# Enabling Efficient SIMD Acceleration for Virtual Radio Access Network

Jianda Wang
Electrical and Computer Engineering Department
The University of Texas
Dallas, Texas, USA
jxw174930@utdallas.edu

Yang Hu
Electrical and Computer Engineering Department
The University of Texas
Dallas, Texas, USA
Yang.Hu4@utdallas.edu

## Abstract

Nowadays, the Radio Access Network (RAN) is resorting to Function Virtualization (NFV) paradigm to enhance its architectural viability. However, our characterization of virtual RAN (vRAN) on modern processors depicts a frustrating picture of Single-Instruction Multi-Data (SIMD) acceleration. The data arrangement processes in vRAN software pipeline do not align data for efficient SIMD processing across the pipeline. Specifically, existing data arrangement processes cannot fully utilize the ALU ports in modern processors, which leads to high backend bound and fails to saturate the memory bandwidth between registers and L1 cache. To overcome the overburden, we thoroughly examine the state-of-the-art CPU architecture and find there are idle ports which could be utilized by the process. Motivated by this observation, we propose "Arithmetic Ports Consciousness Mechanism" (APCM) utilizing these idle ports to eliminate the backend bound and saturate the memory bandwidth. The APCM decreases the data arrangement's backend bound from 45% to 3% and promotes its memory bandwidth utilization by 4X-16X. The CPU time of the data arrangement process can be reduced by 67% - 92% and the overall latency of the vRAN packet transmission is decreased by 12% - 20%.

**ACM Reference Format:**
Jianda Wang and Yang Hu. 2021. Enabling Efficient SIMD Acceleration for Virtual Radio Access Network. In *50th International Conference on Parallel Processing (ICPP '21), August 9–12, 2021, Lemont, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3472456.3472477


*ICPP '21, August 9–12, 2021, Lemont, IL, USA*

ACM ISBN 978-1-4503-9068-2/21/08…$15.00
https://doi.org/10.1145/3472456.3472477

## 1 Introduction

To date, the volume and variety of mobile traffic is exploding, driven primarily by bandwidth intensive applications, such as 4K and 3D video, augmented reality (AR), and virtual reality (VR). Communication service providers (CSPs) are under challenge coping with the unprecedented growth and diverse traffic patterns, meanwhile they are under the pressure since the average revenue per user continues to decline in today's fiercely competitive mobile services market. To preserve profit margins, CSPs focus on the Radio Access Network (RAN), which is not only the most expensive part of the mobile network but also the major source of performance problems that affect the customer experience. To achieve the full potential of cost savings, dynamic capacity scaling, better quality of experience (QoE) and rapid instantiation of new services for the network today while laying the foundation for 5G network in the future, CSPs adopt the RAN architectural evolution from the traditional RAN model to the fully virtual RAN (vRAN) implementation.

The introduction of virtualization in RAN transfers the hardware substrates from specific hardware to general x86 processors. Consequently, the signal processing modules for current vRAN platform are implemented as software components on the state-of-the-art x86 server and the computing intensive modules are accelerated by the Single Instruction Multiple Data (SIMD) mechanism supported by the standard x86 architecture. Hence, the architectural support provided by general processors plays a vital role in delivering the smooth vRAN performance. However, to the best of our knowledge, there is no existing work that provides an extensive and in-depth architectural characterization of emerging x86 processors running vRAN platform. We fill the blank in this work and we explore that the current vRAN platform exhibits severe overhead when running on the start-of-the-art x86 architecture.

Our extensive characterization shows that ***the data arrangement process*** utilized to align the data to the SIMD type for the SIMD accelerated modules is remarkably inefficient. Its under-level SIMD port utilization leads to high backend bound and meanwhile fails to saturate the memory bandwidth between registers and L1 cache. Specifically, the

data arrangement process incurs a high backend bound stall across a variety of CPU micro-architectures, including high-end Xeon-class server CPU and Core-class desktop CPU. Besides, the performance overhead caused by such hardware resource under-utilization is non-trivial, and is expected to be significantly deteriorated in next-generation high throughput architectures. However, the existing platform misses the opportunity to fully exploit the micro-architecture resource of the state-of-the-art server infrastructure for the data arrangement process to mitigate the packet's transmission latency.

Specifically, the data arrangement process constitutes a significant portion of SIMD-related tasks, while the data arrangement process in existing vRAN platform only adopts SIMD data movement instructions. As a result, the overwhelming demands of the SIMD data movement instructions can easily saturate the load and store ports in emerging CPUs, and significantly hurts the memory bandwidth between the registers and L1 cache and packet latency of vRAN platform. Considering future processor architecture will extend the width of registers (e.g. larger than 512 bit in next-generation Intel processor and 4K bit in GPU) and pour more data into L1 cache through load and store ports, current hardware port management will not be sustainable in the next-generation vRAN platform since the SIMD data movement can account for more that 50% of the CPU time.

Our observation motivates an architectural-aware data arrangement process that better leverages the available ALU ports and load/store ports in emerging CPU architecture to reduce the data movement overheads for vRAN. Since the throttled memory bandwidth in load and store ports is mainly caused by the intensive CPU computation spent on data re-organization, we propose an instruction-level data arrangement mechanism named "Arithmetic Ports Consciousness Mechanism" (APCM) that optimizes the data organization for the SIMD data arrangement. Interestingly, the under-utilization of the available ALU ports for the SIMD calculation instructions genuinely motivates us to strategically offload the data re-organization computation to ALU ports which are idle during SIMD data movement. The APCM effectively reduces the pressure of load/store ports and increases the utilization of ALU ports without modifying the hardware architecture. We evaluate the APCM mechanism and our results demonstrate that the APCM can reduce the backend bound from 45% to 3% and improve memory bandwidth utilization by 4X-16X. The CPU time of the data arrangement process can be reduced by 67% - 92% and the overall latency of the RAN packet transmission will be decreased by 12% - 20%.

