PMU and Performance Analysis Methods With Proposals for RISC-V in Cloud

Jiankang Chen
Shuai Xue
Huaixin Chang

Alibaba Cloud
6/26/2025

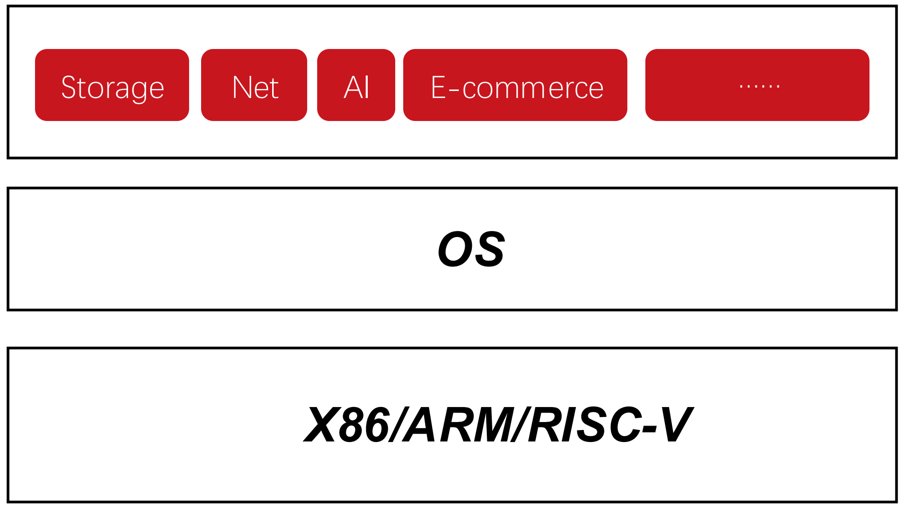
CONTENTS

Challenges and Methods of Performance Analysis

Challenges and Methods of Performance Analysis

Challenges of cloud scenario performance analysis

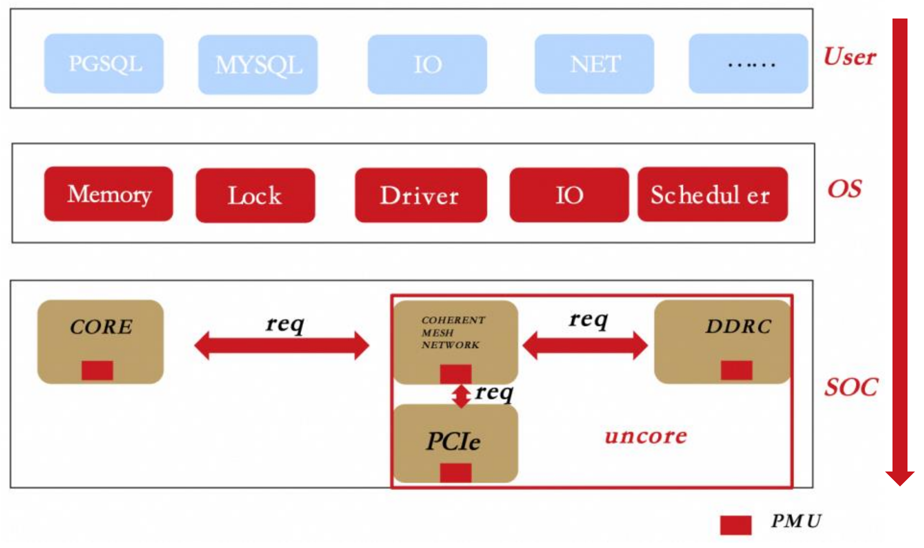
- Need to analyze complex business
  - Storage
    - Memory and IO intensive
  - E-commerce
    - Hybrid deployment
- The business program calling process is complex
  - The software stack involves many modules and components.
- Multi-architecture hardware
  - X86, ARM, RISC-V, GPU, etc
- Need low-overhead real-time monitoring of performance metrics



Challenges and Methods of Performance Analysis

Performance analysis methods for cloud computing scenarios

- From applications to OS, and then to hardware self-directed performance analysis methods
  - Perform full stack analysis through topdown
- Low overhead monitoring via hardware pmu in cloud
- Provides full-architecture analysis tools to solve multi-architecture problems
- Find bottlenecks that affect business performance through underlying hardware metric analysis
  - Use Core pmu to analyze the bottleneck points in the core
  - Analyze application bottlenecks at the soc architecture level through uncore pmu

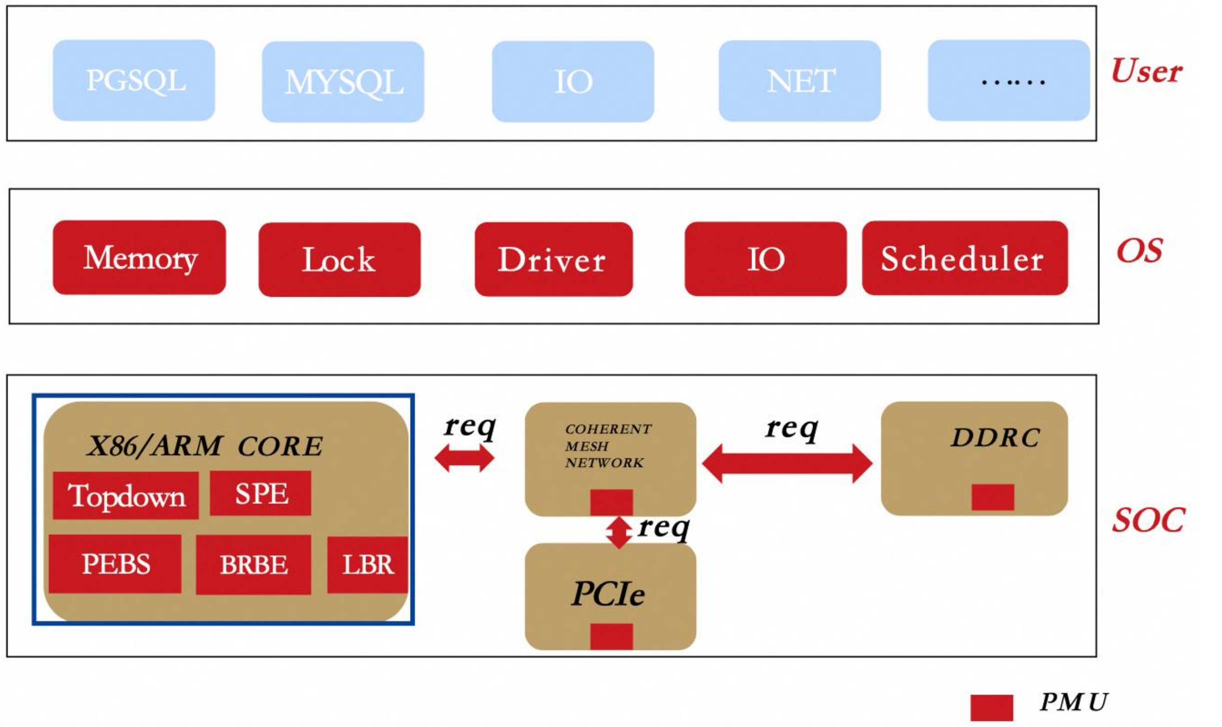


CPU Bottleneck Analysis and Multi-architecture Support

CPU Bottleneck Analysis and Multi-architecture Support

We use the following methods to analyze:

- **Applying the topdown analysis model**
  - Topdown for performance monitoring and analysis by perf and pas tool;
- **Multi-platform analysis framework:**
  - Essential for diverse cloud environments(X86, AMD, ARM, RISC-V)
  - PAS
- **Using advanced core pmu features:**
  - **BRBE**: BRBE is used to perform autoFDO to optimize database services by reducing frontend bound.
  - **SPE/PEBS**: Fine-grained metric analysis, such as instruction behavior and time latency, etc.

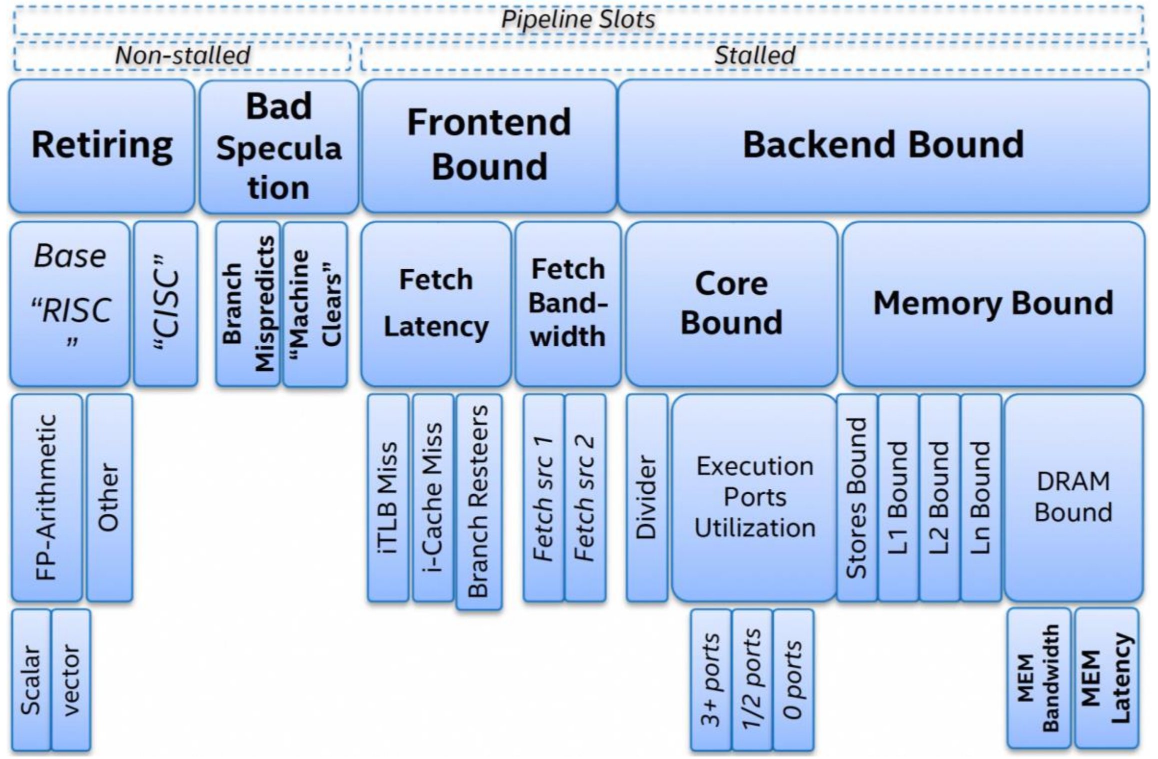


To solve the bottlenecks of the program on the core

Topdown Analysis Model – X86

Frontend Bound:This metric is the percentage of total slots that were stalled in the frontend of the processor.

