time software framework running on a single multi-core server to support FR1 massive MIMO up to 64×16. To demonstrate the unique challenges associated with software baseband processing in FR2 due to the much shorter latency requirements, we run Agora to process the same volume of fronthaul traffic in two settings: (*i*) 8×8 MIMO with 20 MHz bandwidth in FR1 with 1 ms slot, and (*ii*) 2×2 MIMO with 100 MHz bandwidth in FR2 with 0.125 ms slot. For both settings, we set a processing deadline of three slots for a five-slot frame schedule of DDDSU. Fig. 1(a) shows that Agora is able to meet the 3-slot deadline of 3 ms in FR1 with 9 cores. However, it fails to meet the 3-slot deadline of 0.375 ms in FR2, even with more CPU cores, due to the tighter deadline and inefficient matrix operations.

In this paper, we present Savannah[1], a system for efficient mmWave baseband processing using minimal and heterogeneous computing resources, including CPU and eASIC. To address these challenges associated with baseband processing imposed by FR2, Savannah applies techniques for (*i*) optimizing vectorized matrix operations and memory access patterns, (*ii*) supporting heterogeneous computation via offloading LDPC decoding to Intel's ACC100 accelerator [49], and (*iii*) enabling single-core scheduling. We also comprehensively characterize the ACC100 accelerator for its decoding throughput and energy efficiency. Savannah achieves real-time processing for the given FR2 configuration in (*i*) the multi-core variant (Savannah-mc) with five CPU cores, or (*ii*) the single-core variant (Savannah-sc) with heterogeneous computing resources. The two variants show the performance tradeoff between dedicated hardware and CPU, where Savannah-sc outperforms both Savannah-mc and Agora in energy efficiency, as shown in Fig. 1(b).

We also evaluate Savannah without the ACC100 accelerator and compare it to previous software baseband processing works. These works focus on supporting massive MIMO systems in FR1 with extensive computing resources, while Savannah can achieve the same data rate in FR2 with minimal hardware resources. For example, Agora [13] requires 26 cores on a single server to achieve a data rates of 454 Mbps using 64×16 MIMO with 64QAM and 1/3 code rate; Hydra [21] utilizes four 32-core servers (among which 71 cores are allocated for processing) to achieve a data rate of 950 Mbps of 150×32 MIMO following the same FR1 configuration. In FR2, a data rate of 487 Mbps and 976 Mbps can be achieved by 2×2 and 4×4 MIMO links, both with 100 MHz bandwidth, under a similar modulation and coding scheme (MCS). Meanwhile, the number of CPU cores required by Savannah to meet the tighter processing deadline (5 and 21 cores) is reduced by 5.2× and 3.4× compared to Agora and Hydra, respectively.

[1]Savannah is a breed of hybrid cat from crossing a Leptailurus serval (a wild cat native to Africa) with a domestic cat (Felis catus).

We also implement Savannah using real radio units (RUs) in the PAWR COSMOS testbed [12, 43] with FR2 frontends and evaluate it via over-the-air (OTA) experiments.

To summarize, the main contributions of this paper are:

- We design Savannah, a system achieving efficient, real-time mmWave baseband processing through the joint optimization of matrix operations and memory layout, complemented by the incorporation of a hardware LDPC decoder;
- We present a first-of-its-kind in-depth characterization of the performance of Intel's ACC100 accelerator [49] with a focus on the decoding throughput and energy efficiency, as well as the comparison to a software LDPC decoder based on Intel's FlexRAN SDK;
- We experimentally evaluate Savannah and demonstrate its capability to meet the demanding processing deadline across various FR2 PHY configurations.

To the best of our knowledge, this is the first work that targets real-time mmWave baseband processing and shows that the stringent processing deadline imposed by FR2 settings can be met using minimal and heterogeneous computational resources.

Savannah is open-sourced [2].

2 RELATED WORK

mmWave communications and networking. Compared to the sub-7 GHz band, the mmWave band is featured with larger attenuation over distance and weaker multipath effects. To obtain the channel information, beam training [19, 23, 28, 59, 63] is usually adopted before establishing the mmWave link; furthermore, beam tracking [37, 52, 57] is the successive step that maintains the established link over time, especially in dynamic scenarios with user mobility. Due to the heavy overhead incurred, recent efforts have been devoted to accelerating these two steps [23, 57, 59]. In the higher layers, recent research focused on multi-user MIMO and joint transmission (e.g. [62]), integrated access and backhaul (e.g., [38]), and various applications such as sensing and localization (e.g., [60]).

mmWave testbeds. There has been extensive recent work on the design and implementation of mmWave radios and testbeds in both the SISO and MIMO settings. These include the unlicensed 60 GHz band for 802.11ad (e.g., OpenMili [64], X60 [47], MillimeTera [39], M-Cube [65], and MIMORPH [29]), and the 28 GHz band for 5G FR2 (e.g., mMobile [27], and COSMOS [11, 43]). In particular, the PAWR COSMOS testbed provides the research community access to a set of advanced mmWave radios that integrate the IBM 28 GHz 64-element phased array antenna module (PAAM) [46].

LDPC decoding. The optimization of LDPC decoding has gathered significant attention in communication systems, particularly with the advancement of 5G technologies. Deep

Table 1: 5G NR FR2 PHY layer configurations with numerology 3 ($\mu = 3$) and a slot duration of 0.125 ms.

Channel Bandwidth	SCS, Δf	Max # of RBs (# of Subcarriers)	FFT Size	Sampling Rate, F_s
100 MHz	120 kHz	66 (792)	1,024	122.88 MSa/s
200 MHz	120 kHz	132 (1,584)	2,048	245.76 MSa/s
400 MHz	120 kHz	264 (3,168)	4,096	491.52 MSa/s

unfolding technique [58] is adopted to improve performance for binary linear codes while [33] illustrates the architecture that supports frame parallelism to maximize the utilization of processing units in field-programmable gate array (FPGA). Practical GPU-based acceleration for the LDPC decoding has been investigated in [54, 56]. In contrast, Savannah integrates Intel's ACC100 eASIC [49] for LDPC decoding which, to the best of our knowledge, has not been thoroughly characterized or integrated into any open-source framework.

Software baseband processing. The innovation of using commodity CPUs for high-performance PHY processing is first illustrated by Sora [53] on wireless signal processing for WiFi and BigStation [61] expanded the concept to address more complex scenarios specifically multiuser MIMO. More recent studies such as Agora [13] present the processing for massive MIMO in a single server while Hydra [21] expanded the idea to a distributed system across multiple servers. Open source projects such as srsRAN [50] and OpenAirInterface (OAI) [35] benefit the research community with mature software solutions on 5G vRAN. Though srsRAN [50] supports full FR1 and 15/30 kHz subcarrier spacing, the open document does not report any FR2 support upon submission time of this work. In constrast, OAI [35] supports FR2 $\mu = 3$ and up to 100 MHz bandwidth. Savannah explores the real-time baseband processing possibilities under FR2's stringent time requirements with minimal and heterogeneous resources and prototypes 200 MHz bandwidth in the SISO configuration.

3 PRELIMINARIES

3.1 5G NR Specifications

5G NR frame structure and numerology. A radio frame in 5G NR has a fixed duration of 10 ms and consists of 10 subframes, each with a duration of 1 ms. Unlike LTE, which employs a fixed subcarrier spacing (SCS) of 15 kHz, 5G NR introduces a flexible numerology (μ) and supports various SCS values given by $\Delta f = 2^\mu \cdot 15$ kHz. The 3GPP Release 17 specifies seven numerology options [8], each of which can only be used for specific types of physical channels. For example, $\mu = 3$ ($\Delta f = 120$ kHz) can only be used in FR2, including the physical downlink/uplink shared channel (PDSCH/PUSCH), whereas $\mu = 2$ ($\Delta f = 60$ kHz) can be used in both FR1 and FR2. Each 1 ms subframe consists of 2^μ *slots*, indicating that

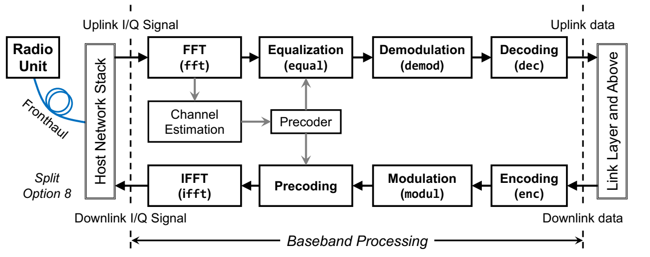


Figure 2: Baseband processing stages for the uplink and downlink with centralized processing under 5G split option 8.

a higher numerology leads to a much shorter slot duration as the SCS increases (e.g., 0.125 ms for FR2 with $\mu = 3$).

5G NR employs PHY based on the orthogonal frequency division multiplexing (OFDM) technique, where data are modulated onto and carried by orthogonal subcarriers in the frequency domain. To convert each OFDM symbol between the frequency domain and time domain (I/Q waveform), Fast Fourier Transform and its inverse (FFT/IFFT) are used. Since each slot contains a fixed number of 14 OFDM symbols, a higher numerology leads to a shorter symbol duration. The FFT size, denoted by N_{FFT}, is selected to cover the required number of resource blocks (RBs), each being 12 consecutive subcarriers. $F_s = N_{\text{FFT}} \times \Delta f$ calculates the PHY sampling rate. Table 1 lists the key PHY parameters in 5G NR FR2.

Baseband processing. Baseband processing refers to the signal processing operations responsible for preparing the data prior to transmission over a communication channel and decoding the received data. Fig. 2 shows the standard baseband processing stages for the uplink and downlink. 5G NR supports different functional split options, which determines how different RAN functions are distributed across the radio unit (RU), distributed unit (DU), and centralized unit (CU). In the context of vRAN, split option 8 [3], where all RAN functionalities are centralized, can largely benefit from centralized baseband processing and facilitate load balancing and sharing of computing resources across distributed RUs. However, it also places the highest demands on the fronthaul network with high capacity and strict latency requirements.

