IEEE Open Journal of
Circuits and Systems

Received 9 August 2020; revised 11 November 2020; accepted 26 November 2020. Date of current version 28 January 2021.

Digital Object Identifier 10.1109/OJCAS.2020.3042448

GPU-Based, LDPC Decoding for 5G and Beyond

CHANCE TARVER [1,2] (Member, IEEE), MATTHEW TONNEMACHER [2], HAO CHEN [2], JIANZHONG ZHANG [2] (Fellow, IEEE) AND JOSEPH R. CAVALLARO [1] (Fellow, IEEE)

[1] Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005 USA

[2] Standard and Mobility Innovation Lab, Samsung Research America, Plano, TX 75023, USA

This article was recommended by Associate Editor L. Liu.

CORRESPONDING AUTHOR: C. TARVER (e-mail: tarver@rice.edu)

This work was supported in part by Samsung Research America, and in part by the U.S. NSF under Grant CNS-1717218, Grant CNS-2016727, and Grant CNS-1827940, for the "PAWR Platform POWDER-RENEW: A Platform for Open Wireless Data-Driven Experimental Research With Massive MIMO Capabilities."

ABSTRACT In 5G New Radio (NR), low-density parity-check (LDPC) codes are included as the error correction codes (ECC) for the data channel. While LDPC codes enable a low, near Shannon capacity, bit error rate (BER), they also become a computational bottleneck in the physical layer processing. Moreover, 5G LDPC has new challenges not seen in previous LDPC implementations, such as Wi-Fi. The LDPC specification in 5G includes many reconfigurations to support a variety of rates, block sizes, and use cases. 5G also creates targets for supporting high-throughput and low-latency applications. For this new, flexible standard, traditional hardware-based solutions in FGPA and ASIC may struggle to support all cases and may be cost-prohibitive at scale. Software solutions can trivially support all possible reconfigurations but struggle with performance. This article demonstrates the high-throughput and low-latency capabilities of graphics processing units (GPUs) for LDPC decoding as an alternative to FPGA and ASIC decoders, effectively providing the high performance needed while maintaining the benefits of a software-based solution. In particular, we highlight how by varying the parallelization strategy for mapping GPU kernels to blocks, we can use the many GPU cores to compute one codeword quickly to target low-latency, or we can use the cores to work on many codewords simultaneously to target high throughput applications. This flexibility is particularly useful for virtualized radio access networks (vRAN), a next-generation technology that is expected to become more prominent in the coming years. In vRAN, the hardware computational resources will become decoupled from the specific computational functions in the RAN through virtualization, allowing for benefits such as load-balancing, improved scalability, and reduced costs. To highlight and investigate how the GPU can accelerate tasks such as LDPC decoding when containerizing vRAN functionality, we integrate our decoder into the Open Air Interface (OAI) NR software stack. With our GPU-based decoder, we measure a best case-latency of 87 μs and a best-case throughput of nearly 4 Gbps using the Titan RTX GPU.

INDEX TERMS LDPC, SDR, GPU, OAI, vRAN.

I. INTRODUCTION

A KEY challenge in 5G and beyond is the flexibility necessary for the radio access network (RAN) to be able to support the many possible applications ranging from 4K video streaming, which requires high data rates, to high-precision remote surgery, which requires ultra-low latency. The various applications are typically described to fit into one of the following categories: enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), and massive machine-type communications (mMTC). These descriptions are used to encapsulate the IMT-2020 5G requirements. eMBB is designed to support peak downlink data throughputs of 20 Gbps and uplink peak rates of 10 Gbps. URLLC is designed to support end-to-end

latencies of 1 ms. The mMTC category is designed to support connection densities of 1 million devices per km^2 [1]. These targets are 50 to 100× above the targets for 4G. To achieve the next 100× improvement beyond 5G, communications systems must continue embracing flexibility.

A. GPU-BASED HIGH-PERFORMANCE BASEBAND PROCESSING

In 5G, flexibility is a major theme. However, providing high-performance computation for all possible use cases and variations in the standard may be difficult for hardware-based accelerators. This flexibility/performance gap can be filled by GPUs which provide many multipurpose cores for high-performance parallelization while maintaining software agility. GPUs have become commonplace in many high-performance computing fields such as deep learning. They have also already been considered for many baseband processing tasks for 4G and 5G implementations. For example, in [2] GPUs are used for detection and beamforming in a multi-user (MU) multiple-input multiple-output (MIMO) base station. Additionally, GPUs can be containerized via tools like NVIDIA-Docker to run on vRAN systems, and the latest generation supports Multi-Instance GPU (MIG), allowing a single GPU to be virtualized into multiple GPUs.

1) LDPC FOR 5G

One particular task well-suited for GPU implementation in 5G is the LDPC ECC. ECCs add redundancy to wirelessly transmitted bits so that the receiver can detect and correct errors. LDPC was chosen to replace turbo codes from 4G for the data traffic in 5G [3]. Although LDPC codes have near capacity-achieving decoding performance, the decoding complexity is high. LDPC decoding creates a computational challenge for the 5G next-generation NodeBs (gNBs), which potentially need to decode many codewords (CWs) from multiple users at high data rates.

GPUs are an excellent platform for LDPC. This combination works due to how well LDPC decoding algorithms cleanly map to the single-instruction multiple-thread (SIMT) parallel architecture of GPUs. Decoding is an iterative process where log-likelihood ratio (LLR) messages are exchanged to correct any bits. Messages can easily be computed in parallel on the thousands of processing elements such as the compute unified device architecture (CUDA) cores found in NVIDIA GPUs. A GPU project can be rapidly developed and deployed similar to any other software project. Scaling up to support more users and higher codeword throughput can also be trivial in that multiple GPUs can work together in parallel. This scalability lends itself to being a natural fit for data-center based cloud radio access network (C-RAN) systems. Moreover, as GPU hardware development continues to progress rapidly, an operator could see throughput and latency improvements by upgrading devices over time without additional development overhead.

A new challenge in 5G for any LDPC implementation that is not found in other radio access technologies (RATs) that use LDPC such as Wi-Fi is the range of parameters, block sizes, and target use cases. In eMBB, a high decoding throughput is necessary. For URLLC, being able to quickly decode a CW will likely be one of the major bottlenecks in round-trip time. For mMTC, it will be challenging to decode CWs from many users simultaneously. However, the scalability and reconfigurability of the GPU make this possible. At runtime, we can reconfigure the GPU to change the configuration to prioritize latency or throughput.

2) RELATED WORKS

GPUs have been used for a variety of ECC. For example, in [4] they are used for turbo codes found in 4G. LDPC codes were implemented in [5], [6], and [7], with the latter work also deploying multiple GPUs together to increase total throughput. However, these works target Wi-Fi and WiMAX. There are few reported architectures and software realizations for the recent LDPC code for 5G NR physical layers. In [8] LDPC is implemented on a Xeon processor using AVX instructions. While the latency results are exceptional at 31 μs, the throughput is limited to 270 Mbps. Intel offers an field programmable gate array (FPGA) based solution based on their FlexRAN platform, but performance benchmarks are not publicly available [9]. Xilinx also offers an FPGA based solution that supports NR LDPC. Their IP is reported to have a peak throughput of up to 1.78 Gbps for eight iterations, but no other details about the exact platform this number was measured on were provided in the product brief [10].

Other works have investigated augmenting LDPC with neural networks (NNs) through a process known as deep unfolding [11]. For example, in [12], an offset factor is learned in an offset min-sum implementation [13]. This additional parameter learned over the Tanner graph can help improve BER performance, potentially reducing the number of iterations and improving throughput and latency. Although we do not adopt these techniques in this work, the GPU is a natural fit to any NN processing, and our methods scale to where the NN enhancements could be added to our kernels.

Recently, Nvidia has announced their Aerial SDK for GPU acceleration in 5G networks. One particular emphasis made by the company is the combination of artificial intelligence (AI) with other network tasks to provide novel services [14]. While AI seems to be a major focus, they also claim to have GPU acceleration for baseband tasks through their cuPHY SDK [15]. However, access is currently limited to these tools, and there are no publicly available details on the performance or architecture of their LDPC decoder.

3) CONTRIBUTIONS

In this work, we provide a GPU solution for 5G NR LDPC decoding and highlight the approach's flexibility. Our main

contributions are the development of a GPU-based solution for LDPC with support for the 5G NR standard with the flexibility necessary to become a research platform for beyond 5G. We present throughput and latency numbers across multiple GPU devices. In our implementation, we present multiple optimizations to improve performance, such as quantization to shorter word lengths to save time on data transfers. We also develop our GPU software as a library to run in a container and use it in the "develop-nr" git branch of Open Air Interface (OAI) [16] as part of a vRAN testbed.