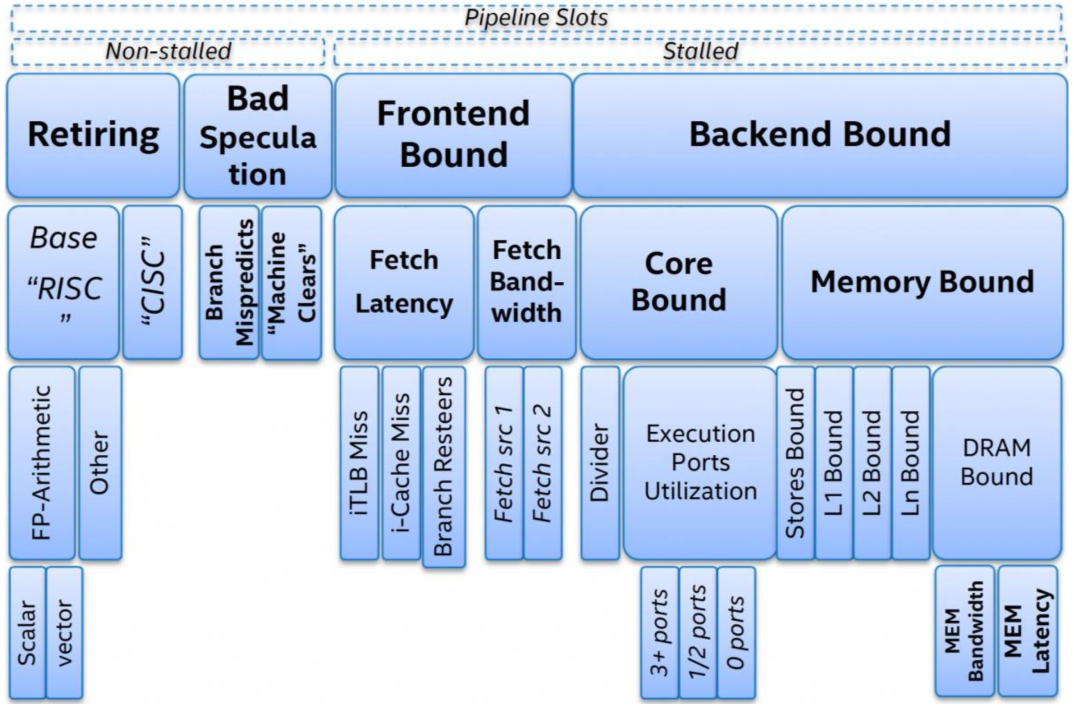
- Fetch Latency: CPU was stalled due to front-end latency issues;
  - Itlb miss, icache miss, branch resteers
- Fetch Bandwidth: CPU was stalled due to front-end bandwidth issues;
  - Inefficiencies in the instruction decoders;
  - Code restrictions for caching in the DSB;



Topdown Analysis Model – X86

Backend Bound: No uOps are being delivered due to a lack of required resources for accepting new uOps in the Back-End.

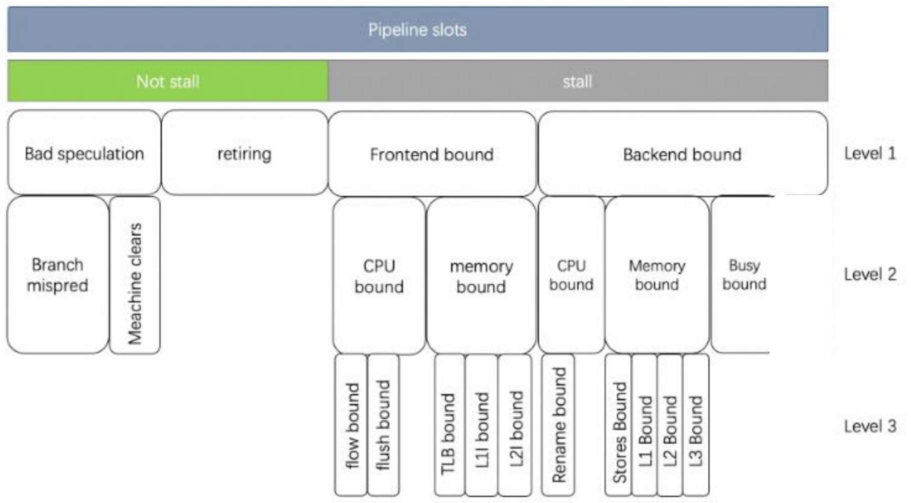
- Core Bound: Shortage in hardware compute resources, or dependencies software's instructions
  - Certain execution units are overloaded;
  - Dependencies in program's data or instruction flow are limiting the performance
- Memory Bound:
  - Store bound: CPU was stalled on store operations
  - Cache miss bound: L1, L2, L3;
  - Dram bound
    - Memory bandwidth
    - Memroy latency



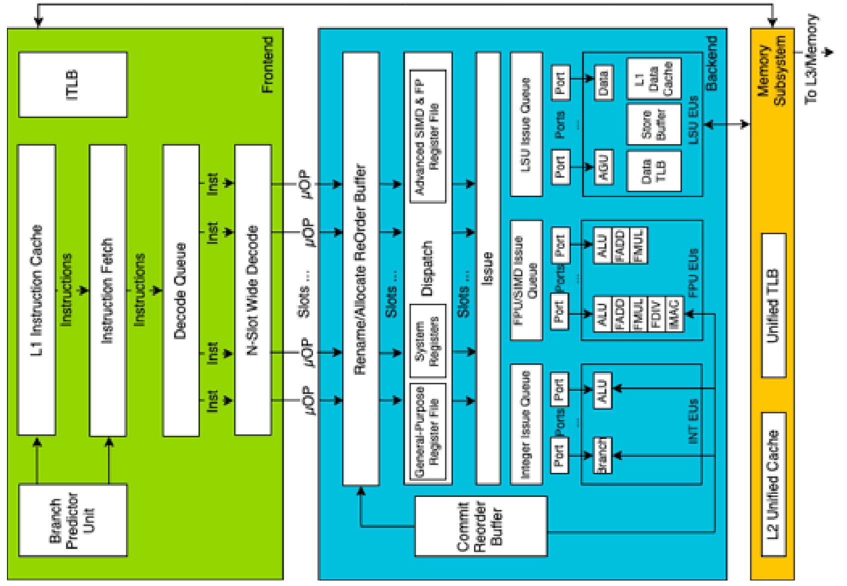
Topdown Analysis Model – ARM

Frontend Bound: This metric is the percentage of total slots that were stalled in the frontend of the processor.

- Memory Bound: Cycle stalled by TLBi miss, L1i miss and L2i miss;
- CPU Bound: Due to frontend core resource constraints not related to instruction fetch latency issues caused by memory access components.
    - Flow Bound: Decode unit is awaiting input from the branch prediction unit.
    - Flush Bound: Recovering from a pipeline flush caused by bad speculation



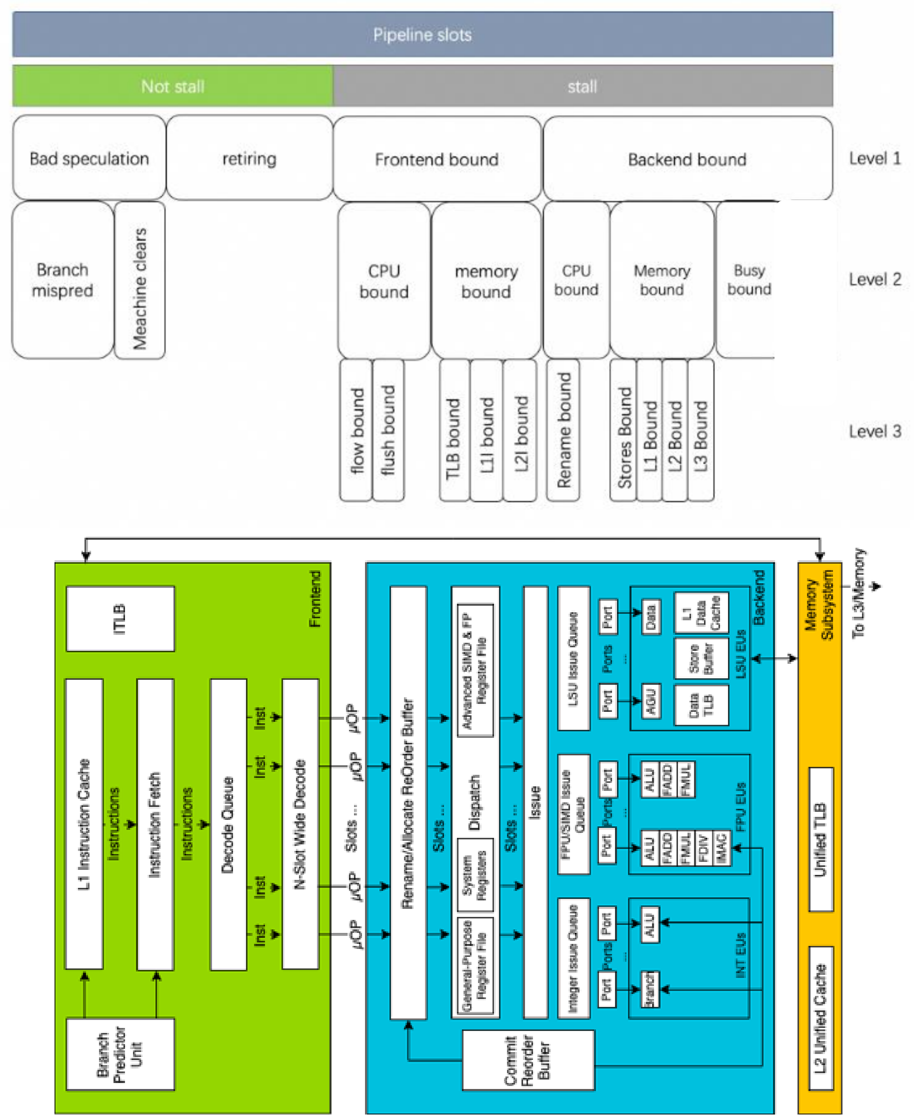
ARM Neoverse v3 core



Topdown Analysis Model – ARM

Backend Bound: No uOps are being delivered due to a lack of required resources for accepting new uOps in the Back-End

- Memory Bound: Data cache miss and slc miss;
- CPU Bound: Because of unavaliable rename regs to produce stall
- Busy Bound: Due to issue queues being full to accept operations



