Enabling Efficient SIMD Acceleration for Virtual Radio Access Network

Jianda Wang*, Yang Hu

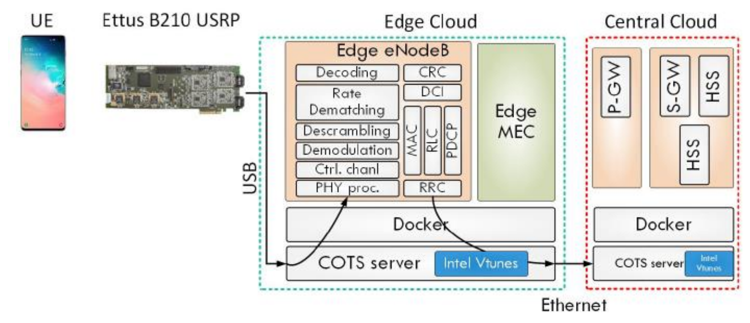
ICPP 21’

Pervasive and Emerging Architecture Research Lab, UT Dallas

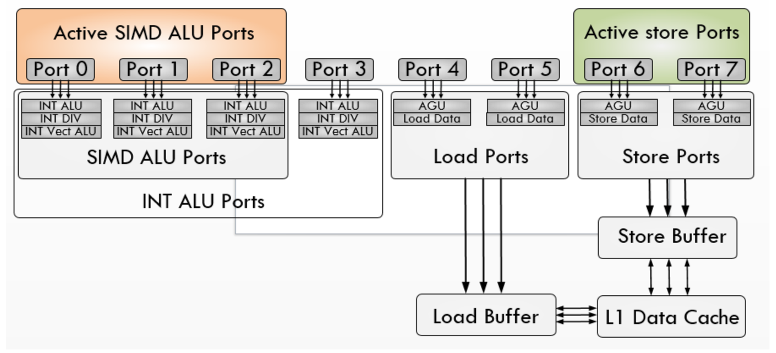
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Pervasive and Emerging Architecture Research Lab (PEARL)