Summarily, we make the following contributions:

- We demonstrate a thorough profiling for the vRAN platform through the detailed architectural characterization. Our extensive characterization shows that the data arrangement process utilized to align the data to the SIMD type for the SIMD accelerated modules is remarkably inefficient. Its under-level SIMD port utilization leads to high backend bound and meanwhile fails to saturate the memory bandwidth between registers and L1 cache. This will become a severe hotspot when further extending the width of the registers.
- To optimize the data arrangement process, we propose a new mechanism "APCM" which utilizes idle ALU ports to eliminate the high backend bound and meanwhile promote the memory bandwidth utilization efficiency. The APCM mechanism decreases the backend bound from 45% to 3% and promotes memory bandwidth utilization by 4X-16X for the data arrangement process. The CPU time of the process can be reduced by 67% - 92% and the overall latency of the vRAN packet transmission is decreased by 12% - 20%.

## 2 Background

**vRAN platforms:** Nowadays, there are several mainstream vRAN platforms, such as Intel FlexRAN[10], Wibench[24], srsLTE[3], and OpenAirInterface[14]. Among them, Intel FlexRAN is the Intel released Software Development Kit (SDK) modules, which is the most highly optimized libraries for LTE and 5G NR Layer 1 workload. However, the Intel FlexRAN does not contain a comprehensive top-down framework. FlexRAN lacks the the data arrangement process for the layer 1 modules and it also does not include the layer 2 and layer 3 modules which are essential to realize the real communication between the base stations and the mobile devices. Compared to the Intel FlexRAN, OpenAirInterface (OAI) is the most complete open-source vRAN experimentation and prototyping platform, which includes layer 1 workloads as well as their data arrangement process code and the complete layer 2 and layer 3 management workloads. The OAI platform includes a full software implementation of mobile cellular systems compliant with 3GPP standards in C under real time Linux optimized for x86. OAI realizes the real end to end (E2E) communication between the base station and the mobile devices. Consequently, we deploy our test-bed on the OAI platform in order to characterize and optimize the vRAN platform top-down instead of concentrating on the performances of the separated workloads.

**Acceleration mechanisms:** Today, there is a trend to explore the acceleration methods for the critical modules of NFV platforms. For example, [8] utilizes the GPU to accelerate the proportional fair scheduling algorithm for the 5G NR platform. [2] proposes the specialized FPGA hardware to speed up the Network Interface Card for the public cloud.

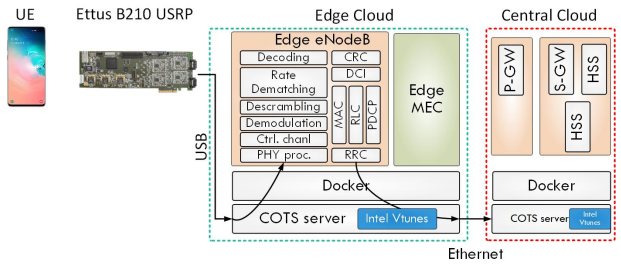



**Figure 1: Typical RAN architecture for the uplink**

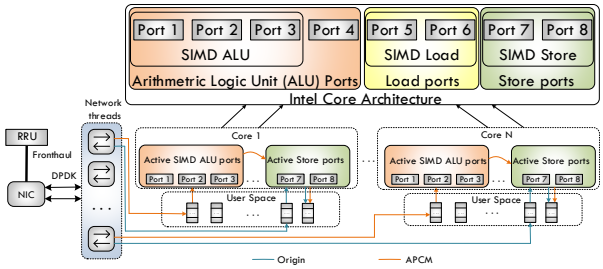



**Figure 2: Intel core micro-architecture and APCM mechanism**

OAI utilizes Single Instruction Multiple Data (SIMD) to accelerate the processing of the network physical layer protocols. The recent popular Data Plane Development Kit (DPDK) mechanism concentrates on the system level optimization, from which the packets can be processed directly on the user space by passing through the kernel space. As shown in Figure 2, DPDK can be deployed by the vRAN platform to accelerate the data transfer process from the NIC to the user space and the SIMD mechanism can be utilized by the vRAN platform to speed up the following protocols computation. In this paper, we focus our analysis on OAI's SIMD optimized top-down framework.

## 3 Methodology

In this section, we first introduce our testbed based on the open-source virtualized RAN framework OpenAirInterface (OAI). We then discuss the COTS CPU architecture and demonstrate the general background of the SIMD acceleration enabled by COTS CPU.

### 3.1 Experimental Platform Overview

We choose the OpenAirInterface (OAI) as our vRAN framework. OAI is the most complete open-source vRAN experimentation and prototyping platform created by EURECOM. The OAI platform includes a full software implementation of mobile cellular systems compliant with 3GPP standards in C under realtime Linux optimized for x86. For the 3GPP Access-Stratum, OAI provides standard-compliant implementations of PHY, MAC, RLC, PDCP and RRC, spanning the entire protocol stack from the physical to networking layer, for both vRAN and UE. For the core network, the OAI provides standard compliant implementations of a subset of 3GPP Evolved Packet Core (EPC) components such as the Serving Gateway (S-GW), the Packet Data Network Gateway (P-GW), the Mobility Management Entity (MME), and the Home Subscriber Server (HSS). Figure 1 shows a typical uplink path and the key network functions in OAI edge and core networks, note that both vRAN and core functions are hosted in containers. **Hardware platform:** As shown in Figure 1, the experimental testbed consists of one/two units of Commercial off-the-shelf (COTS) UE, one unit of vRAN Unit and one unit of EPC. We use Intel Core machines (Core i7-8700 @ 3.20GHz 16GB RAM) for vRAN, Intel Xeon machine (E5405 @ 2.00GHz 4G RAM) for EPC and Huawei Honor 8 as our UE. The testbed is implemented with a real RF front-end (Ettus B210 USRP). **Software platform:** The vRAN version we use is branch 2018-w25. For EPC, the version we use is branch "develop". The operation system used for both machines is Ubuntu 16.04. In order to run the vRAN smoothly, all power management features in the BIOS (p-state, c-state and power clamp) and CPU frequency scaling (CPU frequency control and Intel SpeedStep) are disabled. All the experiments were conducted with the same configuration, namely FDD with 5 MHz bandwidth in band 7. We use Intel VTune Amplifier [12, 20, 21] to profile architectural data of key network functions as illustrated in 1. All the applications are implemented in a docker container environment. The docker version we use is 18.09.1.