Consider *uplink* processing with split option 8, where the RU receives frames from the user and streams the I/Q samples to be processed by the edge server over the fronthaul. These I/Q samples are first converted to frequency domain symbols via FFT (fft), from which the pilot symbols are used to estimate the channel state information (csi). The CSI is used to calculate the precoder (precode) to equalize (equal) the data symbols. The equalized symbols are demodulated (demod), and a decoder (dec) performs forward error correction (FEC) before passing the frame to the medium access control (MAC) layer. 5G NR employs LDPC codes for both the uplink and downlink channels (i.e., PUSCH and PDSCH).

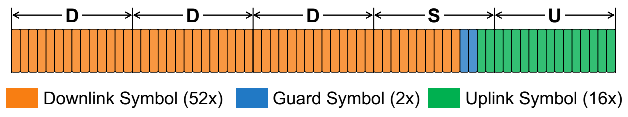


Figure 3: The 5-slot DDDSU frame schedule where the special slot (S) contains 10/2/2 downlink/guard/uplink symbols.

3.2 Low-Density Parity Check (LDPC) Code

In 5G NR, a transport block (TB) is a packet of data passed between the PHY and MAC layers. For LDPC encoding, a TB with size T (bits) is appended with a cyclic redundancy check (CRC) of T_{crc} bits (16 or 24 bits). Let $T' = T + T_{\text{crc}}$ denote the total size of the TB including the CRC. Prior to the LDPC coding process, a critical step is the selection of the base graph (BG), which determines the maximum code block (CB) size and whether CB segmentation is needed. 5G NR specifies two BG types [6], where BG1 is selected if (*i*) $T > 3{,}824$ and the code rate $R > 0.25$, or (*ii*) $T > 292$ and $R > 0.67$. If neither condition is met, BG2 is selected. In the interest of brevity, we elaborate on the scenario where CB segmentation is not involved, and details on scenarios involving CB segmentation can be found in [31]. For the selected BG, the number of information bit columns, denoted by K_b, is determined based on T and R, e.g., $K_b = 22$ for BG1. Following this, the lifting size, Z_c, is determined by selecting the smallest value from the set of supported lifting sizes that satisfies $K_b \cdot Z_c \geq T'$. The final CB size is then given by $K_{\text{cb}} = T' = K_b \cdot Z_c$. If needed, filler bits are added to match the required CB size and maintain the code rate. Finally, LDPC coding produces the sequence of coded bits for each CB.

For LDPC decoding, the same parameters are applied and the log-likelihood ratios (LLRs) produced from the demod stage via soft demodulation are used as input to the decoder, which represent the likelihood of a bit being "0" or "1". Belief propagation algorithms [10, 44] are then used to iteratively refine these likelihoods to converge on the most likely solution. For efficiency, early termination mechanism can be applied to pause the decoding process once a sufficient level of confidence is achieved. LDPC decoding is typically more time-consuming compared to encoding due to the iterative nature of the belief propagation. Hence, it can become the bottleneck in baseband processing when using a software decoder [13]. The hardware accelerator becomes favorable due to specialized parallel processing to speed decoding up.

3.3 PHY Latency Requirements in FR2

3GPP specifies a list of frame structures for 5G NR to support different applications with diverse requirements. In particular, 5G NR supports various *slot formats* [5] to define if each symbol within a specific slot is used for uplink (UL) or downlink (DL). These include the full UL/DL slot (U/D), where all 14 symbols are allocated for uplink/downlink transmission, and the special slot (S), which can include mixed UL/DL/guard symbols. We focus on the TDD frame structure formatted DDDSU as shown in Fig. 3, which includes 52/2/16 DL/guard/UL symbols over five slots. The DDDSU format has been widely adopted [20, 30, 32] due to its potential latency improvement as recommended by 3GPP [16].

In this paper, we refer to the 5-slot TDD schedule DDDSU as a frame and *set a PHY processing latency deadline of three slots (0.375 ms when $\mu = 3$) at the 99.9th percentile.* This allows two slots for the MAC layer to schedule the downlink NACK to the user in the case of a failed uplink transmission. Note that this represents a much tighter PHY processing deadline imposed by FR2, in contrast to the more relaxed deadlines considered by previous works focusing on FR1 settings, e.g., 1–5 ms in Agora [13] and 2.5 ms in Hydra [21].

4 SYSTEM DESIGN

In this section, we present the design of Savannah, which integrates a set of optimization strategies listed in Table 2 to address the stringent PHY processing requirements in FR2.

4.1 Challenges and Motivations

Existing real-time baseband processing systems for FR1 such as Agora [13] cannot be directly applied to FR2 due to the more stringent processing deadline (e.g., 375 μs if $\mu = 3$) or requires more computational resources. FR2 PHY presents its unique opportunities for efficient baseband processing.

Accelerating batch matrix operations. In an $M \times N$ MIMO system, the channel state across the N_{sc} subcarriers can be represented by a tensor $\mathbf{H} \in \mathbb{C}^{M \times N \times N_{\text{sc}}}$, obtained by the csi stage. Since subcarriers in each OFDM symbol can be independent, processing $\mathbf{H}$ becomes batch matrix operations. Massive MIMO systems in FR1 must operate on relatively large matrices, particularly in the csi, precode, and equal stages. FR2 systems, however, typically focus on smaller MIMO dimensions supporting up to 4 or 8 streams (e.g., 4×4 or 8×8 MIMO) [14, 22] with a much larger channel bandwidth and shorter slots to enhance data rates significantly. This fact motivates optimizing matrix operations for FR2 systems.

Existing works such as Agora [13] and Hydra [21] leverage *subcarrier-parallelism* to distribute the precode and equal tasks over CPU cores. However, this parallelism can cause inefficient batch matrix operations and memory access when processing small matrices. As shown in Fig. 1, Agora can meet the deadline in FR1 ($\mu = 0$) but not in FR2 ($\mu = 3$) under the same amount of fronthaul traffic. Careful profiling of Agora in FR2 settings (see §7) reveals that the bottlenecks are the precode, equal, and dec stages, where the former two involve heavy batch matrix operations. We propose vectorization with the SIMD instructions (e.g., AVX-512) provided

Table 2: Savannah focuses on the real-time baseband processing in FR2 with a more stringent latency processing deadline through optimization across various dimensions in comparison to Agora [13].

	Target Scenario	Batch Matrix Operation	Memory Layout	Arithmetic Library	LDPC Decoder	Model
Agora [13]	FR1 ($\mu = 0$)	Subcarrier-parallel	Subcarrier	Armadillo	FlexRAN [25]	Multi-core
Savannah-mc	FR2 ($\mu = 3$)	Vectorized	Antenna	AVX-512	FlexRAN [25]	Multi-core
Savannah-sc	FR2 ($\mu = 3$)	Vectorized	Antenna	AVX-512	ACC100 [49]	Single-core

by CPUs (§4.2) and the corresponding memory layout to optimize performance (§4.3) for FR2 MIMO dimensions.

Minimizing hardware resources. The optimization targeting the smaller MIMO dimension in FR2 not only allows Savannah to meet the processing deadline but also largely reduces the hardware resources required to achieve the same data rate compared to existing solutions. In particular, previous works in FR1 often target solving the computation of massive MIMO with a single or many multi-core servers [13, 21]. In addition, LDPC decoding contributes significantly to the baseband processing latency. To further enhance the system performance, especially in scenarios constrained by computing resources (e.g., a limited number of CPU cores), we propose a heterogeneous computing scheme that incorporates Intel's ACC100 hardware accelerator [49] for LDPC decoding (§4.4). We present first-of-its-kind comprehensive profiling and investigate the decoding capability and energy efficiency of this accelerator (§5). As shown in Fig. 1, this heterogeneous processing paradigm enables Savannah to support 2×2 MIMO over 100 MHz channel with a single CPU core. Our detailed profiling also reveals multi-dimensional tradeoffs between the hardware and software decoders with respect to the decoding throughput and energy efficiency.

Below, we elaborate on the optimization techniques and hardware/software components employed by Savannah.

4.2 Matrix Operation Optimization

We first focus on optimizing the equal and precode stages through vectorized matrix operations, leveraging the antenna-parallelism and seamless integration with SIMD instructions.

Optimizing batch matrix-vector multiplication (MVM) for equalization. For the k-th subcarrier, let $\mathbf{H}^{(k)}$ denote the CSI matrix and $\mathbf{P}^{(k)}$ denote the linear precoder, e.g., based on zero-forcing (ZF). Equalizing an OFDM symbol is a *batch MVM operation* $\vec{\mathbf{y}}^{(k)} = \mathbf{P}^{(k)} \times \vec{\mathbf{x}}^{(k)}$ with a batch size of N_{sc}. $\vec{\mathbf{x}}^{(k)}$ and $\vec{\mathbf{y}}^{(k)}$ denote the vector of I/Q samples on the k-th subcarrier before and after equalization, respectively, with a length determined by the MIMO dimension. Consider 2×2 MIMO with $[\mathbf{P}^{(k)}]_{k=1}^{N_{\text{sc}}} \in \mathbb{C}^{2\times2\times N_{\text{sc}}}$, equalizing an OFDM symbol with a subcarrier-parallel design requires executing N_{sc} times of MVM iteratively across the subcarriers, i.e.,

$$\begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \end{bmatrix} = \begin{bmatrix} p_{1,1}^{(k)} & p_{1,2}^{(k)} \\ p_{2,1}^{(k)} & p_{2,2}^{(k)} \end{bmatrix} \times \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix}, \ \forall k. \quad (1)$$

For small matrics utilized in 2×2 MIMO, each MVM involves heterogeneously four scalar multiplications and two scalar additions, unfriendly to processors with SIMD capabilities.