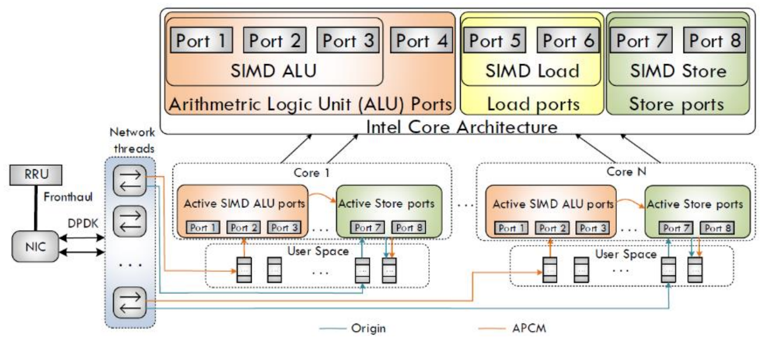
1. Motivation and Background



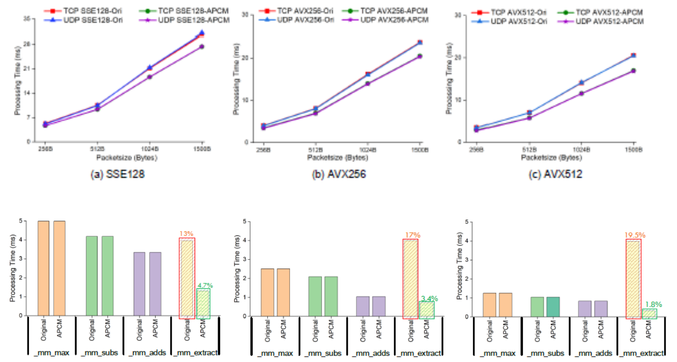
2. Architectural Implications of vRAN



3. vRAN Inefficiency and Optimization



4. Evaluations



1. Motivation and Backgrounds

Autonomous Driving

Home Automation

New Era Network

Smart City

Healthcare

Virtual Reality

1. Motivation and Backgrounds

Ultra-dense Bandwidth

Ultra-high Reliability

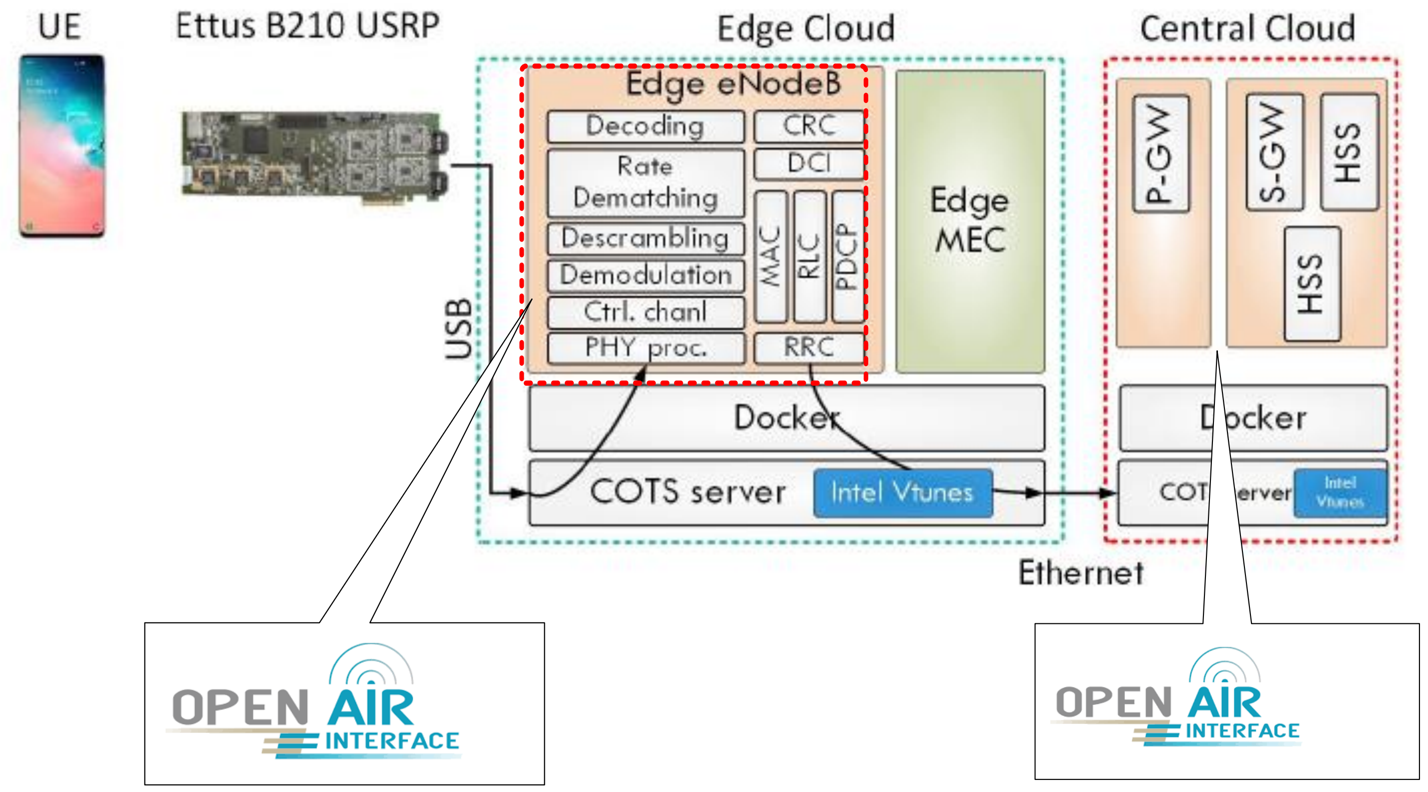
New Era Network

Ultra-low Latency

Network Function Virtualization

Mobile Edge Computing

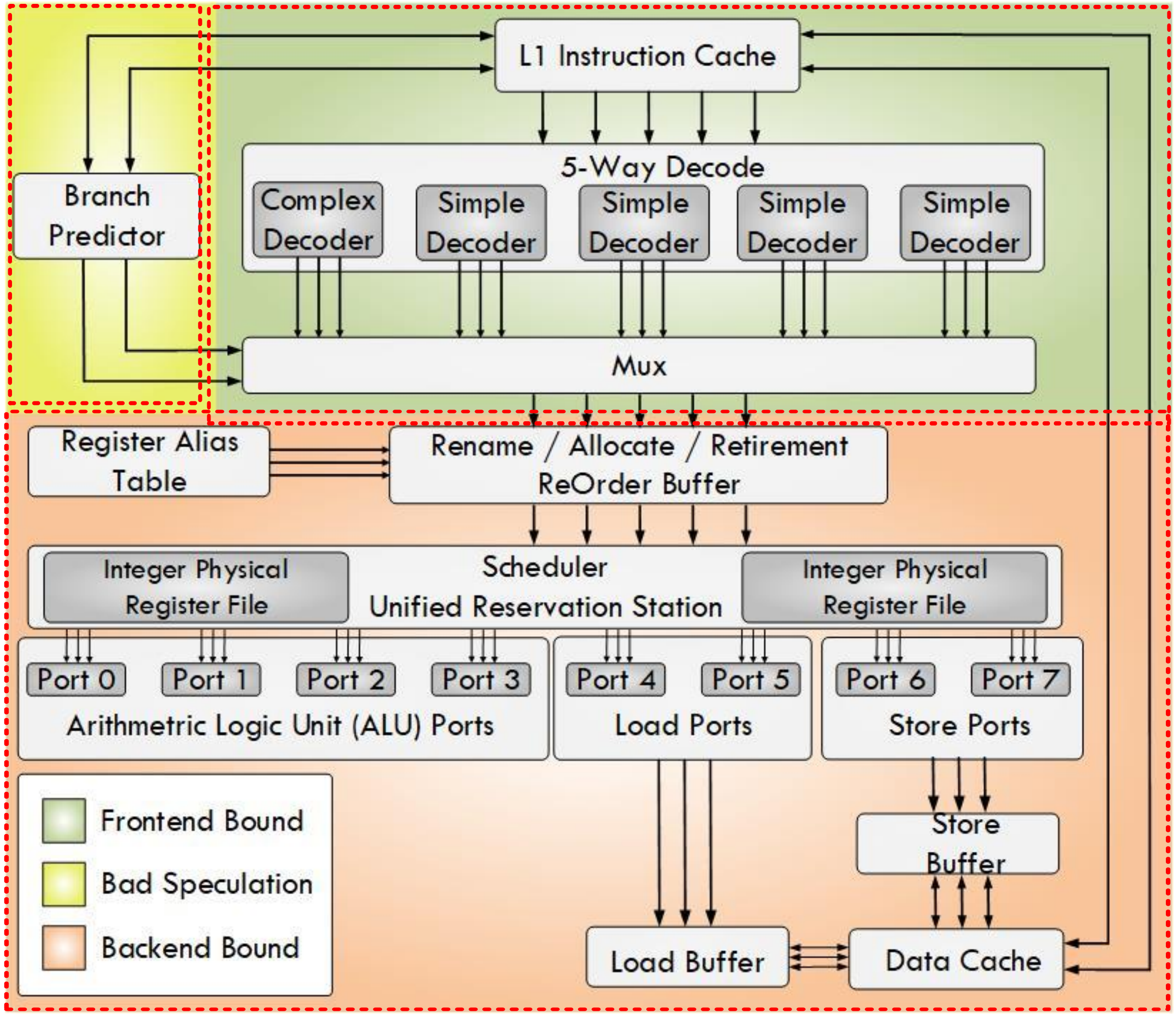
1. Background – Typical vRAN Architecture



OpenAirInterface 2018_w25 (eNodeB)

OpenAirInterface Version Develop

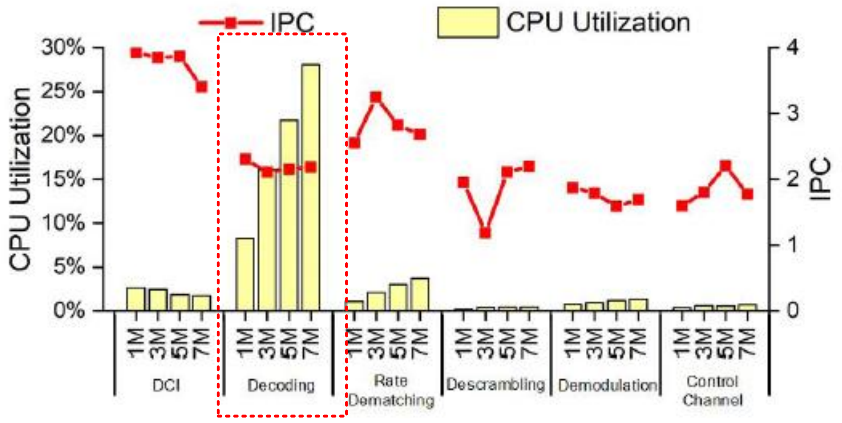
1. Background – Typical Intel Core Architecture