### 3.2 COTS CPU Architecture and SIMD Instruction Overview

The COTS servers we utilized to deploy the OAI platform are Intel Xeon and Core Server. The micro-architecture of our Intel server are Skylake (Xeon) and Coffee Lake (Core). The Skylake and Coffee Lake share the typical Intel Processor architecture [9, 11]. Figure 2 illustrates fundamental port architecture of the state-of-the-art Intel core. Intel Server represents the most up-to-date server platform available from Intel and it fully embraces the Single instruction multiple data (SIMD) extensions, which is one of the most significant capabilities of recent General Purpose Processors (GPPs) which improves the performance of applications with less hardware modification. Intel has introduced SIMD technologies such as Streaming SIMD Extensions (SSE128), Advanced Vector eXtensions (AVX256 and AVX512 sets). For SSE128, AVX256 and AVX512, register width has been extended from 64 bits to 128 bits (xmm), 256 bits (ymm) and 512 bits (zmm) respectively [15, 16]. The work concept of the SIMD is to execute multiple data in single instruction. Wider registers provide more parallelism ways and more registers reduce extra data movement to the cache memory. The OAI platform efficiently utilizes the SIMD to accelerate its critical submodules to decrease the packets' processing time.

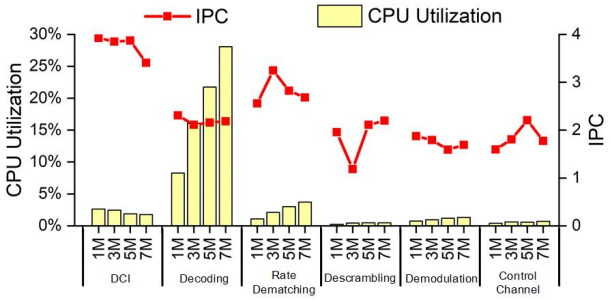



**Figure 3: CPU utilization and IPC for uplink**

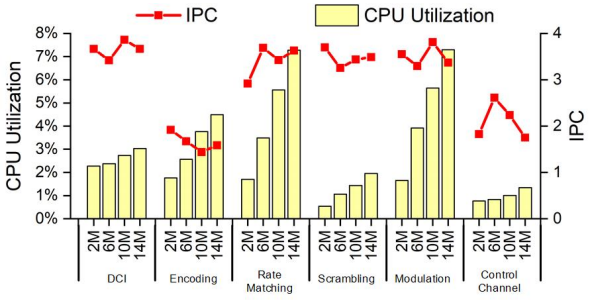



**Figure 4: CPU utilization and IPC for downlink**

# 4 The Architectural Implications of vRAN

We conduct a preliminary characterization and measure that average end-to-end (e2e) delay of the current vRAN software pipeline is 31ms. Clearly, the current e2e latency of vRAN is insufficient to meet the strict requirement for the interactive real-time applications like VR and AR. To discover the potential bottlenecks leading to high e2e delay of the current vRAN platform, we conduct a thorough architectural characterization of the solo-run vRAN. We report our key learnings and illustrate the root cause of the performance bottlenecks of the edge NFV workloads which causes the high e2e delay.

## 4.1 Micro-architectural Inefficiency

**Architectural implication metrics:** The fundamental metrics to describe micro-architectural performance are Instruction per cycle (IPC), frontend bound, bad speculation and backend bound. IPC is a metric indicating the average number of instructions executed for each clock cycle, which is used to measure instruction level parallelism. There are four micro-architectural metrics relates to the IPC – retiring, bad speculation, frontend bound and backend bound. Retiring reflects the issued micro-operations ($\mu$op) eventually get retired while the other 3 metrics mean the issued micro-operations ($\mu$op) get stalled. The high percentage of retiring usually means the high IPC value of an application. The high percentage of the other three categories will hurt the retiring, which will lead to the low IPC. A thorough analysis of frontend bound, bad speculation and backend bound would help us locate the bottlenecks of the edge NFV workloads.

We begin by profiling the IPCs for the main modules inside the vRAN platform. Figure 3 and Figure 4 breakdown the IPC and CPU time of the main modules inside OAI vRAN for uplink and downlink respectively. We can observe that for both uplink and downlink cases, the IPCs for Downlink Control Information (DCI), Rate Matching, and Scrambling are near to the ideal value of 4 for modern Intel processor. However, the IPC of Turbo Decoding module is around only 2.1, which suggests potential headroom for optimization.

Furthermore, we examine the root cause for the poor IPCs by exploring the micro-architectural metrics related to IPC. Frontend bound denotes that instruction-fetch stall will prevent core from making forward progress due to lack of instructions. Bad speculation reflects slots wasted due to incorrect speculations. Backend bound illustrates that no micro-operations ($\mu$op) are being delivered at the issue pipeline, due to lack of required resources in the backend. Figure 5 and Figure 6 demonstrate the micro-architectural profile, the results reveal that across all the modules, the frontend bound and bad speculation overheads are negligible. *The main stall of vRAN applications are concentrated at backend bound*, which means the optimization for backend bound is necessary for vRAN applications. For the most CPU consuming module - turbo decoding, we observe that the backend bound is more than 50%, which is the main reason causing its poor IPC.

**Backend bound bottleneck analysis:** We investigate the source of backend bound by dividing it into two separate metrics: Memory bound and core bound. Memory bound manifests with execution units getting starved after a short while. Core bound manifests either with short execution starvation periods or with sub-optimal execution ports utilization. We can find the root cause of non-uniform backend bound. Memory bound and core bound both suffer from the current vRAN platform. For memory bound, most of the protocols suffer on the L1 and L2 cache bound. Memory bound can be mitigated by simply increasing the cache size of the server, or by leveraging memory which can control the mapping of instruction and data storage for each core. Core bound can be mitigated by prohibiting execution starvation or better ports utilization inside the processor.

