Implementing vRANs on an Arm and GPU system

1st Romain Beurdouche
Communication Systems dept.
EURECOM
Biot, France
romain.beurdouche@eurecom.fr

2nd Raymond Knopp
Communication Systems dept.
EURECOM
Biot, France
raymond.knopp@eurecom.fr

***Abstract*—This work describes our effort to enable the deployment of virtualized Radio Access Networks (vRANs) using the OpenAirInterface 5G software stack on top of a computer architecture based on an Arm processor and a Graphical Processing Unit (GPU). We enhanced the software stack to support this system and leverage its capabilities. We implemented a deployment of several end-to-end vRANs in order to challenge the system with our enhanced stack. We share our observations on this setup, its performance and the behaviors we faced. We discuss the limitations of the software stack and how they can be addressed but also consider how alternative implementations could be used to achieve better results.**

***Index Terms*—5G, vRAN, O-RAN, Distributed Unit, Baseband Processing, GPU, OpenAirInterface, Open Source**

I. Introduction

As the concept of virtualized and open Radio Access Network (RAN) was flourishing, silicon industrials such as Intel, AMD or NVIDIA released several solutions providing the computing capabilities to efficiently implement 5G RANs while addressing the requirements in term of flexibility and scalability of virtualized RANs (vRANs). These solutions can rely on several building blocks such as diverse types of General-Purpose Processors (GPP) with AMD64 or Arm64 architectures or hardware accelerators such as Graphical Processing Units (GPU) or application-specific accelerators.

Open-source 5G software stacks also turn to implementing vRANs following the guidelines of the Open RAN (O-RAN) Alliance [1], [2]. FlexRAN [3], which stands as a reference vRAN implementation, was already used on top of Intel vRAN accelerators [4]. Other stacks also achieved end-to-end vRAN deployments: OCUDU [5] or OpenAirInterface [6]. The latter can leverage Intel's and AMD's vRAN computer architectures [7].

The variety of architectures already supported by the OpenAirInterface stack makes it ideal to compare the performance of different devices. In this work, we describe our effort for adding support in the OpenAirInterface stack for the NVIDIA Grace Hopper 200 (GH200) system, which is based on an Arm processor and a Graphical Processing Unit (GPU), and adding a new channel coding implementation on GPUs. Using this system primarily designed for AI workloads to implement vRAN computing was considered as an optimization for edge computing infrastructures including AI computing capabilities. The same converged infrastructure may be used for hosting AI workloads and vRAN workloads without affording separate computing capabilities for vRANs [8]. Understanding the feasibility and potential of such an architecture requires to know to which extend and with which performance this GH200 system can be used to implement vRANs.

Then, our purpose was to estimate whether the physical layer of the OpenAirInterface stack is fit or can be made fit for implementing vRANs on top of the GH200 system or similar systems. We demonstrate that, on the one hand, our work on the stack enables to implement functional small-cell vRAN instances with decent performance. We also observe, on the other hand, that the scenarios we could implement are rather limited due to some bottlenecks that we identified in the stack. We conclude that enabling reliable and efficient vRAN deployments on the GH200 system with the OpenAirInterface stack would require significant changes in the software architecture of this stack. We also wish to share the lessons learned from working with an open-source 5G software stack on top of an Arm processor and of a GPU with the hope that this experience will be useful to fellow implementers willing to bring up similar setups.

As a first step, Section II gives elements of background about the vRAN architecture and its requirements in terms of computing. It also introduces the target GH200 device and gives an overview of the pre-existing capabilities of the OpenAirInterface stack. Section III gives a summary of our enhancements in the OpenAirInterface stack. In section IV, we challenge our enhanced stack on the GH200 system with a dense deployment of vRANs [7] and observe the capabilities and limits of the stack. The deployment consists in several 5G FR1 small-cell instances sharing a single host system. Finally, in section V, we summarize the capabilities and limitations of the stack to identify necessary improvements and discuss the relevance of using the OpenAirInterface stack on top of the GH200 system.