ARM Neoverse v3 core

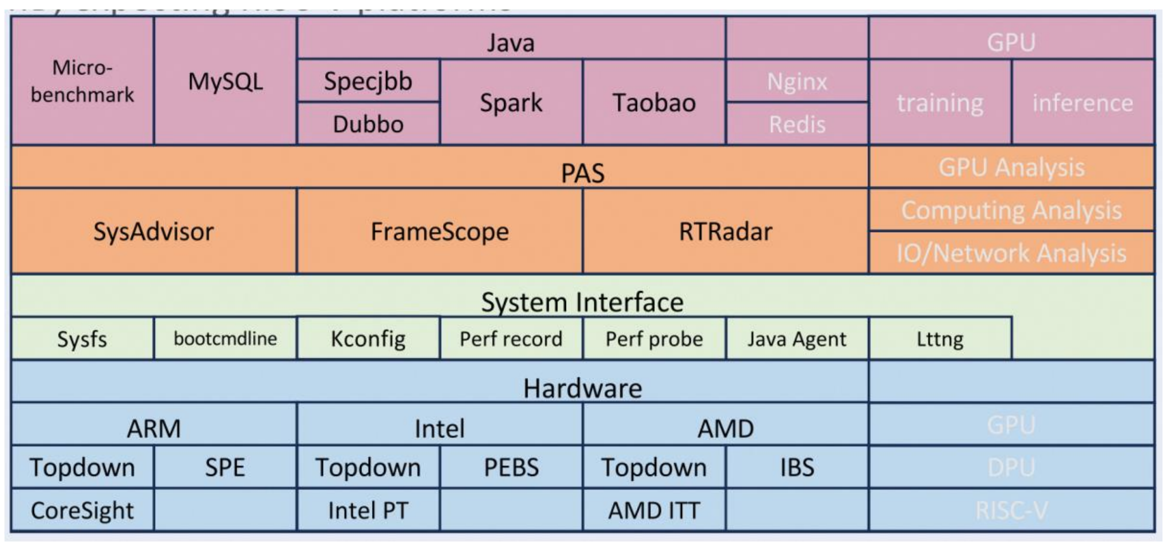
Multi-platform Analysis Framework -- PAS

Why need pas in cloud?

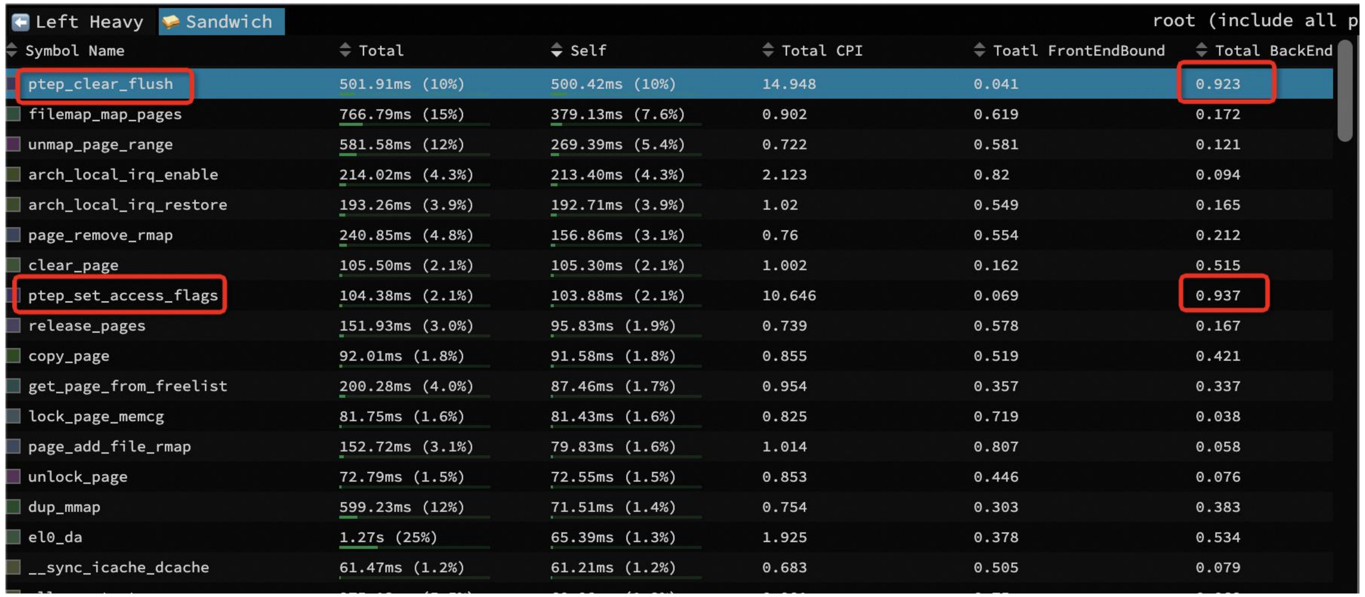
- Multiple architectures exist in cloud
- Need a lightweight analytical framework, perf is too heavy
- Requires fine-grained analysis at the instruction level
- Need to integrate different analytical capabilities to meet multiple business needs

PAS is designed to support multiple platforms

- Use Standard System Interfaces to get pmu data with low overhead
- Supports ARM/Intel/AMD, expecting RISC-V platforms
- FrameScope is the On-CPU analysing tool with improved observability by providing function-level topdown;
- RTRadar is the tool to achieve OS-level full observability, which conduct offcpu analysis.
- SysAdvisor is a tool to check and compare performance configuarations



Use Cases for PAS in Cloud



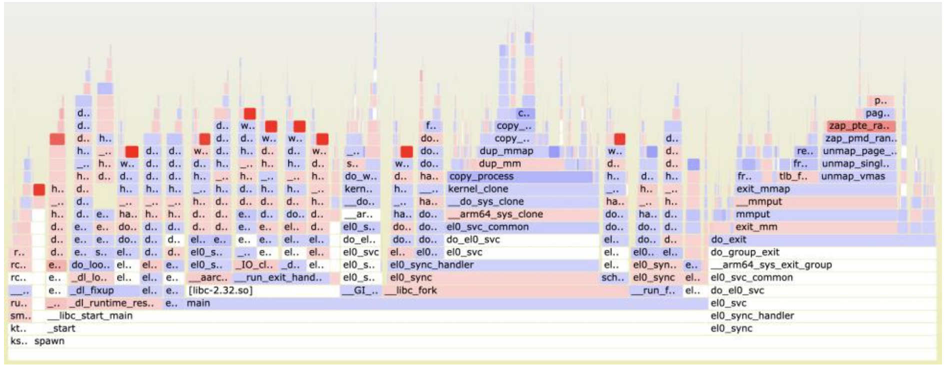
Left Heavy | Sandwich — root (include all p

Symbol Name	Total	Self	Total CPI	Toatl FrontEndBound	Total BackEnd
ptep_clear_flush	501.91ms (10%)	500.42ms (10%)	14.948	0.041	0.923
filemap_map_pages	766.79ms (15%)	379.13ms (7.6%)	0.902	0.619	0.172
unmap_page_range	581.58ms (12%)	269.39ms (5.4%)	0.722	0.581	0.121
arch_local_irq_enable	214.02ms (4.3%)	213.40ms (4.3%)	2.123	0.82	0.094
arch_local_irq_restore	193.26ms (3.9%)	192.71ms (3.9%)	1.02	0.549	0.165
page_remove_rmap	240.85ms (4.8%)	156.86ms (3.1%)	0.76	0.554	0.212
clear_page	105.50ms (2.1%)	105.30ms (2.1%)	1.002	0.162	0.515
ptep_set_access_flags	104.38ms (2.1%)	103.88ms (2.1%)	10.646	0.069	0.937
release_pages	151.93ms (3.0%)	95.83ms (1.9%)	0.739	0.578	0.167
copy_page	92.01ms (1.8%)	91.58ms (1.8%)	0.855	0.519	0.421
get_page_from_freelist	200.28ms (4.0%)	87.46ms (1.7%)	0.954	0.357	0.337
lock_page_memcg	81.75ms (1.6%)	81.43ms (1.6%)	0.825	0.719	0.038
page_add_file_rmap	152.72ms (3.1%)	79.83ms (1.6%)	1.014	0.807	0.058
unlock_page	72.79ms (1.5%)	72.55ms (1.5%)	0.853	0.446	0.076
dup_mmap	599.23ms (12%)	71.51ms (1.4%)	0.754	0.303	0.383
el0_da	1.27s (25%)	65.39ms (1.3%)	1.925	0.378	0.534
__sync_icache_dcache	61.47ms (1.2%)	61.21ms (1.2%)	0.683	0.505	0.079

Both functions call the flush_tlb_page function:

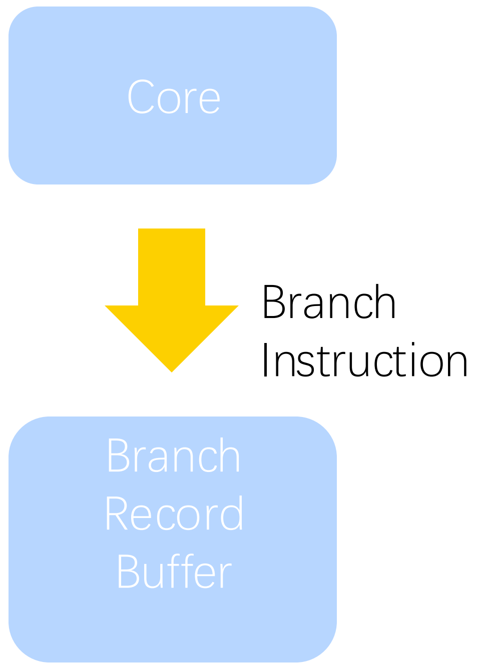
```
static inline void flush_tlb_page(struct vm_area_struct *vma,
                                  unsigned long uaddr)
{
      flush_tlb_page_nosync(vma, uaddr);
      dsb(ish);
}

static inline void flush_tlb_page_nosync(struct vm_area_struct *vma,
                                  unsigned long uaddr)
{
                                  unsigned long addr;

                                  dsb(ishst);
                                  addr = __TLBI_VADDR(uaddr, ASID(vma-
>vm_mm))
                                  __tlbi(vale1is, addr);
                                  __tlbi_user(vale1is, addr);
}
```

- For the instruction: __tlbi(vale1is, addr)
  - va: virtual address
  - l: last level, refers to PTE。
  - is: inner shareable, invalidate all cores TLB in inner shareable
- Ptep_clear_flush has a high backend bound due to tlb flushing;