Savannah *vectorizes* matrix operations by treating the elements across subcarriers as vectors and performing element-wise vector-to-vector operations, i.e., the Hadamard product '$\odot$'. In such a way, (1) can be translated into a vector form,

$$\overrightarrow{\mathbf{y}_1} = \overrightarrow{\mathbf{p}_{1,1}} \odot \overrightarrow{\mathbf{x}_1} + \overrightarrow{\mathbf{p}_{1,2}} \odot \overrightarrow{\mathbf{x}_2} \ \text{ and } \ \overrightarrow{\mathbf{y}_2} = \overrightarrow{\mathbf{p}_{2,1}} \odot \overrightarrow{\mathbf{x}_1} + \overrightarrow{\mathbf{p}_{2,2}} \odot \overrightarrow{\mathbf{x}_2}, \quad (2)$$

where the addition/multiplication operations are performed on vectors of length N_{sc}. This vector-based computation can exploit data parallelism with SIMD instructions such as Intel's AVX-512 [26]. Since AVX-512 only supports arithmetic on real floating point numbers, we implement the complex-valued number counterparts with a combination of AVX-512 intrinsics [26], including bit operations. Via the vectorized batch MVM given by (2), Savannah reduces the equalization time in a 2×2 MIMO, 100 MHz, $\mu = 3$ link by 22.3× from Agora [13], as shown in Fig. 16. Although our discussions focus on a precoder matrix $\mathbf{P} \in \mathbb{C}^{2\times2}$, the proposed method can be easily extended to precoder matrices in arbitrary sizes.

Optimizing batch matrix inversion for precoding. Batch matrix inversion is the core computation for linear precoding, e.g., based on ZF. Similar to optimizing batch MVM for equal, Savannah optimizes precode through vectorizing batch matrix inversion. Consider a square channel matrix $\mathbf{H}^{(k)}$, the ZF precoder can be calculated by $\mathbf{P}^{(k)} = c_0^{(k)} \cdot \left(\mathbf{H}^{(k)}\right)^{-1}$, where $c_0^{(k)}$ is a constant factor for calibration. For 2×2 MIMO, inverting $\mathbf{H}^{(k)} \in \mathbb{C}^{2\times2}$ can be intuitively done following

$$\left(\mathbf{H}^{(k)}\right)^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}. \quad (3)$$

This involves operations such as multiplication and addition, checking non-zero determinants, and finding reciprocal, which Savannah processes using similar vectorization as that applied for equal, illustrated above. The same method can be applied to obtain the inverse of larger $N \times N$ matrices; however, the complexity can grow $O(N^3)$ (e.g., using the adjugate matrix). As we will show in §6, the vectorized precoding for SISO, 2×2 MIMO, and 4×4 MIMO can achieve

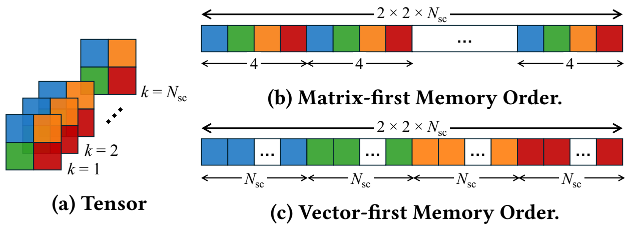

$2 \times 2 \times N_{sc}$
4
4
4
$k = N_{sc}$
$k = 2$
$k = 1$
(a) Tensor
(b) Matrix-first Memory Order.
(c) Vector-first Memory Order.
N_{sc}

Figure 4: The illustration of (a) batched 2×2 matrices as a 3D tensor, where the z dimension is the subcarriers. (b) and (c) are how the tensor can be mapped to the 2D memory space.

a speedup of 267×, 122×, and 34×, respectively, compared to the loop-based precoder calculation employed by Agora. Calculating the linear precoder for a non-square channel matrix requires the computation of its pseudo-inverse, which can similarly be adapted to the SIMD-friendly vector format.

4.3 Efficient Memory Access

The performance gain achieved by Savannah stems from the joint contribution of vectorized computation and vector memory alignment. We now discuss Savannah's memory layout and its compatibility across the DSP stages.

Memory layout. Memory access patterns can significantly impact an application's execution time and are determined by multiple factors, such as the data structure and memory layout. The performance gain of vectorized matrix operations will be limited if the memory access pattern is suboptimal. Fig. 4(a) illustrates a batch of 2×2 matrices, whereas Figs. 4(b) and (c) show two memory arrangements. In a loop-based batch matrix operation, arranging data matrix-by-matrix is appropriate, and Fig. 4(b) shows how to place matrices in a column-major manner. However, this per-matrix layout is inefficient for the proposed vectorized matrix operations, as the SIMD processor needs to fetch each element in a vector with an offset, which is 4 in the case of 2×2 MIMO. Fetching across discontinuous memory space can waste bandwidth on the memory bus, incur high cache miss rates, and induce long memory access time. Savannah adopts the memory layout as shown in Fig. 4(c), which aligns the best with the proposed vectorized matrix by guaranteeing continuous memory fetch.

Parallelism paradigm. In addition to efficient fetching, the vector memory allocation can significantly reduce unnecessary data rearrangement across the DSP stages shown in Fig. 2. Previous research [13, 21] often describes the parallel opportunities in each DSP stage as the antenna-, subcarrier-, or user-parallelism, depending on the dimension of independence. Techniques such as block fusion [13] and delayed shuffling [21] were used to reduce the memory movement across the DSP stages with the same parallelism paradigm to guarantee the proximity of the data to the processor (i.e., CPU cores) within the memory hierarchy for efficient access.

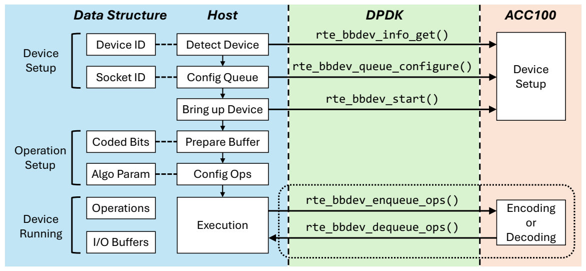


Figure 5: Workflow for configuring and controlling Intel's ACC100 accelerator via the DPDK driver.

However, data rearrangements are still required for a DSP stage to pass the operands to another stage with a different parallelism paradigm. For example, fft is an antenna-parallel stage since one FFT operation is performed for the signals received by each antenna. For each received uplink symbol, fft generates a vector with dimension N_{FFT} corresponding to the frequency domain representation of the received signal. To pass these vectors as matrices to the downstream csi stage that follows subcarrier-parallelism, Agora must distribute the vector elements to an interleaving style, as shown in Fig. 4(b). Interestingly, it can be easily observed that the continuous memory allocation shown in Fig. 4(c) naturally fits the output of fft and, as a result, data rearrangement can be eliminated. Similarly, csi carrying out matrix inversion can keep the output memory layout in the vector form, enabling more efficient data passing to equal stage. Then, passing the equalized symbols from equal to demod can fall back to block fusion [13] since they are both subcarrier-parallel.

4.4 Hardware LDPC Decoder

Savannah integrates a hardware LDPC decoder based on Intel's ACC100 accelerator [49] to enhance baseband processing speed, particularly when a single CPU core fails to meet the latency requirements in FR2. As a baseband device, the ACC100 accelerator provides a polling-mode driver (PMD) interfacing with the user space program via DPDK.

Accessing the ACC100 accelerator via DPDK. The driver functions of the ACC100 accelerator belong to the subset of `rte_bbdev`, standing for the baseband device in the real-time environment. While the ACC100 accelerator supports multiple modes, we select LDPC decoding specifically as an example. Fig. 5 shows the workflow to initiate ACC100 via DPDK function calls, the corresponding operands, and actions by the host computer/server. In particular, the device setup involves detecting the ACC100 accelerator and initializing the communication sockets. The user program can use device ID to select the device; DPDK manages a ring buffer of the operations with the declared and initialized queues, which can be accessed with socket ID. Before execution, the user must prepare the input/output buffer and specify the

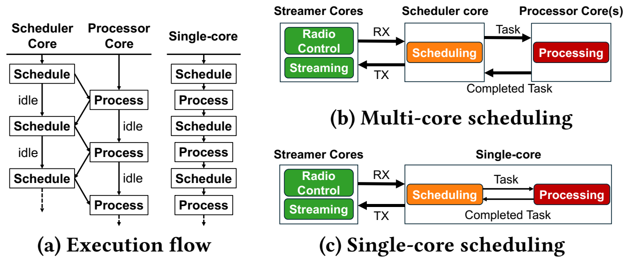


Figure 6: The execution flow and core allocation of multi-core and single-core scheduling.

desired operation. In the case of LDPC decoding, the input buffer contains the bits to be decoded, and one operation describes the buffer pointer and LDPC parameters, such as the MCS and lifting size (Z_c). The execution of the decoding tasks is triggered by enqueuing the pre-determined operations into the queue via DPDK. The host then actively polls the status using the dequeue request. A successful dequeue marks the completion of LDPC decoding for one CB.

Integration with Savannah. At run-time, each OFDM symbol after demod (i.e., the LLR outputs) is reorganized following the DPDK-BBDEV specifications, and is then transferred to the ACC100 device. To best utilize the built-in parallelism of the hardware accelerator, each demodulated OFDM symbol is immediately dispatched to the ACC100 device after the demod task using the `rte_bbdev_enqueue_ops()` function. Then, upon the completion of all software tasks for each frame, the program initiates a bulk dequeue operation using the `rte_bbdev_dequeue_ops()` function and produces all decoded symbols for a specific frame. Such a design ensures a continuous data flow and allows the subsequent software tasks (e.g., fft for the next OFDM symbol) to execute *in parallel* with the ACC100 accelerator decoding tasks. As a result, this parallel enqueue-dequeue strategy significantly enhances the decoding and overall throughput from the sequential counterpart, where the program awaits the decoding completion of the current OFDM symbol before proceeding to the next software task.

4.5 Minimizing Computational Resources

The optimization techniques presented in §4.2 and §4.3 can effectively reduce the number of CPU cores required in a multi-thread system. Given the decoding performance gain provided by the ACC100 accelerator, Savannah reveals the potential to use a single CPU core augmented with the hardware decoder to serve the intensive baseband processing workloads in FR2. To better analyze the contribution of each proposed design, we present two variants of Savannah:

- Savannah-mc uses multiple CPU cores and the software LDPC decoder based on FlexRAN. We refer to Savannah-Nc as Savannah-mc using N CPU cores, e.g., Savannah-4c uses 4 CPU cores that are shared for all processing tasks, including LDPC decoding;
- Savannah-sc integrates the scheduler and DSP processors in a single core with an ACC100 accelerator for decoding.

Both variants adopt the optimized matrix operations.

