[Public]

AMD PQOS White Paper for AMD EPYC™ 9004 and 9005 Series Processors

Publication #	**69127** Revision: **1.00**
Issue Date:	**November 2025**

Advanced Micro Devices

[Public]

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED “AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD Arrow logo, AMD EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. CXL is a trademark of Compute Express Link Consortium, Inc. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries. Ubuntu and the Ubuntu logo are registered trademarks of Canonical Ltd.

© 2025 Advanced Micro Devices, Inc. All rights reserved.

[Public]

Contents

[Public]

[Public]

[Public]

List of Figures

[Public]

List of Tables

[Public]

Revision History

Date	Revision	Description
November 2025	1.00	Initial release

[Public]

Chapter 1 Introduction

Modern computing platforms face increasing pressure to deliver consistent performance across diverse workloads, especially in multi-tenant and latency-sensitive environments. AMD's Platform Quality of Service (PQOS) architecture addresses this challenge by providing hardware-enforced mechanisms for monitoring, allocating, and enforcing shared resources such as last level (L3) cache and memory bandwidth.

The purpose of this document is to introduce the PQOS extension that is already available on AMD platforms and to demonstrate how to use its features to achieve predictable performance, workload isolation, and fairness in shared-resource systems.

These PQOS features are available on AMD EPYC™ 9004 series processors, codenamed "Genoa," and AMD EPYC 9005 series processors, codenamed "Turin," enabling customers to deploy them for applications that include these products.

1.1 Intended Audience

This white paper is intended for system architects, performance engineers, virtualization specialists, and platform software developers working in cloud, data-center, edge, and embedded environments.

1.2 Definitions

The following table defines the terms and acronyms that are used in this document.

Table 1. Definitions

Term	Description
ABMC	Assignable Bandwidth Monitoring Counters
BMEC	Bandwidth Monitor Event Configuration
CAT	Cache Allocation Technology
CBM	Cache Bitmask
CCX	Core complex
CDP	Code and Data Prioritization
COS	Class of Service
CMT	Cache Monitoring Technology
CXL	Compute Express Link
L2 cache	Level 2 cache

[Public]

Term	Description
L3 cache	Level 3 cache (last level cache)
LLC	Last level cache (L3 cache)
LRU	Least Recently Used
MBA	Memory Bandwidth Allocation
MBM	Memory Bandwidth Monitoring
MLC	Memory Latency Checker
NUMA	Non-Uniform Memory Access
PQE	PQOS Enforcement
PQM	PQOS Monitoring
PQOS	Platform Quality of Service
RMID	Resource Monitoring ID
SDCIAE	Smart Data Cache Injection Allocation Enforcement
SMBA	Slow Memory Bandwidth Allocation
UMC	Unified Memory Controller
VM	Virtual machine

1.3 Reference Documents

The following table lists official documentation and technical resources that support AMD PQOS features, including architectural specifications, software enablement guides, and performance analysis tools.

Table 2. Reference Documents

Reference	Publication #	Link
AMD64 Architecture Programmer's Manual, Volume 2: System Programming	24593	https://docs.amd.com/search/all?query=24593&content-lang=en-US
Smart Data Cache Injection (SDCI) White Paper	58725	https://docs.amd.com/search/all?query=58725&content-lang=en-US
Linux® resctrl options	—	https://kernel.org/doc/Documentation/x86/resctrl.rst
Ubuntu® manpage: pqos, pqos-msr, pqos-os	—	https://manpages.ubuntu.com/manpages/noble/man8/pqos-os.8.html

[Public]

1.4 Background

As part of our ongoing innovation roadmap, AMD continues to evolve the PQOS architecture with additional advanced features. These enhancements are currently a work in progress and aim to further expand the capabilities of resource control, monitoring granularity, and enforcement precision. By showcasing real-world use cases and performance profiling data, this paper demonstrates the practical benefits of AMD PQOS features and encourages customers to explore their use in production environments.

1.5 Benefits

AMD PQOS features, including cache and memory bandwidth control, are fully accessible through standard Linux interfaces such as the resctrl filesystem and the pqos tool. No proprietary software, vendor-specific utilities, or additional platform integration are required. These tools offer a simple and portable way to configure and monitor quality of service (QoS) policies across workloads. They enable fine-grained control over resource usage, allowing system administrators and developers to isolate workloads, enforce service-level guarantees, and reduce performance jitters.

[Public]

Chapter 2 PQOS Feature Overview

To meet the growing complexity of modern workloads, systems must balance performance guarantees with equitable resource distribution and responsiveness to real-time demands. AMD's PQOS extension equips platforms with tools to observe, manage, and regulate shared resources such as cache and memory bandwidth, ensuring that critical applications maintain consistent behavior under varying system loads.

PQOS is designed to operate in domains where resource contention is most critical, such as L3 cache, I/O subsystems, and memory bandwidth. A PQOS domain is a group of system components that share resources, such as a set of processor cores, memory controllers, or cache slices.

Goals of AMD PQOS include:

- Monitor usage of shared resources per thread and process
- Enforce resource limits to ensure fairness and performance isolation
- Allocate resources dynamically based on workload priority
- Support real-time and latency-sensitive applications
- Overall: improve performance and throughput

For details about PQOS, see the "Platform Quality of Service (PQOS) Extension" chapter of *AMD64 Architecture Programmer's Manual, Volume 2*, publication #24593.

2.1 Feature Categories

AMD PQOS includes both monitoring and enforcement technologies.

2.1.1 Monitoring Technologies

- Cache Monitoring Technology (CMT)
- Memory Bandwidth Monitoring (MBM)
- Bandwidth Monitor Event Configuration (BMEC)

2.1.2 Enforcement Technologies

- Cache Allocation Technology (CAT)
- Memory Bandwidth Allocation (MBA)
- Code and Data Prioritization (CDP)
- Slow Memory Bandwidth Allocation (SMBA)
- Smart Data Cache Injection Allocation Enforcement (SDCIAE)

[Public]

These features are enabled via architectural tags and hardware registers and are managed by the OS or hypervisor using interfaces like model-specific registers (MSRs), CPUID functions, and the resctrl filesystem.