II. Background

A. vRAN High-PHY

In the O-RAN architecture, the RAN protocol stack is scattered across many network functions. The Radio Unit (O-RU) on the cell site implements the lower physical (Low-PHY) layer while the Distributed Unit (O-DU) located at the edge, close to the cell sites, implements the upper physical layer (High-PHY) and some other layers above. The RAN protocol

stack is therefore split at a determined point known as the 7.2 split where a standard network interface known as the O-RAN 7.2 fronthaul [9] ensures the connection of the O-DU and O-RU and of the two halves of the physical layer. High-PHY corresponds to the part of the RAN protocol stack located between the top of the physical layer and the 7.2 split. It operates under tight real-time constraints to respect the physical transmission timing over the air. By seating in an edge location, the O-DU is the first network function up from the cell site that can be virtualized in a shared edge infrastructure. An O-DU can share the infrastructure with other O-DUs, other user-plane network functions and local end services. The High-PHY layer consists of a digital signal processing pipeline with a downlink (DL) and an uplink (UL) direction. Channel coding is a part of High-PHY that relies on Low-Density Parity-Check (LDPC) codes [10] to fix transmission errors introduced by the channel. Its UL component, channel decoding is the dominant computing consumers in UL High-PHY. Its DL counterpart, channel encoding is one of the dominant computing consumers in DL High-PHY together with precoding, which reckons DL data per antenna port to generate radio beams [11]. The computing weight of different components also depends on the balance of UL and DL configured on the network.

B. Target system

Our target system is the Grace Hopper 200 Superchip from NVIDIA [12]. This platform can be deployed for implementing at the same time mobile networking and local end applications that could involve AI [8].

The device relies on the combination of the 'Grace' Arm processor and 'Hopper' GPU. The 'Grace' processor features 72 Arm Neoverse V2 cores operating at a frequency of 3.0GHz. The 'Hopper' 200 GPU is interfaced with the processor via a 'NVLink-C2C'. The system features a NVIDIA BlueField-3 Data Processing Unit (DPU) [13] interfaced via PCIe gen5 x16 and which can act as a simple Network Interface Card (NIC). It connects to our fronthaul network with a 100 Gb/s QSFP fiber link. The system runs Ubuntu 22.04 with kernel 6.8.0-1025-nvidia-64k as host Operating System.

SIMD: The cores of the 'Grace' processor feature Single-Instruction Multiple-Data (SIMD) extensions. SIMD extensions add instructions that perform some operations on data vectors with large size holding multiple smaller size variables. Different SIMD extensions exist for different processor architectures. For instance, AMD64 processors commonly feature SIMD extensions of the AVX family which offer operations on vectors up to 512 bits large in the AVX512 family [14]. The Neoverse V2 Arm cores of the 'Grace' processor feature several Arm SIMD extensions such as Neon, SVE and SVE2 [15].

C. Initial situation

vRANs with OpenAirInterface: The OpenAirInterface stack already enabled to deploy vRANs on top of a variety of computer architectures. The O-RAN 7.2 fronthaul was available and commonly used on servers based on AMD64 processors, the platform usually used to host this stack. The High-PHY layer could execute entirely on an AMD64 processor with or without the help of a hardware accelerator for channel coding. To implement the 7.2 Fronthaul, the stack relied on the Data-Plane Development Kit (DPDK) and on the reference implementation of the interface from the O-RAN software community [16]. The stack is used to implement vRANs for experimental setups but it was also demonstrated in dense vRAN deployments trying to achieve a high resource usage efficiency in a mobile-broadband small-cell scenario [7].

Aerial cuPHY: OpenAirInterface was also used on top of the GH200 system as a Virtual Network Function (VNF), on top of a High-PHY layer or Physical Network Function (PNF) from NVIDIA: Aerial cuPHY [17]. In this approach, many cells can share the same PNF and the High-PHY processing is entirely offloaded to the GPU. This is a form of *inline* acceleration. The entire stack is offloaded to a processing unit or accelerator — the GPU in this case — below a given point in the stack.