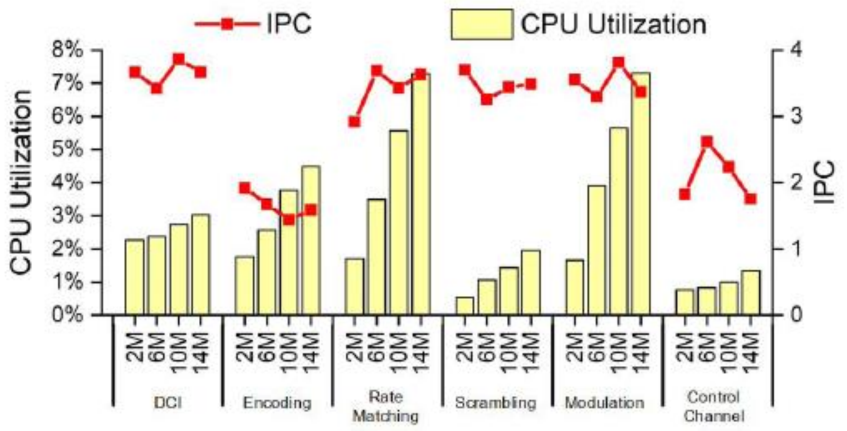


- Frontend Bound
- Bad Speculation
- Backend Bound

2. Architecture Implications of the RAN system

IPC and CPU Utilization

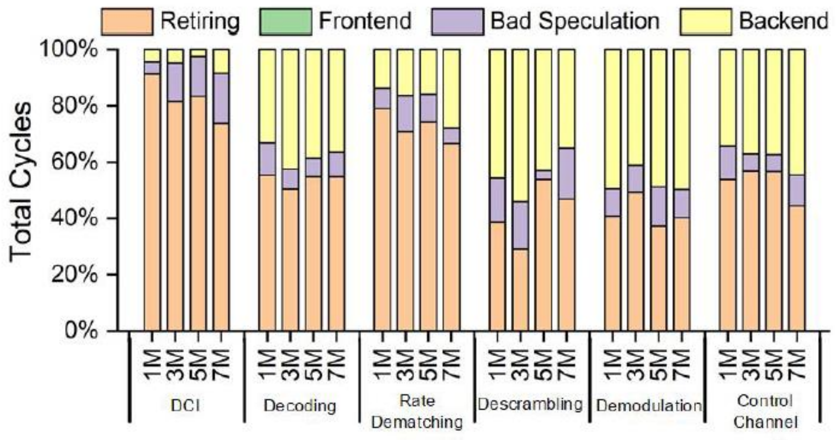


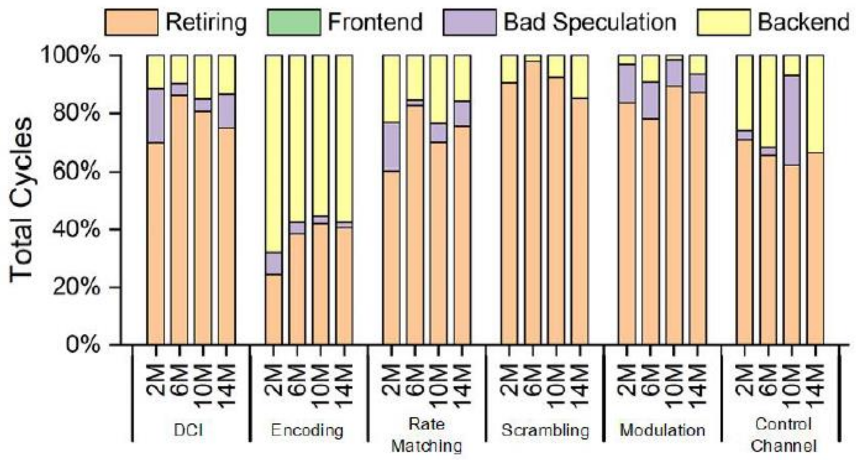


- We can observe that for both uplink and downlink cases, the IPCs for Downlink Control Information (DCI), Rate Matching, and Scrambling are near to the ideal value of 4 for modern Intel processor. All these modules are not CPU consuming.
- However, the IPC for the most CPU consuming module - Turbo Decoding is only around 2.1, which suggests potential headroom for optimization.

2. Architecture Implications of the RAN system

Micro-architecture value for the RAN system





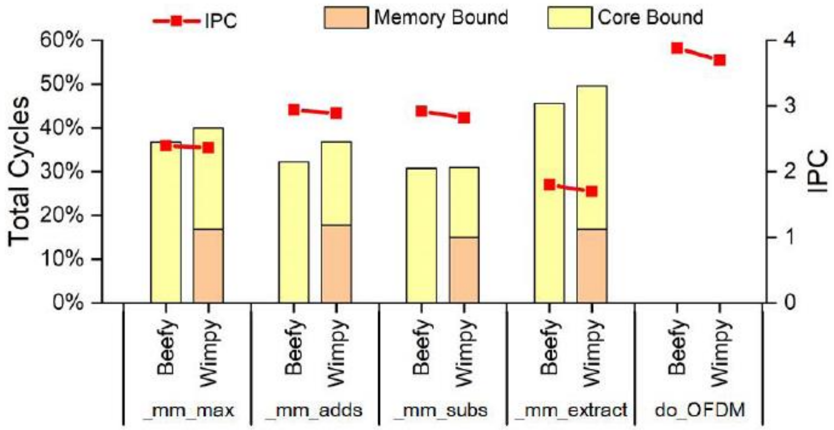
- As demonstrated in the micro-architectural profile, the results reveal that across all the modules, the frontend bound and bad speculation overheads are negligible.
- The main stall of vRAN applications are concentrated on the backend bound, which means the optimization for backend bound is necessary for vRAN applications.