NOTE: For details about SDCI, see *Smart Data Cache Injection (SDCI) White Paper*, publication #58725. For details about SDCIAE, see the "Platform Quality of Service (PQOS) Extension" chapter of *AMD64 Architecture Programmer's Manual, Volume 2*, publication #24593.

2.2 Key Concepts

Three foundational elements are essential for understanding PQOS features.

2.2.1 Resource Monitoring ID (RMID)

RMIDs are numeric tags assigned to logical threads or processes to track their usage of shared resources. PQOS allows the user to measure metrics like L3 cache occupancy and memory bandwidth with per-RMID granularity, enabling flexible grouping of hardware threads. RMIDs are configured via the PQR_ASSOC MSR.

- RMID monitoring is non-intrusive and has no hardware overhead
- RMIDs can be dynamically reassigned
- Monitoring granularity is per-thread or per-process

2.2.2 Class of Service (COS) ID

COS IDs enforce resource allocation policies. Threads assigned to a COS share the resource limits defined for that class. COS IDs are configured via MSRs such as L3_MASK_n (for cache) and L3QOS_BW_CONTROL_n (for bandwidth).

- Up to 16 COS IDs per system
- COS IDs are used for partitioning and throttling
- Enforcement is hardware-driven and may be based on feedback loops

2.2.3 resctrl Filesystem

resctrl is a Linux kernel interface that exposes PQOS features to user space. It allows administrators to:

- Create resource groups
- Assign tasks or CPUs to groups
- Configure cache and bandwidth allocation
- Monitor resource usage

[Public]

Chapter 3 Use Cases

AMD PQOS technologies are engineered to solve real-world performance and predictability challenges across diverse computing environments from hyperscale cloud data centers to latency-critical industrial systems. This chapter summarizes key use cases that illustrate how PQOS features can improve fairness, isolation, and performance.

3.1 Cloud Data Center Optimization

Challenge:
In multi-tenant cloud environments, virtual machines (VMs) often compete for shared resources. This competition leads to performance degradation for critical workloads due to "noisy neighbors," prompting customers to search for better performance by frequently switching nodes—a behavior known as node jumping.

Solution:
PQOS enables administrators to enforce per-VM resource limits using COS IDs. By bounding cache and memory bandwidth usage, PQOS increases predictable performance and reduces node jumping.

Features Used:

- MBA
- CAT

3.2 Memory Oversubscription Jitter Reduction

Challenge:
When VMs oversubscribe memory, data often spills into slower memory tiers like Compute Express Link (CXL®) or NVDIMM, causing access latency spikes and performance jitter that violate SLAs.

Solution:
PQOS uses SMBA to throttle traffic to slower memory tiers and prioritize access to fast memory (e.g., DDR). This management reduces jitter and ensures consistent performance for DDR-intensive workloads.

Features Used:

- SMBA
- MBM

[Public]

3.3 Hybrid Cloud Networking Protection

Challenge:
Networking VMs such as virtual switches and firewalls run performance-critical code loops that must stay resident in cache to maintain low-latency packet processing. In shared environments, cache evictions caused by other workloads can lead to frequent reloads from slower memory, resulting in latency spikes and packet loss.

Solution:
Use CDP to ensure that critical networking code and data remain in cache. This approach minimizes evictions and maintains fast, consistent packet handling.

Features Used:

- CDP

3.4 Industrial Control System Optimization

Challenge:
Real-time control loops in industrial systems require deterministic execution and minimal jitter. Resource contention—especially cache and memory bandwidth saturation—can disrupt control logic and compromise system safety.

Solution:
Apply CAT and MBA to enforce strict resource boundaries. This practice isolates control workloads from interference, ensuring predictable timing and safer operation.

Features Used:

- CAT
- MBA

3.5 Data Center Networking Infrastructure Optimization

Challenge:
Virtual switches and network endpoints require consistent low-latency performance under heavy traffic loads. Shared-resource contention can cause unpredictable delays.

Solution:
Combine latency-aware thread prioritization with cache and bandwidth enforcement to ensure deterministic performance. Use CAT to isolate cache usage and MBA to cap bandwidth for non-critical workloads.

[Public]

Features Used:

- CAT
- MBA

3.6 Gaming Platform Optimization

Challenge:
Gaming workloads rely on “hero” threads for real-time responsiveness. Background “worker” threads can consume shared resources, causing frame drops and input lag.

Solution:
Assign hero threads to COS IDs with full cache and bandwidth access. Throttle worker threads using MBA and CAT to ensure smooth gameplay and responsiveness.

Features Used:

- CAT
- MBA

[Public]

Chapter 4 Software Enablement Overview

AMD PQOS features are exposed to system software through a combination of architectural registers, CPUID instructions, and kernel interfaces. This section outlines how to detect, configure, and use PQOS capabilities in Linux environments.

4.1 CPUID-Based Feature Detection and Enumeration

To determine PQOS support and capabilities, use the following CPUID functions.