It is to be distinguished from *look-aside* acceleration which consists in offloading a chosen component of the stack to an accelerator and returning the result to the host processor. This is the approach for channel coding acceleration in OpenAirInterface High-PHY. Another difference with Aerial cuPHY is that OpenAirInterface High-PHY relies on one High-PHY instance per cell. With multiple cells, multiple High-PHY instances execute concurrently and share the computing resources.

Limitations: The OpenAirInterface High-PHY layer was not supporting the GH200 platform. Support of Arm based host systems had to be added in OpenAirInterface High-PHY. With regards to using GPUs, an implementation of LDPC coding using CUDA was distributed with the stack. But it was not actively maintained and was actually not functional in an end-to-end network.

III. Enhancement of the OpenAirInterface RAN stack

We contributed a few enhancements in the OpenAirInterface stack in order that it supports the GH200 system. Granting a basic support of the platform mostly required to replace sections of the stack that relied on SIMD instructions of the AVX family [14]. Then, we also added a new offload of channel coding to the GPU.

These enhancements of the stack are available from the OpenAirInterface public code repository as mainstream feature from the tag '2026.w34' of August 2026. Although this work mostly focuses on the GH200 system, our contribution to the stack enable using a variety of systems featuring diverse Arm processors or GPUs to implement vRANs or User Equipments (UE).

A. Support of the Arm processor

O-RAN 7.2 Fronthaul: The reference implementation of the O-RAN 7.2 fronthaul interface from the O-RAN software community [16], on which relies the OpenAirInterface stack,

uses AVX in order to efficiently perform payload compression over the interface. Fortunately, the Arm RAN Acceleration Library (ArmRAL) [18] provided an alternative implementation of data compression for the O-RAN 7.2 fronthaul interface optimized for Arm systems. We added this implementation to the stack to enable the O-RAN 7.2 fronthaul on Arm systems.

SIMD in High-PHY: The OpenAirInterface High-PHY layer also heavily relies on AVX for some of its routines. It uses AVX intrinsics [14], a series of C functions that enables to use directly the AVX instructions without leaving the C compiler to decide to resort to these instructions or not. These intrinsics give low-level control on the series of instructions that is executed. Then, some routines of the OpenAirInterface stack had to be rewritten to use Arm SIMD extensions instead of AVX. This was done mostly by using the SIMD everywhere (SIMDe) library [19] which provides a series of functions corresponding to the AVX intrinsics but implemented with SIMD instructions available on the target architecture. Most of the AVX intrinsics in the stack only had to be slightly modified to call the corresponding SIMDe function. But the SIMDe library did not provide equivalent functions for all AVX intrinsics and was not providing low-level control on the final series of instructions that was executed. Then, some parts of the stack had instead to be rewritten with Arm SIMD intrinsics.

B. Channel coding offload

As explained previously, channel coding with the LDPC code is the most important computation consumer in High-PHY UL and one of the most important in DL. The computing demand of channel decoding can be addressed by leveraging ASICs or GPUs. The DecodeX [20] benchmark demonstrated that the Hopper 200 GPU with the Aerial software stack achieves comparable performance as state-of-the-art ASICs at a comprehensive set of LDPC decoding tasks.

According to our own testing, when channel decoding is executed on the 'Grace' processor of the GH200 system, at least 4 cores are necessary to support the UL throughput in a full-stack 100MHz numerology 1 cell with 1 UL layer and modulation order 8 and a maximum number of LDPC iterations set to 15. Over one second of recording of channel decoding times, we measure an average decoding time of 1189µs per full UL slot and a maximum of 1765µs.

In order to reduce this processing time and save processor cores, we added a new implementation of LDPC coding on the GPU in the OpenAirInterface stack. Then, we compared decoding and encoding times on the same end-to-end test as above with offloading to the GPU. The reduction of the decoding time enabled by the GPU is significant with new measures giving an average of 443µs and a maximum of 455µs. The decoding time is not only lower on average but the higher percentiles are also closer to median values.