Advanced Core PMU Features -- Branch Record Buffer Extension

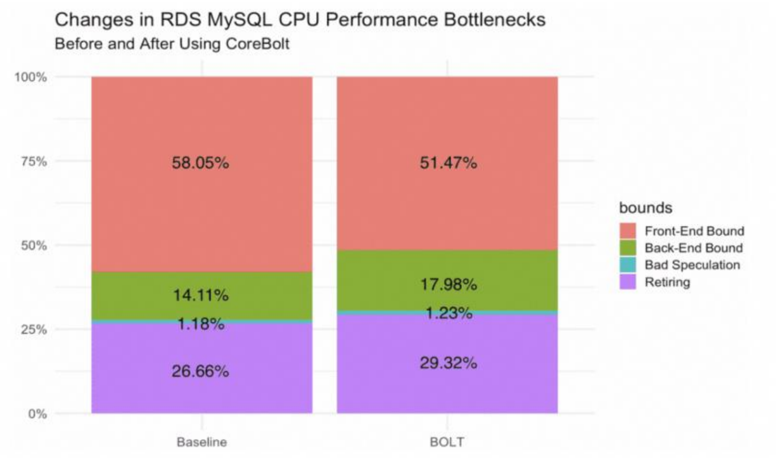
- We will use BRBE to optimize the frontend bound of database applications with autoFDO.
- Objective
  - Capture a recent sequence of branches in an easy-to-consume format
- Requirements
  - Low compute and memory overheads for capture/analysis;
  - Low overheads while recording
  - Low cost of context switch and on reading
- Branch Record
  - Source VA of taken branch/exception
  - Target VA of taken branch/exception
- Branch Record Buffer
  - EL1 feature, for recording EL0 and/or EL1;
- For risc-v the smctr (Control Transfer Records extension) is the same as ARM BRBE;



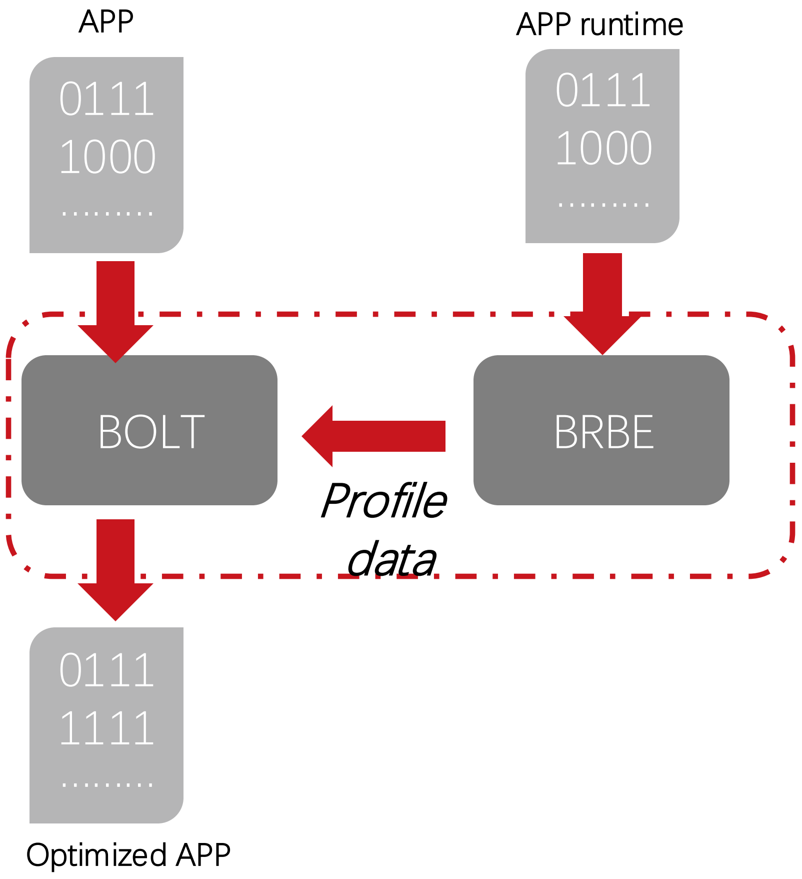
Practice of the BRBE In Cloud

We optimize database services by combining Bolt and BRBE

- Bolt's optimization principle
    - Bolt(binary optimization and layout tool)is a post-link optimizition tool
    - We collects the profile data(branchs track of program) of business programs during runtime;
    - Bolt using this profile data to directly modifies and optimizes the binary files;
    - Reducing the jump instructions on the critical execution path;
- Comparison of TPS of RDS MySQL:*read_only increased by 20%, read_write increased by 16%, and write_only increased by 22%.*

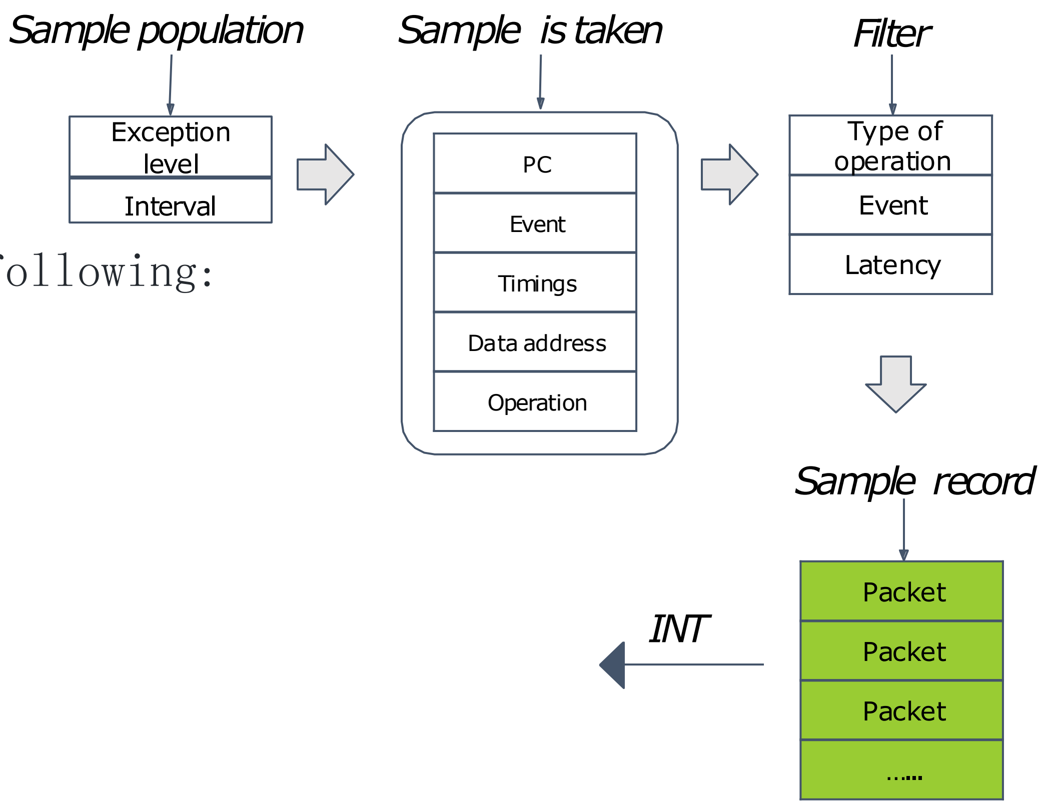


Compared with before optimization, frontend bound decreased



Advanced Core PMU Features -- Statistical Profiling Extension

- **PMU Defects:** PMU Due to the existence of interrupts, the PMU sampling instruction will deviate from the actual instruction position
- SPE advantage: can perform instruction-level accurate sampling
- The sampling process of SPE can be divided into the following:
  - 1. Choose an operation
  - 2. Collect data about the operation
  - 3. Optionally discard the record based on a filter
  - 4. Write the record to memory
  - 5. Interrupt when the buffer is full
- SPE trace the processor pipeline
  - Record operations (memory, exception, SVE, etc)
  - Gather associated information for the operation (PC value, data address, event type, timestamp, etc.)



Advanced Core PMU Features -- Statistical Profiling Extension

SPE Data Recording Format

- One record represents the tracking record of an operation
- One record contains multiple packets.
- Different packets represent different information, including:
    - Address: Virtual PC
    - Event Source: Where in the system the data was returned from. E.g. Level 1 cache, Last Level cache, or another socket in a multi-socket system;
    - ISSUE: Number of cycles taken from Dispatch to Issue
    - TOT: Number of cycles taken from Dispatch to Complete
    - XLAT: Number of cycles spent on pagetable walk
    - Events: Events occur: L1D-ACCESS
    - Type: Operation type
    - Address: Virtual address that instruction access
    - Timestamp
- ARM supports multiple events in patckets;
- The Operation type in packet:
    - Load/store/branch/sve,etc instructions

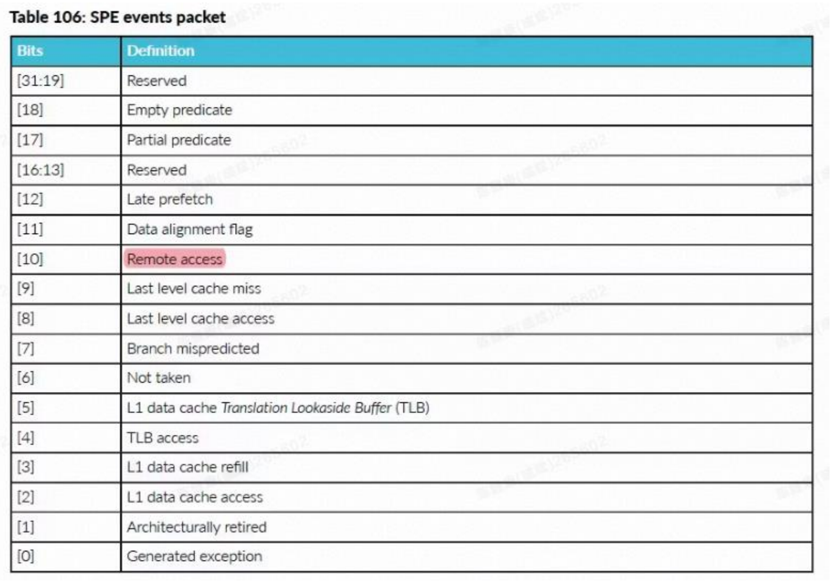
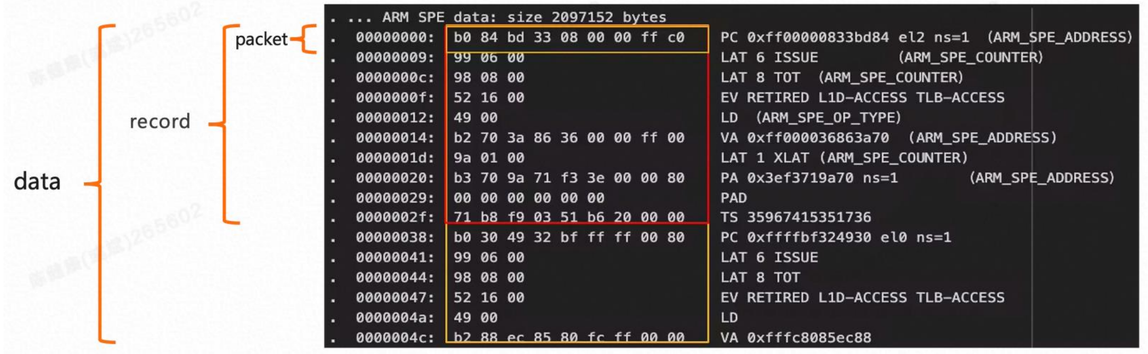