Table 3. CPUID-Based Feature Detection and Enumeration

CPUID Function	Register	PQOS Feature Detection and Enumeration
0x0000_0007	EBX (ECX=0)	Bit 12 indicates PQOS Monitoring (PQM) support
		Bit 15 indicates PQOS Enforcement (PQE) support
0x0000_000F	EBX (ECX=0)	Enumerates maximum RMIDs supported for monitoring
	EDX (ECX=0)	Bit 1 indicates L3 cache monitoring support
	EAX (ECX=1)	Bits 7-0 enumerate counter width for L3 cache monitoring
0x0000_0010	EDX (ECX=0)	Bit 1 indicates CAT support
	EAX (ECX=1)	Bits 4-0 enumerate cache bitmask length for CAT and CDP
	ECX (ECX=1)	Bit 2 indicates CDP support
	EDX (ECX=1)	Bits 15-0 enumerate maximum COS IDs for CAT and CDP
0x8000_0020	EBX (ECX=0)	Indicates support for extended PQOS features: • Bit 6 indicates SDCIAE support • Bit 5 indicates ABMC support • Bit 3 indicates BMEC support • Bit 2 indicates SMBA support • Bit 1 indicates MBA support
	EAX (ECX=1)	Enumerates bandwidth field width for COS-based enforcement and MBA

[Public]

CPUID Function	Register	PQOS Feature Detection and Enumeration
	EDX (ECX=1)	Enumerates maximum COS IDs for COS-based enforcement and MBA
	EAX (ECX=2)	Enumerates bandwidth field width for SMBA
	EDX (ECX=2)	Enumerates maximum COS IDs for SMBA

These values guide how many RMIDs and COS IDs can be used and how granular the monitoring and enforcement can be. For a complete and up-to-date list of CPUID functions that pertain to PQOS, refer to the "Platform Quality of Service (PQOS) Extension" chapter of *AMD64 Architecture Programmer's Manual, Volume 2*, publication #24593.

4.2 Kernel Configuration

1. Ensure the Linux kernel is built with PQOS support:

```
CONFIG_X86_CPU_RESCTRL=y
CONFIG_PROC_CPU_RESCTRL=y
```

2. Verify:

```
cat /boot/config-$(uname -r) | grep -i resctrl
```

4.3 resctrl Filesystem Setup

Example setup to mount resctrl, create a cos0, and add cores 0-3 to cpus_list:

1. Mount the resctrl filesystem:

```
mount -t resctrl resctrl /sys/fs/resctrl
```

2. Create a directory cos0:

```
mkdir /sys/fs/resctrl/cos0
```

3. Write 0-3 to cpus_list:

```
echo 0-3 > /sys/fs/resctrl/cos0/cpus_list
```

[Public]

4. Consider a process with this PID: 12345. To apply the resource constraints defined in the schemata, add the PID to tasks of cos0:

```
echo 12345 | sudo tee /sys/fs/resctrl/cos0/tasks
```

4.4 MSR Interfaces

Key MSRs used for PQOS include:

Table 4. MSR Interfaces

MSR Name (Address)	PQOS Purpose
PQR_ASSOC (0xC8F)	Assign RMID or COS to logical thread
QM_EVTSEL (0xC8D)	Select RMID and event to monitor
QM_CTR (0xC8E)	Read counter value for RMID/event
L3_MASK_n (0xC90 + n)	Configure cache allocation per COS
L3QOS_BW_CONTROL_n (0xC000_0200 + n) L3QOS_SMBW_CONTROL_n (0xC000_0280 + n)	Set bandwidth ceiling per COS

The following figures illustrate PQR_ASSOC, QM_EVTSEL, and QM_CTR.

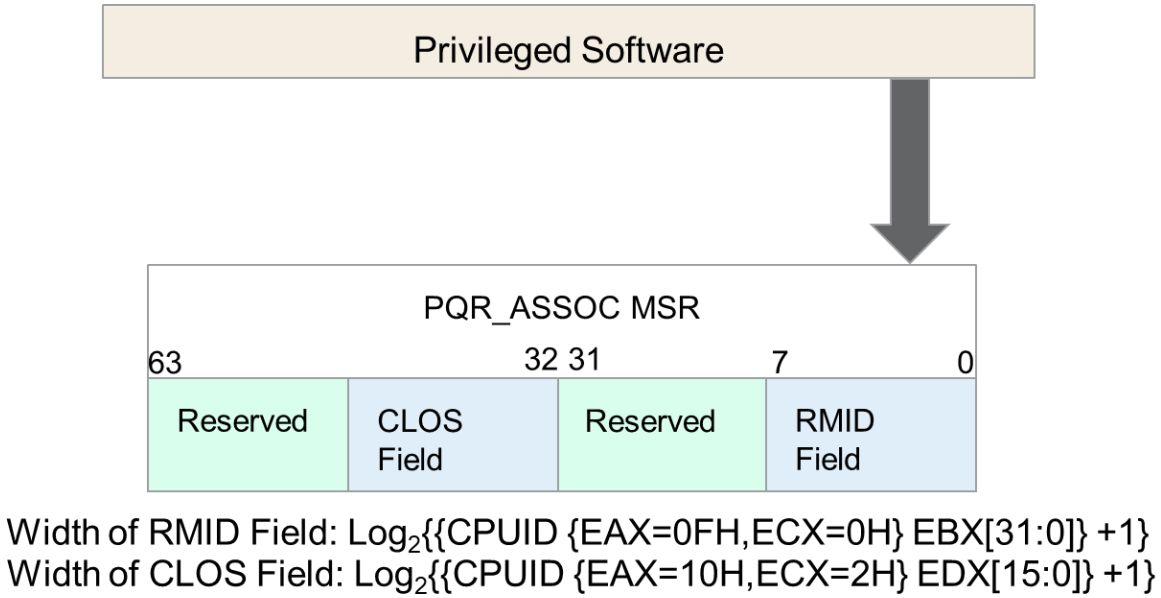


Figure 1. PQR_ASSOC MSR

[Public]