Our first effort to mitigate the backend bound stall is to prohibit the memory bound by deploying the vRAN on an alternative high-end beefy COTS server platform (W2195 @ 2.30GHz, 128GB RAM) with higher cache size. Table 1 compares the cache size of the wimpy server and the beefy server. Figure 7 shows the memory bound and core bound details of the dominant functions in vRAN on the wimpy server and the beefy server. *Our finding is that although the memory bound is significantly mitigated by the larger cache resources, the core bound overhead deteriorates on the beefy server. Thus the counteracts of lower memory bound and higher*

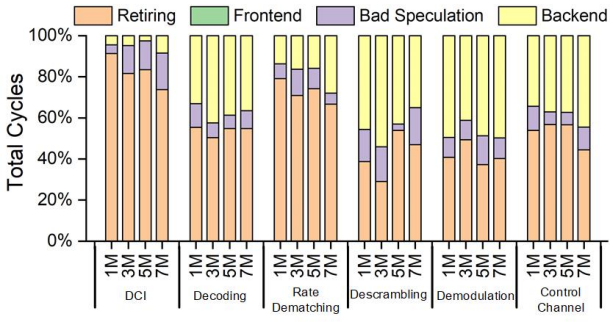



**Figure 5: Micro-architecture value for uplink**

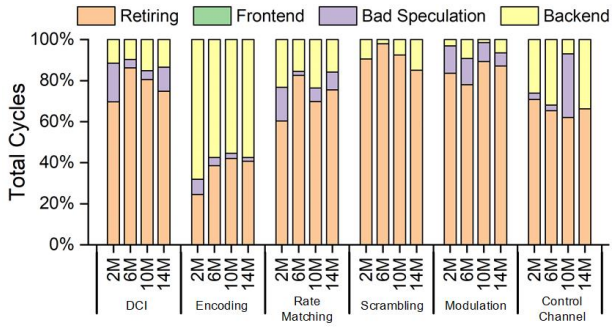



**Figure 6: Micro-architecture value for downlink**

*core bound on beefy server platform makes the overall backend bound stay almost the same to the wimpy server platform.*

Consequently, increasing the cache size cannot effectively improve the vRAN performance on general COTS processors. To achieve better performance, further architectural exploration for the core bound is requested to find out the better state-of-the-art CPU utilization opportunity to mitigate the overall backend bound stall of vRAN platform.

## 4.2 SIMD Processing Causes Backend Bound Inefficiency

In this section, we choose to use the beefy server as our basic platform whose cache size is large enough to rule out the memory bound impacts. We look into the OAI platform's submodules to find out the root cause of the high backend bound.

**Low IPC caused by SIMD instructions:** By examining the submodules of the OAI platform, we observe that the *submodules involving the SIMD instructions such as* _mm_adds, _mm_subs, _mm_max *and* _mm_extract *incur significant high backend bound overheads* compared to the general scalar instructions. The high backend bound overhead is the dominant reason causing the low IPC. The SIMD instructions can be categorized as two types based on their functionalities, one is the SIMD calculation instructions and the other is the SIMD data movement instructions. As shown in Figure 7, the average IPC for the SIMD calculation instructions such as _mm_adds, _mm_subs, and _mm_max is around 2.5, whose backend bound is around 35%. The average IPC for the SIMD data movement instruction such as _mm_extract is around 1.5, whose backend bound is around 55%. While utilizing scalar instructions (do_OFDM) whose backend bound is negligible, the IPC can reach around 3.8, which is near the ideal value (i.e. 4) of the Intel processors.

**Table 1: Cache size and frequency in wimpy and beefy node**

| | Wimpy Node | Beefy Node |
|---|---|---|
| L1 cache | 384KB | 1152KB |
| L2 cache | 1536KB | 18432KB |
| L3 cache | 12288KB | 25344KB |

**Why SIMD instructions cause high backend bound overheads?** Since the high backend bound is the root cause of the low IPC, we further investigate the reason for the high backend bound overheads. The backend bound can be divided into memory bound and core bound. As shown in Figure 7, we can see the memory bound is completely eliminated by utilizing high-end beefy COTS server platform with large cache resources, thus the core bound becomes the dominant reason for the high backend bound. Core bound manifests either with short execution starvation periods or with sub-optimal execution ports utilization. We will go through the micro-architecture of the Intel CPU core to find out whether the sub-optimal execution port utilization is the root cause for the high core bound overhead of the SIMD instruction.

By examining the state-of-the-art CPU architecture, we find that the reason for the high core bound overhead of the SIMD calculation instruction is *due to the limitation of available ports for SIMD instructions.* As shown in Figure 2, the SIMD calculation instructions sustainable ALU ports are port 0, 1 and 2, while the general scalar ALU ports are port 0, 1, 2 and 3. Consequently, the maximum IPC value involved in the SIMD calculation is 3 compared to the ideal IPC value of the scalar instructions value, which is 4. For data movement SIMD instructions, port 4 and 5 hold the load instruction and port 6 and 7 hold the store instruction, which means the ideal IPC value for the data movement instructions is 2. This is the reason why the IPC value for _mm_extract is below 2.

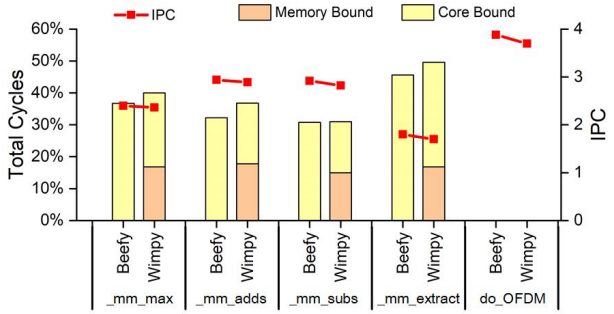
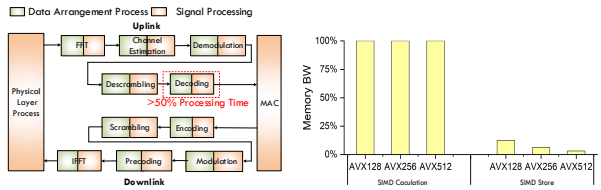