Table 106: SPE events packet

Bits	Definition
[31:19]	Reserved
[18]	Empty predicate
[17]	Partial predicate
[16:13]	Reserved
[12]	Late prefetch
[11]	Data alignment flag
[10]	Remote access
[9]	Last level cache miss
[8]	Last level cache access
[7]	Branch mispredicted
[6]	Not taken
[5]	L1 data cache *Translation Lookaside Buffer* (TLB)
[4]	TLB access
[3]	L1 data cache refill
[2]	L1 data cache access
[1]	Architecturally retired
[0]	Generated exception



```
. ... ARM SPE data: size 2097152 bytes
.  00000000:  b0 84 bd 33 08 00 00 ff c0    PC 0xff00000833bd84 el2 ns=1  (ARM_SPE_ADDRESS)
.  00000009:  99 06 00                      LAT 6 ISSUE          (ARM_SPE_COUNTER)
.  0000000c:  98 08 00                      LAT 8 TOT  (ARM_SPE_COUNTER)
.  0000000f:  52 16 00                      EV RETIRED L1D-ACCESS TLB-ACCESS
.  00000012:  49 00                         LD  (ARM_SPE_OP_TYPE)
.  00000014:  b2 70 3a 86 36 00 00 ff 00    VA 0xff000036863a70  (ARM_SPE_ADDRESS)
.  0000001d:  9a 01 00                      LAT 1 XLAT (ARM_SPE_COUNTER)
.  00000020:  b3 70 9a 71 f3 3e 00 00 80    PA 0x3ef3719a70 ns=1         (ARM_SPE_ADDRESS)
.  00000029:  00 00 00 00 00 00             PAD
.  0000002f:  71 b8 f9 03 51 b6 20 00 00    TS 35967415351736
.  00000038:  b0 30 49 32 bf ff ff 00 80    PC 0xffffbf324930 el0 ns=1
.  00000041:  99 06 00                      LAT 6 ISSUE
.  00000044:  98 08 00                      LAT 8 TOT
.  00000047:  52 16 00                      EV RETIRED L1D-ACCESS TLB-ACCESS
.  0000004a:  49 00                         LD
.  0000004c:  b2 88 ec 85 80 fc ff 00 00    VA 0xfffc8085ec88
```

Practice of SPE in Cloud

- DDRC statistics only count the latency of requests within DDRC;
- The ddr latency calculated by DDRC PMU may be low.

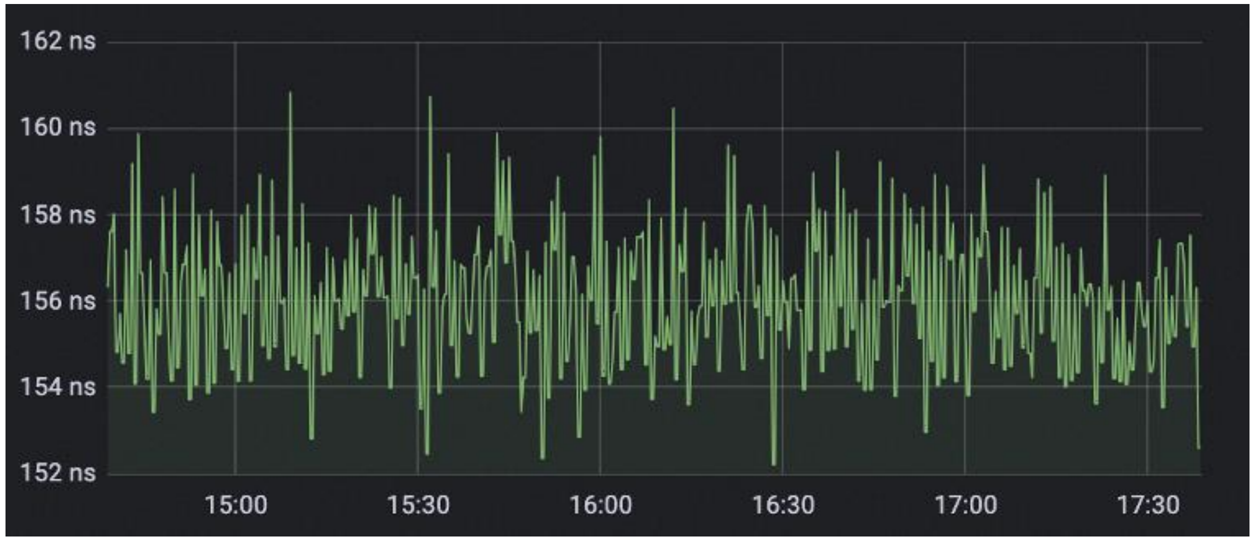
- Therefore, we can monitor the memory access latency by SPE;

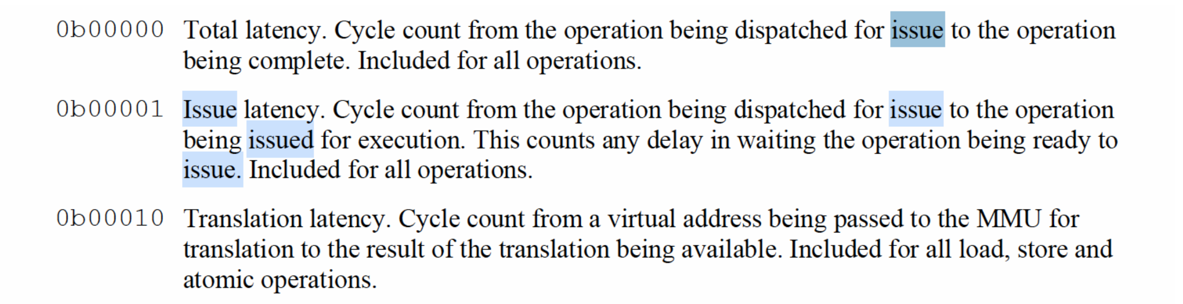
- We know that spe can count the three latency of load instructions:;
  - Total latency(total lat);
  - Issue latency(issue lat);
  - Xlat latency (xlat);

- Memory latency equals total latency minus address translation and issue latency

mem lat = total lat - xlat – issue lat

DDR latency





0b00000	Total latency. Cycle count from the operation being dispatched for issue to the operation being complete. Included for all operations.
0b00001	Issue latency. Cycle count from the operation being dispatched for issue to the operation being issued for execution. This counts any delay in waiting the operation being ready to issue. Included for all operations.
0b00010	Translation latency. Cycle count from a virtual address being passed to the MMU for translation to the result of the translation being available. Included for all load, store and atomic operations.

Disadvantages of this approach: The tool needs to be customized;
Demand : A new independent PMU is needed to count latency, for example, AMD has the XI pmu

Advanced Core PMU Features -- Statistical Profiling Extension

Packet Data Source:

- N2 TRM: Data source: Refers to where does core get data from;
- There are currently 8 data sources in spec, but the number of implementations varies for different SOC.

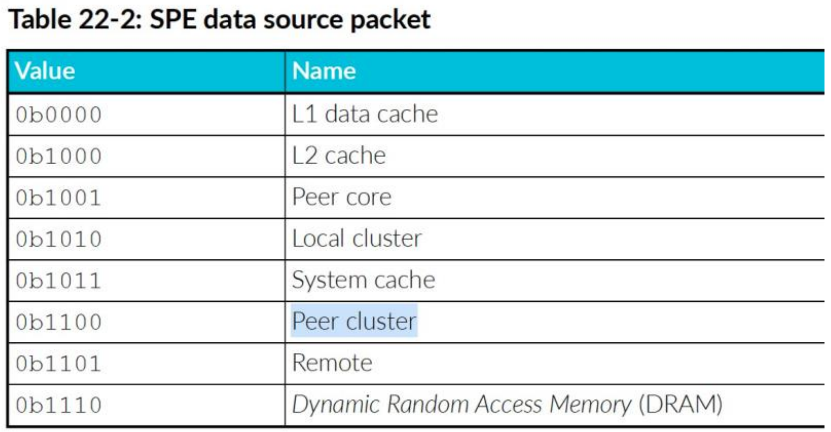
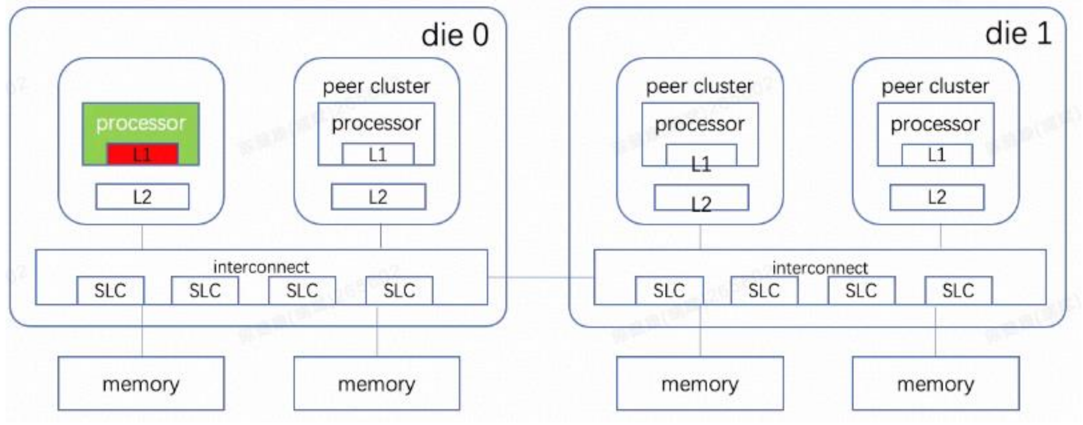