This article is a journal extension of the conference version from [17]. In this journal extension, we include more discussion of the GPU algorithm implementation and architecture, present new results with more 5G LDPC configurations, extend our analysis to better understand the throughput/latency tradeoffs, provide a timing breakdown of the GPU-based LDPC decoding, and provide more details throughout.

The rest of the paper is as follows. In Section II, we present an overview of LDPC, including its role in 5G NR. In Section III, we explain how we map the decoding algorithm to the GPU by describing the GPU architecture then detailing each of our kernels. Section IV presents the results including comparisons with other works. Section V includes an overview of implementation efforts in software-defined radio (SDR) platforms. We then conclude the paper in Section VI.

II. LDPC OVERVIEW

A binary LDPC code is defined by an $M \times N$ sparse parity check matrix, $\mathbf{H}$, and a $N \times 1$ CW vector, $\mathbf{x}$. For the vector $\mathbf{x}$ to be a valid CW, $\mathbf{Hx} = \mathbf{0}$ must be true. In a communications system, we only transmit valid CWs. If we receive a CW that is not valid, we assume that elements of $\mathbf{x}$ are incorrect and attempt to find the valid CW that was most likely sent.

A sequence of K information bits, $\mathbf{s}$, is encoded to create valid codewords where the encoding is often performed using a generator matrix derived from $\mathbf{H}$. The rate of an LDPC code, R, is determined by the ratio of information bits to codeword bits, $R = K/N$. LDPC codes, including those used in 5G, are often "systemic" meaning that the first K bits in a given codeword are the information bits. The remaining $N - K$ bits are redundancy bits.

One common family of codes is quasi-cyclic (QC)-LDPC codes. Here, each parity check matrix is constructed from a $m_b \times n_b$ base matrix, which is an array of shifted identity matrices of size Z known as the lifting factor. The two QC base graphs of 5G NR are shown in Fig. 1.

From the parity check matrix, a bipartite Tanner graph can be constructed with rows in $\mathbf{H}$ corresponding to check nodes (CNs), columns corresponding to variable nodes (VNs), and $H_{ji} = 1$ corresponding to an edge connecting CN j and VN i.

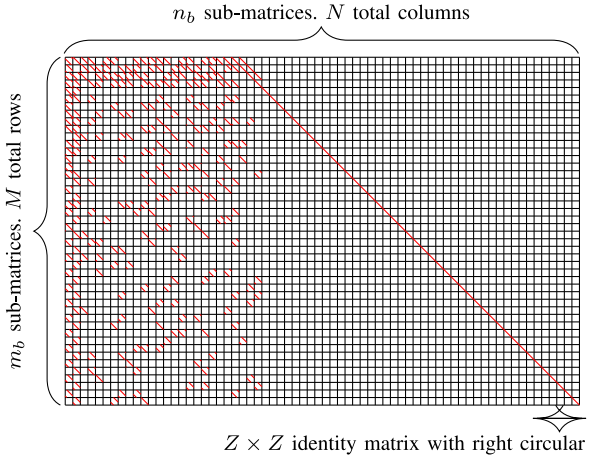
A. DECODING ALGORITHMS

The main idea behind most LDPC decoders is that we can send messages back-and-forth from VNs to CNs to find and correct any incorrect bits. Decoding is typically performed via iterative message passing on the Tanner graph between CNs and VNs. The calculation of *a posteriori* probabilities (APP) is done through a sum-product algorithm (SPA). However, a common simplification that we implement in this work is the min-sum algorithm (MSA). When implemented with a scaling parameter or an offset, it offers low complexity with minor loss in BER performance. For complete details on SPA, MSA, and other LDPC algorithm details, see [18]. For each, we assume that soft-decision LLR values are given to the decoder from the demodulator as opposed to hard bits. The LLR value corresponding to bit n, L_n, for a BPSK signal with constellation $\{-1, 1\}$ is given as

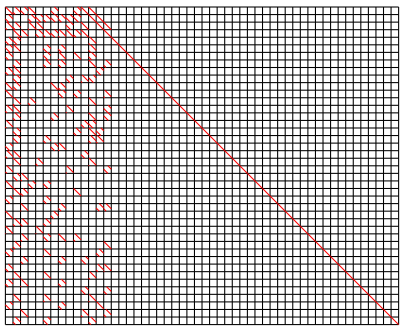
$$L_n = \log\left(\frac{\text{Prob}(x_n = 0)}{\text{Prob}(x_n = 1)}\right) = \frac{2y_n}{\sigma^2}, \tag{1}$$

where y_n is the analog value of the received symbol and σ^2 is the noise power variance of the channel.

a) *Min-Sum Algorithm:* The MSA and its variants, such as the scaled min-sum algorithm and the offset min-sum algorithm, are simplifications made on the SPA to improve decoding throughput with a minor loss in decoding performance. In its basic form, it allows for decoding to be performed with only comparison operations and additions.



(a) Base graph 1. Here, $m_b = 46$, $n_b = 68$, and the set index $i_{LS} = 1$.



(b) Base graph 2. Here, $m_b = 42$, $n_b = 52$, and the set index $i_{LS} = 0$.

FIGURE 1. Illustration of the QC-LDPC parity check base graphs, $\mathbf{H_{BG}}$, used in 5G NR. Red slashes correspond to ones in the sub-matrices.

The algorithm begins with check node processing. The initial messages to each CN is set to the corresponding initial LLRs, $Q_{mn}^{(0)} = L_n^{(0)}$. Each CN proceeds to compute the messages to send to each connected VN, R_{mn}. Eq. (2) shows this calculation where m and n are the index of the check and variable nodes respectively, N_m is the set of all variable nodes connected to check node m, and γ is the MSA scale factor:

$$R_{mn}^{(i+1)} = \gamma \min_{n' \in N_m \backslash n} \left| \left(Q_{mn'}^{(i)} \right) \right| \prod_{n' \in N_m \backslash n} \text{sign}\left(Q_{mn'}^{(i)} \right). \quad (2)$$

The variable nodes update the LLR with the APP in (3),

$$L_n^{(i+1)} = L_n^{(i)} + \sum_m \left(R_{mn}^{(i+1)} - R_{mn}^{(i)} \right), \quad (3)$$

and then compute updated messages to send to each of the connected check nodes in (4),

$$Q_{mn}^{(i+1)} = L_n^{(i+1)} - R_{mn}^{(i+1)}. \quad (4)$$

Once the maximum number of iterations has been performed, I, a final hard decision can be made as shown in (5),

$$x_n = \begin{cases} 1 & L_n^{(T)} \leq 0 \\ 0 & L_n^{(T)} > 0 \end{cases}. \quad (5)$$

B. LDPC FOR 5G NR

The LDPC encoding and decoding specification for 5G NR is defined in 3GPP TS.38.212 of Rel. 15. In 5G, LDPC is used for the downlink shared channel (DL-SCH), uplink shared channel (UL-SCH), and paging channel (PCH) transport channels, which mostly carry data. In contrast, 5G uses polar codes for the broadcast channel (BCH), which mostly carries system information. In the below section, we present a brief overview of the relevant parts of the specification for reference. See [19] for the complete details and [20] for a discussion on the design of LDPC for 5G.

There are two QC base graphs (BGs) used to derive the parity check matrices. In general, BG 1, shown in Fig. 1(a), is used for larger, higher-rate data while BG 2, shown in Fig. 1(b), is reserved for short payload sizes and code rates less than 0.25. For BG 1 there are $m_b = 46$ rows and $n_b = 68$ columns while BG 2 has $m_b = 42$ and $n_b = 52$. The maximum length of a sequence of bits in a code block, K_{cb}, to be encoded is 8448 bits for BG 1 and 3840 bits for BG 2. Each supports a variety of lifting factors, Z, from 2 up to 384. When using $Z = 384$, the largest possible 5G NR code is an irregular (25344, 8448) rate 1/3 code which uses BG 1.