**Figure 7: IPC, memory and core bound under beefy and wimpy server**




**(a) Data arrangement process** **(b) Memory BW utilization**

**Figure 8: Data arrangement process in vRAN and memory bandwidth utilization for SIMD**

**SIMD calculation instructions analysis:** We investigate the SIMD calculation instruction code inside submodule functions to see if there is optimization headroom to promote the performance of these submodules. We extract the C code and the corresponding assembly code for the turbo decoding submodule functions. We observe that for the SIMD calculation instructions, the codes are well organized. For _mm_adds, _mm_subs, when the data is moved into the SIMD 128 bits register (xmm), they do all the operations required before the data are moved out of the xmm registers. The procedures involving in the SIMD calculation instructions utilize all the available ports inside the CPU core. As a result, the IPC value for the _mm_adds, _mm_subs are 2.8 and 2.7 respectively, which is approaching the ideal value 3. For the _mm_max instruction, the IPC for it is around 2.2, which is a little lower compared to _mm_adds and _mm_subs. The reason is that the decoding algorithm involved in _mm_max has some unavoidable data dependencies, which leads to the low IPC value for the _mm_max compared to the _mm_adds and _mm_subs. Besides, as illustrated in Figure 8b, all of the available SIMD ports are efficiently utilized. Especially, the memory bandwidth between SIMD ports' registers and cache are fully occupied. Therefore, when extending the width of the register, the process involving the SIMD calculation instructions can be accelerated significantly, as illustrated in Figure 9. Consequently, these processes will not become hotspot of the OAI platform for the further development.

**SIMD data movement instructions analysis:** We go through the data arrangement process which contains the data movement instruction _mm_extract. As illustrated in Figure 8a, the data movement instructions are utilized by the data arrangement process of the signal processing modules of the vRAN platform. The data arrangement process is utilized to align the data into the SIMD type so that the data can be used as the input of the SIMD calculation instructions. *Our profiling reveals that even though the process utilizes all the available SIMD data movement ports inside CPU core, it suffers severe low memory bandwidth utilization between the ports' registers and L1 cache for this submodule function.* The data arrangement generates the input values systematic1, yparity1 and yparity2 for the gamma, alpha, beta and ext calculations. The data arrangement operations are 16 bits one time and thus the data arrangement operation times is 8 for 128 bits register. However, as shown in Figure 8b, the bandwidth between xmm register and cache is 128 bits, which means that only 12.5% are used. For 256 bits ymm registers and 512 bits zmm registers, the store operation times will be 16 and 32 respectively, the bandwidth utilization ratio will be 6.25% and 3.125% respectively. The store operation times will be extremely high and the bandwidth utilization will be significantly low when further utilizing GPU to do the acceleration. As shown in Figure 9, the operation time proportion of the data arrangement will become larger and larger compared to the modules involving SIMD calculation instructions.

The performance impacts of data arrangement process in the vRAN platform is neglected by most of the current vRAN platforms, such as FlexRAN. The current platforms always concentrated on the optimization of the signal processing modules' calculation process rather than the data arrangement process between the modules. However, we have to pay attention to this elephant in the room as it turns out to be a major performance issue for a vRAN system and can generalize to other SIMD applications.

## 5 Optimization for the Data Arrangement

Data arrangement process occurs in SIMD accelerated modules of the vRAN platform. The data arrangements' underoptimization widely exist in the vRAN platform. Since the decoding module occupies more than 50% of the processing time of the vRAN platform, we choose the data arrangement process of decoding as our optimization object in this paper. The main reason for the low IPC of the data arrangement process is that only store ports are utilized by the process. Besides, the bandwidth between registers and L1 cache is not efficiently utilized during the data transmission. Therefore, it is essential to efficiently utilize the Intel CPU core's hardware to accelerate the data arrangement process to eliminate its severe effect.

### 5.1 Arithmetic Ports Consciousness Mechanism

By examining the micro-architecture of the state-of-the-art CPU, we find that ALU ports are paralleled with the data movement ports and these ports are idle when implementing the data arrangement. Motivated by this observation, we propose "Arithmetic Ports Consciousness Mechanism"(APCM) to leverage these idle ALU ports concurrently with the store ports to solve the data arrangement inefficiency problem. As shown in Figure 2, after the data is transferred directly into the user space by DPDK, APCM utilizes the ALU ports to alleviate the backend bound since the process utilizes not only the data movement ports but also the ALU ports inside CPU core compared to the original solely store ports utilization

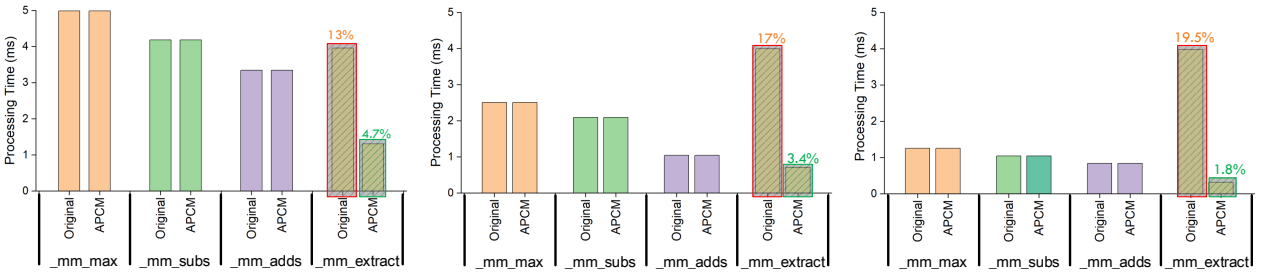



**Figure 9: SIMD module processing time under SSE128, AVX256 and AVX512**

mechanism. The ALU vector ports are utilized to batch the systematic, yparity1 and yparity2 data in segregated registers. After the batching process, the data are moved batch by batch from the register to L1 cache by data movement ports. The APCM will promote the utilization of the memory bandwidth between registers and L1 caches.