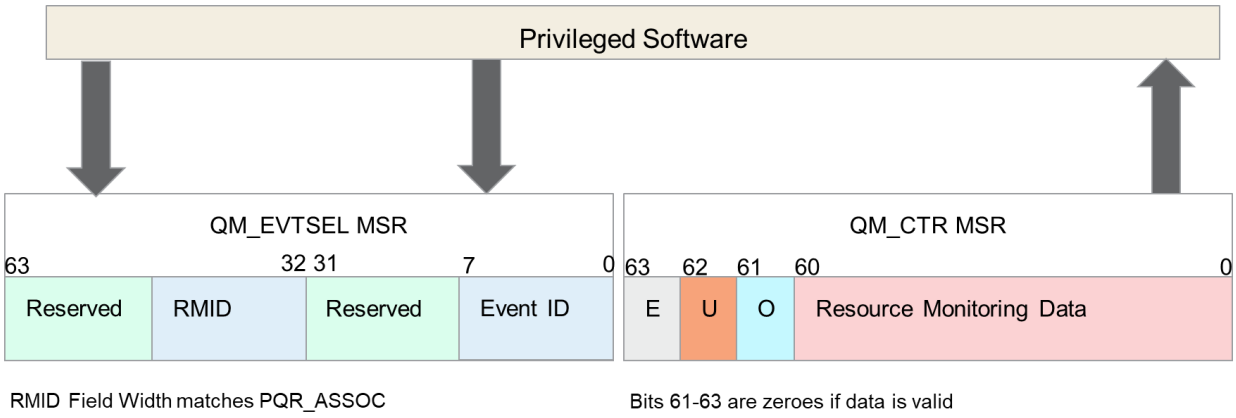


Figure 2. QM_EVTSEL and QM_CTR MSRs

4.5 User-Space Tools

The pqos tool simplifies configuration and monitoring of PQOS features. It supports both MSR and kernel interfaces. However, using both interfaces together may lead to unexpected behavior, so AMD recommends using either the MSR interface or the kernel interface consistently.

- To inspect available PQOS features:

```
pqos -d
```

Sample output:

```
OS capabilities (Linux kernel 5.15.0-73-generic)
Monitoring
    Cache Monitoring Technology (CMT): LLC Occupancy
    Memory Bandwidth Monitoring (MBM): LMEM, TMEM, RMEM
    PMU events: LLC misses, IPC
Allocation
    Cache Allocation Technology (CAT): L3 CAT, CDP disabled, Num COS: 16
    Memory Bandwidth Allocation (MBA): Num COS: 16
```

- To reset configuration:

```
pqos -r
```

For deeper hardware-level feature detection, use CPUID functions 0x0000_000F, 0x0000_0010, and 0x8000_0020 to query support for specific PQOS sub-features.

[Public]

4.6 Feature Detection via dmesg

- Use the following command to search for CAT or MBA support in the CPU:

```
lscpu | grep -o 'cat\|mba'
```

Then, check the kernel log to see if CAT or MBA has been initialized:

```
dmesg | grep -i 'cat\|mba'
```

If they appear in the output, it means the features are likely enabled and supported by your system.

- Check if resctrl is active:

```
dmesg | grep -i resctrl
```

If you see output related to resctrl, it indicates that the resctrl filesystem has been initialized and is available for use.

Example output:

```
[ 13.034171] resctrl: L3 allocation detected
[ 13.044547] resctrl: L3DATA allocation detected
[ 13.050066] resctrl: L3CODE allocation detected
[ 13.055117] resctrl: MB allocation detected
```

[Public]

Chapter 5 Monitoring Features

5.1 RMID-Based Monitoring

RMID-based monitoring is a hardware feature that enables fine-grained tracking of shared resources—such as L3 cache occupancy and memory bandwidth—by associating each logical thread or process with a unique RMID. The hardware maintains counters for each active RMID, allowing software to observe resource consumption per thread, process, or VM. This capability is foundational for QoS enforcement and performance analysis on AMD platforms.

The PQR_ASSOC MSR assigns RMIDs on a per-hardware-thread basis. After an RMID is assigned, the hardware automatically tracks usage, and MSRs store usage counters. Monitoring granularity is configurable, allowing administrators or OS managers to group workloads—such as applications, containers, or VMs—and map them to specific RMIDs for targeted analysis.