To better understand LDPC for 5G, we discuss the encoding procedure below. Given a sequence of input bits to be encoded, represented by the vector **a**, we start by appending a cyclic redundancy check (CRC) to make a new vector, **b**. For **b** with length B and the code rate, R, provided by the modulation and coding scheme (MCS), a BG is chosen according to the above rules. If $B > K_{cb}$ for the selected BG then the input sequence is segmented into multiple code blocks, each with their own CRC forming a new sequence, **c**. The size of the individual code blocks is used to calculate the appropriate lifting factor, Z. Based on the lifting factor, a table lookup provides the set index, i_{LS}, which is used to construct the final parity check matrix, **H**. There are eight set indices in total, leading to eight possible variations of each BG. In the process of encoding, a vector in the nullspace of **H** is constructed in the form of $\begin{bmatrix} \mathbf{c} & \mathbf{w} \end{bmatrix}^T$ where **w** is a vector of parity bits to be calculated with length $N + 2Z - K$ where $N = 66Z$. After encoding, the CW bits corresponding to the indices after $2Z$ undergo rate matching via bit selection and interleaving. If segmentation took place, the code blocks are concatenated together before modulation and transmission.

A critical insight from the above exploration of the 5G standard is that the standard is extensive. Decoders should support 51 different lifting factors from Z = 2 to 384, giving rise to eight reconfigurations of two separate parity check base graphs, all derived from a wide range of possible code rates and block lengths. Supporting all possible valid configurations is challenging for hardware-based decoding implementations. Moreover, meeting the various latency and throughput targets based on the real-time demands is another serious challenge. In contrast, the GPUs flexible, software-based nature makes decoding straightforward without sacrificing performance, as we will demonstrate in the remainder of the paper.

1) HARQ AND LDPC

hybrid automatic repeat request (HARQ) is implemented in 5G NR to provide incremental redundancy. For uplink transmissions, the eNB must provide a success (ACK) or failure (NACK) acknowledgment within 4 ms. We will use this as a frame of reference for the LDPC decoding latency in the results section. However, all uplink processing through the CRC check must be complete to update the appropriate HARQ process, so additional time must be budgeted to perform the appropriate demodulation steps as well as the LDPC decoding.

C. SUMMARY OF NOTATION

In this section, we include a list of all symbols used throughout the paper. Matrices, such as **H**, are denoted in bold and capitalized. Scalar constants, such as K are unbolded capital letters. Vectors, such as **c**, are bolded lower-case symbols. The l0 norm is given as $\|.\|_0$ and returns the number of non-zero elements in a vector. We include a list of all symbols and definitions in Table 1.

III. IMPLEMENTATION DETAILS

A. GPU OVERVIEW

A GPU consists of an array of streaming multiprocessors (SMs), which each often contain 32 or 64 vector processing lanes. NVIDIA provides C++ language extensions, known as CUDA, to write highly parallel applications that target these devices. The developer writes kernels that will utilize a certain number of blocks and threads, which are roughly a

TABLE 1. LDPC notation.

Symbol	Definition
K	Number of information bits to encode
N	Number of coded bits and VNs
M	Number of CNs
R	Code rate (K/N)
$\mathbf{H}$	Parity check matrix with dimensions $M \times N$
m_b	Number of rows of sub-matrices for a quasicyclic $\mathbf{H}$
n_b	Number of columns of sub-matrices for a quasicyclic $\mathbf{H}$
Z	Lifting factor
$\mathbf{s}$	Information (systemic) bits; column vector of length K
$\mathbf{x}$	Codeword bits; column vector of length M
I	Total number of decoding iterations
$L_n^{(i)}$	LLR for VN n at iteration i
$R_{mn}^{(i)}$	Message from CN m to VN n at iteration i
$Q_{nm}^{(i)}$	Message from VN n to CN m at iteration i
η	Number of CUDA streams
α	Number of CWs per MCW
β	Number of MCWs
T	Data type used to represent LLRs
$r_{H\leftrightarrow D}$	Average memory bandwidth between the host and the device
r_{DDR}	Average memory bandwidth to global memory
$t_{H\rightarrow D}$	Average time per host-to-device transfer
t_{CN}	Average time per CN kernel call
t_{VN}	Average time per VN kernel call
t_{BP}	Average time per bit-packing kernel call
$t_{D\rightarrow H}$	Average time per device-to-host transfer
C	LDPC codeword decoding throughput

programming abstraction of the SMs and CUDA cores. There are multiple tiers of memory on the GPU. Global memory is typically a GDDR-based, off-chip memory available to all kernels over the lifetime of the application. Threads in a block all execute on the same SM and have access to the same low-latency, high-throughput, on-chip SRAM memory known as "shared" memory for the kernel's lifetime. Each thread also has its own local registers. There is also an on-chip "constant" memory that is globally available to all threads, but it is read-only. The global memory is typically the slowest, so it is desirable to put data in constant and shared memory whenever possible.

Each generation of GPU computing has consistently seen substantial improvements over the previous. The 2020 Ampere generation from Nvidia [21] includes new features such as multi-instance GPU, which will allow for the virtualization of one GPU into multiple smaller GPUs, which may be attractive for vRAN applications. Direct GPU-to-GPU communication through NVLink, the inclusion of PCIe4 on many platforms, and support for Mellanox-based RDMA [22] will make significant progress on overcoming the issues of moving data to-and-from GPUs that previously limited adoption for real-time problems such as PHY processing.

B. MAPPING OF LDPC TO GPU

When implementing any algorithm on GPU, effectively mapping the algorithm to the hardware is critical for achieving the desired acceleration. For a GPU system, this mapping involves deciding where to draw the kernel boundaries within an algorithm, deciding how the computation should be parallelized across the threads, structuring the data so that it can be placed appropriately in the memory hierarchy, and using the profiling tools to iterate on design decisions. There are a near limitless number of ways algorithms such as LDPC decoding can be structured to run on the GPU. In this work, we extend the architecture described in [5] and [7] where there is a kernel devoted to check node processing and a kernel devoted to variable node processing and individual threads will perform the computation related to one check node or variable node. For additional information on tradeoffs for the design decisions related to the architecture, see [5].

We create two main kernels: a CN kernel and a VN kernel. Multiple CWs are decoded in parallel via two main mechanisms. First, we pack $\alpha \in \mathbb{N}$ CWs into what we refer to as a macro-codeword (MCW). These CWs are evaluated concurrently within the same CUDA block. Secondly, we copy $\beta \in \mathbb{N}$ MCWs to the GPU in a batched transfer to execute on separate blocks. Each block will work on one row of the base graph with the individual threads working on the subrows after applying the lifting factor. This strategy allows for $\alpha \cdot \beta$ total codewords to be transferred to the GPU and decoded concurrently.

When operating the system, selection of α and β will depend on the hardware, lifting factor Z, and the operators' desired tradeoff between latency and throughput. In CUDA, there is a maximum number of threads per block (usually 1024). There is therefore a constraint that $\alpha Z \leq 1024$. As the number of total codewords being transfered to the GPU per batch increases, given by $\alpha \cdot \beta$ the latency per codeword increases. Throughput typically increases as the total codewords per batch increases. Once the total number of codewords in a batch is sufficiently high so that the GPU cores can remain fully backlogged, there is no benefit to further increases in either parameter. It is recommended that the system designer experiment with the Nvidia profiling tools to find the best set of parameters for their hardware platform and design goals.

Fig. 2 presents a top-level diagram of each of the main arrays and kernels. In Table 1, we provide a description of each array in memory as well as the length. For most of these arrays, the final size in memory per GPU stream is given by multiplying the array's length by the size of the selected data type. In addition to these primary arrays, we store a structure in constant memory with various parameters needed for decoding such as, Z and I. In the following subsections, we will examine each kernel in more detail.