Figure 10 shows the detail of the APCM process. The original registers contain the combined data from the clusters systematic (S1), yparity1 (YP1) and yparity2 (YP2). We firstly sampling out each element in each cluster and then congregate elements of the same cluster in segregated registers. After the congregation, we left rotate elements in YP1 16 bits and YP2 32 bits to align elements in S1, YP1 and YP2. Figure 11 illustrates this process inside the processor. As shown in Figure 11, 3 instructions can be implemented in one cycle by 3 parallel ports. Since completing batching S1, YP1 and YP2 will totally require 17 instructions and 5.7 ((17instructions)/(3instructionspercycle)) cycles. Therefore, the memory bandwidth per cycle under APCM will be around 67 bits/cycle ((128bits/register)∗(3register)/(5.7cycles)). When extending the width of the registers, the total instructions and cycles required for the APCM will stay the same. Theoretically, the memory bandwidth utilization will be 134 bits/cycle ((256bits/register) ∗ (3register)/(5.7cycles)) and 270 bits/cycle ((512bits/register)∗(3register)/(5.7cycles)) for AVX256 and AVX512 respectively. The promoted memory

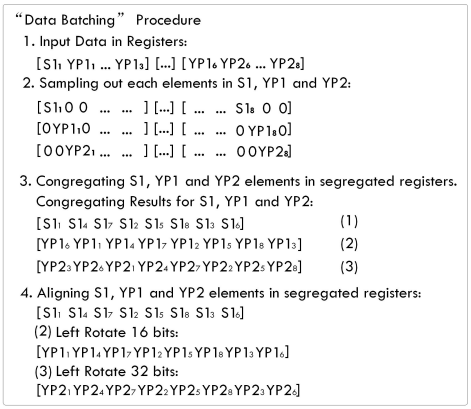



**Figure 10: Data arrangement process under APCM**

bandwidth will decrease the CPU time of the data arrangement process.

## 5.2 Implementation of the Optimization

In this section, we first demonstrate the original data arrangement process. For 128 bits registers (xmm), it utilizes 'pextrw' instruction to extract 16 bits data from register to cache. For 256 bits registers (ymm), it utilizes the same instruction to extract data for the lower 128 bits of ymm from register to cache. After completing the movement of the lower 128 bits, it uses 'vextracti128' instruction to move the upper 128 bits data of ymm register to the lower 128 bits register and do the data extraction since no instruction can extract data directly from upper 256 bits registers (ymm). For the 512 bits registers, it uses 'vextracti32*8 $0 ∗ 0' to extract the lower 256 bits data from zmm to ymm and then do the same process as the 256 bits registers. After completing the data arrangement for the lower 256 bits of zmm, it needs to reload the data into the zmm since utilizing 'vextracti32*8 $0 ∗ 0' will remove the upper 256 bits data automatically. After the data reload, 'vextracti32*8 $0 ∗ 1' is utilized to load the upper 256 bits data from zmm to ymm and do the same data arrangement process as ymm.

Furthermore, we create our optimized code according to the APCM. For 128 bits xmm registers and 256 bits ymm registers, we utilize vpand and vpor to filter out and combine the data for S1, YP1 and YP2. For the 512 bits zmm registers, we use 'vpandd' and 'vpord' to do the data filtering and combination. Since the SIMD register does not provide left rotate instruction directly, we show our mimic process in Figure 12. For YP1, we load extra $YP1_6$ after the YP1 array in the cache. When utilizing YP1 array, we extract the data with the start address at 2nd YP1 element. We do the similar thing for YP2, we load extra $YP2_3$ and $YP2_6$ after the YP2 array in the cache. When utilizing YP2 array, we extract the data with the start address at 3rd YP2 element.

# 6 Evaluation Results

The data arrangement performance is determined primarily by its processing time, IPC and its architectural metrics.

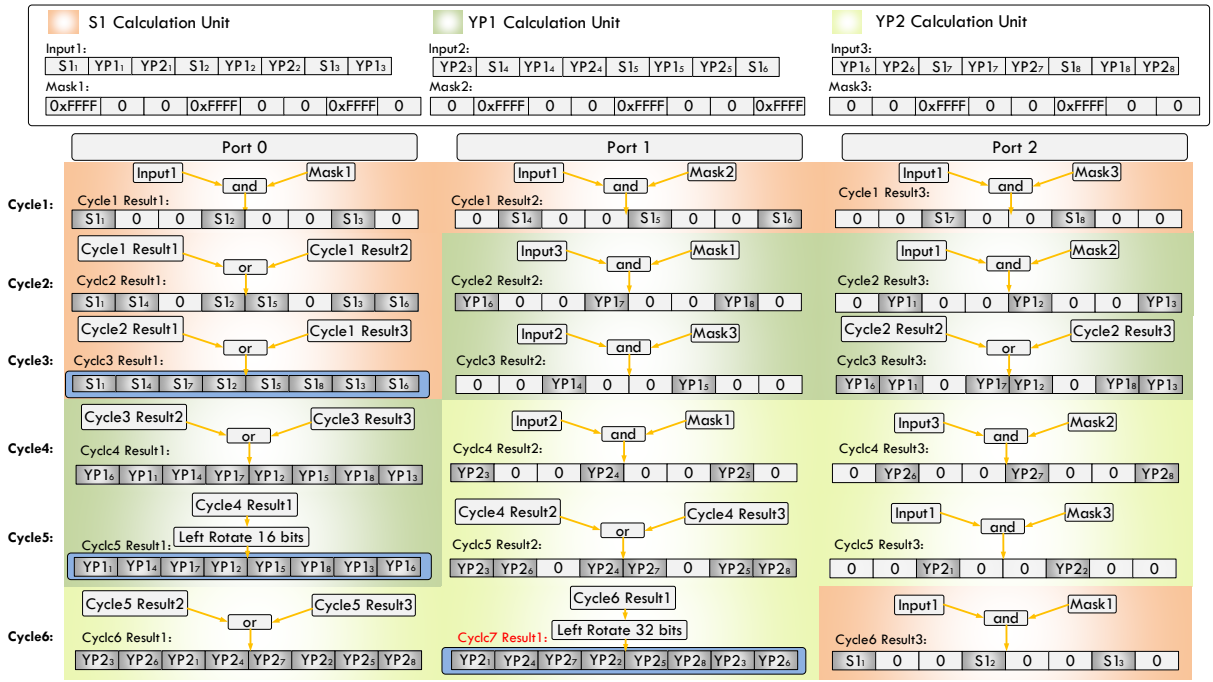



**Figure 11: Micro-architectural data arrangement process under APCM**

We begin with examining these metrics in this section. Besides, we estimate the submodules proportion time inside decoding process and the overall transmission latency to ensure the performance enhancement of the APCM mechanism. Eventually, we evaluate the maximum bandwidth each COTS processor's core can provide and the core numbers required to support the current RAN station before and after the optimization to indict the equipment utilization promotion collected by the APCM mechanism.