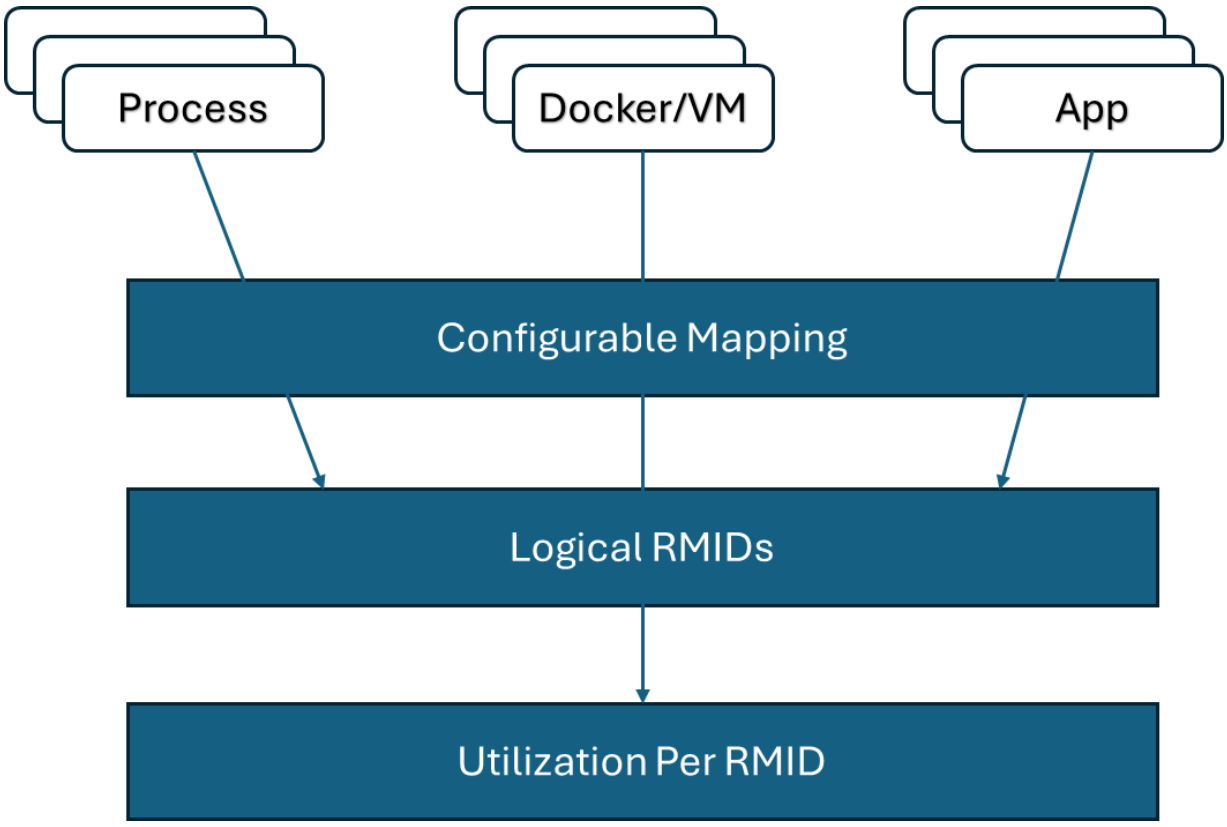


Figure 3. RMID-Based Monitoring

[Public]

5.1.1 RMID Pinning

Assignable Bandwidth Monitoring Counters (ABMC) enable users to explicitly assign hardware counters to RMID-event pairs, a practice also called QoS RMID pinning. This practice guarantees that hardware continuously tracks the assigned RMID until it is explicitly unassigned.

CPUID 0x8000_0020, EBX bit 5 (ECX=0) indicates ABMC support.

The ABMC implementation introduces new interfaces under the resctrl filesystem:

- mbm_assign_mode: Lists assignable monitoring features supported.
- num_mbm_cntrs: Identifies the total number of assignable counters per domain.
- available_mbm_cntrs: Identifies the number of counters currently available for assignment.
- event_configs: Contains configuration directories for each MBM event, including filters for transaction types.
- mbm_L3_assignments: Provides a per-group interface to list or modify RMID-event assignments.

Assignments are domain-specific and can be:

- Exclusive (e): Counter is dedicated to a specific RMID-event pair.
- Shared (s): Counter is shared across multiple groups, with scheduling handled by the system.
- Unassigned (_): No counter is currently assigned.

Example of use:

- To check if ABMC is supported:

```
# cat /sys/fs/resctrl/info/L3_MON/mbm_assign_mode
[mbm event]
```

- To view available counters:

```
# cat /sys/fs/resctrl/info/L3_MON/available_mbm_cntrs
0=30;1=30
```

- To assign a counter associated with the mbm_total_bytes event on all domains in exclusive mode:

[Public]


```
# echo "mbm_total_bytes:*=e" > /sys/fs/resctrl/mbm_L3_assignments
# cat /sys/fs/resctrl/mbm_L3_assignments
mbm_total_bytes:0=e;1=e
mbm_local_bytes:0=e;1=e
```

- To unassign a counter:

```
# echo "mbm_total_bytes:0=" > /sys/fs/resctrl/mbm_L3_assignments
```

5.1.2 Prerequisites

- **Hardware Support**: Platform must support PQM, indicated by CPUID 0x0000_0007, EBX bit 12 (ECX=0).
- **RMID Enumeration**: Maximum RMIDs supported (typically up to 4095) is given by CPUID 0x0000_000F, EBX (ECX=0).
- **Software Support**: OS or monitoring tools must be able to program PQR_ASSOC and read counters.
- **QoS Domain Awareness**: Understanding core complex (CCX)/L3 topology and thread mapping is required.

5.1.3 Interfaces

5.1.3.1 CPUID Enumeration

- 0x0000_000F, EBX (ECX=0): Max_RMID indicates the largest RMID supported.
- 0x0000_000F, EDX bit 1 (ECX=0): Indicates support for L3 cache monitoring.

5.1.3.2 MSRs and Registers

- PQR_ASSOC (C8Fh): Assign RMID to logical thread.
- QM_EVTSEL (C8Dh): Select RMID and event type.
- QM_CTR (C8Eh): Read counter value.
- ChL3QosEvtCfg0/1: Configure bandwidth sources per event.

5.1.4 Software Workflow

1. **Enumerate RMID Support:** Use CPUID functions to determine RMID capacity and supported events.
2. **Assign RMIDs:** Write desired RMID to PQR_ASSOC MSR for each thread or process.
3. **Configure Monitoring Events:** Use QM_EVTSEL to select RMID and event type (e.g., L3 cache occupancy or memory bandwidth) to monitor.
4. **Read Counters:** Read QM_CTR to obtain the current value for the selected RMID/event. Convert the counter value to bytes using the conversion factor from CPUID.

[Public]

5. **Interpret Results:** Analyze per-RMID resource usage using the counters. Ignore counters if U bit of QM_CTR is set.
6. **Reassign Dynamically:** RMIDs can be reassigned at runtime by updating PQR_ASSOC.

5.1.5 Feature Operation

- **Initialization**: Hardware enumerates RMIDs and supported events.
- **Assignment**: OS or tool assigns RMIDs to threads via PQR_ASSOC.
- **Monitoring**: Hardware tracks all RMIDs currently assigned to active threads; counters are queried via QM_CTR.
- **Event Selection**: Events like L3 cache occupancy or memory bandwidth are selected via QM_EVTSEL.
- **Counter Reading**: QM_CTR provides a 44-bit value, converted to bytes using CPUID-provided scaling factors.
- **Overflow Handling**: Counters are large enough to avoid overflow for ~1 hour at peak bandwidth; periodic reads are recommended.
- **Inactive RMIDs**: Hardware retains counters for active RMIDs and a limited number for recently used inactive RMIDs (Least Recently Used [LRU] policy).
- **Error Handling**: If an invalid RMID or event is selected, the E bit in QM_CTR is set and the counter value should be ignored. If the U bit is set, the counter value is not valid (e.g., RMID not tracked).