1) CHECK NODE PROCESSING

In this kernel, each thread operates on a row of $\mathbf{H}$. Using the two-min algorithm for the scaled MSA, messages are computed for each connected variable node. These messages are then stored in global memory. The CUDA grid is deployed

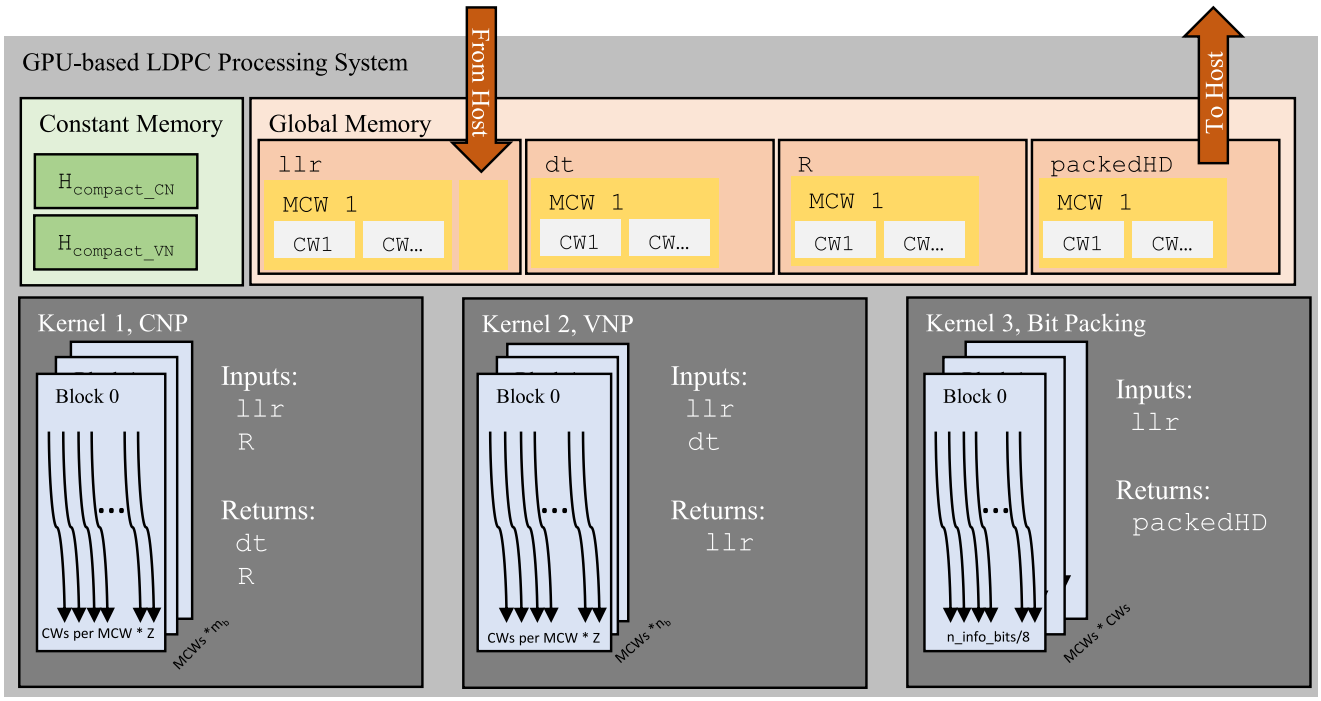


FIGURE 2. GPU Architecture Diagram. The LLRs for multiple MCWs are copied from the host to the device. The LDPC decoding is performed by iterating *I* times between Kernel 1 and 2 for check-node and variable-node processing. After *I* iterations, a hard decision is made, and Kernel 3 packs the hard decision bits into a smaller memory footprint format for transfer back to the host.

TABLE 2. Summary of utilized GPU memory.

Name	Description	Location	Length
`h_compact_cn`	Condensed form of H arranged with non-zero elements of rows of $\mathbf{H}$	Constant	$m_b \cdot \max(\|\text{row}(\mathbf{H})\|_0)$
`h_compact_vn`	Condensed form of H arranged with non-zero elements of columns of $\mathbf{H}$	Constant	$n_b \cdot \max(\|\text{col}(\mathbf{H})\|_0)$
`llr`	Array of LLRs for all deployed CWs	Global	$\alpha \cdot \beta \cdot N$
`dt`	Change in message from each CN to each connected VN, $R_{mn}^{(i+1)} - R_{mn}^{(i)}$	Global	$\alpha \cdot \beta \cdot M \cdot \max(\|\text{row}(\mathbf{H})\|_0)$
`R`	Current message from each check node to connected variable nodes, $R_{mn}^{(i)}$	Global	$\alpha \cdot \beta \cdot M \cdot \max(\|\text{row}(\mathbf{H})\|_0)$
`R_cache`	Messages, $R_{mn}^{(i)}$, read into CN kernel during first recursion to be reused in the second recursion	Shared	$\max(\|\text{row}(\mathbf{H})\|_0)$
`packed_hd`	Final hard-decision bits to be packed and sent to the host	Global	$\alpha \cdot \beta \cdot \lceil K/8 \rceil$

with blocks of dimensions $(m_b, \beta, 1)$ and threads of dimensions $(Z, \alpha, 1)$. The check node kernel is shown in Kernel 1. The main idea of the kernel is that the one thread will be working on one check node. The individual thread will load a compact parity check matrix to determine which variable nodes it is connected to. It will then access the incoming messages from global memory, keeping track of the minimum and the second minimum incoming message. Then, in a second recursion, it computes messages to each VN as per (2).

There are two optimizations to note in the approach for Kernel 1. First, the check node computes the messages sent by each variable node as per (4). By having the check nodes compute Q instead of loading it, we reduce the amount of global memory accesses required and improve performance. Second, since the kernel already has access to two iterations of R, we store the difference in its own array `dt` for use by Kernel 2, again reducing memory accesses.

2) VARIABLE NODE PROCESSING

For the VN kernel, each thread corresponds with a column of $\mathbf{H}$ and computes its APP LLR based on the change in R stored in `dt` by Kernel 1. The kernel accesses elements of `dt` according to `h_compact_vn` and accumulates the change in messages to update the LLR L_n according to (3). The updated LLR is then stored in global memory. The full algorithm is shown in Kernel 2. The CUDA grid is deployed with blocks of dimensions $(n_b, \beta, 1)$ and threads of dimensions $(Z, \alpha, 1)$.

C. OPTIMIZATIONS STRATEGIES

1) REDUCED WORD LENGTH

The main optimization that we offer in this work compared to previous GPU-based solutions is the reduction of the word lengths down to the 8-bit char format. Using as few as six bits is known to be adequate [23] and is common in many

Kernel 1 Check Node Processing

```
i_subrow, i_CW ← Thread Indices
i_blockrow, i_MCW← Block Indices
i_thisCW = α· i_MCW + i_CW
load h_compact_cn from constant memory
rmin1, rmin2, imin, qSigns, signProd = 0, 0, 0, 0, 1
// First Recursion. Search for Min
for i = 0 to len(h_compact_cn) do
    load llr and R as L and R
    Q = L − R;                          // Calculate VN message to CN as in (4)
    save R in R_cache
    s = Q < 0                           // Check sign for (2)
    qsign |= s << i                     // Store this sign bit in register
    signProd *= (1 − 2s)                // Update product of signs for (2)
    if |Q| < rmin1 then
        rmin2 ← rmin1, rmin1 ← Q, imin ← i
    else if |Q| <rmin2 then
        rmin2 ← Q
    end if
end for
// Second Recursion. Calculate Messages to VNs
for i = 0 to len(h_compact_cn) do
    s = 1 − 2((qsign>> i)&0x01)
    save R_new = 0.75signProd ·s · (i ==imin? rmin2:rmin1)
    load R from R_cache
    Δ = R_new − R
    save Δ in dt    // Save difference for (3) to be used in Kernel 2
end for
```

Kernel 2 Variable Node Processing

```
i_subcol, i_CW ← Thread Indices
i_blockcol, i_MCW← Block Indices
i_thisVN = (Z· i_blockcol) + i_subcol
load h_compact_cn from constant memory
load L ← llr[i_thisVN] from global memory
for i = 0 to len(h_compact_vn) do
    load dt[i] from global memory
    L += dt[i]    // Compute new LLR by accumulating change in messages as in (3)
end for
save llr[i_thisVN] ← L
```

FPGA and ASIC implementations. We use this reduced word length to save up to 4x on data transfer times between host and device and for global memory accesses while casting to and from floats for each threads' computation.

2) PACKING FINAL HARD DECISION

After I total iterations, a hard decision can be made, and each LLR can be represented by a single bit. Past works often did not take advantage of this and stored the result with unnecessary data types. We implement a bit-packing kernel before the final transfer to the host and only copy the final, hard-decision information bits to the host instead of the entire decoded CW. This kernel is expressed as Kernel 3. Each thread works to grab eight elements from `llr` and then make the hard decision according to (5). This is then bit-shift accordingly into a byte. Once all eight `llr` elements that a thread is responsible for have been decided upon, the byte is stored in the `packed_hd` array in global memory for the host application to transfer back to the CPU.

Kernel 3 Bit-Packing Kernel

```
i_subcol, i_CW ← Thread Indices
i_blockcol, i_MCW← Block Indices
i_baseVN = (Z· i_blockcol) + i_subcol
load LLR for this VN
for i = 0; i < 8 do
    load L ← llr[i_baseVN + i] from global memory
    L > 0?b = 0:b = 1          // Hard decision for the LLR as per (5)
    packedbyte |= b << i
end for
save packed_hd[i_baseVN] = packedbyte to global memory
```

3) STREAMS

To further alleviate the memory transfer between the host and device, we utilize $\eta \in \mathbb{N}$ CUDA streams. Each stream acts like a separate process on a CPU application to be scheduled and executed in parallel. This allows for one stream to be computing while another may be copying new data to the GPU. This helps to ensure that the computational resources are fully utilized most of the time and that the device is not waiting for new data.