Fig. 6(a) (*left*) shows the workflow of a multi-core processing system (e.g., Agora and Savannah-mc), where the scheduler core assigns tasks to the processor core(s), and then busy waiting for the task completion to schedule the next task. However, the idle time appears when the load is imbalanced, especially when the number of processor cores is small. In the case of a single processor core, the scheduler and processor cores will idle half of the time, waiting for each other to complete execution. Under this condition, all scheduling and processing tasks can be merged into a single execution flow to improve core utilization and reduce inter-core communication, as shown in Fig. 6(a) (*right*).

Figs. 6(b) and 6(c) show the core allocation under the multi-core and single-core scheduling scheme, respectively. The inter-core communication is marked in thick arrows, while the intra-core counterpart is marked in thin arrows. The streamer cores are dedicated and can always be busy to keep up with the high volume of fronthaul traffic. On the other hand, the scheduler and processor cores can be merged in some cases to avoid inter-core communications. Inter-core communication must be handled with synchronization, such as lock-free concurrent queues, which incur overhead when the individual processing core is unnecessary. In addition to the software optimization, Savannah can process all the DSP stages in one core by offloading the LDPC decoding to dedicated hardware. This leads to the design of Savannah-sc following the scheduling strategy shown in Fig. 6(c).

5 UNDERSTANDING THE PERFORMANCE OF INTEL'S ACC100 ACCELERATOR

In this section, we detail the characterization of Intel's ACC100 accelerator to understand its LDPC decoding performance.[2] We use Silicom's Lisbon P2 ACC100 FEC accelerator server adapter [49] that integrates an ACC100 eASIC. To the best of our knowledge, this is the first comprehensive performance benchmark for the ACC100 accelerator in terms of decoding throughput and energy efficiency.

5.1 Decoding Throughput Characterization

Decoding unit test setup. We develop a unit test program based on the DPDK's `test-bbdev` benchmarking tool [55], which is widely used in the bbdev API for FEC processing in vRANs. We consider the setting where each OFDM symbol

[2]We focus on LDPC decoding for uplink processing in this work. The LDPC encoding performance can be characterized using a similar method.

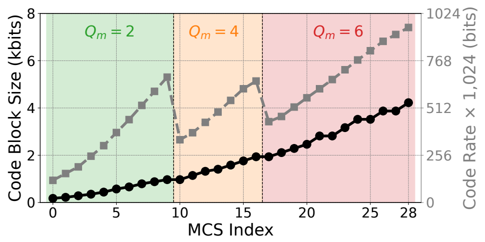


Figure 7: CB size (*solid black line*) and code rate (*dashed gray line*) corresponding to each MCS index (MCS0–MCS28).

carries at most one CB, i.e., with no CB segmentation (see §3.2), and expand the test-bbdev tool to support the 29 MCS (MCS0–MCS28) for the 5G NR PUSCH/PDSCH as specified in [4, Table 5.1.3.1-1]. For each MCS with modulation order, Q_m, and code rate, R, we derive the LDPC parameters using the unit test for Intel FlexRAN. Fig. 7 shows the mapping between the MCS index and its associated CB size and code rate. Subsequently, the test vectors for each MCS are prepared. Once all the assigned decoding tasks are completed, the unit test records the time duration, calculates the throughput, and validates the block error rate (BLER) upon the ground truth. One key aspect of utilizing the ACC100 accelerator is determining the number of CBs that can be "queued" into the hardware for concurrent processing. We refer to this as the *level of parallelism*, defined by NUM_CB in the unit test, where multiple CBs can be enqueued without the need to wait for the completion of the dequeue operation of the previous CBs. This unit test design also aligns with how the ACC100 device is integrated into Savannah (see §4.4).

In addition, we develop a unit test for the software LDPC decoder based on Intel's FlexRAN SDK [25], which can be configured with identical parameters as used by the unit test for the ACC100 accelerator. It utilizes OpenMP (Open Multi-Processing) version 4.5 [36] for execution across multiple CPU cores, which facilitates the comparison of the performance tradeoffs between hardware and software decoders across varied computing resources.

Performance metrics. Upon completing the unit test and validating the output BLER, we analyze the *decoding throughput* (dec_tput) across MCS and LDPC parameters. The decoding throughput is defined as the number of *uncoded* information bits that can be decoded per second, i.e.,

$$\texttt{dec\_tput} = (K_{\text{cb}} \times \texttt{NUM\_CB}) \;/\; \texttt{dec\_time}, \tag{4}$$

where the *decoding time* (dec_time) denotes the time duration required for the LDPC decoder to complete the decoding for a total number of NUM_CB CBs. We use the Read Time-Stamp Counter (RDTSC) instruction for precise time measurements.

Results and comparison to the software decoder. We first evaluate the LDPC decoding throughput of the ACC100 accelerator across varying NUM_CB and CB size values, K_{cb}. Note that there is a one-to-one mapping between CB size

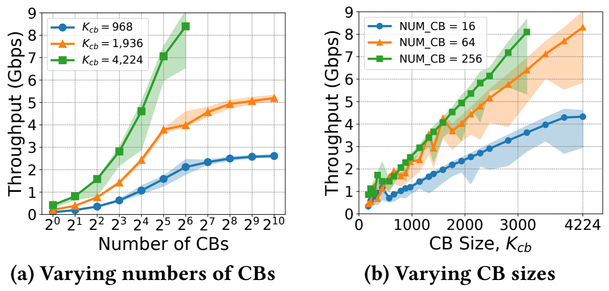
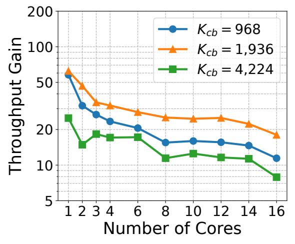


Figure 8: LDPC decoding throughput profiling for the Intel ACC100 eASIC within Silicom's Lisbon-2 FEC Accelerator.

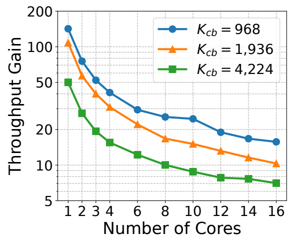
and MCS (see Fig. 7). For each configuration, we execute the unit test 5,000 times and collect the detailed decoding throughput statistics. Fig. 8(a) presents the average LDPC decoding throughput for the ACC100 accelerator when processing different numbers of CBs in parallel, with MCS10 (K_{cb} = 968), MCS17 (K_{cb} = 1,936), and MCS28 (K_{cb} = 4,224). We empirically determine the maximum number of CBs to be 2^{10}. Overall, the decoding throughput improves with increased parallelism levels indicated by NUM_CB. With MCS28 and NUM_CB = 2^6, the ACC100 accelerator achieves the peak decoding throughput of 8.30 Gbps, which aligns with the hardware specifications [49]. For lower MCS such as MCS17 and MCS10, the it achieves a maximum decoding throughput of 5.20 Gbps and 2.61 Gbps, respectively, with NUM_CB = 2^{10}.

Fig. 8(b) shows the average LDPC decoding throughput for the ACC100 accelerator with different CB sizes and parallelism levels, NUM_CB $\in \{2^4, 2^6, 2^8\}$. In general, the decoding throughput increases with larger CB sizes and, as a result, the number of CBs to be processed in parallel by the ACC100 accelerator needs to be carefully selected for optimal performance. This is particularly crucial for higher MCS that involve larger CB sizes, for which a threshold on NUM_CB exists that will impact the decoding performance.

Fig. 9 shows the average decoding throughput gain for the ACC100 accelerator compared to the software decoder using different numbers of CPU cores. Note that since the hardware and software decoders process the same number of CBs in the unit tests, the throughput gain is equivalent to the speedup in decoding latency for the ACC100 accelerator. Here, the selection of 16 and 64 CBs matches with the full traffic load for SISO and 4×4 MIMO in the DDDSU frame structure (see §3.3 and Fig. 3). For example, under the moderate MCS17, the ACC100 accelerator achieves a decoding throughput gain of 31.8× and 30.9× with 16 and 64 CBs, respectively, compared to the software decoder with four cores. These gain values degrade to 25.2× and 16.7× when the software decoder uses doubled computing resources, i.e., eight cores. While the software decoder is able to achieve high decoding throughput, it usually requires intensive computational resources and can lead to poor energy efficiency.

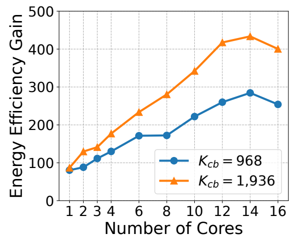


(a) NUM_CB = 16 CBs

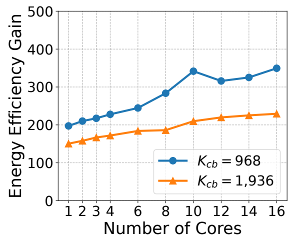


(b) NUM_CB = 64 CBs

Figure 9: LDPC decoding throughput gain for the ACC100 accelerator compared to the software decoder (FlexRAN).

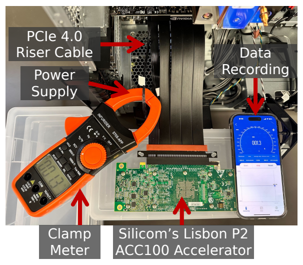


(a) NUM_CB = 16 CBs



(b) NUM_CB = 64 CBs

Figure 11: Energy efficiency gain achieved by the ACC100 accelerator compared to the software decoder (FlexRAN).



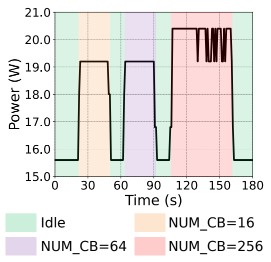


Figure 10: (*Left*) Setup for measuring the power consumption of Silicom's Lisbon P2 ACC100 accelerator; (*Right*) Example power consumption profiles during parallel decoding.