5.1.6 Product-Specific RMID Availability

Table 5. Product-Specific RMID Availability

Product	Largest RMID Supported for Any Resource: CPUID 0x0000_000F, EBX (ECX=0)
AMD EPYC 9004 series processors, codenamed “Genoa”	255
AMD EPYC 9005 series processors, codenamed “Turin”	4095

5.1.7 Using pqos for Monitoring

RMIDs are automatically assigned when monitoring starts.

Use the `-m` or `-p` option to monitor cores or processes, respectively.

- Core monitoring:

```
pqos -m all:0,2,4-10
```

- PID monitoring:

[Public]


```
pqos -I -p llc:22,25673
```

Monitoring options:

```
llc: Last Level Cache occupancy
mbl: Local memory bandwidth
mbr: Remote memory bandwidth
mbt: Total memory bandwidth
all: All available events
```

5.2 L3 Cache and Memory Bandwidth Monitoring

L3 Cache Monitoring and Bandwidth Monitoring are hardware-based QoS features on AMD platforms that enable real-time, fine-grained tracking of shared resource usage. These features operate by associating each logical thread or process with an RMID. The hardware maintains counters per RMID to track:

- **L3 Cache Occupancy:** Tracks how much of the L3 cache is occupied by a specific thread, process, or VM. This tracking helps identify cache-heavy workloads and analyze cache contention. High occupancy may indicate inefficient cache usage or the need for cache partitioning.
- **Local and Remote Memory Bandwidth:** Tracks how much memory bandwidth is used locally (same Non-Uniform Memory Access [NUMA] node) vs. remotely (other NUMA nodes). This tracking helps identify inefficient memory access patterns.
- **Non-Temporal Writes:** Monitors memory writes that bypass the cache, typically used in streaming workloads. This is tracked separately for local and remote memory.
- **Slow Memory Access:** Tracks reads to slower memory tiers such as NVDIMM-P, CXL-attached memory, or peer memory extensions. These accesses are typically higher latency and lower bandwidth than DDR. Monitoring slow memory usage helps ensure latency-sensitive workloads are not routed to slow memory unintentionally.
- **"Dirty Victim" Traffic:** Measures bandwidth consumed by evicted dirty cache lines. High dirty victim traffic may indicate frequent cache evictions and write-back pressure, which can degrade performance and increase memory bandwidth usage.

This monitoring capability forms the foundation for performance analysis, workload isolation, and dynamic resource management in multi-tenant and latency-sensitive environments.

5.2.1 Prerequisites

- **Hardware Support**: Platform must support PQM, indicated by CPUID 0x0000_0007, EBX bit 12 (ECX=0).
- **RMID Enumeration**: Maximum RMIDs supported (typically 4095) is given by CPUID 0x0000_000F, EBX (ECX=0).

[Public]

- **Software Support**: OS or monitoring software must program PQR_ASSOC MSR and read counters.
- **QoS Domain Awareness**: Understanding CCX/L3 topology and thread mapping is essential.

5.2.2 Interfaces

5.2.2.1 CPUID Enumeration

- 0x0000_000F, EBX (ECX=0): Max_RMID indicates the largest RMID supported.
- 0x0000_000F, EDX bit 1 (ECX=0): Support for L3 cache monitoring.
- 0x0000_000F, EAX bits 7-0 (ECX=1): CounterSize (e.g., 44-bit counters).

5.2.2.2 MSRs and Registers

- PQR_ASSOC (C8Fh): Assign RMID to logical thread.
- QM_EVTSEL (C8Dh): Select RMID and event type.
- QM_CTR (C8Eh): Read counter value.
- ChL3QosEvtCfg0/1: Configure bandwidth sources per event.

5.2.3 Software Setup and Workflow

1. **Enumerate RMID Support**: Use CPUID to determine RMID capacity and supported events.
2. **Assign RMIDs**: Write RMID to PQR_ASSOC MSR for each thread or process.
3. **Configure Monitoring Events**: Use QM_EVTSEL to select RMID and event type.
4. **Read Counters**: Use QM_CTR to get usage data; convert using CPUID scaling.
5. **Interpret Results**: Analyze per-RMID resource usage. Check U and E bits for counter validity.
6. **Reassign Dynamically**: RMIDs can be reassigned at runtime by updating PQR_ASSOC.

5.2.4 Using resctrl for Monitoring

5.2.4.1 Last Level Cache (LLC) Occupancy

To monitor LLC (L3 cache) occupancy, use the following file:

```
/sys/fs/resctrl/<COS>/mon_data/mon_L3_<CCX_id>/llc_occupancy
```

5.2.4.2 Local DRAM Bandwidth

To measure bandwidth to local DRAM from the L3 domain:

[Public]


```
# script_mbm_local.sh
cd /sys/fs/resctrl/test
m0=$(cat mon_data/mon_L3_00/mbm_local_bytes)
sleep 1
m1=$(cat mon_data/mon_L3_00/mbm_local_bytes)
delta=$(( m1 - m0 ))
mb=$(echo "scale=3; ${delta}/1024/1024" | bc -l)
echo "Local memory bandwidth: ${mb} MB/s"
```

To measure total DRAM bandwidth: read mbm_total_bytes.

5.2.5 Using pqos for Monitoring

To measure bandwidth to DRAM from the L3 domain:

```
# Local bandwidth monitoring
pqos --mon-core=mbl:0-5
# Remote bandwidth monitoring
 pqos --mon-core=mbr:0-5
# Total bandwidth monitoring
  pqos --mon-core=mbt:0-5
```

5.3 Bandwidth Monitoring Event Configuration (BMEC)

BMEC is an advanced feature that allows users to customize which types of memory traffic are counted by the memory bandwidth monitoring infrastructure. This feature enables more targeted monitoring of memory behavior across workloads.

CPUID 0x8000_0020, EBX bit 3 (ECX=0) indicates BMEC support.