2. Architecture Implications of the RAN system

Wimpy and Beefy node Setup

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

IPC, memory and core bound under wimpy and beefy node

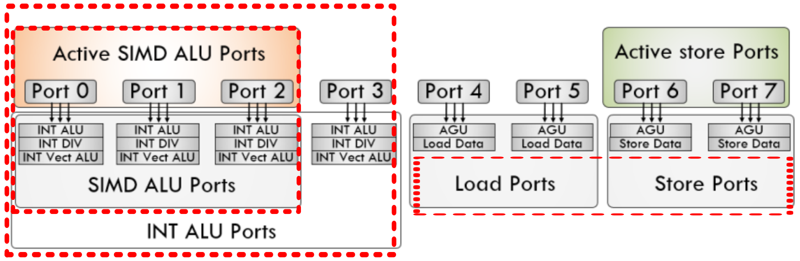


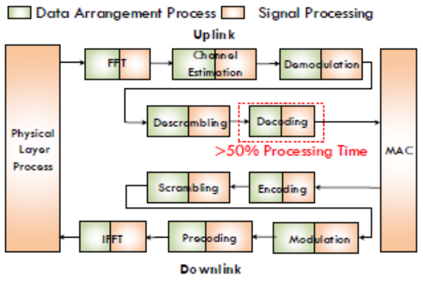
- The figure shows the memory bound and core bound on the wimpy server and the beefy server. Our finding is that although the memory bound is significantly mitigated by the larger cache resources, the core bound overhead becomes worse on the beefy server. The counteracts of the lower memory bound and the higher core bound on beefy server platform makes the overall backend bound stay almost the same to the wimpy server platform.

2. Architecture Implications of the RAN system

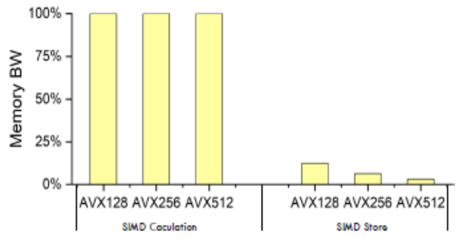
Intel processor architecture

Data arrangement process inefficiency for the RAN system





(a) Data arrangement process



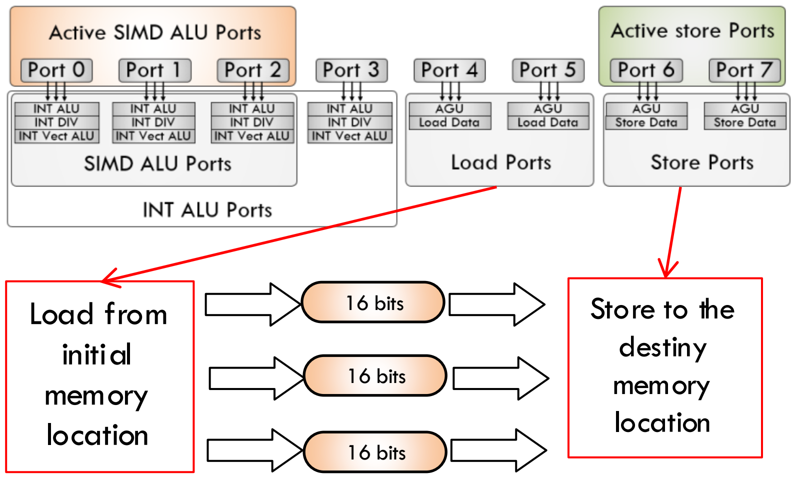
(b) Memory BW utilization

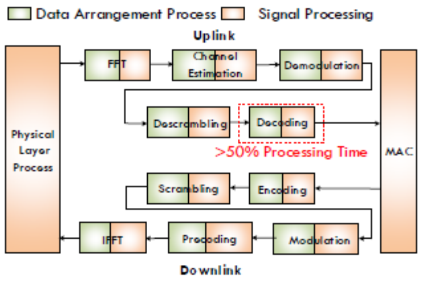
- Our profiling reveals that the data arrangement process suffers severe low memory bandwidth utilization between the ports’ registers and L1 cache since it only utilizes the load and store ports in the processor