5.2 Energy Efficiency Characterization

Power profiling setup. We utilize two approaches to profile the power consumption of the hardware and software LDPC decoders. Fig. 10 depicts the setup for measuring the power consumption of Silicom's Lisbon P2 ACC100 accelerator server adapter. In particular, we interface the ACC100 device with the host through a PCIe 4.0 riser cable, and isolate the wires associated with the 12 V power pins from the riser cable [45]. This setup allows us to apply a digital power clamp meter (Infurider 570S-APP), which records the current draw across the isolated wires. The measurement accuracy is validated using an NVIDIA GTX1070 GPU with the `nvidia-smi` tool. Fig. 10 also shows an example power consumption profile for the ACC100 accelerator. It can be seen that the ACC100 device consumes 20.4 W of power (4.8 W in addition to the idle power of 15.6 W) when achieving an LDPC decoding throughput of 4.93 Gbps, with $K_{\text{cb}} = 1{,}936$ and NUM_CB = 256. To accurately measure the CPU power consumption for the software LDCP decoder, we utilize Intel's performance counter monitor (PCM) tool [1] and allocate specific CPU cores exclusively for running the FlexRAN unit test on one socket. We also measure the system's idle power to obtain the power consumption for decoding.

Performance metrics. We consider the *energy efficiency* of the LDPC decoder, which is defined as the ratio of its power consumption (W) to its decoding throughput (Gbps) and measured with the unit of nJ/bit. We exclude the idle power consumption of the eASIC and CPU hardware when calculating the energy efficiency of the LDPC decoder.

Results and comparison to the software decoder. Fig. 11 presents the average energy efficiency gain achieved by the ACC100 accelerator compared to the software decoder using different numbers of CPU cores. Our profiling results reveal the superior energy efficiency of the hardware decoder. In particular, with 16 CBs, the ACC100 accelerator achieves an energy efficiency of 3.18 nJ/bit and 1.54 nJ/bit for MCS10 and MCS17, respectively. With a higher parallelism level of 64 CBs, the energy efficiency numbers are improved to 1.33 nJ/bit and 0.90 nJ/bit for MCS10 and MCS17, respectively.

On the other hand, the software decoder underperforms in energy efficiency, primarily due to (*i*) the lower decoding throughput and (*ii*) CPU power consumption increases linearly to the number of cores. In particular, the software decoder peaks its energy efficiency when only a single CPU core is used for decoding. Under this condition, the ACC100 accelerator achieves an energy efficiency gain of 81–87× and 151–197× with 16 and 64 CBs, respectively. The ACC100 accelerator can achieve an energy efficiency gain of two orders of magnitude when more CPU cores are used for decoding. Such an improved energy efficiency, therefore, offsets the upfront capital cost of the hardware. As shown in Figs. 1(a) and 11, the energy efficiency of five CPU cores is 200× worse than that of the ACC100 accelerator to achieve comparable decoding performance. Moreover, an ACC100 accelerator can be configured as multiple virtual instances and support multiple cells simultaneously through DPDK. For example, a 2×2 100 MHz link with MCS10 and full uplink traffic load requires a decoding throughput of 0.38 Gbps, which is only 5% of the peak decoding throughput of 8 Gbps [9].

6 IMPLEMENTATION

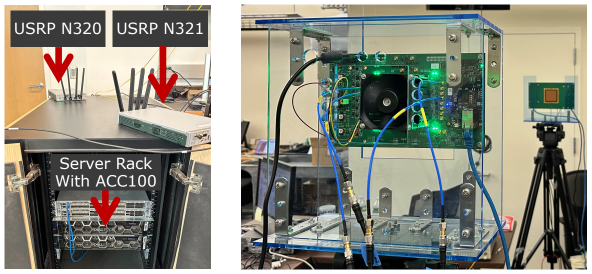
We implement Savannah in 10K lines of source code in C++ on top of the Agora codebase [13], including 5K lines of functional code and 5K lines of testing code. Specifically, Savannah optimizes matrix operations following vectorized computation and memory arrangement. Savannah-mc applies Intel's FlexRAN SDK for LDPC decoding and runs in a multi-core model, whereas Savannah-sc incorporates Intel's ACC100 accelerator and runs with a single CPU core.

Matrix operations and memory layout. Savannah features vectorized matrix operations and SIMD-friendly memory layout for software-based processing. Savannah leverages continuous memory layout with antenna-continuous data arrangement and uses AVX-512 provided by the Intel Xeon CPUs. We implement required complex-number arithmetic with real floating point numbers and/or bit operations provided by Intel intrinsics [26]. Savannah accesses the memory with raw pointers and manages the resource using C++ standard libraries, where the memory layout is written manually and handled case-by-case with respect to the matrix dimensions. Note that Agora [13] leverages Armadillo [48, 51], a C++ library for linear algebra & scientific computing, to handle matrix operations and memory layout.

Server configuration. We perform experiments on a Dell PowerEdge R750 server, equipped with a 56-core Intel Xeon Gold 6348 CPU @2.6 GHz and runs Ubuntu OS (20.04.6 LTS). To guarantee real-time processing, we enable core isolation and mark specific cores dedicated for Savannah. The server runs DPDK 21.11.2 to interface with Silicom's Lisbon P2 ACC100 card. To optimize performance and minimize OS context switches, we configure the CPU to operate in performance mode and execute Savannah as a real-time process with the highest scheduling priority. We also disable hyper-threading and enable thread affinity to mitigate unpredictable hazards in high-throughput applications due to shared resources between logical cores. Similar to Agora, two additional threads are dedicated to handling the network I/O, i.e., streaming I/Q samples between the server and RU.

ACC100 accelerator binding. We integrate Silicom's Lisbon P2 ACC100 accelerator with Savannah using DPDK's `igb_uio` driver, which is the userspace I/O (UIO) driver for 5G FEC device and facilitates direct access to the ACC100 accelerator from userspace, bypassing Linux kernel's networking stack. We employ DPDK's `dpdk-devbind` tool to bind the ACC100 accelerator to the `igb_uio` driver and configure two virtual functions (VFs) (`max_vfs = 2`), and set up the I/O memory management unit (IOMMU) for managing the memory address visible to the VFs. To configure the ACC100 device with specific FEC parameters, we use Intel's physical function bbdev configuration application (`pf_bb_config`).

Emulated and real RUs. We use both emulated and real RUs, as shown in Fig. 12, to evaluate the performance of Savannah. For emulated RUs, we use one edge server that integrates an ACC100 accelerator, whose configurations are described above, to emulate RU traffic with different PHY parameters and MIMO dimensions. We run Savannah where the RU and user are allocated on two separate NUMA nodes of the same server. For real RUs support over-the-air (OTA) transmissions, we use a local testbed in a **Lab** environment, which consists of two USRP N32x software-defined radios (SDRs). The radios are connected to the same server used for the emulated RU and controlled by UHD [34]. Due to the lack of mmWave front ends, we set up a 3 GHz wireless link between the radios, while employing PHY parameters that reflect FR2 settings with SISO 100 MHz configuration.



(a) USRP SDRs in *Lab* **(b) 28 GHz SDRs in *COSMOS***

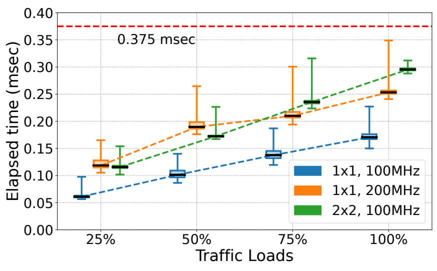
Figure 12: OTA evaluation setup using USRP SDRs in Lab (sub-6 GHz front ends) and 28 GHz SDRs in COSMOS (mmWvae front ends), both with FR2 PHY configurations.

To further evaluate Savannah with real RUs equipped with FR2 front ends, we leverage the open-access PAWR **COSMOS** testbed [40, 43]. In particular, we conduct experiments using the COSMOS Sandbox 2 (sb2), which includes two IBM 28 GHz phased array antenna module (PAAM) boards (see Fig. 12(b)) and two USRP N310 SDRs. Both the 28 GHz front ends and USRP SDRs are connected to and controlled by a Dell PowerEdge R740 server equipped with a 48-core Intel Xeon Gold 6226 CPU operating @2.7 GHz. We apply similar server configurations as described above to the COSMOS server. Due to the limited bandwidth supported by the USRP N310 and lack of the ACC100 accelerator in the COSMOS server, we implement Savannah-mc with SISO 100 MHz configuration and conduct OTA experiments over a 28 GHz link.

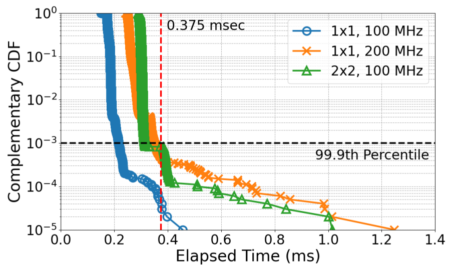
7 EVALUATION

We now present the evaluation results for Savannah with the PHY configurations listed in Table 1. For each configuration, the achievable data rate is a function of channel bandwidth, MCS, and MIMO dimension. We consider MIMO dimensions of 1×1 (SISO), 2×2, and 4×4, and the MCS listed in Table 3, from which the corresponding data rate can be obtained. For example, with MCS17 and 100 MHz bandwidth, the data rates for SISO, 2×2 MIMO, and 4×4 MIMO configurations are 243.9 Mbps, 487.8 Mbps, and 975.7 Mbps, respectively.

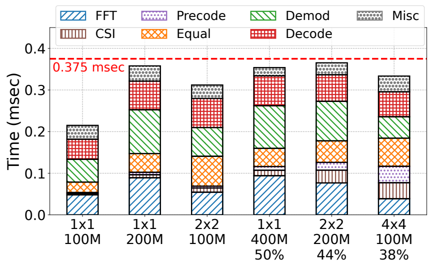
As described in §3.3, we focus on the baseband processing of uplink traffic and refer to *full (100%) traffic load* with 16 uplink symbols corresponding to the `DDDSU` frame schedule (see Fig. 3). The traffic load can also be varied by adjusting the number of uplink symbols in the frame. We set a PHY processing latency deadline of 0.375 ms, or three slots for FR2



(a) Elapsed time distribution



(b) Complementary CDF of elapsed time



(c) 99.9th percentile elapsed time breakdown

Figure 13: Performace evaluation for Savannah-sc over FR2 configurations. (a) Each box indicates the inter-quartile range, and the top/bottom whisker indicates the 99.9th percentile/minimum latency. (b) Complementary CDF under full traffic load. (c) Breakdown of the 99.9th percentile elapsed time, where partially supported traffic load is marked in %.

Table 3: Considered MCS used for the experimental evaluation of Savannah and the corresponding spectral efficiency.