When the platform supports BMEC, the following monitoring features are available:

```
# cat /sys/fs/resctrl/info/L3_MON/

mbm_total_bytes_config
mbm_local_bytes_config
```

The presence of mbm_total_bytes_config and mbm_local_bytes_config indicates that BMEC is supported and configurable.

BMEC allows users to selectively monitor memory traffic, contributing to:

- mbm_total_bytes: Total memory bandwidth usage
- mbm_local_bytes: Local memory bandwidth usage

[Public]

Each configuration is domain specific and affects all CPUs within that domain. When a configuration is changed, the bandwidth counters for all RMIDs in that domain are cleared. The next read returns “Unavailable” until new data is collected.

5.3.1 Supported Event Types

Each configuration is a bitmask where each bit enables a specific type of memory traffic:

Table 6. Supported Event Types

Bit	Description
6	Dirty victims from the QoS domain to all memory types
5	Reads to slow memory in the non-local NUMA domain
4	Reads to slow memory in the local NUMA domain
3	Non-temporal writes to the non-local NUMA domain
2	Non-temporal writes to the local NUMA domain
1	Reads to memory in the non-local NUMA domain
0	Reads to memory in the local NUMA domain

Default values:

- mbm_total_bytes_config: 0x7F (all event types enabled).
- mbm_local_bytes_config: 0x15 (local memory reads and writes).

5.3.2 Examples of Use

5.3.2.1 Count Only Reads on Domain 0

To count only read traffic (bits 0, 1, 4, 5), set the bitmask to 0x33:

```
# echo "0=0x33" > /sys/fs/resctrl/info/L3_MON/mbm_total_bytes_config
```

5.3.2.2 Count Only Slow Memory Reads on Domains 0 and 1

To count only slow memory reads (bits 4 and 5), set the bitmask to 0x30:

```
# echo "0=0x30;1=0x30" >
/sys/fs/resctrl/info/L3_MON/mbm_local_bytes_config
```

[Public]

Chapter 6 Enforcement Features

6.1 COS-Based Enforcement

COS-based enforcement is a hardware-enabled QoS mechanism that allows system software to partition and control shared resources such as L3 cache and memory bandwidth by assigning threads or processes to distinct COS IDs. Each COS can be independently configured with resource allocation policies, enabling fine-grained control over how resources are distributed across workloads.

Each logical core uses the PQR_ASSOC MSR to associate a thread with a COS. Enforcement is applied via cache masks and bandwidth ceilings, allowing the system to throttle or isolate workloads based on priority or performance requirements.

The impact of dynamically updating MSR values is not instantaneous. After an MSR update, a delay occurs before the change takes effect, and the update itself may take additional time depending on current CPU load and memory access pressure.

6.1.1 Interfaces

6.1.1.1 CPUID Enumeration

- 0x0000_0007, EBX bit 15 (ECX=0): Indicates PQE support.
- 0x8000_0020, EDX (ECX=1): COS_MAX indicates maximum number of supported COS (typically 16).
- 0x8000_0020, EAX (ECX=1): BW_LEN indicates width of bandwidth ceiling field.

6.1.1.2 MSRs and Registers

- PQR_ASSOC (C8Fh): Assigns COS to logical thread.
- L3QOS_BW_CONTROL_n (C000_0200h + n): Bandwidth ceiling per COS.
- L3_MASK_n (C90h + n): Cache allocation mask per COS.

6.1.2 Software Setup and Workflow

1. **Enumerate COS Support:** Use CPUID to determine available COS IDs and enforcement capabilities.
2. **Assign Threads to COS:** Write the desired COS ID to PQR_ASSOC MSR for each thread or process.
3. **Configure Enforcement Policies:**
    - Set bandwidth ceilings via L3QOS_BW_CONTROL_n.

[Public]

- Configure cache allocation via L3_MASK_n (optional).

4. **Monitor and Adjust:** Use QoS monitoring features to observe per-COS resource usage and adjust policies as needed.
5. **Reassign Dynamically:** Threads can be reassigned to different COS IDs at runtime to adapt to workload changes.

6.1.3 Feature Operation

- **Resource Partitioning**: Each COS is configured with independent limits for cache and bandwidth. Threads assigned to a COS share these resources competitively.
- **Enforcement**: Hardware enforces limits via
  - Throttling (for bandwidth).
  - Allocation masks (for cache).
- **Feedback Loop**: At the end of each feedback interval (e.g., 128 µs), the hardware evaluates usage and adjusts throttle levels accordingly.
- **Unlimited Mode**: Setting the U bit in the ceiling register disables enforcement for that COS.

This mechanism ensures that bandwidth-intensive workloads do not interfere with latency-sensitive or critical tasks, and it supports dynamic adjustment based on workload behavior.

6.2 L3 Cache Allocation Technology (CAT)

L3 CAT is a hardware-based QoS feature that enables fine-grained partitioning of the shared L3 cache across workloads. These mechanisms allow system software to partition, isolate, and reserve cache regions for specific workloads, enabling predictable performance and cache residency. By assigning each logical thread or process to a COS and configuring a Cache Bitmask (CBM) for each COS via MSRs, the system can control which cache “ways” are accessible to each group.

6.2.1 Prerequisites

- **Hardware Support:** Platform must support
  - PQE, indicated by CPUID 0x0000_0007, EBX bit 15 (ECX=0).
  - L3 Cache Allocation Enforcement, indicated by CPUID 0x0000_0010, EDX bit 1 (ECX=0).

- **BIOS/SMU Enablement**: Both features must be enabled in firmware.
- **Operating System/Driver**: Must support MSR programming and thread-to-COS assignment.
- **QoS Domain Awareness**: Understanding CCX/L3 topology and thread mapping is required.

[Public]

6.2.2 Interfaces

6.2.2.1 CPUID Enumeration

- 0x0000_0010, EAX bits 4-0 (ECX=1): CBM_LEN indicates length of cache bitmask (typically 15 for 16 ways).
- 0x0000_0010, EDX bits 15-0 (ECX=1): COS_MAX indicates maximum number of supported COS IDs.