4) CACHING OF **H**

A row-major and column-major compact version of the parity check matrices, `h_compact_cn` and `h_compact_vn`, are loaded by the host onto the GPU's constant memory. The row or column-major version is used by the check or variable node kernels, respectively, to determine the memory index of a connected VN or CN to allow for fast, coalesced memory accesses [5].

D. MEMORY TRANSFER TIME OVERVIEW

The fundamental bottleneck for LDPC processing is the speed data can be moved. Firstly, we have to get the LLRs onto the GPU which is often limited by a bus such as PCIe. Once data is in the GPU's global memory, it must be transferred to the SMs for processing by the kernels. This data movement is done for both kernels for each iteration. In Table 2, we provided an overview of each array in memory for the LDPC decoding. In this section, we discuss the memory transfer time required as a function of the number of iterations, I, and each bus throughput rate, r, in bytes per second.

Firstly, we have the memory transfer into the GPU. This transfer time is given as $t_{H\to D}$ in (6),

$$t_{H\to D} = \alpha \cdot \beta \cdot N \cdot \text{sizeof(T)} \cdot r_{H\leftrightarrow D}. \tag{6}$$

Here, T represents the data type and sizeof(.) returns the number of bytes in T. In this work, we consider `char`, `half`, and `float` which have 1, 2, and 4 bytes per word. The average time per iteration of the CN kernel, t_{CN}, is given in (7),

$$t_{CN} = \epsilon_{CN} + \alpha \cdot \beta \cdot N \cdot \text{sizeof(T)} \cdot r_{DDR} + 2\alpha \cdot \beta \cdot M \cdot \max(\|\text{row}(\mathbf{H})\|_0)\text{sizeof(T)} \cdot r_{DDR} + \tag{7}$$

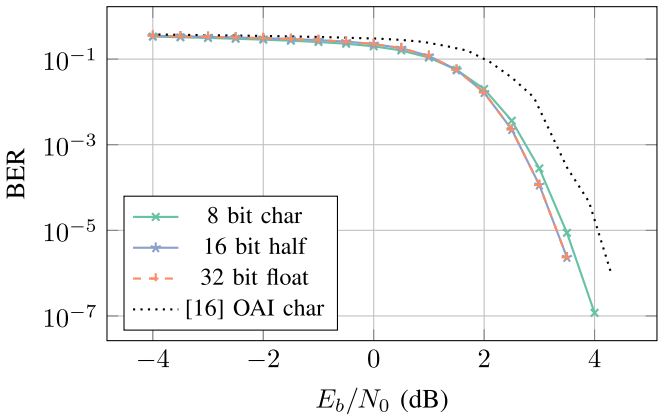


FIGURE 3. BER curve comparing performance when using various data storage precisions on GPU. The (25344, 8448) code was used with 5 iterations.

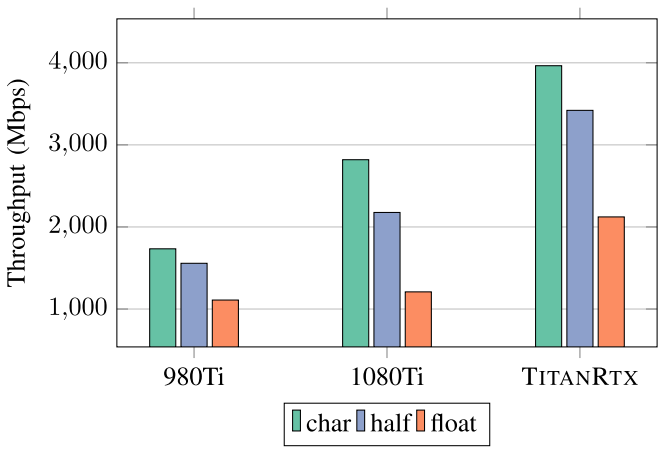


FIGURE 4. Throughput for the considered datatypes on three different GPUs.

In (7) we account for the time moving `llr, R,` and `dt` from the global memory to the threads. The time spent computing results is represented as ϵ_{CN}. The average time per iteration of the VN kernel, t_{VN}, is given in (8),

$$t_{VN} = \epsilon_{VN} + \alpha \cdot \beta \cdot N \cdot \text{sizeof(T)} \cdot r_{DDR} + \alpha \cdot \beta \cdot M \cdot \max(\|\text{row}(\mathbf{H})\|_0)\text{sizeof(T)} \cdot r_{DDR} + \tag{8}$$

For the VN kernel, `llr` and `dt` are transported from global memory, accounting for the bulk of the time. In (4), ϵ_{VN} represents the time computing in the kernel. The bit-packing kernel time is given below in (9).

$$t_{BP} = \epsilon_{BP} + \alpha \cdot \beta \cdot N \cdot \text{sizeof(T)} \cdot r_{DDR} + \alpha \cdot \beta \cdot \lceil K/8 \rceil \cdot r_{DDR} \tag{9}$$

The average time per LDPC batch is therefore given as t_{LDPC} in (10),

$$t_{LDPC} = t_{H\to D} + I(t_{CN} + t_{VN}) + t_{BP} + t_{D\to H}. \tag{10}$$

The codeword throughput,C, in codewords per second is then given in (11),

$$C = \alpha \cdot \beta \cdot t_{LDPC}. \tag{11}$$

Given this modeling, for a given platform with fixed $r_{H\leftrightarrow D}$ and r_{DDR}, one could choose optimal parameters α and β for a specific goal, such as minimizing latency per batch, t_{LDPC}, or maximizing the codeword throughput, C. Many of these tradeoffs are further examined in the following Results section.

IV. GPU MEASUREMENT RESULTS

In this section, we examine the performance and a variety of tradeoffs for the GPU implementation of the LDPC decoder. Throughout this section, we study the case where $Z = 384$ for base graph 1. This corresponds to a $R = 1/3$ code with $N = 26112$ total VNs.

a) *Performance:* We begin by evaluating the BER performance as a function of energy per bit to noise power spectral density (E_b/N_0). This tests the LDPC decoding performance and shows the performance for the various word lengths, as shown in Fig. 3. The float and half-precision data types offer nearly identical performance while there is a minor increase in BER when quantizing down to the 8-bit char for data storage. For comparison, we also include the OAI implementation from [16] which performs decoding on CPU using char formats throughout the data storage and computation. For a given decoder algorithm, such as min-sum, performance should be consistent across implementations. Any gap can be attributed to a difference in algorithms and quantization. Here, the gap between the OAI implementation and ours can be attributed to the difference between the min-sum algorithm implemented in OAI and the scaled min-sum implemented in this work.

b) *Throughput vs. Latency:* For three different GPUs, we tested the maximum throughput. For this test, we configured the GPU to use 6 streams, 20 MCWs per transfer, and 2 CW per MCWs. This configuration helped to ensure the GPU was always full, and the effect of the PCIe transfers was reduced. In Fig. 4, we see that with each generation of GPU performance increases as the number of CUDA cores also increases. The best performance was for the char data type on the TITAN RTX with 3964 Mbps of decoding throughput, including the transport time to-and-from the CPU and GPU. For this configuration, the latency is increased and on the order of 700 μs, though this can be acceptable for eMBB applications.

The GPU can be reconfigured to target URLLC to reduce the decoding times. To achieve lower latency, we can reduce the number of MCWs, CWs per MCW, and streams. In cases where there is a good signal-to-noise ratio (SNR), the number of decoding iterations can be reduced as well or early termination can be applied. When testing across multiple GPUs as shown in Fig. 5, we see that the TITAN RTX is able to achieve latencies as low as 87 μs, including the transport time to-and-from the CPU and GPU. For this configuration, the throughput is 290 Mbps.

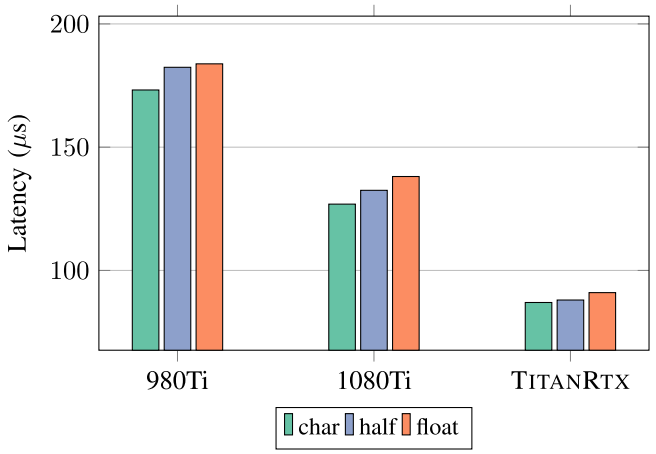


FIGURE 5. Latency for the considered datatypes on three different GPUs.