Table 22-2: SPE data source packet

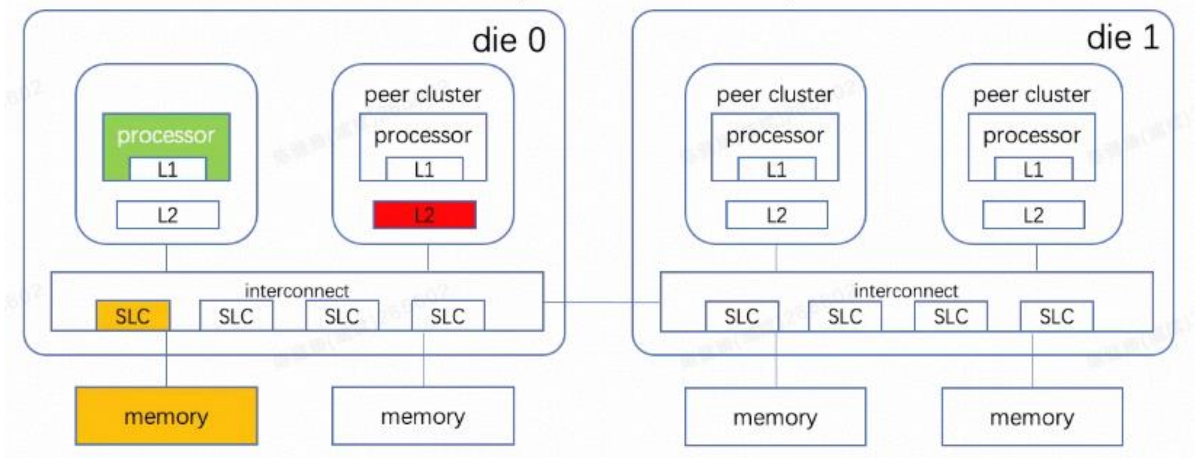
Value	Name
0b0000	L1 data cache
0b1000	L2 cache
0b1001	Peer core
0b1010	Local cluster
0b1011	System cache
0b1100	Peer cluster
0b1101	Remote
0b1110	*Dynamic Random Access Memory* (DRAM)

- N2: One core per cluster

data source is l1 cache



data source is peer cluster.



🟩 the core requesting data

🟥 new data

🟨 old data

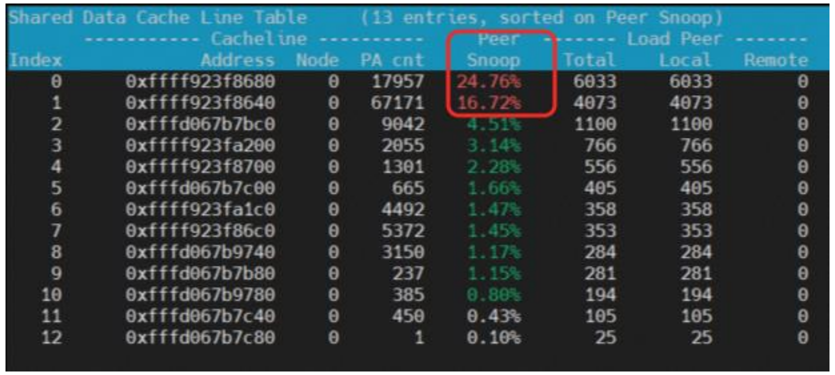
Practice of SPE in Cloud

Another important use case is detecting false sharing in X265 by SPE data source

False sharing: multiple threads concurrently modifying variables within the same cache line lead to performance degradation;

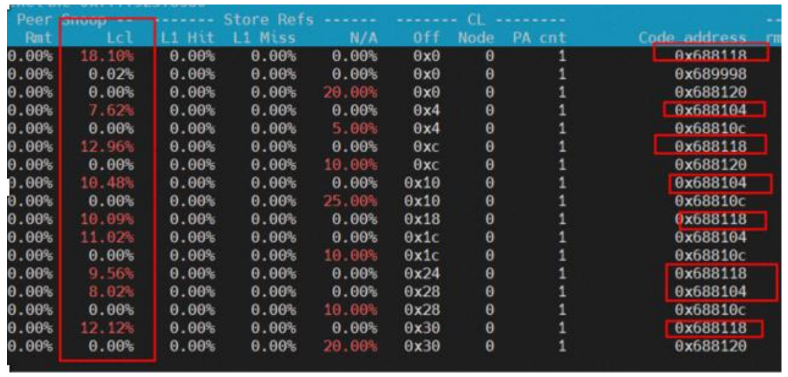
We use perf c2c cmd to collect the sampling data of X265

- In the perf c2c statistics, each row here represents a cacheline.
- The peer snoop field describes the percentage of peer snoop.
- A high peer snoop value indicates false sharing has occurred.
- It can be seen that the addresses where false sharing occurs are mainly 0x688118 / 0x688104
- Through spe, we can find the code line where false sharing occurs and solve the problem
- After optimization, performance increased by 8.6%



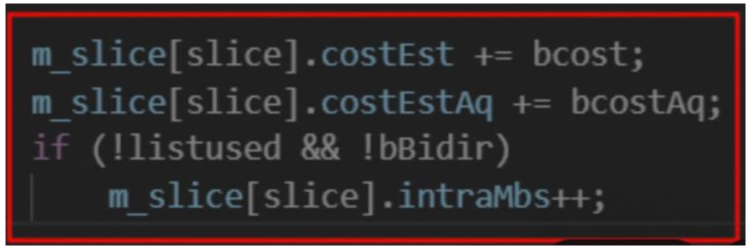
```
Shared Data Cache Line Table     (13 entries, sorted on Peer Snoop)
      ----------- Cacheline ----------     Peer   ------- Load Peer -------
Index           Address  Node  PA cnt     Snoop    Total     Local    Remote
    0    0xffff923f8680     0   17957    24.76%     6033      6033         0
    1    0xffff923f8640     0   67171    16.72%     4073      4073         0
    2    0xfffd067b7bc0     0    9042     4.51%     1100      1100         0
    3    0xffff923fa200     0    2055     3.14%      766       766         0
    4    0xffff923f8700     0    1301     2.28%      556       556         0
    5    0xfffd067b7c00     0     665     1.66%      405       405         0
    6    0xffff923fa1c0     0    4492     1.47%      358       358         0
    7    0xffff923f86c0     0    5372     1.45%      353       353         0
    8    0xfffd067b9740     0    3150     1.17%      284       284         0
    9    0xfffd067b7b80     0     237     1.15%      281       281         0
   10    0xfffd067b9780     0     385     0.80%      194       194         0
   11    0xfffd067b7c40     0     450     0.43%      105       105         0
   12    0xfffd067b7c80     0       1     0.10%       25        25         0
```

perf c2c statistics



```
 Peer Snoop ---------- Store Refs ------   ------- CL --------                 --
  Rmt      Lcl   L1 Hit  L1 Miss      N/A     Off  Node  PA cnt       Code address  rm
0.00%   18.10%    0.00%    0.00%    0.00%     0x0     0       1           0x688118
0.00%    0.02%    0.00%    0.00%    0.00%     0x0     0       1           0x689998
0.00%    0.00%    0.00%    0.00%   20.00%     0x0     0       1           0x688120
0.00%    7.62%    0.00%    0.00%    0.00%     0x4     0       1           0x688104
0.00%    0.00%    0.00%    0.00%    5.00%     0x4     0       1           0x68810c
0.00%   12.96%    0.00%    0.00%    0.00%     0xc     0       1           0x688118
0.00%    0.00%    0.00%    0.00%   10.00%     0xc     0       1           0x688120
0.00%   10.48%    0.00%    0.00%    0.00%    0x10     0       1           0x688104
0.00%    0.00%    0.00%    0.00%   25.00%    0x10     0       1           0x68810c
0.00%   10.09%    0.00%    0.00%    0.00%    0x18     0       1           0x688118
0.00%   11.02%    0.00%    0.00%    0.00%    0x1c     0       1           0x688104
0.00%    0.00%    0.00%    0.00%   10.00%    0x1c     0       1           0x68810c
0.00%    9.56%    0.00%    0.00%    0.00%    0x24     0       1           0x688118
0.00%    8.02%    0.00%    0.00%    0.00%    0x28     0       1           0x688104
0.00%    0.00%    0.00%    0.00%   10.00%    0x28     0       1           0x68810c
0.00%   12.12%    0.00%    0.00%    0.00%    0x30     0       1           0x688118
0.00%    0.00%    0.00%    0.00%   20.00%    0x30     0       1           0x688120
```

Detailed information corresponding to cache line 0xffff923f8680



```
m_slice[slice].costEst += bcost;
m_slice[slice].costEstAq += bcostAq;
if (!listused && !bBidir)
    m_slice[slice].intraMbs++;
```

Hot code

Uncore PMU Analysis and Unified PMU Support

Uncore PMU Analysis and Unified PMU Support

The main usage scenarios of uncore pmu in cloud are as follows:

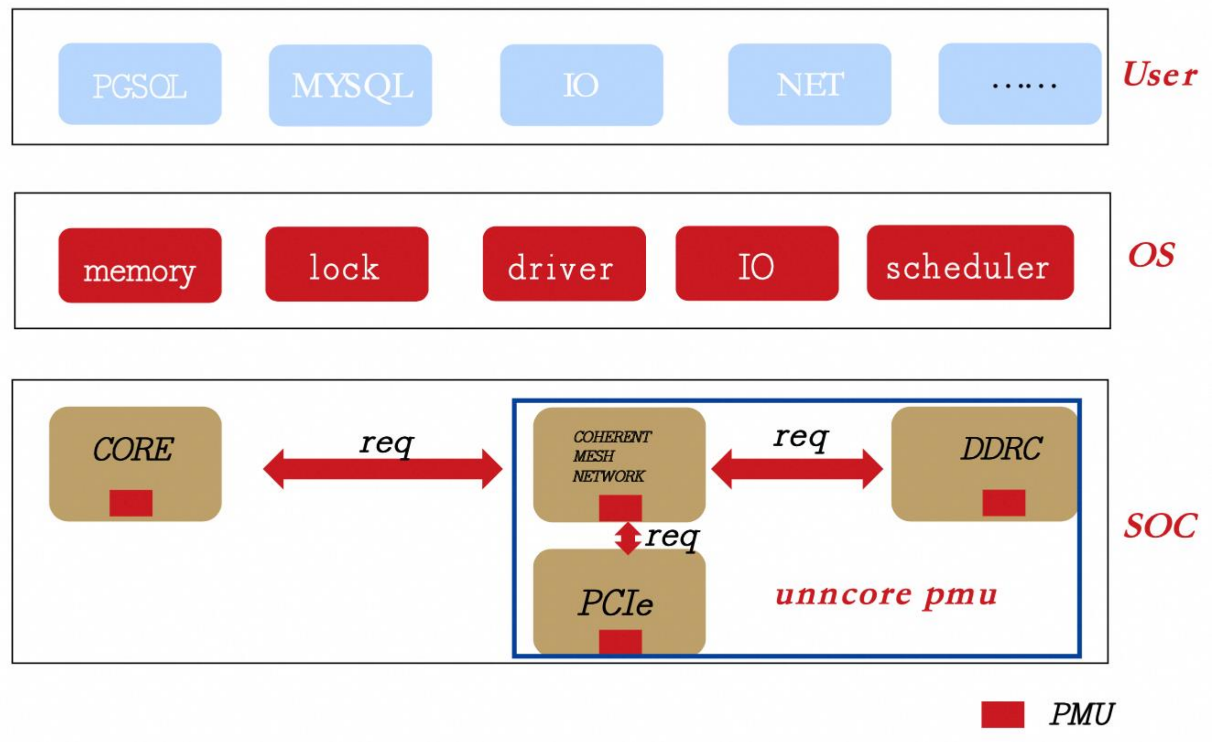
- **Low-overhead monitoring**
- **Bandwidth and Latency:** Cloud services focus on bandwidth and latency between components.
  - Inter-die bandwidth
  - Inter-socket bandwidth
  - Memory bandwidth
  - Memory latency
  - Some IO and network services also focus on PCIe bandwidth;
- **Last level cache pmu events**