**Processing time per packet under different packet sizes:** Processing time is a critical metric to depict the performance of the packet transmission. Figure 13 demonstrates the processing time per packet under different packet size for both UDP and TCP packet before and after APCM optimization. We observe that the APCM mechanism efficiently decreases the processing time for both UDP and TCP packets. For the same packet size, we can observe that the APCM decrease the processing time for both UDP and TCP packet from 12% (SSE128) to 20% (AVX512).


**Figure 12: Rotation mimic mechanism**

**Processing time proportion for different process:** To illustrate the efficiency of the APCM, we present the processing time for the data arrangement processing time and calculation processing time under the standard network transmission packet size (1500 Bytes) in Figure 14. For the registers with sizes of 128 bits, 256 bits, and 512 bits, the processing time for the data arrangement process reduce 67%, 82%, and 92%, respectively compared to the baseline. As a result, the proportion of the data arrangement process on the whole packet processing becomes trivial so that it will not become a hotspot when extending the registers' width.

Based on the results shown in Figure 14, we see that the performance deteriorates when extending the registers' width for the same volume of the workload with the original mechanism. Comparing the 128 bits registers and the 256 bits registers, 2.2% more CPU time is required for 256 bits registers. The reason is that the ymm (256 bits register) does not have an extract instruction to move upper 128 bits data directly, an operation to move the data from upper 128 bits to lower 128 bits is required before executing data extraction. A similar result is depicted when comparing the 256 bits registers and the 512 bits registers, 6.4% more CPU time is required for 512 bits registers. For the zmm (512 bits register), it needs to move the data from zmm (512 bits register) to ymm (256 bits register) before executing data extraction. Besides, when utilizing vextracti32*8 to move lower 256 bits

data to ymm register, the upper 256 bits in zmm will be removed. Therefore, when moving the data for the upper 256 bits, another load operation (vmovdqa64) is required.

To ensure the efficiency of the APCM, it is also critical to compare the processing time under different width of the registers. As illustrated in Figure 14, we can see the CPU time decreases proportionally when extending the width of the registers for the same volume of the workload. Comparing the optimized code for 128 bits registers and 256 bits registers, the 256 bits registers' CPU time decreases 49%. Comparing the optimized code for 512 bits registers and 256 bits registers, the 512 bits registers CPU time decrease 51%, which demonstrates that the extended width of the registers do accelerate the data arrangement process under the APCM mechanism.

**The micro-architectural metrics and IPC:** Figure 15 illustrates the micro-architectural and IPC value of original and APCM data arrangement process. For the same volume of the workload, we can observe that for registers with the width 128 bits, 256 bits and 512 bits, the retiring percentage increases from 55.6% to 97%, from 52% to 96% and from 48% to 95% respectively. The main backend bound stall decreases from 44.4% to 3%, from 48.2% to 4% and from 52% to 5% respectively. The IPC soar from 1.2, 1.1, and 1.05 to 3.6, 3.5, and 3.3, respectively, which demonstrates that the APCM can utilize the CPU ports far more efficiently by exploiting the idle ALU ports comparing to the original mechanism.

**Submodule processing time proportion:** The submodule functions' processing time reflects the efficiency of the APCM. Figure 9 shows the submodule functions' processing time under the APCM, we can see that data arrangement module's processing time is decreased significantly when further extending the width of the register. For the registers with the width 128 bits, 256 bits and 512 bits, the data arrangement process time proportion decrease from 13%, 17%, and 19.5% to 4.7%, 3.4%, and 1.8% respectively. This illustrates that the data arrangement will not become a hotspot when extending the width of the registers.

**Bandwidth per core and system utilization:** We calculate the bandwidth which can be supported by each core and the core numbers required to provide 300 Mbps for current RAN [19] before and after the optimization to illustrate the equipment utilization efficiency of the APCM. As shown in Figure 16, we can see that for the SSE1, AVX2 and AVX512, the maximum bandwidth per core increase from 16.4 Mbps to 18.5 Mbps, from 21.6 Mbps to 26.0 Mbps and from 25.5 Mbps to 32.9 Mbps respectively, which means the system utilization increase around 12% to 29%. The core numbers required to provide 300 Mbps for the SSE1, AVX2 and AVX512 decreases from 18 to 16, 14 to 12 and 12 to 9, respectively.

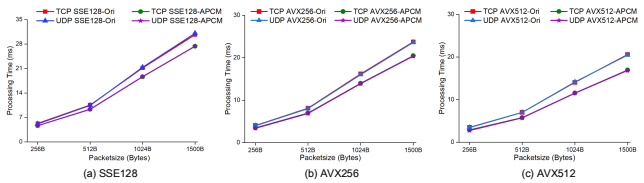



**Figure 13: Processing time for UDP and TCP under different packetsize with original mechanism and APCM**

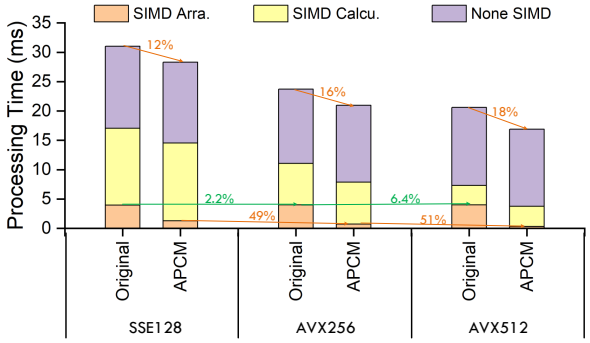



**Figure 14: Processing time for procedures and modules under original mechanism and APCM**

## 7 Related Work

**Main network NFV workloads:** Network Virtualization Networks (NFV) has received substantial attention from the communities in recent years with both academia and industry recognizing its benefits on operational mobile networks. Among the works of the NFV area, some focus on the core network virtualization [1, 4–6, 17] and others concentrate on the edge network [14, 18, 23]. While the above-mentioned work includes the RAN virtualization and network slicing realization, they lack a detailed architectural characterization of the virtual network platform.