There is, on the other hand, no significant improvements regarding the encoding time. The encoding time on GPU is very close to the time on the processor with average times of 176µs on the processor and 174µs on the GPU and maximums of 205µs on the processor and 182µs on the GPU for encoding a DL full slot with 4 DL layers and a modulation order of 8. Nevertheless, the offload of channel encoding can be used to remove a workload from the processor and better use the GPU without introducing any significant overhead. Then, we will offload the whole channel coding to the GPU in the following experiment.

IV. Experimental vRAN deployment

We implemented a deployment of several small-cell 5g O-DUs under traffic load sharing a single GH200 system. We evaluated the ability of the O-DUs to sustain a given mobile-broadband Quality of Service (QoS). We monitored as well its computing resource and energy usage. We do not try to perform an exhaustive evaluation of the GH200 system and OpenAirInterface stack with this deployment. The objective of this experiment is rather to observe their behavior under operational conditions to learn about their strength, weaknesses and fitness for some use-cases.

A. Implemented deployment

Our setup didn't offer enough radio appliances to accommodate many O-DU instances. Fortunately, the OpenAirInterface stack offers a 'phy-test' mode that emulates the connection of a User Equipment (UE). This 'phy-test' mode can be used together with an O-RAN 7.2 fronthaul executing as it would with an O-RU connected but without a physical connection to an O-RU. It only requires an active network interface. Then, each instance has an O-RAN 7.2 fronthaul, but only one instance connects to a real O-RU. The instance with a real O-RU enables to assess the network performance with end-to-end tests. The other instances execute in 'phy-test' mode to create a fair load on the system. All the O-DUs operate with 4 DL antennas and 4 UL antennas, a bandwidth of 100 MHz on frequency band n77 and numerology 1, corresponding to a Transmission Time Interval (TTI) of 500 µs. DL and UL traffic separation is enforced through Time Domain Division (TDD). For each period of 5 slots, there are 3 slots dedicated to DL traffic, one slot shared between DL and UL traffics and one slot dedicated to UL traffic. On the O-DU instances in 'phy-test' mode, the emulated connection of a UE has 4 layers and modulation order 8 in the DL direction and 2 layers and modulation order 6 in the UL direction. Each O-DU instance is executed within a dedicated container. Containerization enforces processor cores partitioning. The cores assigned to the O-DU containers are isolated in the kernel parameters so that they are not considered for scheduling other processes. We use the Bluefield 3 DPU as a NIC for the O-RAN 7.2 fronthaul. This single NIC is shared by all the O-DUs by using Single Root Input & Output Virtualization (SR-IOV) and the VFIO driver for DPDK [21].

We evaluated the ability of the system to provide a typical mobile-broadband QoS for such O-DUs, with end-to-end throughput tests with a physical UE and the O-DU instance with a physical O-RU. The end-to-end DL throughput should reach 1200Mb/s with 4 layers and a modulation order of 8

and the end-to-end UL throughput should reach 90Mb/s with 2 layers and a modulation order of 6.

B. Resource Assignment

Processor resource requirements: Dense deployments of OpenAirInterface [7] require to assign 8 cores to each O-DU instance on AMD64 systems. 6 of these cores are assigned statically to specific tasks within the stack. 4 cores are used as a 'thread pool' to execute worker threads forked out of the main High-PHY processing threads. Then, 2 cores overlap between cores assigned to specific tasks and the thread pool. The assignment of cores is as follow:

- 4 single task cores: DPDK 'IO', DPDK 'worker', 'L1 TX thread' and 'L1 RX thread' cores.
- 2 cores: DPDK 'system' and 'RU thread' cores overlapping with the thread pool.
- 2 cores for thread pool only.