Currently the key components of uncore pmu are:

- CMN PMU
- PCIe PMU
- DDRC PMU

ARM propose a new uncore pmu architecture

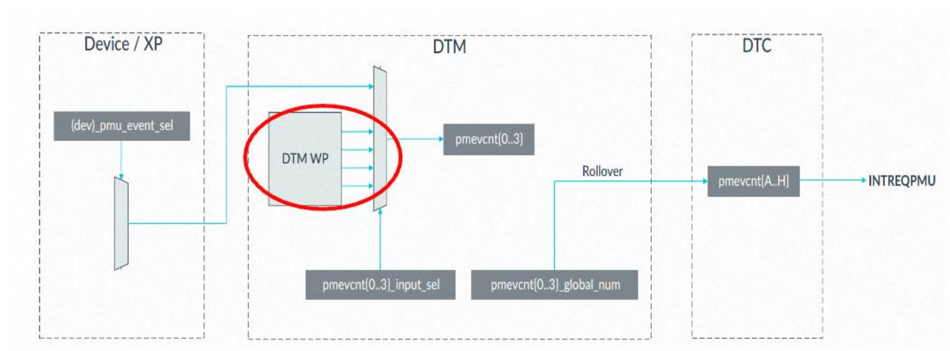
- Use coresight pmu replace all uncore pmu

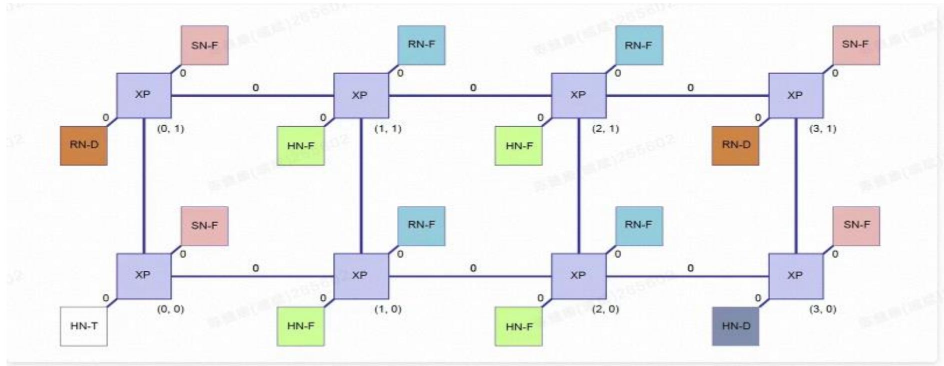


Bandwidth and Cache Analysis of CMN PMU

In ARM,CMN PMU is mainly used to analyze cross-die data bandwidth and last level cache.

- CMN is composed of many different types of nodes in a cpu die
  - XP: Crosspoint – A switch or router logic module.
  - RN-F(a CPU core), HN-F (SLC cache module), SN-F(connect Memory Controller)
- The CMN PMU consists of DTM (local in xp) and DTC(global in die) components;
  - The DTM is located in each XP and has 4 local counters that can handle up to 4 events of the connected device nodes and the XP itself;
  - DTC has only 2 counters per CMN instance and 8 global counters;
  - The local counter in DTM overflows and carries into the global counter, and support overflow interrupt.
  - Each node supports different events, for example, HNF supports last level cache related events.

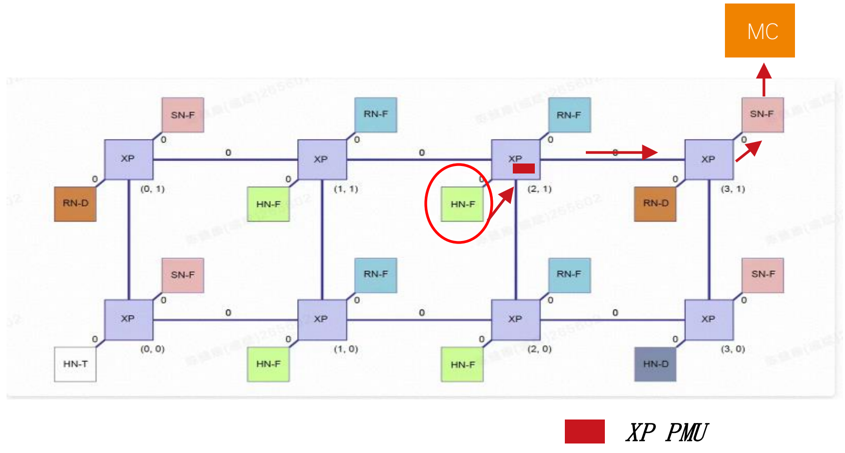




Bandwidth and Cache Analysis of CMN PMU

- HNF contains last level cache miss rate events and some memory request events
- HNF SLC cache miss rate can be obtained through the following two events
  - PMU_HN_CACHE_MISS_EVENT (0x1)
    - Counts the total cache misses.
  - PMU_HNSLC_SF_CACHE_ACCESS_EVENT (0x2)
    - The total number of cache accesses.
  - Metric:

  Cache miss rate (%) = Total cache misses / Total cache accesses

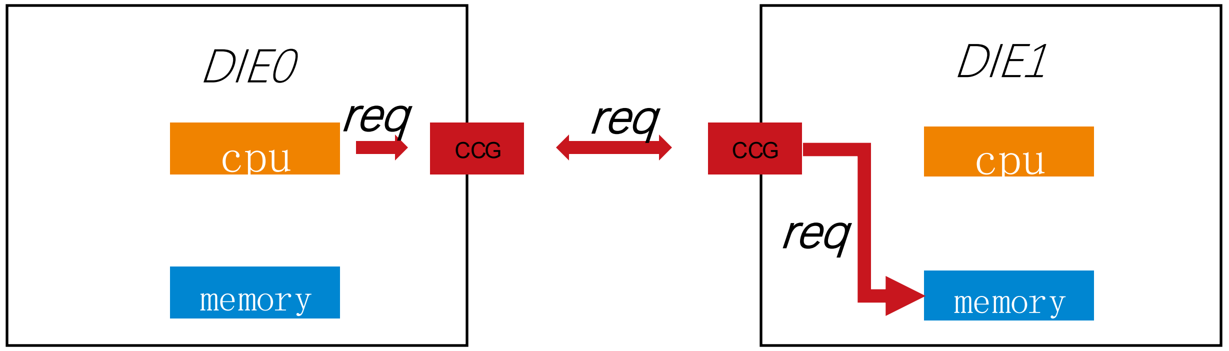


- If have a cache miss, hnf can send a memroy request
- If HNF resend memory requests frequently, the memory controller has become a bottleneck.
- MC message retry rate mainly include two events:
  - PMU_HN_MC_RETRIES_EVENT Number of requests that are retried to the memory controller.
  - PMU_HN_MC_REQS_EVENT Total number of requests that are sent to the memory controller.
  - metric:

  MC message retry rate (%) = MC total messages retried / MC total messages received x 100

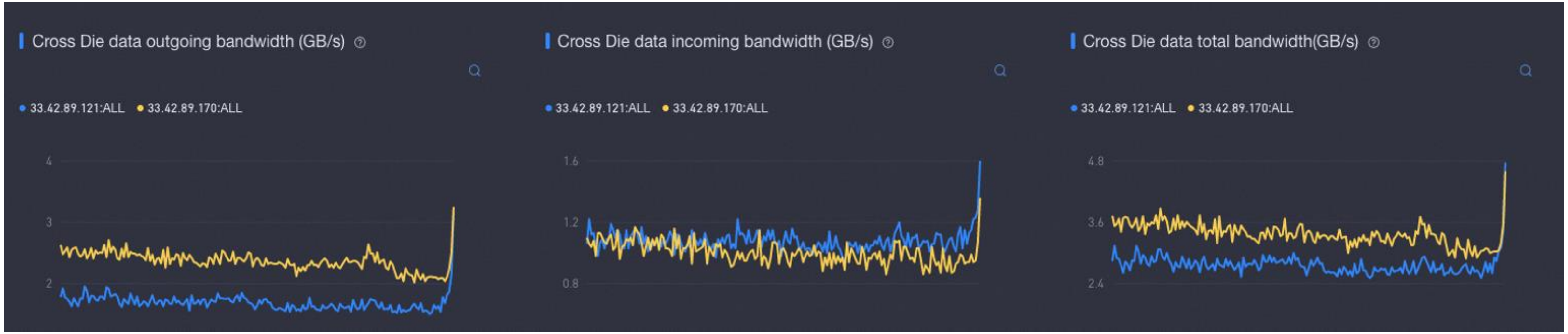
Bandwidth and Cache Analysis of CMN PMU

- CCG connects different CPU dies and is responsible for exchanging data between CPU dies.
- The most important events for CCG are to count the data bandwidth across the die.
- Data Bandwidth events metric:
  - Inter-Socket RX DATA Bandwidth = Flit_Size * CCG_WP0_RX_DATA_FLIT * CMN_Freq / PMU_CYCLE_COUNTER
  - Inter-Socket TX DATA Bandwidth = Flit_Size * CCG_WP0_TX_DATA_FLIT * CMN_Freq / PMU_CYCLE_COUNTER
  - Inter-Die RX DATA Bandwidth = Flit_Size * CCG_WP0_RX_DATA_FLIT * CMN_Freq / PMU_CYCLE_COUNTER
  - Inter-Die TX DATA Bandwidth = Flit_Size * CCG_WP0_TX_DATA_FLIT * CMN_Freq / PMU_CYCLE_COUNTER
  - Flit_Size = 32 Bytes
- CCG_WP0_RX_DATA_FLIT: The CMN PMU can count the number of data flit passing through CCG
- PMU_CYCLE_COUNTER: CPU cycle spent transferring data