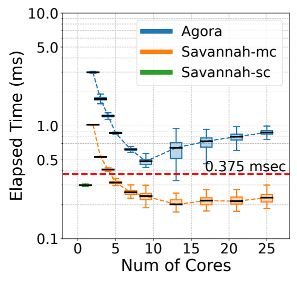
	Modulation Order	Code Rate	Spectral Efficiency
MCS10	16QAM ($Q_m = 4$)	340/1,024	1.328 bit/s/Hz
MCS17	64QAM ($Q_m = 6$)	438/1,024	2.566 bit/s/Hz
MCS22	64QAM ($Q_m = 6$)	666/1,024	3.902 bit/s/Hz
MCS28	64QAM ($Q_m = 6$)	948/1,024	5.555 bit/s/Hz

$\mu = 3$, at the 99.9th percentile for Savannah (see §3.3). We focus on the **elapsed time**, which is the overall PHY processing latency from the reception of each packet to the completion of LDPC decoding. We also consider the **CPU time**, which is the time consumed across all DSP tasks, excluding the scheduling and inter-core communication overhead.

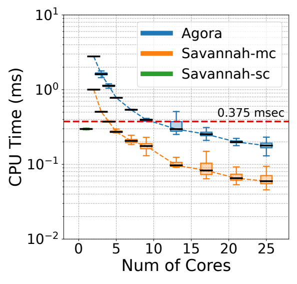
7.1 End-to-End Evaluation

Feasibility of Savannah-sc. We first evaluate the capability of Savannah-sc with different FR2 PHY configurations and Fig. 13 (a) shows the median and 99.9th percentile of the elapsed time achieved by Savannah-sc with varying uplink traffic loads. For each configuration, we run Savannah-sc for 100K frames and collect the baseband processing time. The results show that using a single CPU core and the ACC100 accelerator, Savannah-sc supports real-time processing for SISO 200 MHz and 2×2 100 MHz under *full uplink traffic*, both corresponding to a data rate of 487 Mbps with MCS17. With the same configuration and MCS28, Savannah-sc can achieve a data rate of 1.06 Gbps. Fig. 13 (b) shows the complementary cumulative distribution function (CDF) of the elapsed time achieved by Savannah-sc, measured across 100K frames. Under full uplink traffic, Savannah-sc achieves a 99.9th percentile elapsed time of 349 µs and 312 µs for SISO 200 MHz and 2×2 100 MHz, respectively. We also run Savannah-sc with the 2×2 100 MHz and different MCS listed in Table 3, and observe similar latency with less than 11% variation.

Fig. 13 (c) shows the breakdown of 99.9th percentile elapsed time achieved by Savannah-sc with different FR2 channel



(a) Elapsed Time



(b) CPU Time

Figure 14: Elapsed time and CPU time distributions for Savannah in 2×2 100 MHz configuration with full traffic load.

bandwidth and MIMO dimensions. With the support of the ACC100 accelerator, the most time-consuming tasks for software processing include fft, demod, and equal. We also consider more computationally intensive PHY configurations with increased bandwidth and MIMO dimensions. These include SISO 400 MHz, 2×2 200 MHz, and 4×4 100 MHz, where the maximum uplink traffic loads that can be supported by Savannah-sc are 50%, 43.75%, and 37.5%, respectively.

ACC100 accelerator utilization and cost. One instance of Savannah-sc can support a 2×2 MIMO link with 100 MHz bandwidth using one CPU core and an ACC100 accelerator, where the latter is not utilized at its full capacity when serving a single cell (see §5.2). The under-utilization of the ACC100 accelerator reveals an opportunity for resource sharing in a multi-cell system. Theoretically, one ACC100 accelerator and 20 CPU cores can support 20 instances of Savannah-sc, each running 2×2 MIMO and 100 MHz bandwidth, whereas without the ACC100 accelerator requires 100 CPU cores. Thus, the operating cost of an ACC100 accelerator will be amortized when shared across multiple cells.

Comparison to Agora [13] Fig. 14 presents the distributions of the elapsed time and CPU time for Savannah-mc and Agora, across different numbers of CPU cores with the 2×2 100 MHz configuration and full traffic load. Note that the 99.9th percentile elapsed time for Savannah-sc under

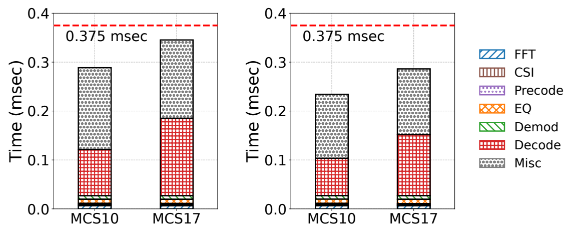
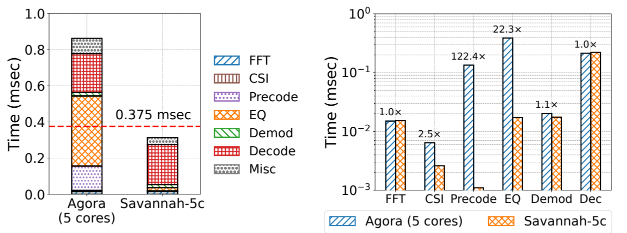


Figure 15: Breakdown of the 99.9th percentile (*left*) and average (*right*) elapsed time in 4×4 100 MHz for Savannah-21c.



(a) Oveall breakdown **(b) Breakdown for each DSP stage**

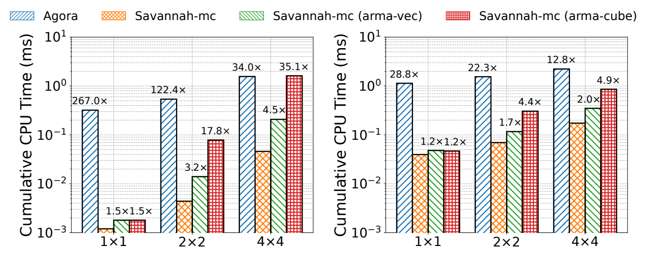
Figure 16: Breakdown of the average processing time in 2×2 100 MHz for Savannah-5c and Agora, both using 5 CPU cores.

the same configuration is 0.312 ms. Specifically, to meet the processing deadline, Savannah-mc requires only five CPU cores, whereas Agora fails to support the desired FR2 configuration, demonstrating the performance gain achieved by the optimization techniques employed by Savannah. It can also be observed that although the CPU time decreases with an increased number of cores for both Savannah-mc and Agora, the elapsed time does not further reduce due the overhead in scheduling and inter-core communication.

Furthermore, we consider a more computationally intensive configuration of 4×4 100 MHz, where Savannah-mc requires 21 cores to meet the 3-slot deadline, achieving a data rate of 975 Mbps with MCS17. Fig. 15 shows the breakdown of the 99.9th percentile elapsed time, which is dominated by the software LDPC decoder and increased overhead due to inter-core communication. Note that Agora fails to support this configuration even with 32 cores. While the current implementation of Savannah-mc supports only the software LDPC decoder, we believe that the required number of cores can be reduced by 4× with the ACC100 accelerator based on the profiling results with `NUM_CB` = 64 (see §5.1).

7.2 Effectiveness of Optimization

Vectorized matrix operations and memory layout. To evaluate the effectiveness of the proposed matrix operation and memory layout scheme, we present the latency breakdown of Savannah-5c and Agora processing a 2×2 100 MHz link. Note that the minimum number of cores required for Savannah-mc to meet the 3-slot deadline is 5, while Agora's processing latency exceeds the deadline given the same hardware resources, as shown in Fig. 16(a). On the other hand, Fig. 16(b) shows the performance gain achieved by Savannah across the DSP stages. It can be seen that the precode time is improved by 122.4× with vectorized matrix inversion, and equal time is improved by 22.3× with vectorized MVM. Note that each DSP stage has the antenna-, subcarrier-, or user-parallelism that can be exploited [13, Table. 2]. However, the transition between parallelism often requires data re-ordering [21]. Savannah's vectorization in precode and equal can be viewed as continuing the antenna-parallelism from the fft stage. Thus, csi time is reduced by 2.5× even without any modification on the fft stage. This is because the vector memory layout fits the csi output format and thus eliminates additional memory access for data rearrangement.



(a) Precoding (precode) **(b) Equalization (equal)**

Figure 17: Average cumulative matrix operation time across CPU cores with 100 MHz bandwidth. precode and equal involve matrix inversion and MVM operations, respectively.

Varying MIMO dimensions. We also evaluate the performance of the proposed antenna-parallel, AVX512-based matrix operations across different MIMO dimensions (SISO, 2×2, and 4×4). Specifically, Savannah-mc requires at least 4, 5, and 21 cores to meet the 3-slot processing deadline for SISO, 2×2 MIMO, and 4×4 MIMO with 100 MHz bandwidth, under full traffic load. Note that Agora cannot meet the deadline with the same number of cores. Fig. 17 demonstrates the average matrix operation time accumulated across the CPU cores for each MIMO configuration, which represents the overall computation time and complexity for each DSP stage. The results show that compared to Agora, Savannah achieves a CPU time reduction of 30–250× and 10–40× time reduction for precode and equal, respectively. It can also be seen from Fig. 17 that the proposed solution yields a more significant time reduction for smaller MIMO dimensions.

Overhead of generalized matrix operations. Note that Armadillo [48] provides an abstraction layer, including operator and data structure, for vectors, matrices, and 3D tensors (or "cube") with performance overhead, as shown in Fig. 17. Savannah-mc (arma-vec) vectorizes matrix operation and memory layout in the same way as Savannah-mc but uses Armadillo's vector APIs instead of AVX-512, showing that the overhead of the abstraction layer leads to at least 1.5×

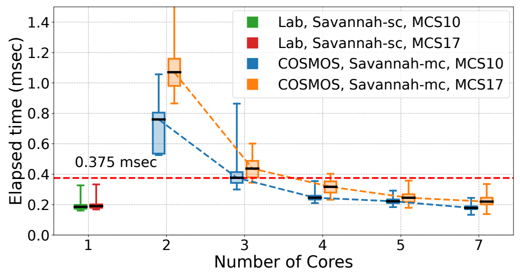


Figure 18: Elapsed time achieved by Savannah-sc and Savannah-mc with real RUs in the Lab setup and COSMOS testbed, respectively, under the SISO 100 MHz configuration.

precoder calculation and 1.2× equalization time. Savannah-mc (arma-cube) keeps Armadillo's "cube" memory layout and uses its "cube" APIs to perform vectorization. In SISO (1×1), Savannah-mc (arma-cube) reduces to Savannah-mc (arma-vec). However, as the MIMO size grows, Savannah-mc (arma-cube) suffers from mismatched computation and memory layout, diminishing the benefit of vectorization. Baseline Agora uses "cube" data structure/APIs without vectorization.