2. Architecture Implications of the RAN system

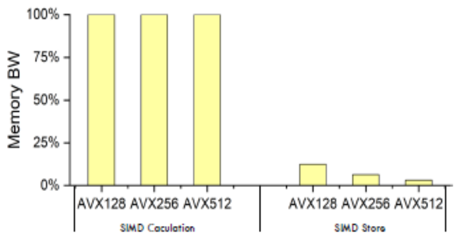
Intel processor architecture

Data arrangement process inefficiency for the RAN system





(a) Data arrangement process

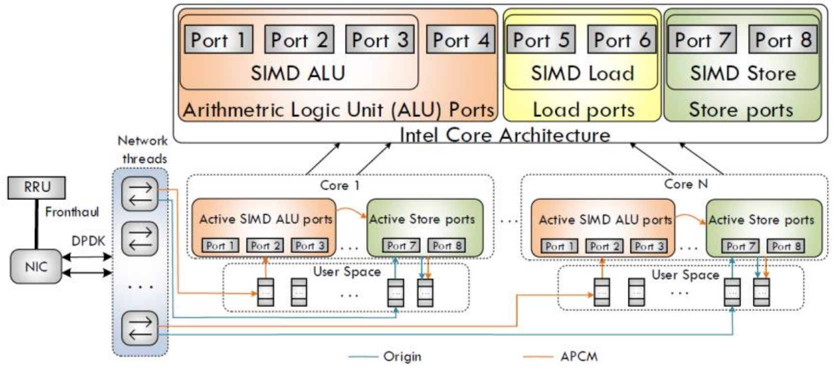


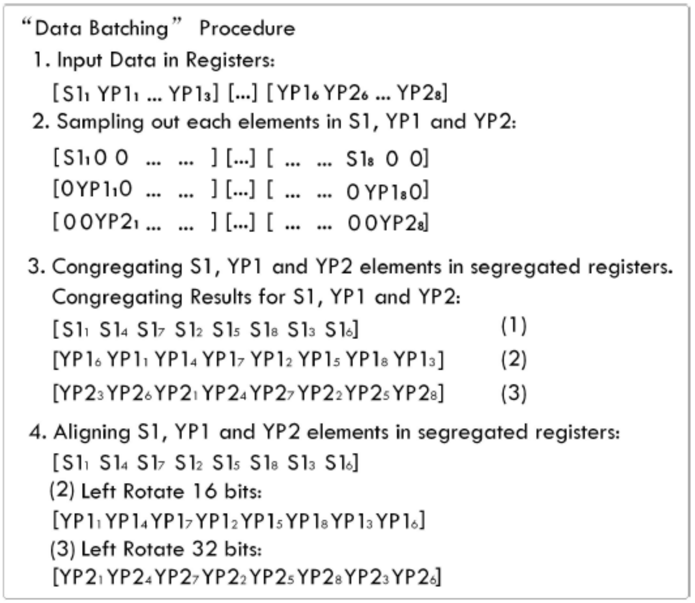
(b) Memory BW utilization

- The bandwidth utilization for the original process is 16 bits/cycle

3. Optimization of the current RAN system

Data arrangement process under APCM




"Data Batching" Procedure

1. Input Data in Registers:

   $[S1_1\ YP1_1 \dots YP1_3]\ [\dots]\ [YP1_6\ YP2_6 \dots YP2_8]$

2. Sampling out each elements in S1, YP1 and YP2:

   $[S1_1\ 0\ 0\ \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ S1_8\ 0\ 0]$

   $[0\ YP1_1\ 0\ \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ 0\ YP1_8\ 0]$

   $[0\ 0\ YP2_1 \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ 0\ 0\ YP2_8]$

3. Congregating S1, YP1 and YP2 elements in segregated registers.

   Congregating Results for S1, YP1 and YP2:

   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$ (1)

   $[YP1_6\ YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3]$ (2)

   $[YP2_3\ YP2_6\ YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8]$ (3)

4. Aligning S1, YP1 and YP2 elements in segregated registers:

   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$

   (2) Left Rotate 16 bits:

   $[YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3\ YP1_6]$

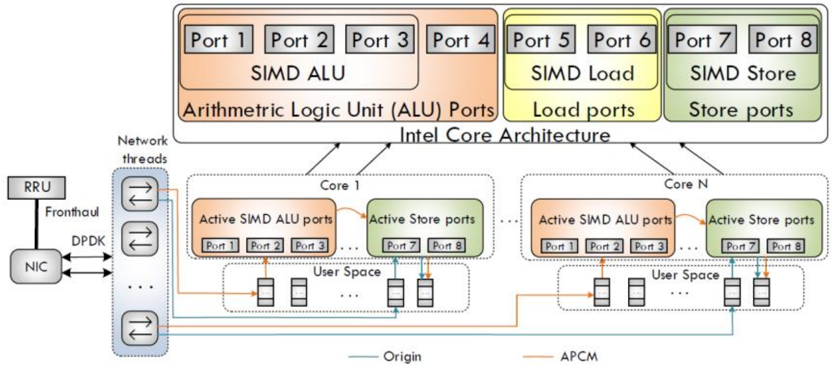
   (3) Left Rotate 32 bits:

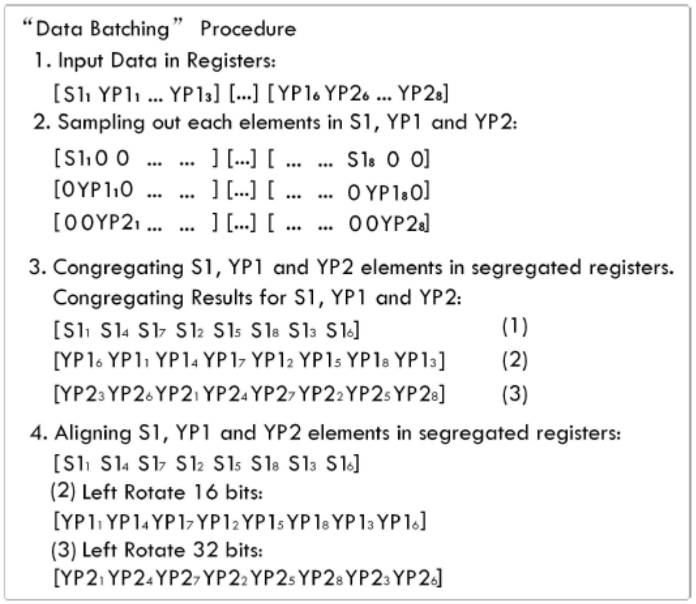
   $[YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8\ YP2_3\ YP2_6]$

- We propose "Arithmetic Ports Consciousness Mechanism"(APCM) to leverage these idle ALU ports concurrently with the load and store ports to solve the data arrangement inefficiency problem.

3. Optimization of the current RAN system

Data arrangement process under APCM





"Data Batching" Procedure

1. Input Data in Registers:

   $[S1_1\ YP1_1 \dots YP1_3]\ [\dots]\ [YP1_6\ YP2_6 \dots YP2_8]$

2. Sampling out each elements in S1, YP1 and YP2:

   $[S1_1\ 0\ 0\ \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ S1_8\ 0\ 0]$

   $[0\ YP1_1\ 0\ \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ 0\ YP1_8\ 0]$

   $[0\ 0\ YP2_1\ \dots\ \dots\ ]\ [\dots]\ [\ \dots\ \dots\ 0\ 0\ YP2_8]$

3. Congregating S1, YP1 and YP2 elements in segregated registers.

   Congregating Results for S1, YP1 and YP2:

   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$ (1)

   $[YP1_6\ YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3]$ (2)

   $[YP2_3\ YP2_6\ YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8]$ (3)

4. Aligning S1, YP1 and YP2 elements in segregated registers:

   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$

   (2) Left Rotate 16 bits:

   $[YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3\ YP1_6]$

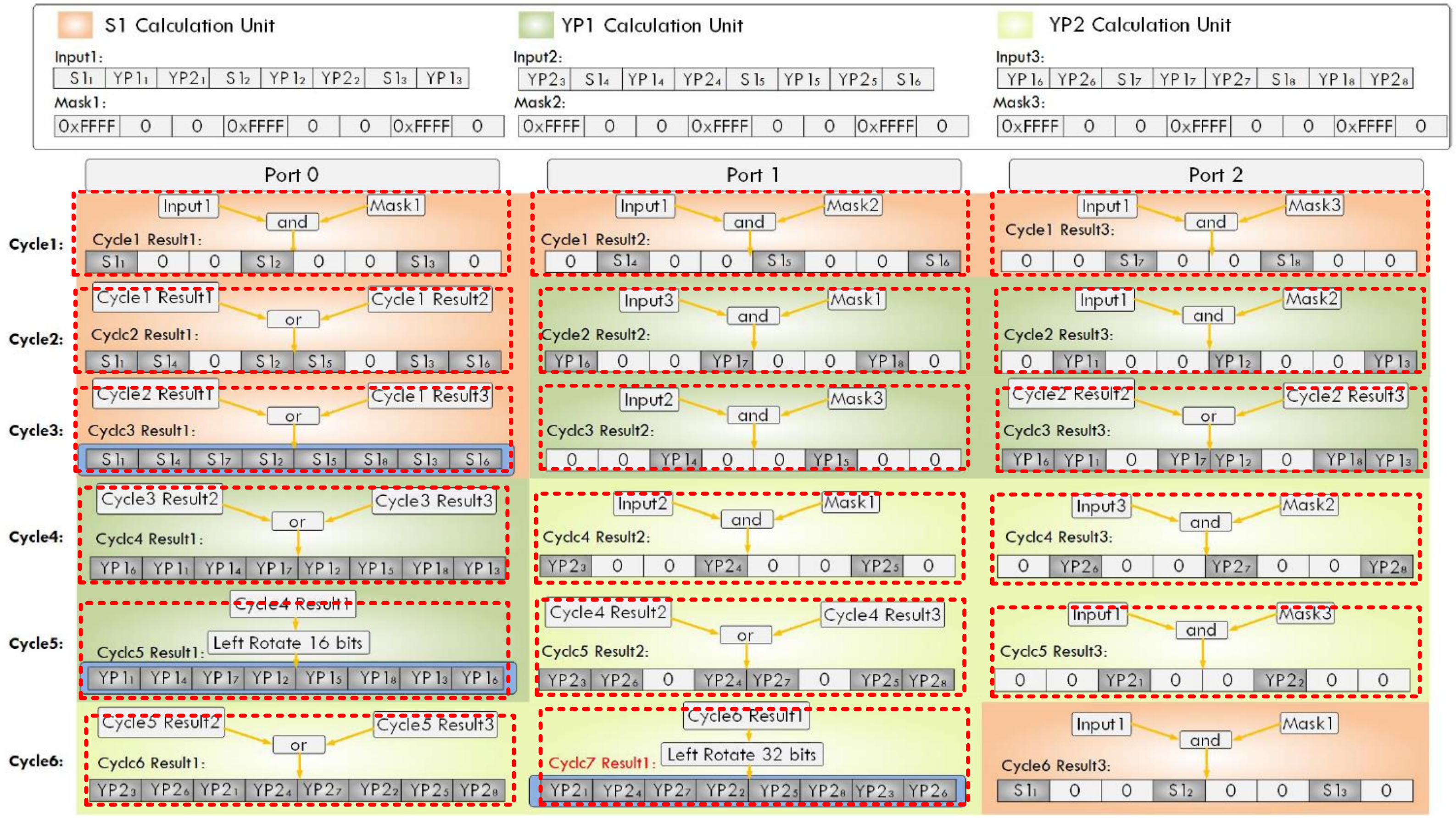
   (3) Left Rotate 32 bits:

   $[YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8\ YP2_3\ YP2_6]$

- As shown in the figure, after the data is transferred directly into the user space by DPDK, APCM utilizes the ALU ports to batch the data before storing them back to the cache. This will alleviate the backend bound and meanwhile promote the bandwidth utilization between the registers and the cache.

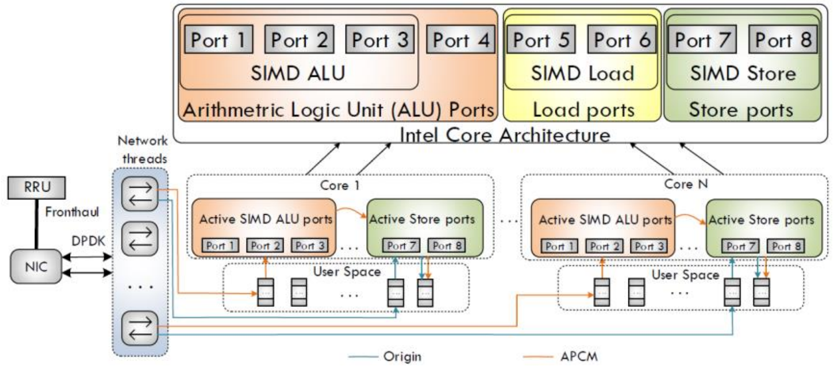
3. Optimization of the current RAN system