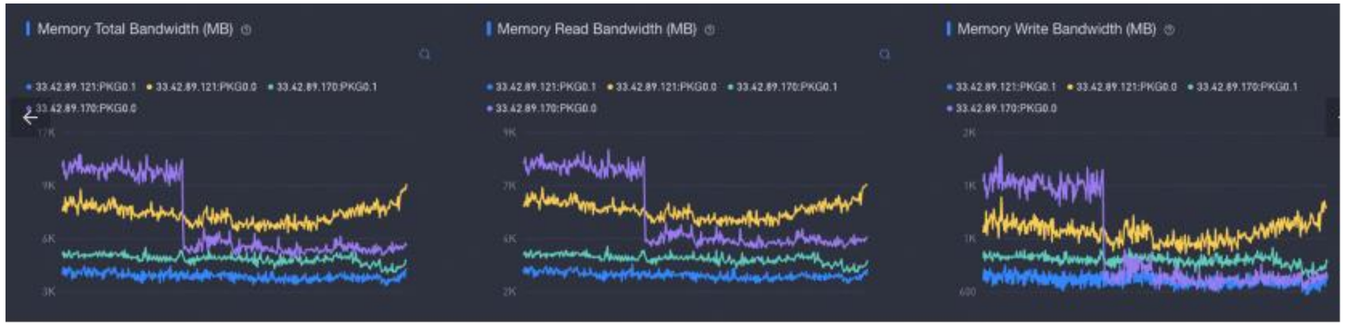
Bandwidth and Cache Analysis of CMN PMU

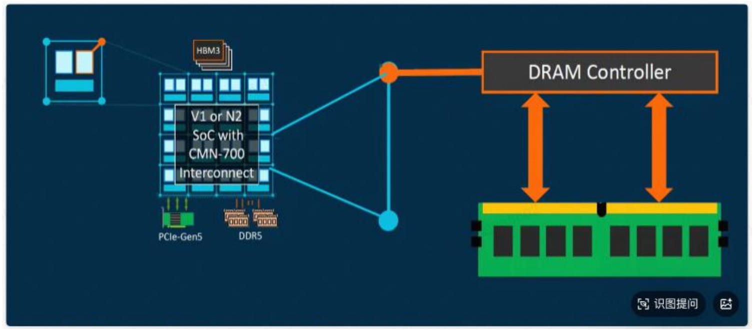
- In actual use, we will monitor the real-time data bandwidth across the die using the CCG XP PMU.
- The following figure shows the real-time monitoring of the business:



Memory bandwidth Analysis Via DDRC PMU

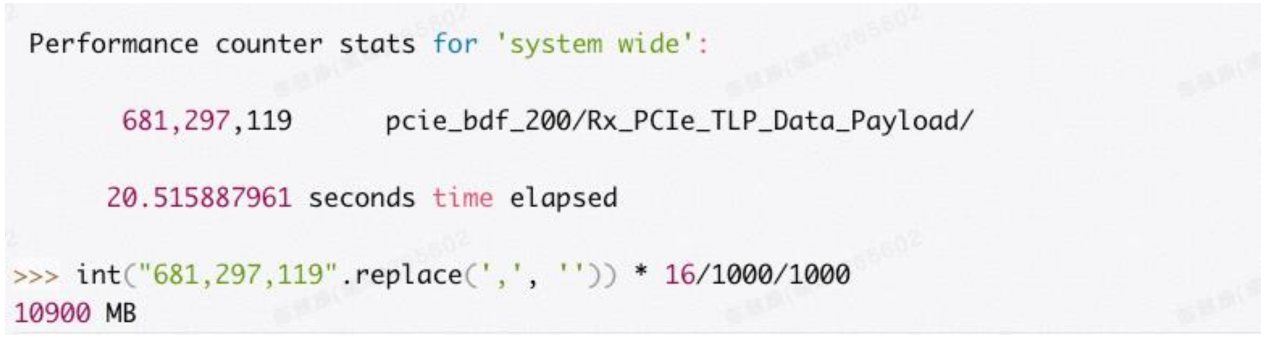
- For DDRC PMU, we often use it to monitor the DDR memory bandwidth of the entire machine.
- In addition, we will monitor the memory bandwidth usage of each machine in the cloud in real time online.
- And the main events we monitor include the following:
  - Memory total bandwith
  - memory read bandwidth
  - memory write bandwidth;





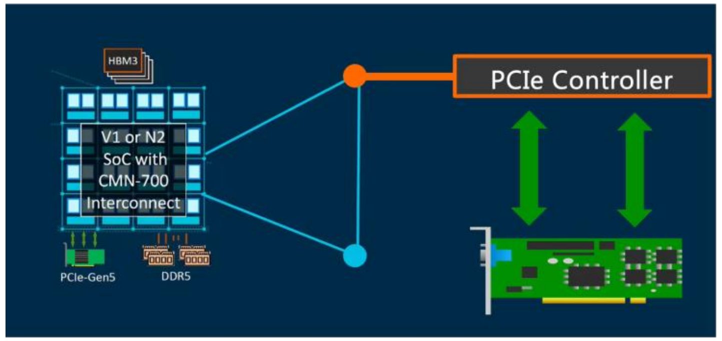
Device Bandwidth Analysis of PCIe PMU

- In cloud, we also use PCIe PMU to count the bandwidth of IO and Net on the PCIe link.
- PCIe controller PMU provides:
  - Time Based Analysis (RX/TX data throughput and time spent in each state)
  - Lane Event counters (Error and Non-Error for lanes)
- **Time Based Analysis**: the counters are 64-bit width and measure data in two categories,
  - Percentage of time does the controller stay in each state in a configurable duration
  - Amount of data processed (Units of 16 bytes).
- **Lane Event counters**: Event (Error and Non-Error) across different layers of PCIe protocol, a 32 bit counter
  - Multiple event counters group, and each group include multiple events;
- For PCIe PMU, the events we commonly use are TLP Bandwidth, including
  - PCIe RX Bandwidth: PCIE_RX_DATA * 16B / Measure_Time_Window
  - PCIe TX Bandwidth: PCIE_TX_DATA * 16B / Measure_Time_Window



```
Performance counter stats for 'system wide':

     681,297,119      pcie_bdf_200/Rx_PCIe_TLP_Data_Payload/

    20.515887961 seconds time elapsed

>>> int("681,297,119".replace(',', '')) * 16/1000/1000
10900 MB
```

FIO

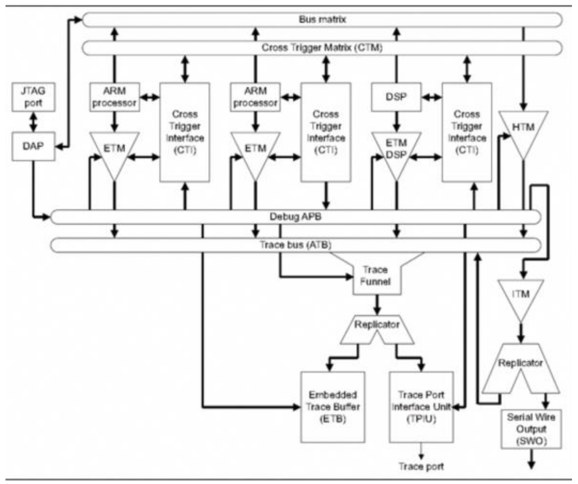


Use Coresight PMU to Uniformly Manage Uncore

- Traditional uncore pmu architectural features
  - In the traditional uncore pmu architecture, all pmu components are independent;
  - Poor maintainability and scalability, as every new uncore architecture requires a unique driver.

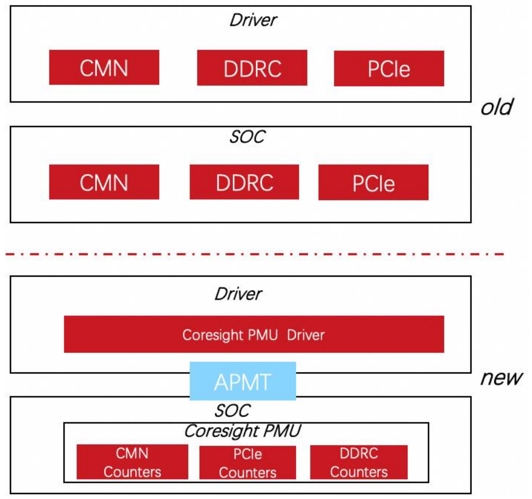
- New CoreSight PMU architecture
  - All uncore pmu events are managed by a coresight pmu hardware component;
  - There is only one coresight pmu driver in the driver layer;
  - Different uncore pmu component events can be extended in the same driver

- Introduced APMT new acpi table for extending uncore pmu nodes;
  - APMT(ARM Performance Monitoring Unit Table)
  - APMT supports multiple node types:
    - Memory controller, smmu, pcie, cpu cache



Bus matrix
Cross Trigger Matrix (CTM)
JTAG port
DAP
ARM processor
ETM
Cross Trigger Interface (CTI)
ARM processor
ETM
Cross Trigger Interface (CTI)
DSP
ETM DSP
Cross Trigger Interface (CTI)
HTM
Debug APB
Trace bus (ATB)
Trace Funnel
Replicator
ITM
Replicator
Embedded Trace Buffer (ETB)
Trace Port Interface Unit (TPIU)
Trace port
Serial Wire Output (SWO)

Coresight can be used to trace branch and memory access instructions.



Uncore PMU new mode

RISC-V proposal

- RFC Proposal1:

Motivtion:
Mixed deployment runs multiple containers on the same node with both online and offline applications. Since online applications are latency-sensitive, we need a Latency PMU to monitor request latency.

Issue:
- DDRC statistics only count the latency of requests within DDRC.
- It is possible that the foreground request is blocked in CMN, while the DDR is still relatively idle. As a result, even if the overall latency is longer, the latency seen only from DDR is relatively small.

Solution:
- Add a new pmu component on the risc-v core;
- The new core pmu component needs to be able to count the latency of the entire life cycle of a request from the core.
- The new core pmu needs to support the collection of latency indicators such as memory read request latency, write request latency, and total latency.

RISC-V proposal

- RFC Proposal2:

Motivation:

- Lack of unified PMU specifications results in numerous redundant perf drivers in the kernel with repetitive code for device discovery, interrupt registration, and counter enablement. ARM's Coresight PMU specification addresses this by providing a unified design for components like DDR, SMMU, and PCIe Root Complex.

Issue:

- Lack of a unified uncore pmu for the risc-v architecture

Solution:

- Add an uncore unified architecture pmu adapted to the risc-v architecture
- Add new acpi table to adapt risc-v uncore pmu node;
- The pmu needs to support multi-component event statistics and add multi-component counters
- Design supports overflow interrupt

Q & A