7.3 Over-the-Air Evaluation

We evaluate the performance of Savannah-sc and Savannah-mc with real RUs connected to a server using the **Lab** setup and **COSMOS** testbed, whose settings are described in §6. For each setup, a SISO 100 MHz link is established between two radios under full uplink traffic load, and we empirically set the link signal-to-noise ratio (SNR) to be 23.2 dB and 28.5 dB for MCS10 and MCS17, respectively. These SNR values ensure a zero BLER over the wireless link for the corresponding MCS. Fig. 18 shows the distribution of the elapsed time for Savannah-sc and Savannah-mc, where the number of CPU cores is varied for the latter setup. The experimental results show that Savannah-sc achieves similar performance compared to Savannah-mc using 5 CPU cores. Note that compared to Fig. 13 (a), the elapsed time achieved with real RUs exhibits a longer tail performance due to the potential jitter from the NIC that streams baseband signal between the RU and server. In addition, the 99.9th percentile elapsed time for Savannah-mc is 0.291 ms and 0.357 ms for MCS10 and MCS17, respectively, when using 5 cores. We acknowledge that UHD also provides DPDK support to reduce this jitter, and integrating DPDK for both streaming and baseband device control is left to future work.

8 LIMITATIONS

Scaling to massive MIMO in FR2. Savannah demonstrates that an FR2 system can achieve comparable data rates to an FR1 system when employing smaller MIMO dimensions and larger signal bandwidth, while requiring less intensive computing resources. However, spectrum and spatial efficiency are orthogonal, and the service providers may prioritize the data rates over cost. As illustrated in Fig. 17, the vectorization of matrix operations works better in a smaller MIMO dimension. Enabling massive MIMO in the FR2 band needs a scalable matrix operation algorithm. In addition, evaluation of Savannah in OTA experiments would require off-the-shelf FR2 front ends that support large MIMO dimensions.

Adaptive scheduling. Savannah-sc supports up to 4×4 MIMO with a single CPU core and ACC100 accelerator in FR2, and scaling to larger antenna counts requires more hardware resources. Integrating the ACC100 accelerator into a multi-core system requires a precise prediction of task execution time and an adaptive scheduler, since it requires the host CPU to poll the finish signal, and busy-wait polling leads to inefficiency. Further exploration into cross-layer design is necessary, particularly in co-optimizing the baseband processing and resource allocations in higher layers.

Tradeoffs between programmability and system performance. As mentioned in §7.2, existing arithmetic libraries can lead to overhead. However, programmability is essential for the rapid adoption and deployment of the system. Savannah provides a solution to accelerate the matrix operations but still requires the developers to be familiar with the techniques in §4.2 and AVX-512 instructions. A general code generation tool to "hide" the implementation details from the developer will increase the accessibility of Savannah.

9 CONCLUSION

We present Savannah, a real-time baseband processing framework targeting the FR2 signals in the context of 5G vRAN. Savannah leverages techniques such as vectorizing small-matrix operations, efficient memory access patterns, single-core processing, and incorporating ACC100 accelerator for LDPC decoding. Savannah achieves a data rate of 487 Mbps via minimal and heterogeneous computing resources in a 2×2, 100 MHz link. Future research directions include migrating low PHY to hardware, evaluating the ACC100 accelerator in a multi-core model to facilitate resource sharing, and evaluating Savannah with real FR2 MIMO frontends.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-1827923, CNS-2211944, AST-2232458, and an IBM Academic Award. This work was also supported in part by the Center for Ubiquitous Connectivity (CUbiC), sponsored by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program. We thank Arun Paidimarri, Bodhisatwa Sadhu, and Alberto Valdes-Garcia (IBM Research), Rahman Doost-Mohammady and Andrew Sedlmayr (Rice University), and Ivan Seskar (Rutgers University) for their contributions to this work.

REFERENCES

[1] 2024. Intel Performance Counter Monitor (Intel PCM). https://github.com/intel/pcm.

[2] 2024. Savannah: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources. https://github.com/functions-lab/Savannah.

[3] 3GPP. 2017. *Study on new radio access technology: Radio access architecture and interfaces.* Technical Specification (TS) 38.801. 3rd Generation Partnership Project (3GPP). Version 14.0.0.

[4] 3GPP. 2018. *5G; NR; Physical Layer Procedures for Data.* Technical Specification (TS) 38.214. 3rd Generation Partnership Project (3GPP). https://www.etsi.org/deliver/etsi_ts/138200_138299/138214/15.02.00_60/ts_138214v150200p.pdf Version 15.2.0.

[5] 3GPP. 2019. *5G; NR; Physical Layer Procedures for Control.* Technical Specification (TS) 38.213. 3rd Generation Partnership Project (3GPP). https://www.etsi.org/deliver/etsi_ts/138200_138299/138213/15.06.00_60/ts_138213v150600p.pdf Version 15.6.0.

[6] 3GPP. 2020. *5G; NR; Multiplexing and channel coding.* Technical Specification (TS) 38.212. 3rd Generation Partnership Project (3GPP). https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/16.02.00_60/ts_138212v160200p.pdf Version 16.2.0.

[7] 3GPP. 2022. *5G; NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.* Technical Specification (TS) 38.101-1. 3rd Generation Partnership Project (3GPP). https://www.etsi.org/deliver/etsi_ts/138100_138199/13810101/17.05.00_60/ts_13810101v170500p.pdf Version 17.5.0.

[8] 3GPP. 2022. *General Aspects for User Equipment (UE) Radio Frequency (RF) for NR.* Technical Specification (TS) 38.817-01. 3rd Generation Partnership Project (3GPP). Version 16.4.0.

[9] ADLINK. 2022. FEC Accelerator Based On Intel® vRAN Dedicated Accelerator ACC100. . Version 1.2.

[10] Jinghu Chen and Marc PC Fossorier. 2002. Near optimum universal belief propagation based decoding of low-density parity check codes. *IEEE Trans. Commun.* 50, 3 (2002), 406–414.

[11] Tingjun Chen, Prasanthi Maddala, Panagiotis Skrimponis, Jakub Kolodziejski, Abhishek Adhikari, Hang Hu, Zhihui Gao, Arun Paidimarri, Alberto Valdes-Garcia, Myung Lee, et al. 2023. Open-access millimeter-wave software-defined radios in the PAWR COSMOS testbed: Design, deployment, and experimentation. *Computer Networks* 234 (2023), 109922.

[12] Tingjun Chen, Jiakai Yu, Arthur Minakhmetov, Craig Gutterman, Michael Sherman, Shengxiang Zhu, Steven Santaniello, Aishik Biswas, Ivan Seskar, Gil Zussman, et al. 2022. A software-defined programmable testbed for beyond 5G optical-wireless experimentation at city-scale. *IEEE Network* 36, 2 (2022), 90–99.

[13] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. 2020. Agora: Real-time massive MIMO baseband processing in software. In *Proc. ACM CoNEXT'20.*

[14] Ryan M Dreifuerst and Robert W Heath. 2023. Massive MIMO in 5G: How beamforming, codebooks, and feedback enable larger arrays. *IEEE Commun. Mag.* 61, 12 (2023), 18–23.

[15] Jinfeng Du, Dmitry Chizhik, Reinaldo A Valenzuela, Rodolfo Feick, Guillermo Castro, Mauricio Rodriguez, Tingjun Chen, Manav Kohli, and Gil Zussman. 2020. Directional measurements in urban street canyons from macro rooftop sites at 28 GHz for 90% outdoor coverage. *IEEE Trans. Antennas Propag.* 69, 6 (2020), 3459–3469.

[16] Rostand AK Fezeu, Jason Carpenter, Claudio Fiandrino, Eman Ramadan, Wei Ye, Joerg Widmer, Feng Qian, and Zhi-Li Zhang. 2023. Mid-Band 5G: A measurement study in Europe and US. *arXiv preprint arXiv:2310.11000* (2023).

[17] Zhihui Gao, Ang Li, Yunfan Gao, Yu Wang, and Yiran Chen. 2021. Hermes: Decentralized dynamic spectrum access system for massive devices deployment in 5G. In *Proc. EWSN'21.*

[18] Zhihui Gao, Zhenzhou Qi, and Tingjun Chen. 2024. Mambas: Maneuvering Analog Multi-User Beamforming using an Array of Subarrays in mmWave Networks. In *Proc. ACM MobiCom'24.*

[19] Yasaman Ghasempour, Muhammad K Haider, Carlos Cordeiro, Dimitrios Koutsonikolas, and Edward Knightly. 2018. Multi-stream beam-training for mmWave MIMO networks. In *Proc. ACM MobiCom'18.*

[20] Global System for Mobile Communications. 2020. 5G TDD synchronisation guidelines and recommendations for the coexistence of TDD networks in the 3.5 GHz range. https://www.gsma.com/spectrum/wp-content/uploads/2020/04/3.5-GHz-5G-TDD-Synchronisation.pdf.

[21] Junzhi Gong, Anuj Kalia, and Minlan Yu. 2023. Scalable distributed massive {MIMO} baseband processing. In *Proc. USENIX NSDI'23).*

[22] Xiaoxiong Gu, Arun Paidimarri, Bodhisatwa Sadhu, Christian Baks, Stanislav Lukashov, Mark Yeck, Young Kwark, Tingjun Chen, Gil Zussman, Ivan Seskar, et al. 2021. Development of a compact 28-GHz software-defined phased array for a city-scale wireless research testbed. In *Proc. IEEE IMS'21.*

[23] Haitham Hassanieh, Omid Abari, Michael Rodriguez, Mohammed Abdelghany, Dina Katabi, and Piotr Indyk. 2018. Fast millimeter wave beam alignment. In *Proc. ACM SIGCOMM'18.*

[24] David Hunt, Kristen Angell, Zhenzhou Qi, Tingjun Chen, and Miroslav Pajic. 2024. MadRadar: A Black-Box physical layer attack framework on mmWave automotive FMCW radars. *Proc. NDSS'24* (2024).