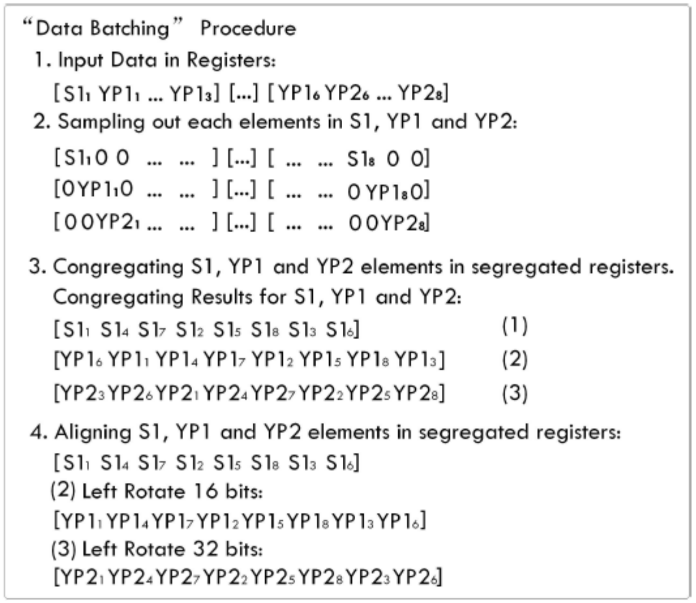
Data arrangement process under APCM


S1 Calculation Unit
YP1 Calculation Unit
YP2 Calculation Unit
Input1: $S1_1$ | $YP1_1$ | $YP2_1$ | $S1_2$ | $YP1_2$ | $YP2_2$ | $S1_3$ | $YP1_3$
Mask1: 0xFFFF | 0 | 0 | 0xFFFF | 0 | 0 | 0xFFFF | 0
Input2: $YP2_3$ | $S1_4$ | $YP1_4$ | $YP2_4$ | $S1_5$ | $YP1_5$ | $YP2_5$ | $S1_6$
Mask2: 0xFFFF | 0 | 0 | 0xFFFF | 0 | 0 | 0xFFFF | 0
Input3: $YP1_6$ | $YP2_6$ | $S1_7$ | $YP1_7$ | $YP2_7$ | $S1_8$ | $YP1_8$ | $YP2_8$
Mask3: 0xFFFF | 0 | 0 | 0xFFFF | 0 | 0 | 0xFFFF | 0
Port 0
Port 1
Port 2
Cycle1:
Input1 and Mask1 — Cycle1 Result1: $S1_1$ | 0 | 0 | $S1_2$ | 0 | 0 | $S1_3$ | 0
Input1 and Mask2 — Cycle1 Result2: 0 | $S1_4$ | 0 | 0 | $S1_5$ | 0 | 0 | $S1_6$
Input1 and Mask3 — Cycle1 Result3: 0 | 0 | $S1_7$ | 0 | 0 | $S1_8$ | 0 | 0
Cycle2:
Cycle1 Result1 or Cycle1 Result2 — Cyclc2 Result1: $S1_1$ | $S1_4$ | 0 | $S1_2$ | $S1_5$ | 0 | $S1_3$ | $S1_6$
Input3 and Mask1 — Cycle2 Result2: $YP1_6$ | 0 | 0 | $YP1_7$ | 0 | 0 | $YP1_8$ | 0
Input1 and Mask2 — Cycle2 Result3: 0 | $YP1_1$ | 0 | 0 | $YP1_2$ | 0 | 0 | $YP1_3$
Cycle3:
Cycle2 Result1 or Cycle1 Result3 — Cyclc3 Result1: $S1_1$ | $S1_4$ | $S1_7$ | $S1_2$ | $S1_5$ | $S1_8$ | $S1_3$ | $S1_6$
Input2 and Mask3 — Cyclc3 Result2: 0 | 0 | $YP1_4$ | 0 | 0 | $YP1_5$ | 0 | 0
Cycle2 Result2 or Cycle2 Result3 — Cyclc3 Result3: $YP1_6$ | $YP1_1$ | 0 | $YP1_7$ | $YP1_2$ | 0 | $YP1_8$ | $YP1_3$
Cycle4:
Cycle3 Result2 or Cycle3 Result3 — Cyclc4 Result1: $YP1_6$ | $YP1_1$ | $YP1_4$ | $YP1_7$ | $YP1_2$ | $YP1_5$ | $YP1_8$ | $YP1_3$
Input2 and Mask1 — Cyclc4 Result2: $YP2_3$ | 0 | 0 | $YP2_4$ | 0 | 0 | $YP2_5$ | 0
Input3 and Mask2 — Cyclc4 Result3: 0 | $YP2_6$ | 0 | 0 | $YP2_7$ | 0 | 0 | $YP2_8$
Cycle5:
Cycle4 Result1 — Left Rotate 16 bits — Cyclc5 Result1: $YP1_1$ | $YP1_4$ | $YP1_7$ | $YP1_2$ | $YP1_5$ | $YP1_8$ | $YP1_3$ | $YP1_6$
Cycle4 Result2 or Cycle4 Result3 — Cyclc5 Result2: $YP2_3$ | $YP2_6$ | 0 | $YP2_4$ | $YP2_7$ | 0 | $YP2_5$ | $YP2_8$
Input1 and Mask3 — Cyclc5 Result3: 0 | 0 | $YP2_1$ | 0 | 0 | $YP2_2$ | 0 | 0
Cycle6:
Cycle5 Result2 or Cycle5 Result3 — Cyclc6 Result1: $YP2_3$ | $YP2_6$ | $YP2_1$ | $YP2_4$ | $YP2_7$ | $YP2_2$ | $YP2_5$ | $YP2_8$
Cycle6 Result1 — Left Rotate 32 bits — Cyclc7 Result1: $YP2_1$ | $YP2_4$ | $YP2_7$ | $YP2_2$ | $YP2_5$ | $YP2_8$ | $YP2_3$ | $YP2_6$
Input1 and Mask1 — Cyclc6 Result3: $S1_1$ | 0 | 0 | $S1_2$ | 0 | 0 | $S1_3$ | 0

3. Optimization of the current RAN system

Data arrangement process under APCM





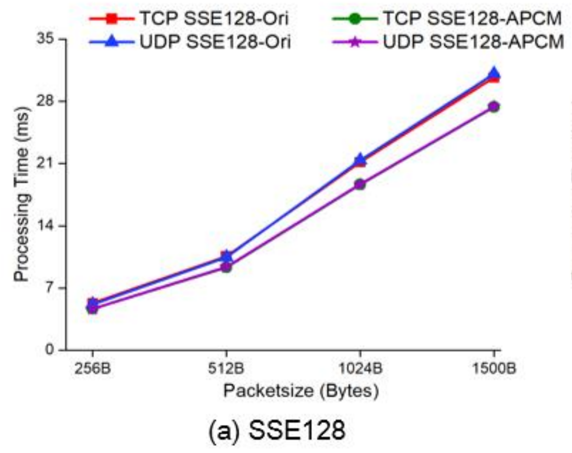
"Data Batching" Procedure

1. Input Data in Registers:
   $[S1_1\ YP1_1 \dots YP1_3]$ [...] $[YP1_6\ YP2_6 \dots YP2_8]$
2. Sampling out each elements in S1, YP1 and YP2:
   $[S1_1\ 0\ 0\ \dots\ \dots\ ]$ [...] $[\ \dots\ \dots\ S1_8\ 0\ 0]$
   $[0\ YP1_1\ 0\ \dots\ \dots\ ]$ [...] $[\ \dots\ \dots\ 0\ YP1_8\ 0]$
   $[0\ 0\ YP2_1 \dots\ \dots\ ]$ [...] $[\ \dots\ \dots\ 0\ 0\ YP2_8]$
3. Congregating S1, YP1 and YP2 elements in segregated registers.
   Congregating Results for S1, YP1 and YP2:
   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$ (1)
   $[YP1_6\ YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3]$ (2)
   $[YP2_3\ YP2_6\ YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8]$ (3)
4. Aligning S1, YP1 and YP2 elements in segregated registers:
   $[S1_1\ S1_4\ S1_7\ S1_2\ S1_5\ S1_8\ S1_3\ S1_6]$
   (2) Left Rotate 16 bits:
   $[YP1_1\ YP1_4\ YP1_7\ YP1_2\ YP1_5\ YP1_8\ YP1_3\ YP1_6]$
   (3) Left Rotate 32 bits:
   $[YP2_1\ YP2_4\ YP2_7\ YP2_2\ YP2_5\ YP2_8\ YP2_3\ YP2_6]$