On the 'Grace' processor with channel coding offloaded to the GPU, the same requirements exist but the thread pool only required 3 cores instead of 4 to provide the target QoS. Then, the minimum required resources were 7 processor cores per instance. As we keep 9 cores for the host operating system and other network components hosted on the same system, we could in theory fit up to 9 O-DUs on the 72 cores of the 'Grace' processor

C. Observations

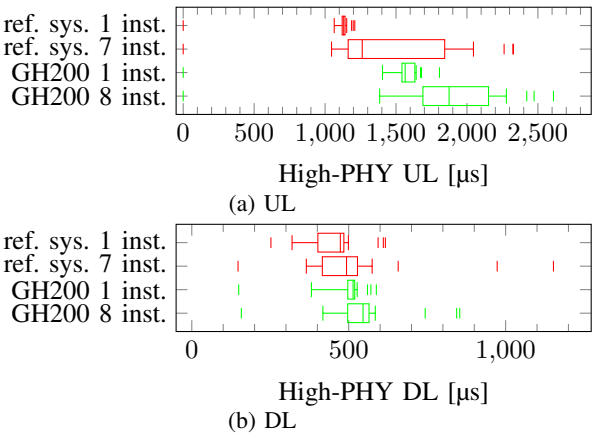
Our experiment shows that our enhanced OpenAirInterface stack can provide the target QoS for a limited number of O-DU instances. We try to explain the limitations of our setup by comparing its behavior with a reference system that we reproduced and on top of which we ran the same experiment. This reference system relied on a 64-cores AMD EPYC processor of the 'Siena' family and on an AMD T2 RFSoC and was already fully supported by the OpenAirInterface stack [7].

Functional O-DU instances: While we expected to host up to 9 O-DU instances on the system after partitioning the processor resource, we did not manage to reliably run this number of instances and provide the target QoS. With 9 instances, the behavior of the O-DU instances is unstable. Some O-DUs may stop working after experiencing real-time deadlines infringement.

Meanwhile, the reference system implemented 7 instances with the target UL and DL throughputs. This is the expected number of instances after partitioning the processor resources of this system in a similar fashion as we did for the GH200.

UL: As Figure 1a shows, the times for processing a UL slot is already rather high for 1 instance with an average of 1443µs to be compared with 1076µs on reference system with 1 instance. The origin of these high UL processing times mostly seats in channel compensation which sets the average time for UL processing excluding channel decoding to 1265µs. Channel compensation is therefore the current main bottleneck of UL processing with this system and stack.

When multiple O-DU instances execute in parallel, we observe a behavior common to both systems. The channel decoding times significantly increase and diverge on both systems as the RFSoC on the reference system and GPU on the GH200 system are shared by many O-DUs.



(a) UL

(b) DL

Fig. 1: Distribution of High-PHY processing time of a slot over 3 seconds of recording. Whiskers are first and last decile, fliers are minimum, 99th and 99.9th percentiles and maximum.

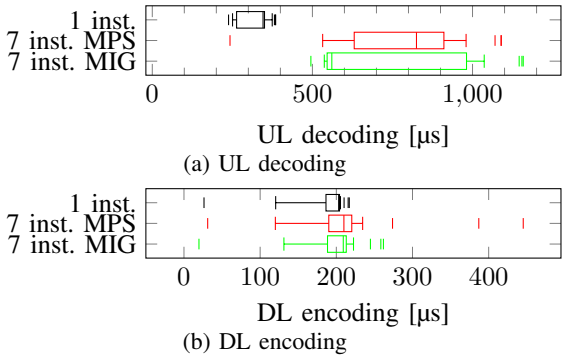
DL: The behavior in the DL direction is to some extend similar to UL. As Figure 1b shows, the processing time of a slot with only 1 instance is still above the times observed on the reference system on average but closer. The average time for processing a DL slot is only 447µs on the reference system while it is 490µs on the GH200 system.

We notice as well that when the systems host many O-DU instances, the higher percentiles in the distribution of processing times significantly increased in the same way as for UL but to a lesser extent. For instance, the maximum time for processing a DL slot increase from 588µs for 1 instance to 844µs for 8 instances on the GH200 system.

On the GH200 system, we also notice signs of saturation of DL processing when 8 instances execute in parallel. First of all, the average recorded processing times of 526µs is rather close to the average time available to process one slot. The O-DU is configured so that 3 full slots and a part of the shared slot are dedicated to DL per 5 slots TDD period while DL processing of one O-DU is managed sequentially on a single DL core. Then, only 625µs are available to process a slot on average without saturating this DL core. We actually noticed that the DL core reports a level of usage of more than 90% when 8 O-DU instances share the system against 75% with only 1 instance on the system. This deterioration of DL processing performance of a single O-DU when the system is shared may explain that we do not manage to host 9 stable O-DUs on the system.