[25] Intel Corporation. 2019. FlexRAN LTE and 5G NR FEC software development kit modules. https://www.intel.com/content/www/us/en/developer/articles/technical/flexran-lte-and-5g-nr-fec-software-development-kit-modules.html.

[26] Intel Corporation. 2023. Intel ® intrinsics guide. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

[27] Ish Kumar Jain, Raghav Subbaraman, Tejas Harekrishna Sadarahalli, Xiangwei Shao, Hou-Wei Lin, and Dinesh Bharadia. 2020. mMobile: Building a mmWave testbed to evaluate and address mobility effects. In *Proc ACM mmNets'20.*

[28] Suraj Jog, Jiaming Wang, Junfeng Guan, Thomas Moon, Haitham Hassanieh, and Romit Roy Choudhury. 2019. Many-to-many beam alignment in millimeter wave networks. In *Proc. USENIX NSDI'19.*

[29] Jesus O Lacruz, Rafael Ruiz Ortiz, and Joerg Widmer. 2021. A real-time experimentation platform for sub-6 GHz and millimeter-wave MIMO systems. In *Proc. ACM MobiSys'21.*

[30] Nikita Lazarev, Tao Ji, Anuj Kalia, Daehyeok Kim, Ilias Marinos, Francis Y Yan, Christina Delimitrou, Zhiru Zhang, and Aditya Akella. 2023. Resilient baseband processing in virtualized RANs with slingshot. In *Proc. ACM SIGCOMM'23.*

[31] Xingqin Lin and Namyoon Lee. 2021. *5G and Beyond: Fundamentals and Standards* (1st ed.). Springer, Cham, Switzerland.

[32] Biraja Mahapatra. 2022. Intel + radisys FlexRAN success stories. https://networkbuilders-cdn.s3.us-east-2.amazonaws.com/Intel_Radisys_FlexRAN_Success_Stories_Biraja_Prasad_Mahapatra.pdf.

[33] Jérémy Nadal and Amer Baghdadi. 2021. Parallel and flexible 5G LDPC decoder architecture targeting FPGA. *IEEE VLSI* 29, 6 (2021), 1141–1151.

[34] National Instruments Corp. 2024. UHD (USRP Hardware Driver). https://www.ettus.com/sdr-software/uhd-usrp-hardware-driver/.

[35] OpenAirInterface. 2023. openairinterface5G. https://gitlab.eurecom.fr/oai/openairinterface5g/blob/develop/doc/FEATURE_SET.md.

[36] OpenMP Architecture Review Board. 2015. OpenMP application program interface version 4.5. https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.

[37] Joan Palacios, Daniel Steinmetzer, Adrian Loch, Matthias Hollick, and Joerg Widmer. 2018. Adaptive codebook optimization for beam training on off-the-shelf IEEE 802.11 ad devices. In *Proc. ACM MobiCom'18*.

[38] Michele Polese, Marco Giordani, Tommaso Zugno, Arnab Roy, Sanjay Goyal, Douglas Castor, and Michele Zorzi. 2020. Integrated access and backhaul in 5G mmWave networks: Potential and challenges. *IEEE Commun. Mag.* 58, 3 (2020), 62–68.

[39] Michele Polese, Francesco Restuccia, Abhimanyu Gosain, Josep Jornet, Shubhendu Bhardwaj, Viduneth Ariyarathna, Soumyajit Mandal, Kai Zheng, Aditya Dhananjay, Marco Mezzavilla, et al. 2019. MillimeTera: Toward a large-scale open-source mmWave and terahertz experimental testbed. In *Proc. ACM mmNets'19*.

[40] Zhenzhou Qi, Zhihui Gao, Chung-Hsuan Tung, and Tingjun Chen. 2023. Programmable millimeter-wave MIMO radios with real-time baseband processing. In *Proc. ACM WiNTECH'23*.

[41] Sundeep Rangan, Theodore S Rappaport, and Elza Erkip. 2014. Millimeter-wave cellular wireless networks: Potentials and challenges. *Proc. of the IEEE* 102, 3 (2014), 366–385.

[42] Theodore S Rappaport, Shu Sun, Rimma Mayzus, Hang Zhao, Yaniv Azar, Kevin Wang, George N Wong, Jocelyn K Schulz, Mathew Samimi, and Felix Gutierrez. 2013. Millimeter wave mobile communications for 5G cellular: It will work! *IEEE Access* 1 (2013), 335–349.

[43] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, et al. 2020. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. ACM MobiCom'20*.

[44] Michaelraj Kingston Roberts and Parthibaraj Anguraj. 2021. A comparative review of recent advances in decoding algorithms for low-density parity-check (LDPC) codes and their applications. *Archives of Computational Methods in Engineering* 28, 4 (2021), 2225–2251.

[45] John W Romein and Bram Veenboer. 2018. Powersensor 2: A fast power measurement tool. In *Proc. IEEE ISPASS'18*.

[46] Bodhisatwa Sadhu, Yahya Tousi, Joakim Hallin, Stefan Sahl, Scott K Reynolds, Örjan Renström, Kristoffer Sjögren, Olov Haapalahti, Nadav Mazor, Bo Bokinge, et al. 2017. A 28-GHz 32-element TRX phased-array IC with concurrent dual-polarized operation and orthogonal phase and gain control for 5G communications. *IEEE J. Solid-State Circuits* 52, 12 (2017), 3373–3391.

[47] Swetank Kumar Saha, Yasaman Ghasempour, Muhammad Kumail Haider, Tariq Siddiqui, Paulo De Melo, Neerad Somanchi, Luke Zakrajsek, Arjun Singh, Owen Torres, Daniel Uvaydov, et al. 2017. X60: A programmable testbed for wideband 60 GHz WLANs with phased arrays. In *Proc. ACM WiNTECH'17*.

[48] Conrad Sanderson and Ryan Curtin. 2016. Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software* 1, 2 (2016), 26.

[49] Silicom Ltd. 2023. Lisbon P2 ACC100 FEC accelerator extended temp. https://www.silicom-usa.com/wp-content/uploads/2023/09/Lisbon-P2-ACC100-FEC-Accelerator-Extended-temp-1v1.pdf.

[50] srsRAN Project. 2023. srsRAN Project Documentation - Features and Roadmap. https://docs.srsran.com/projects/project/en/latest/general/source/2_features_and_roadmap.html.

[51] Jay Kumar Sundararajan, Hwan-Joon Kwon, Olufunmilola Awoniyi-Oteri, Yuchul Kim, Chih-Ping Li, Jelena Damnjanovic, Shanyu Zhou, Ruifeng Ma, Yeliz Tokgoz, Prashanth Hande, et al. 2021. Performance evaluation of extended reality applications in 5G NR system. In *Proc. IEEE PIMRC'21*.

[52] Sanjib Sur, Xinyu Zhang, Parmesh Ramanathan, and Ranveer Chandra. 2016. BeamSpy: Enabling robust 60 GHz links under blockage. In *Proc. USENIX NSDI'16*.

[53] Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M Voelker. 2011. Sora: high-performance software radio using general-purpose multi-core processors. *Commun. ACM* 54, 1 (2011), 99–107.

[54] Chance Tarver, Matthew Tonnemacher, Hao Chen, Jianzhong Zhang, and Joseph R Cavallaro. 2021. GPU-based, LDPC decoding for 5G and beyond. *IEEE Open Journal of Circuits and Systems* 2 (2021), 278–290.

[55] The Linux Foundation. 2023. Data plane development kit (DPDK). https://doc.dpdk.org/guides-21.11/tools/testbbdev.html.

[56] Guohui Wang, Michael Wu, Bei Yin, and Joseph R Cavallaro. 2013. High throughput low latency LDPC decoding on GPU for SDR systems. In *Proc. IEEE GlobalSIP'13*.

[57] Song Wang, Jingqi Huang, and Xinyu Zhang. 2020. Demystifying millimeter-wave V2X: Towards robust and efficient directional connectivity under high mobility. In *Proc. ACM MobiCom'20*.

[58] Yi Wei, Ming-Min Zhao, Min-Jian Zhao, and Ming Lei. 2020. ADMM-based decoder for binary linear codes aided by deep learning. *IEEE Commun. Lett.* 24, 5 (2020), 1028–1032.

[59] Timothy Woodford, Xinyu Zhang, Eugene Chai, Karthikeyan Sundaresan, and Amir Khojastepour. 2021. Spacebeam: Lidar-driven one-shot mmwave beam management. In *Proc. ACM MobiSys'21*.

[60] Chenshu Wu, Feng Zhang, Beibei Wang, and KJ Ray Liu. 2020. mmTrack: Passive multi-person localization using commodity millimeter wave radio. In *Proc. IEEE INFOCOM'20*.

[61] Qing Yang, Xiaoxiao Li, Hongyi Yao, Ji Fang, Kun Tan, Wenjun Hu, Jiansong Zhang, and Yongguang Zhang. 2013. BigStation: Enabling scalable real-time signal processing large MU-MIMO systems. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 399–410.

[62] Ding Zhang, Mihir Garude, and Parth H Pathak. 2018. mmChoir: Exploiting joint transmissions for reliable 60 GHz mmWave WLANs. In *Proc. ACM MobiHoc'18*.

[63] Ding Zhang, Panneer Selvam Santhalingam, Parth Pathak, and Zizhan Zheng. 2023. CoBF: Coordinated beamforming in dense mmWave networks. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 2 (2023), 1–26.

[64] Jialiang Zhang, Xinyu Zhang, Pushkar Kulkarni, and Parameswaran Ramanathan. 2016. OpenMili: A 60 GHz software radio platform with a reconfigurable phased-array antenna. In *Proc. ACM MobiCom'16*.

[65] Renjie Zhao, Timothy Woodford, Teng Wei, Kun Qian, and Xinyu Zhang. 2020. M-Cube: A millimeter-wave massive MIMO software radio. In *Proc. ACM MobiCom'20*.