**RAN characterization:** In the last few years, several works [3, 7, 13, 22, 24] provide the performance analysis and study on the vRAN platform. [13, 22] introduce the concepts and architecture of the vRAN platform. [22] validates two MAC schedulers and analyzes the vRAN platform, in terms of memory occupancy and execution time. [13] performs thorough profiling of OAI, in terms of execution time, on the user plane data flow.

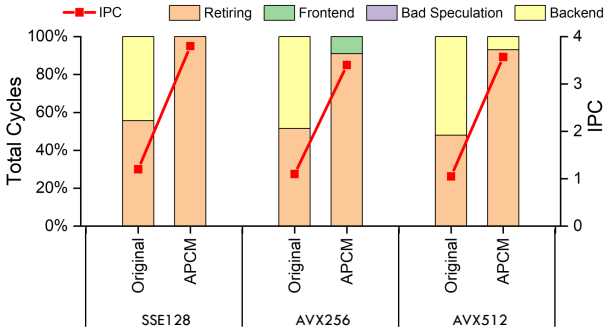



**Figure 15: Micro-architecture value under original mechanism and APCM**

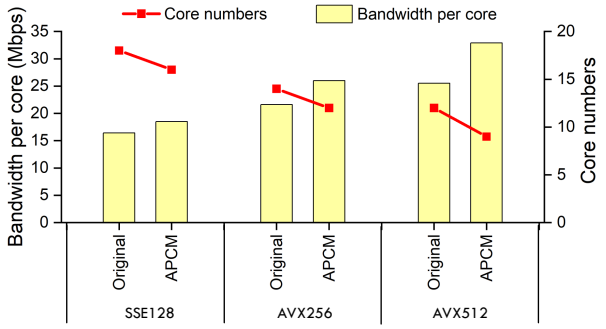



**Figure 16: Bandwidth and core numbers under original mechanism and APCM**

# 8 Conclusion

In this paper, we propose a mechanism named "Arithmetic Ports Consciousness Mechanism" (APCM) to overcome the inefficiency of the data arrangement process of a mainstream edge network vRAN platform. The APCM will decrease the data arrangement's Backend Bound from 45% to 3% and promote its memory bandwidth utilization by 4X-16X. Its CPU time can be reduced by 67% - 92% and the overall latency of the RAN packet transmission will be decreased by 12% - 20%.

# Acknowledgments

This work is supported by NSF grant CCF-1822985 and CCF-1943490 (CAREER). We thank the Intel for their generous donation of servers. Yang Hu is the correspondence author.

# References

[1] Bruno Chatras, U Steve Tsang Kwong, and Nicolas Bihannic. 2017. NFV enabling network slicing for 5G. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IEEE, 219–225.

[2] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 51–66.

[3] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. 2016. srsLTE: an open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*. 25–32.

[4] Yang Hu and Tao Li. 2016. Towards efficient server architecture for virtualized network function deployment: Implications and implementations. In *2016 49th annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 1–12.

[5] Yang Hu and Tao Li. 2018. Enabling efficient network service function chain deployment on heterogeneous server platform. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 27–39.

[6] Yang Hu, Mingcong Song, and Tao Li. 2017. Towards" Full Containerization" in Containerized Network Function Virtualization. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 467–481.

[7] Yang Hu and Jianda Wang. 2019. Architectural and Cost Implications of the 5G Edge NFV Systems. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 594–603.

[8] Yan Huang, Shaoran Li, Y Thomas Hou, and Wenjing Lou. 2018. GPF: A GPU-based Design to Achieve˜ 100 $\mu$s Scheduling for 5G NR. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 207–222.

[9] Intel. [n.d.]. Intel Coffee Lake microarchitecture. https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake/.

[10] Intel. [n.d.]. Intel FlexRAN. https://software.intel.com/content/www/us/en/develop/articles/flexran-lte-and-5g-nr-fec-software-development-kit-modules.html.

[11] Intel. [n.d.]. Intel Skylake microarchitecture. https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)/.

[12] RG Intel. [n.d.]. VTuneTM Amplifier XE 2013.

[13] Po-Chiang Lin and Sheng-Lun Huang. 2018. Performance profiling of cloud radio access networks using openairinterface. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 454–458.

[14] Navid Nikaein, Mahesh K Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. 2014. OpenAirInterface: A flexible platform for 5G research. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 33–38.

[15] Simon J Pennycook, Chris J Hughes, Mikhail Smelyanskiy, and Stephen A Jarvis. 2013. Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi coprocessors. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 1085–1097.

[16] SIMD. [n.d.]. Design of Parallel and High-Performance Computing. http://spcl.inf.ethz.ch/Teaching/2018-dphpc/lectures/lecture8-simd.pdf.

[17] Kun Suo, Yong Zhao, Wei Chen, and Jia Rao. 2018. An analysis and empirical study of container networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 189–197.

[18] Antonio Virdis, Giovanni Stea, and Giovanni Nardini. 2014. SimuLTE-A modular system-level simulator for LTE/LTE-A networks based on OMNeT++. In *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*. IEEE, 59–70.

[19] Kimio Watanabe and Mamoru Machida. 2012. Outdoor lte infrastructure equipment (enodeb). *FUJITSU Sci. Tech. Journal* 48, 1 (2012), 27–32.

[20] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 35–44.

[21] Ahmad Yasin, Yosi Ben-Asher, and Avi Mendelson. 2014. Deep-dive analysis of the data analytics workload in cloudsuite. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 202–211.

[22] Chun Yeow Yeoh, Mohammad Harris Mokhtar, Abdul Aziz Abdul Rahman, and Ahmad Kamsani Samingan. 2016. Performance study of LTE experimental testbed using OpenAirInterface. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 617–622.

[23] Lu Zhang, Chao Li, Pengyu Wang, Yunxin Liu, Yang Hu, Quan Chen, and Minyi Guo. 2019. Characterizing and orchestrating NFV-ready servers for efficient edge data processing. In *Proceedings of the International Symposium on Quality of Service*. 1–10.

[24] Qi Zheng, Yajing Chen, Ronald Dreslinski, Chaitali Chakrabarti, Achilleas Anastasopoulos, Scott Mahlke, and Trevor Mudge. 2013. WiBench: An open source kernel suite for benchmarking wireless systems. In *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 123–132.