Efficiency: The limitation of the number of O-DU instances on the system and the price of the system determine the infrastructure expense for one instance. Then, the efficiency with regard to infrastructure cost could be better with the GH200 if the OpenAirInterface stack enabled to implement more instances.

Another component of the efficiency of a network is the power consumption per instance. We are able to measure the intake electrical power at the power supply on the GH200 and



(a) UL decoding

(b) DL encoding

Fig. 2: Distribution of time for channel coding of a slot over 3 seconds of recording. Whiskers are first and last decile, fliers are minimum, 99th and 99.9th percentiles and maximum.

reference systems through a standard Redfish [22] management interface. The reference system displays a maximum total consumption of 41W per instance. Meanwhile, the GH200 system consumes at most 787W with 8 instances which makes 98W per instance. There is then a wide margin for improvement. Assuming a total consumption of 787W of the GH200 system, at least 20 instances should be implemented to match the consumption per instance of the reference system.

D. GPU sharing & Real-Time

As we experienced with both the reference and GH200 system, sharing a HA between many O-DUs is not a trivial problem as it may affect significantly the processing times and their predictability. But the GH200 system has the specificity that it offers several different mechanisms to distribute the GPU resources between several O-DUs.

MPS: In the previous experiment, we used the Multi-Process Service (MPS) to distribute the GPU resources. An MPS server distributes shares of the GPU resources to the O-DUs depending on their demands. As we already observed with this approach, the processing times on the GPU may increase arbitrarily. For UL channel decoding, we observed a general increase of the processing times. The average decoding time increased from 311µs with 1 instance to 667µs with 8 instances. The maximum increased from 381µs to 1191µs. For DL channel encoding, the increase is mostly limited to the higher percentiles of the processing time. For instance, the maximum encoding time increased from 233µs to 511µs.

MIG: An alternative to MPS on NVIDIA GPUs is Multi-Instance GPU (MIG). Each application gets exclusive access to an isolated partition of the GPU. We replicated our deployment with 7 O-DUs using MPS in a first trial and MIG in a second trial. As Figure 2b shows, MIG mitigates the increase of the maximum and higher percentiles of channel encoding time with 7 O-DUs.

But, each O-DU instance uses at most one seventh of the GPU resources. These reduced resources were sufficient for encoding to execute seamlessly but, as Figure 2a shows, channel decoding is significantly slowed down. Then, for UL channel decoding, MIG offers no gain against MPS. Another problem of MIG is that the GPU cannot be partitioned in more than 7 GPU instances which limits the number of O-DUs to 7.

Other approaches: We should also mention that other approaches to GPU sharing exist. Some works consider to co-locate RAN workloads and best-effort AI or HPC workloads on a single system [8]. Instead of hosting many O-DUs which cannot be prioritized against each-other, an optimal usage of the system is obtained through co-hosting a low number of O-DUs that have priority and the other best-effort low-priority workloads. Contention for the GPU can be mitigated with a RAN aware MPS-based resource allocation method such as YinYangRAN [23].

V. Discussions

The observations we made in the previous experiment enable us to summarize the capabilities and limitations of the OpenAirInterface stack on the GH200 system. Then, we can reflect on possible improvements to the stack and on the relevance of using this stack on the GH200 system.

Capabilities of the stack: The enhancements we contributed in the OpenAirInterface stack did enable new possibilities with the stack on systems based on Arm processors and GPUs such as the GH200 system:

- An O-RAN DU can be implemented on top of an Arm processor using the OpenAirInterface stack as the O-RAN 7.2 split and OpenAirInterface High-PHY were enabled on this type of processor.
- An end-to-end mobile-broadband small-cell can be implemented on top of the GH200 system with decent supported throughputs of 1200Mb/s in the DL direction and 90Mb/s in the UL direction.
- Several O-DU instances can execute in parallel on a single GH200 system.