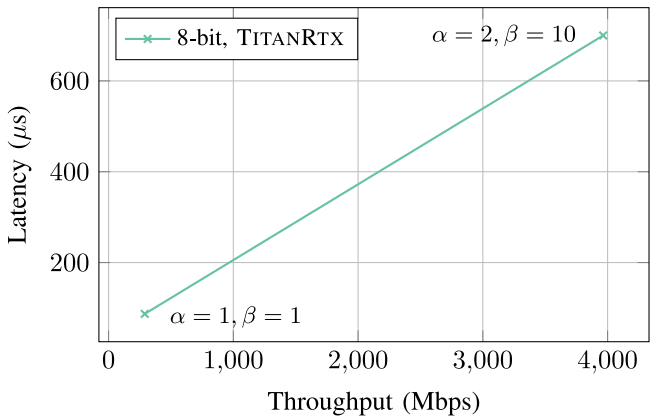


FIGURE 6. Latency vs throughput for five iterations. The GPU system architecture can be reconfigured to target latency constraints or throughput.

To thoroughly highlight the tradeoffs between latency and throughput, the two are plotted against each other in Fig. 6. When batching more codewords together to increase the throughput, the latency increases. With this batched transfer, the latency seen to transfer all the codewords to the GPU, decode all, and transfer back is 700 μs. This latency is well beneath the 4 ms deadline to send any HARQ feedback to the user equipment (UE).

c) *Latency vs. Iteration:* In Fig. 7, we examine the latency as a function of the number of LDPC iterations, I. This also gives an estimation of the performance if early termination were to be used where decoding ends before reaching I if the current LLR vector is determined to represent a valid codeword. As the number of iterations increase, the latency also increases at a linear rate. For the 8-bit LLR data with the 1080 TI, the latency increases at a rate of 13.7 μs per iteration.

d) *Timing Breakdown:* In Table 3, we break down the average timing to examine the amount of time moving data versus computing. For the 1080 Ti, we perform a variety of tests where we vary the number of MCW, α and the number of CWs per MCW, β. By varying these, we increase the

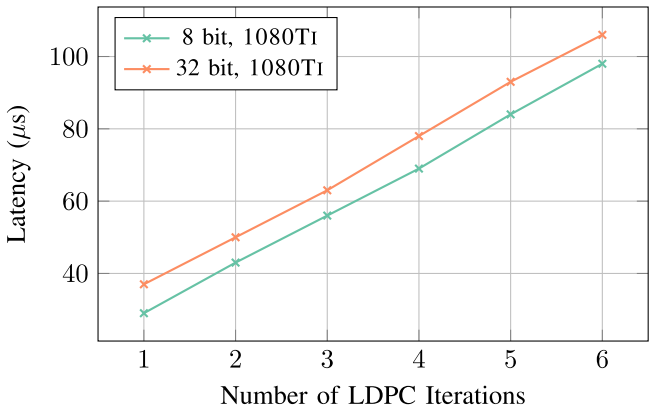


FIGURE 7. Latency vs. Iteration. As the number of iterations I increases, we see the latency increase at a linear rate of 13.7 μs per iteration. In this test, we configure the GPU for minimum latency by setting $\alpha = 1$ and $\beta = 1$.

TABLE 3. Timing breakdown, 1080 Ti (μs).

T	α	β	$t_{H\to D}$	t_{CN}	t_{VN}	t_{BP}	$t_{D\to H}$
char	1	1	5.8	7.4	5.3	1.6	1.0
char	1	2	9.5	8.3	5.9	1.8	1.3
char	10	2	73.3	39.6	28.9	6.2	7.1

total number of codewords per batch and we see that as the total number of codewords in the batch increase, there are proportional increases in each of the times due to the increase in memory transfer that occurs.

e) *Comparison to Other Works:* Currently, there are few published results available publicly for 5G LDPC decoders. To understand how our decoder fits into the space of 5G NR LDPC decoders in the open literature, we performed a survey of the available metrics, as shown in Table 4. In this work, we emphasize flexibility between maximum throughput and minimum latency and have arranged Table 4 to highlight these tradeoffs. However, not all works reported both the lowest latency and the highest throughput, which may occur for different codeword configurations. Many of the commercial product briefs do not specify the exact LDPC code used or the number of iterations used to perform the decoding. While we do specify the implementation platform of each in the table, it is not entirely possible to fairly compare each without knowing the exact resource utilization and power consumption of each. This table is presented to provide a relative idea of performances in the space for context. However, it should not be used to draw hard conclusions about how our decoder compares to others.

In Table 4, we restrict our comparison to 5G implementations. At the time of this submission, there are no ASIC implementations in the open literature. However, there are reference ASIC implementations for other standards, such as Wi-Fi. For an overview of ASIC implementations for other standards, see [24].

OAI provides a software-based implementation and reports results in [25]. This result was verified on the same platform and a corresponding latency was measured. The LDPC code

TABLE 4. Comparison of NR LDPC decoders.

		Low-Latency Configuration		High-Throughput Configuration	
	Platform	Latency (μs)	Throughput (Mbps)	Latency (μs)	Throughput (Mbps)
This work	TITAN RTX	87	290	700	3964
[25]	Intel i7-6700K	177.7 [1]	30	177.7	30
[8]	Xeon Gold 6154. 1 Core.	31.08	271.80	31.08	271.80
[8]	Xeon Gold 6154. 18 Cores.	NA	NA	NA	5000
[26]	i7-4770	240	NA	NA	NA
[27]	Unknown FPGA	NA	NA	NA	574
[28]	2080Ti GPU	NA	NA	NA	1380

used was the (25344, 8448) code from BG1 with 5 iterations. [8] also used a software-based solution. However, this work also takes advantage of the AVX SIMD instructions for the architecture. When using a single core, they can achieve a performance of 271.8 Mbps throughput with 31.08 μs latency for 10 iterations of a layered decoder. When using 18 cores, they report a max throughput of 5 Gbps. [26] also implements a software-based decoder for OAI. a layered min-sum decoder is adopted using AVX-256 instructions. Using an i7, a decode time of 240 μs is reported for an unspecified code and number of iterations. Creonic provides an FPGA-based, NR, LDPC decoder accelerator that is used throughout the OAI community. In their product brief, they specify a max throughput of 574 Mbps for an unspecified FPGA and LDPC code configuration [27]. [28] provides another GPU benchmark for an Nvidia 2080Ti for the (2080, 1760) code using 10 iterations of a layered min-sum decoder. They achieve a max throughput of 1380 Mbps with unspecified latency.

V. VRAN DISCUSSION AND OAI INTEGRATION

vRAN will be a key technology going forward. In this section, we discuss vRAN and show how GPU solutions could be used for a variety of baseband computational tasks. As an example, we then discuss the integration of the our GPU-based LDPC decoder into OAI.

A. THE NEED FOR SOFTWARE-BASED, VRAN SYSTEMS

Software-based solutions excel at flexibility, and the development of standards-compliant SDR projects is an emerging theme that highlights this. For example, OAI and srsLTE are examples of software-defined platforms for 4G and 5G systems [29], [30]. Through these systems, one can now run 4G base stations and core networks entirely on commodity CPUs while supporting multiple commercially available off-the-shelf (COTS) UEs. This software-based approach can be rapidly developed and tested and provides an opportunity for researchers to experiment with new algorithms for any task in the stack.

Software solutions such as OAI are a more natural fit to possible future deployments based on C-RAN, where operators perform baseband processing at a central cloud data center on enterprise-grade servers.

Recently, an extension to C-RAN called vRAN has been proposed [31], [32]. We illustrate this concept in Fig. 8. Here, a cloud compute center provides a baseband processing accelerator pool that can be leveraged by multiple remote radio heads and multiple RAT standards. In vRAN, virtualization is added to the compute center. By virtualizing the baseband functions, the hardware resources can be decoupled from processing tasks. Many telecommunication companies are involved in vRAN efforts through the O-RAN Alliance which is pushing for RAN virtualization through the development of new standards and software [33].

