Enabling Efficient SIMD Acceleration for Virtual Radio Access Network

Jienda Wang*, Yang Hu

ICPP 21’
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

Ultra-low Latency

Network Function Virtualization

Mobile Edge Computing

New Era Network
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

<table>
<thead>
<tr>
<th></th>
<th>Wimpy Node</th>
<th>Beefy Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache</td>
<td>384KB</td>
<td>1152KB</td>
</tr>
<tr>
<td>L2 cache</td>
<td>1536KB</td>
<td>18432KB</td>
</tr>
<tr>
<td>L3 cache</td>
<td>12288KB</td>
<td>25344KB</td>
</tr>
</tbody>
</table>

- 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

- 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**

- 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:
     \[ \text{Data}_{S1} [\ldots, \text{Data}_{Y1}, \ldots, \text{Data}_{YP2}] \]
  2. Sampling out each elements in S1, YP1 and YP2:
     \[ [\text{S1}_{0} \text{ YP1}_{0} \ldots ] [\ldots ] [\ldots \text{S1s} \text{ 0} \text{ 0}] \]
     \[ [\text{YP1}_{0} \text{ YP1}_{0} \ldots ] [\ldots ] [\ldots \text{0} \text{ YP1s} \text{ 0}] \]
     \[ [\text{YP2}_{0} \text{ YP2}_{0} \ldots ] [\ldots ] [\ldots \text{0} \text{ YP2s}] \]
  3. Congregating S1, YP1 and YP2 elements in segregated registers.
     Congregating Results for S1, YP1 and YP2:
     \[ [\text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}] \]
     \[ [\text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}] \]
     \[ [\text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}] \]
  4. Aligning S1, YP1 and YP2 elements in segregated registers:
     \[ [\text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}, \text{S1}_{1}] \]
     \(2\) Left Rotate 16 bits:
     \[ [\text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}, \text{YP1}_{1}] \]
     \(3\) Left Rotate 32 bits:
     \[ [\text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}, \text{YP2}_{1}] \]

- 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

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
3. Optimization of the current RAN system

Data arrangement process under APCM

“Data Batching” Procedure
1. Input Data in Registers:
   \[ \{ S_1 \ YP_{11} \ ... \ YP_{1s} \} \ [\ ... \ \{ YP_{1s} \ YP_{2s} \ ... \ YP_{2s} \} \]
2. Sampling out each elements in S1, YP1 and YP2:
   \[ \{ S_1 \ 0 \ 0 \ ... \ \} \ [\ ... \ \{ S_1 \ 0 \ 0 \} \]
   \[ \{ 0 YP_{11} \ 0 \ ... \ \} \ [\ ... \ \{ 0 YP_{1s} \ 0 \} \]
   \[ \{ 0 \ 0 \ YP_{21} \ ... \ \} \ [\ ... \ \{ 0 \ 0 \ YP_{2s} \} \]
3. Congregating S1, YP1 and YP2 elements in segregated registers.
   Congregating Results for S1, YP1 and YP2:
   \[ \{ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \} \]
   \[ \{ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \} \]
   \[ \{ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \} \]
4. Aligning S1, YP1 and YP2 elements in segregated registers:
   \[ \{ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \ S_1 \} \]
   (2) Left Rotate 16 bits:
   \[ \{ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \ YP_{11} \} \]
   (3) Left Rotate 32 bits:
   \[ \{ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \ YP_{21} \} \]

- 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**

- 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

- 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?