This setup can be used to evaluate the OpenAirInterface stack on top of the GH200 system. It is also a solid base for future enhancements of the stack to execute on top of the GH200 system or other Arm based systems, other system leveraging GPUs or even other Reduced Instruction Set Computer (RISC) systems such as RISC-V systems.

Limitations: We saw that DL processing in the OpenAirInterface High-PHY layer becomes saturated by the DL traffic of a small-cell as the number of network instances increases. Most of the DL processing is performed by a single processor core. Only channel coding can be offloaded to a *look-aside* accelerator and only precoding can be processed on multiple processor cores. This design is convenient on other systems supported by the stack but, on the GH200 system, the main DL processing core is saturated. Meanwhile, other processor cores used for executing worker threads of the physical layer are underutilized. The GPU as well seems to be underutilized with reported levels of usage of 66% of the GPU computing capability, 73% of the memory and 34% of the power capability for 8 O-DU instances with traffic.

These clues suggest that DL processing could be better adapted for the GH200 system. More parallelization across the processor cores would certainly improve the behavior of

the stack. The stack could also offload further DL workloads to the GPU. But offloading processing comes at a cost as transfers to the GPU take time. Then, it may be better to offload a whole sequence of the DL processing in the fashion of *inline* acceleration rather than individual computing blocks in the fashion of *look-aside* acceleration. This would require to port an entire sequence of the physical layer to the GPU.

Relevance: Since we consider to resort to *inline* acceleration with the GPU of the physical layer of OpenAirInterface, it may be more relevant to use a physical layer that is already performing *inline* acceleration with the GPU on the GH200 system: NVIDIA's Aerial cuPHY [17]. This physical layer enables to implement several cells. Then, several OpenAirInterface VNFs can sit on top of a single cuPHY instance in order to implement several RAN instances on top of a single GH200 system. Then, the number and performance of cells achievable with such a setup needs to be determined.

Finally, we can also question the relevance of hosting many vRAN instances on a single GH200 system. The point of using the GH200 system for mobile networking was to implement converged systems hosting at the same time AI applications and the vRANs that connect mobile devices to these applications [8]. Then, maximizing the usage of the GH200 by vRANs may not only be hard as we experienced in this work but also pointless. An efficient usage of computing infrastructures may be achieved instead by simply removing vRAN computing infrastructures and scattering vRANs through the same infrastructure that provides AI services. The scenario where one vRAN instance shares a GH200 system with best-effort workloads with a mechanism like YinYangRAN to enable the sharing of the GPU may actually be the most relevant perspective for using the GH200 system for mobile networking [23].

VI. Conclusion

In this work, we demonstrated that we enabled the OpenAirInterface 5G Radio Access Network stack to support the NVIDIA Grace Hopper 200 system for implementing an O-RAN Distributed Unit. We explained how we enabled the O-RAN 7.2 fronthaul and the upper physical layer of the stack to execute on the 'Grace' processor. We also implemented and evaluated the offload of channel coding to the 'Hopper' GPU. Thanks to this enhancement of the OpenAirInterface stack, we have implemented a deployment of several vRANs on top of the GH200 system. We identified that the performance of the vRANs was limited by the architecture of downlink processing in the OpenAirInterface physical layer. This bottleneck can be removed by better leveraging the computing resources on the GH200 system, for example by offloading the physical layer to the GPU. But, if the physical layer shall be offloaded to the GPU, it may be more relevant to use another physical layer implementation such as Aerial cuPHY and OpenAirInterface for the upper layers of the stack. It may also be more relevant to share the system between a vRAN and best-effort workloads for an efficient usage rather than sharing the system between many vRANs.

Acknowledgment

This research is funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101092598 (COREnext) and by the French National Research Agency under the France 2030 label NF-Plateforme XG ANR-22-PEFT-00011.