Virtualization provides many benefits such as improved system management, enhanced error resilience, better scalability, and reduced costs. By decoupling the hardware resources from the computation, it becomes possible for a more dynamic orchestration of the available resources based on real-time network demands and constraints. This decoupling translates directly to better resilience. Whenever some hardware resources fail or are taken offline for maintenance or upgrades, the orchestration management scheme can use other available resources.

Scalability becomes a matter of adding new hardware resources to the pool of available resources. It also becomes possible to scale across multiple standards by adding the appropriate software and connecting it to the hardware resource pool. C-RAN/vRAN systems also are economical since enterprise-grade servers and other widely available COTS equipment performs the processing tasks instead of expensive, highly specialized equipment. Beyond the above benefits, centralized processing can allow for new algorithms and architectures such as coordinated multipoint (CoMP) and cell-free.

These future vRAN systems will likely be heterogeneous computing clusters including CPUs, FPGAs, application-specific integrated circuits (ASICs), GPUs, and even tensor processing units (TPUs). CPUs may be used for serial control tasks such as scheduling and medium access control (MAC) functionalities. FPGAs and ASICs can accelerate specific tasks such as the precoding and detection. GPUs can fill the performance/flexibility gap for any tasks that need both. GPUs can also provide high performance for any algorithms where dedicated FPGA/ASIC accelerators have not been developed. In next-generation applications, machine learning has been proposed for a variety of communications tasks

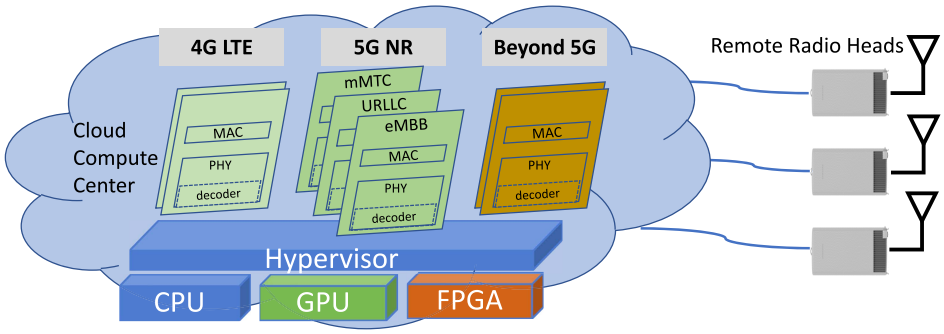


FIGURE 8. Illustration of the flexible, multi-standard vRAN concept. A cloud-compute center can be used in beyond 5G to provide high-performance, baseband processing in a virtualized manner across various hardware accelerators, including GPU and FPGA.

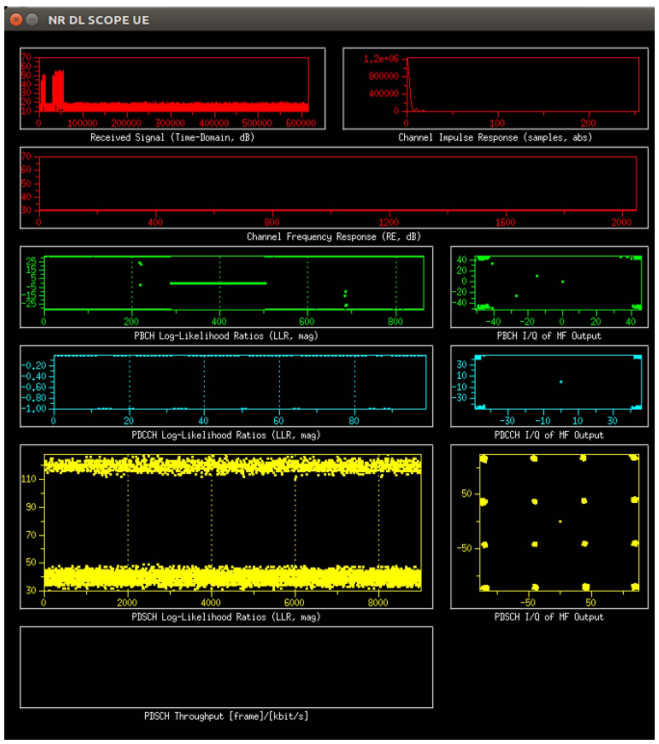


FIGURE 9. OAI's NR Scope showing an example of the constellations for each physical channel.

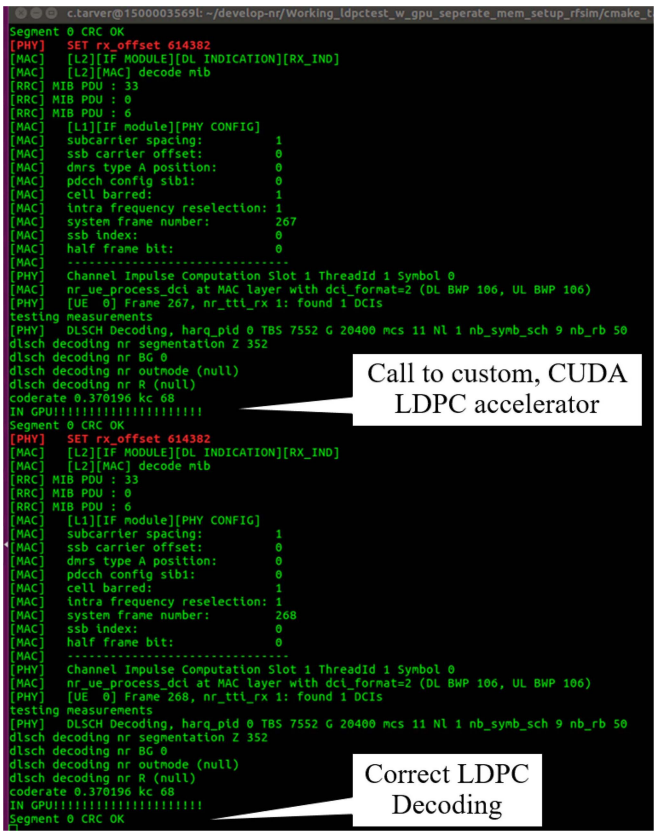


FIGURE 10. Example calling the custom, LDPC accelerator from the OAI ldpctest.

and TPUs can be available to provide dedicated processing resources for such tasks.

B. OAI INTEGRATION

Our GPU software is designed to be compiled into a generic C++ library that can accelerate SDR platforms for a vRAN system. To highlight this portability to SDR platforms, we integrated our decoder into OAI. The integration is done by including our CMAKE file in the master CMAKE and updating function calls to the decoder.

We evaluated the performance of our GPU decoder in two scenarios. First, we tested with the OAI standalone "ldpctest" target. The "ldpctest" evaluates BER performance and latency. When testing OAI's current C-based LDPC decoder in the "ldpctest," the best decoding latency achieved was 178 μs In contrast, our GPU-based decoder had a latency of 87 μs, a 51% reduction in latency. With the BER data from this test, we were able to verify performance and collect the data used in Fig. 3.

We then ported the GPU-based LDPC decoder to the larger "RFSimulator" target, which includes the entire 5G NR protocol stack. When testing with the full 5G stack, we were able to integrate with OAI and verify that the CRC passed as expected. Two screenshots from the OAI tests are shown. First, in Fig. 9, we show OAI's NR display with each of the constellations for the physical channels. Fig. 10 shows a screenshot of the terminal while running OAI. For this test, individual segments are passed to the GPU for decoding.

Our top-level GPU broker code manages to take in new codewords and to launch the kernels to run them. Here, we add a message for the log file indicating that the function call to the GPU LDPC accelerator was successful. When the decoded information bits are passed back to OAI, a CRC check is performed for HARQ. In Fig. 10, we emphasize the CRC check passing after the call to the GPU.

VI. CONCLUSION

In this work, we presented a flexible, 5G NR compliant LDPC decoder for GPU that was targeted for enabling 5G and beyond-5G vRAN. The GPU-based solution, being software-based, is shown to have the flexibility needed to support all possible configurations of LDPC for 5G and beyond. Moreover, by adjusting system parameters, the GPU can be configured to target codeword throughput or latency depending on the needs of the base station. For future work, we plan on extending the vRAN testbed to include more UEs and gNBs to explore hardware orchestration.

REFERENCES

[1] International Telecommunication Union. (Nov. 2017). *Minimum Requirements Related to Technical Performance for IMT-2020 Radio Interface(s)*. [Online]. Available: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf

[2] K. Li, "Decentralized baseband processing for massive MU-MIMO systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Rice Univ., Houston, TX, USA, 2019.