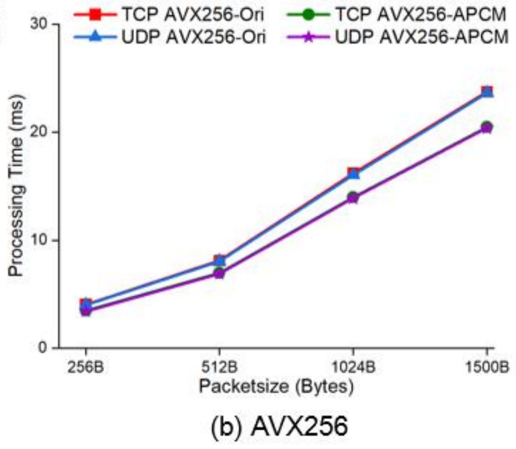
- Based on the procedure of the APCM mechanism, we can calculate that the bandwidth per cycle under APCM will be around 67 bits/cycle, 134 bits/cycle and 270 bits/cycle for the SSE128, AVX256 and AVX512.

4. Evaluations

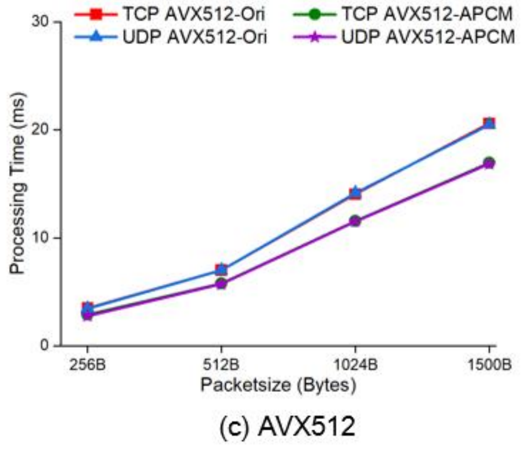
Processing time for UDP and TCP under different packetsize with original and APCM



(a) SSE128



(b) AVX256

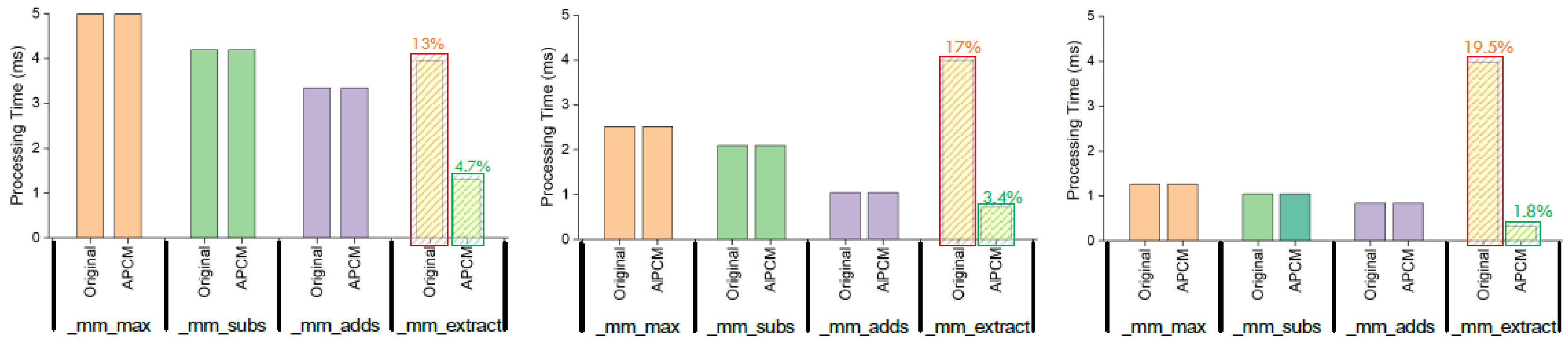


(c) AVX512

- The figure demonstrates the APCM decrease the processing time for both UDP and TCP packet from 12% (SSE128) to 20% (AVX512).

4. Evaluations

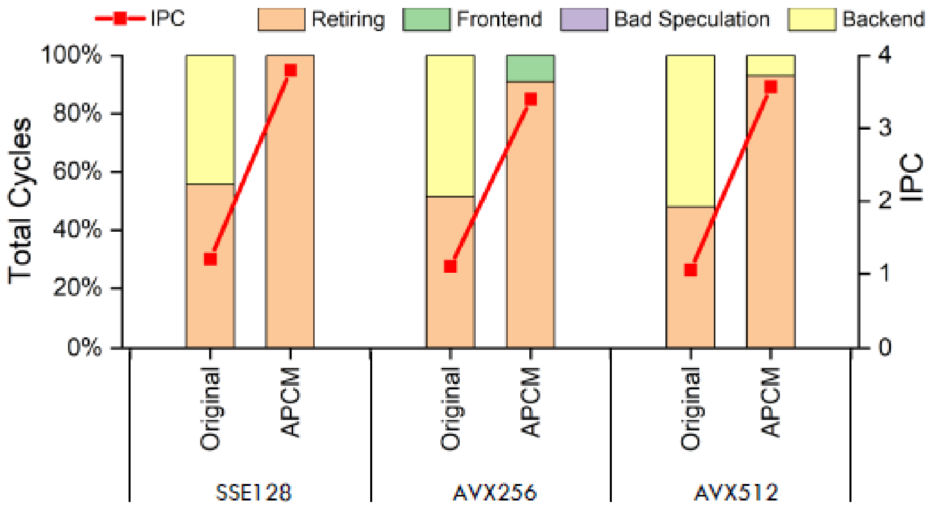
SIMD Module Processing Time under SSE128, AVX256 and AVX512



- 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%.

4. Evaluations

Microarchitectural Value under Original Mechanism and APCM



- As shown in the figure, we can observe that for registers with the width 128 bits, 256 bits and 512 bits, the retiring percentage increases from 55%, 52% and 48% to 97%, 96% and 95%. The main backend bound stall decreases from 44%, 48% and 52% to 3%, 4% and 5%.

- The IPC soars from 1.2, 1.1, and 1.05 to 3.6, 3.5, and 3.3.

5. Questions?