References

[1] "O-RAN: Towards an Open and Smart RAN," O-RAN Alliance, White Paper, October 2018. [Online]. Available: https://www.o-ran.org/o-ran-resources

[2] *O-RAN Architecture Description*, Technical Specification, O-RAN Alliance, October 2025. [Online]. Available: https://specifications.o-ran.org/download?id=932

[3] Intel, "FlexRAN," September 2025. [Online]. Available: https://github.com/intel/FlexRAN/tree/-c441ed3b3d692457428ee855e9e13d76b0a36167

[4] Z. Qi, C.-H. Tung, A. Kalia, and T. Chen, "Savannah: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources," in *ACM MobiCom '24*, 2024, p. 1500–1514.

[5] "OCUDU Ecosystem Foundation," [Accessed: 22-April-2026]. [Online]. Available: https://ocudu.org/

[6] OpenAirInterface Software Alliance, "openairinterface5g," OpenAirInterface Git Repository, August 2026. [Online]. Available: https://github.com/duranta-project/openairinterface5g/tree/2026.w35

[7] R. Beurdouche and R. Knopp, "A framework for experimental dense vRAN deployments," in *WiNTECH '26*. Austin, TX, USA: ACM, 2026. [Online]. Available: https://doi.org/10.1145/3831662.3844162

[8] A. Kelkar and C. Dick, "NVIDIA Aerial GPU Hosted AI-on-5G," in *IEEE 5GWF 2021*, 2021, pp. 64–69.

[9] *Control, User and Synchronization Plane Specification*, Technical Specification, O-RAN Alliance, October 2025. [Online]. Available: https://specifications.o-ran.org/download?id=952

[10] M. Sy, "Demystifying 5g polar and ldpc codes: A comprehensive review and foundations," *TechRxiv*, vol. 2025, no. 0925, 2025. [Online]. Available: https://www.techrxiv.org/doi/abs/10.36227/-techrxiv.174002467.71268212/v2

[11] X. Foukas and B. Radunovic, "Concordia: teaching the 5G vRAN to share compute," in *ACM SIGCOMM 2021*, 2021, p. 580–596.

[12] *NVIDIA GH200 Grace Hopper Superchip*, [Accessed: 15-May-2026]. [Online]. Available: https://nvdam.widen.net/s/rrgqqnpbz8/grace-datasheet-gh200-grace-hopper-superchip-3773000%20

[13] *NVIDIA BlueField-3 Networking Platform*, [Accessed: 15-May-2026]. [Online]. Available: https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield?lx=LbHvpR&topic=networking-cloud

[14] Intel, "Intel® intrinsics guide," [Accessed: 17-May-2026]. [Online]. Available: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

[15] "SVE," [Accessed: 5-June-2026]. [Online]. Available: https://developer.arm.com/Architectures/Scalable%20Vector%20Extensions

[16] "o-du/phy," [Accessed: 17-May-2026]. [Online]. Available: https://gerrit.o-ran-sc.org/r/admin/repos/o-du/phy,general

[17] NVIDIA, "Nvidia aerial™ cuda-accelerated ran," 2025-2026. [Online]. Available: https://github.com/NVIDIA/aerial-cuda-accelerated-ran

[18] "ral," [Accessed: 21-May-2026]. [Online]. Available: https://gitlab.arm.com/networking/ral/-/tree/main

[19] "simd-everywhere/simde," [Accessed: 17-May-2026]. [Online]. Available: https://github.com/simd-everywhere/simde

[20] Z. Qi, Y. Yao, Y. Li, C.-H. Tung, J. Zheng, D. Zhuo, and T. Chen, "Decodex: Exploring and benchmarking of ldpc decoding across cpu, gpu, and asic platforms," in *HotMobile '26*. New York, NY, USA: ACM, 2026, p. 127–132.

[21] DPDK, "7. Linux Drivers," DPDK Documentation, [Accessed: 17-April-2026]. [Online]. Available: https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html

[22] DMTF, "Redfish®," Redfish website, [Accessed: 17-April-2026]. [Online]. Available: https://www.dmtf.org/standards/redfish

[23] L. L. Schiavo, J. A. Ayala-Romero, A. Garcia-Saavedra, M. Fiore, and X. Costa-Perez, "YinYangRAN: Resource Multiplexing in GPU-Accelerated Virtualized RANs," in *IEEE INFOCOM 2024*, 2024, pp. 721–730.