[3] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson, and L. Grosjean, "Channel coding in 5G new radio: A tutorial overview and performance comparison with 4G LTE," *IEEE Veh. Technol. Mag.*, vol. 13, no. 4, pp. 60–69, Dec. 2018.

[4] M. Wu, G. Wang, B. Yin, C. Studer, and J. R. Cavallaro, "HSPA;/LTE-A turbo decoder on GPU and multicore CPU," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2013, pp. 824–828.

[5] G. Wang, M. Wu, Y. Sun, and J. R. Cavallaro, "A massively parallel implementation of QC-LDPC decoder on GPU," in *Proc. Symp. Appl. Specific Processors (SASP)*, 2011, pp. 82–85.

[6] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 309–322, Feb. 2011.

[7] G. Wang, M. Wu, B. Yin, and J. R. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Proc. IEEE Glob. Conf. Signal Inf. Process.*, Dec. 2013, pp. 1258–1261.

[8] Y. Xu, W. Wang, Z. Xu, and X. Gao, "AVX-512 based software decoding for 5G LDPC codes," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, 2019, pp. 54–59.

[9] (Jun. 2020). *5G LDPC Intel FPGA IP User Guide*. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/ond1481066696968.html

[10] (Dec. 2019). *LDPC Encoder Decoder v2.0 LogiCORE IP Product Brief (PB052)*. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ldpc/v2_0/pb052-ldpc.pdf

[11] A. Balatsoukas-Stimming and C. Studer, "Deep unfolding for communications systems: A survey and some new directions," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, 2019, pp. 266–271.

[12] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2017, pp. 1361–1365.

[13] L. Lugosch and W. J. Gross, "Learning from the syndrome," in *Proc. 52nd Asilomar Conf. Signals Syst. Comput.*, 2018, pp. 594–598.

[14] S. Velayutham. (Oct. 2019). *NVIDIA Aerial Software Accelerates 5G on Nvidia Gpus—NVIDIA Blog*. [Online]. Available: https://blogs.nvidia.com/blog/2019/10/21/aerial-application-framework-5g-networks/

[15] Nvidia. (Jul. 2020). *5G CloudRAN and Edge AI End-to-End System Featuring NVIDIA Aerial SDK and EGX Platform*. [Online]. Available: https://news.developer.nvidia.com/5g-cloudran-and-edge-ai-end-to-end-system-featuring-nvidia-aerial-sdk-and-egx-platform/

[16] (Nov. 2019). *OAI Develop-nr Gitlab Repository*. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/tree/develop-nr

[17] C. Tarver, M. Tonnemacher, H. Chen, J. C. Zhang, and J. R. Cavallaro, "GPU-based LDPC decoding for vRAN systems in 5G and beyond," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2020, pp. 1–5.

[18] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[19] *NR; Multiplexing and Channel Coding Version 15.7.0*, 3GPP Standard TS 36.212, Sep. 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214

[20] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 28–34, Mar. 2018.

[21] (2020). *NVIDIA A100 Tensor Core GPU Architecture*. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf

[22] *Mellanox OFED GPUDirect RDMA*. Accessed: Aug. 9, 2020. [Online]. Available: https://www.mellanox.com/products/GPUDirect-RDMA

[23] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low density parity check codes decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 4, May 2001, pp. 202–205.

[24] S. Shao *et al.*, "Survey of turbo, LDPC, and polar decoder ASIC implementations," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2309–2333, 3rd Quart., 2019.

[25] F. Kaltenberger. (Sep. 2019). *5G-NR-Development-and-Releases*. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/5g-nr-development-and-releases

[26] W. Ji, Z. Wu, K. Zheng, L. Zhao, and Y. Liu, "Design and implementation of a 5G NR system based on LDPC in open source SDR," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2018, pp. 1–6.

[27] Creonic. (2019). *5G LDPC Decoder and HARQ Buffers Product Brief*. [Online]. Available: https://www.creonic.com/wp-content/uploads/PB_Creonic_5G_RL15_Decoder.pdf

[28] R. Li, X. Zhou, H. Pan, H. Su, and Y. Dou, "A high-throughput LDPC decoder based on GPUs for 5G new radio," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2020, pp. 1–7.

[29] N. Nikaein *et al.*, "OpenAirInterface: A flexible platform for 5G research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2677046.2677053

[30] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An open-source platform for LTE evolution and experimentation," in *Proc. 10th ACM Int. Workshop Wireless Netw. Testbeds Exp. Eval. Characterization*, 2016, pp. 25–32.

[31] Wind River. Nov. 2017. *vRAN: The Next Step in Network Transformation*. [Online]. Available: https://builders.intel.com/docs/networkbuilders/vran-the-next-step-in-network-transformation.pdf

[32] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vRAN/MEC orchestration," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2366–2374.

[33] O-RAN Alliance. *O-RAN Use Cases and Deployment Scenarios*. Accessed: Aug. 9, 2020. [Online]. Available: https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5e95a0a306c6ab2d1cbca4d3/1586864301196/O-RAN+Use+Cases+and+Deployment+Scenarios+Whitepaper+February+2020.pdf

CHANCE TARVER (Member, IEEE) received the B.S. degree in electrical engineering from Louisiana Tech University, Ruston, LA, USA, in 2014, and the M.S. degree in electrical and computer engineering from Rice University, Houston, TX, USA, in 2016, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. He has been with Samsung Research America, Plano, TX, USA, since 2020, as a Wireless Systems Engineer. His current research interests include computationally efficient implementations of a variety of signal processing algorithms for next-generation wireless communications.

MATTHEW TONNEMACHER received the B.S., M.S., and Ph.D. degrees in electrical engineering from Southern Methodist University, Dallas, TX, USA, in 2011, 2013, and 2019, respectively. He has been with Samsung Research America, Plano, TX, USA, since 2016, as a Wireless Systems Engineer. His current research interests include spectrum sharing, software-defined radio, and next-generation wireless technologies.

HAO CHEN received the B.S. and M.S. degrees in information engineering from Xi'an Jiaotong University, Xi'an, in 2010 and 2013, respectively, and the Ph.D. degree in electrical engineering from the University of Kansas, Lawrence, KS, USA, in 2017. Since 2016, he has been a Research Engineer with the Standards and Mobility Innovation Laboratory, Samsung Research America. His research interests include network optimization, machine learning, and 5G cellular systems.

JIANZHONG (CHARLIE) ZHANG (Fellow, IEEE) received the Ph.D. degree from the University of Wisconsin–Madison, Madison, WI, USA. He is the SVP and the Head of the Standards and Mobility Innovation Lab, Samsung Research America, where he leads research, prototyping, and standards for 5G and future multimedia networks. From 2009 to 2013, he served as the Vice Chairman of the 3GPP RAN1 Working Group and led development of LTE and LTE-advanced technologies, such as 3-D channel modeling, UL-MIMO, CoMP, and carrier aggregation for TD-LTE.

JOSEPH R. CAVALLARO (Fellow, IEEE) received the B.S. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1981, the M.S. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 1982, and the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY, USA, in 1988. From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ, USA. In 1988, he joined the Faculty of Rice University, Houston, TX, USA, where he is currently a Professor of Electrical and Computer Engineering and the Associate Chair. From 1996 to 1997, he served with the U.S. National Science Foundation as the Director of the Prototyping Tools and Methodology Program. He was a Nokia Foundation Fellow and a Visiting Professor with the University of Oulu, Finland, in 2005 and continues his affiliation there as an Adjunct Professor. He is currently the Director of the Center for Multimedia Communication, Rice University. His research interests include computer arithmetic, and DSP, GPU, FPGA, and VLSI architectures for applications in wireless communications. He is an Advisory Board Member of the IEEE SPS TC on Design and Implementation of Signal Processing Systems and the Chair of the IEEE CAS TC on Circuits and Systems for Communications. He was an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING and the IEEE SIGNAL PROCESSING LETTERS, and currently serves as an Associate Editor for the *Journal of Signal Processing Systems*. He was the General/Program Co-Chair of the IEEE International Conference on Application-Specific Systems, Architectures and Processor in 2003, 2004, and 2011, and the General/Program Co-Chair for the ACM/IEEE GLSVLSI conferences in 2012 and 2014. At the IEEE SiPS workshop, he was the TPC Co-Chair in 2016 and the General Co-Chair in 2020. At the IEEE Asilomar Conference on Signals, Systems, and Computers, he was the TPC Chair in 2017 and the General Chair in 2020. He served on the IEEE CAS Society Board of Governors in 2014.