6.2.2.2 MSRs and Registers

- PQR_ASSOC (C8Fh): Assigns COS to logical threads.
- L3_MASK_n (C90h + n): Per-COS cache allocation mask.
- ChL3QosAllocMask{15-0}: Hardware register backing the MSR.

6.2.3 Software Setup and Workflow

1. **Enumerate Support**: Use CPUID to determine COS count and cache way availability.
2. **Assign Threads to COS**: Write COS ID to PQR_ASSOC MSR for each thread.
3. **Configure Cache Allocation Masks**: Set L3_MASK_n with desired bitmask. Example: 0x00FF for lower 8 ways.
4. **Monitor and Adjust**: Use QoS tools to observe cache usage and adjust masks or reserved ranges.
5. **Reassign Dynamically**: Threads and cache masks can be updated at runtime. Reserved regions can be reconfigured as workloads change.

6.2.4 Feature Operation

- **Cache Partitioning**: Each COS is assigned a CBM that defines which L3 cache ways it can use. Threads in that COS can only fill or evict lines in those ways.
- **Competitive Sharing**: Overlapping masks result in shared cache ways among COS groups.
- **Isolation**: Exclusive cache ways ensure strong isolation between workloads.
- **Unlimited Mode**: Setting all bits in the mask allows full cache access.
- **Dynamic Adjustment**: RMID and COS assignments can be modified at runtime.

6.2.5 Using resctrl for L3 CAT

- Mount resctrl and create a COS:

```
umount /sys/fs/resctrl
mount -t resctrl resctrl /sys/fs/resctrl
cd /sys/fs/resctrl
mkdir cos0
```

[Public]


```
cd cos0
```

- View the current schemata:

```
cat schemata
```

- Assign cores 0–3 to the COS:

```
echo 0-3 > cpus_list
```

- Assign a specific PID to the COS:

```
echo <pid> tasks
```

6.2.6 Performance Profiling and Trade-Offs

6.2.6.1 Scenario 1: Default L3 Allocation (Full Cache Access)

Run the Memory Latency Checker (MLC) benchmark on core 2 with default L3 allocation:

```
# ./mlc --peak_injection_bandwidth -m2
```

With the cache bitmask set to 0xFFFF (default), the thread has access to all 16 L3 cache ways.

MLC Results (Application Perspective):

- ALL Reads: 64,189.2 MB/s
- 3:1 Reads-Writes: 79,931.5 MB/s
- 2:1 Reads-Writes: 85,160.9 MB/s
- 1:1 Reads-Writes: 108,459.7 MB/s
- Stream-triad: 94,766.9 MB/s

Unified Memory Controller (UMC) Bandwidth (Memory Controller Perspective):

- Estimated Read Bandwidth: 39.86 GB/s
- Estimated Write Bandwidth: 17.71 GB/s

6.2.6.2 Scenario 2: Restricted L3 Allocation (1 Cache Way)

Restrict the L3 COS bitmask from 0xFFFF to 0x0001:

[Public]


```
/sys/fs/resctrl/test# echo "L3:0=0001" > schemata
```

This setting limits the thread to only 1 cache way.

MLC Results (Application Perspective):

- ALL Reads: 52,777.4 MB/s
- 3:1 Reads-Writes: 70,076.6 MB/s
- 2:1 Reads-Writes: 77,455.6 MB/s
- 1:1 Reads-Writes: 97,809.0 MB/s
- Stream-triad: 76,137.8 MB/s

UMC Bandwidth (Memory Controller Perspective):

- Estimated Read Bandwidth: 41.80 GB/s
- Estimated Write Bandwidth: 17.86 GB/s

6.2.6.3 Conclusions

The results indicate that restricting cache access via CAT reduces application-observed memory bandwidth, especially for read-intensive and mixed workloads.

- With cache access reduced from 16 ways to 1 way, MLC bandwidth—including stream-triad performance, which simulates real-world memory access patterns—can decrease a relative amount of up to 20%.
- UMC bandwidth can increase slightly when the cache is restricted. This result is expected: with fewer cache ways available, more memory accesses bypass the cache and go directly to DDR, increasing observed bandwidth at the memory controller.

This behavior confirms that cache availability directly influences memory throughput from both the core and memory perspectives. CAT can thus effectively throttle or isolate workloads by controlling their cache residency and memory access behavior.

6.3 Memory Bandwidth Allocation (MBA) and Slow MBA (SMBA)

MBA is a platform-level QoS feature that enables the enforcement of upper bounds on memory bandwidth consumed by threads or processes within a QoS domain (typically a CCX or L3 domain). By assigning bandwidth ceilings to specific COSs, MBA ensures:

[Public]

- Fair sharing of memory resources and predictable performance for critical workloads.
- Prevention of bandwidth starvation.

Administrators can allocate any COS ID from 0 to 15 and apply it to a resource group to enforce bandwidth limits.

SMBA is an AMD PQOS feature designed to control and allocate memory bandwidth for slower memory tiers (such as CXL). It works similarly to MBA but specifically targets slow memory regions to allocate bandwidth resources and prevents jitter caused by oversubscription of slow memory in systems with heterogeneous memory.

6.3.1 Use Cases

6.3.1.1 Thread Admission Control

This scenario ensures minimum guaranteed bandwidth for latency-sensitive workloads:

- A background process continuously consumes memory bandwidth.
- When a new process arrives requiring Y GB/s, the system checks if the remaining threads (e.g., 4 out of 8 in a CCD) can be throttled to (X − Y)/4 GB/s.
    - If so, the new process is scheduled accordingly.
    - If not, the new process is not scheduled.

6.3.1.2 Fairness Across VMs

Bandwidth allocation is proportional to the number of cores used by each VM, which:

- Prevents one VM from monopolizing memory bandwidth and degrading others.
- Enables multi-tenant fairness in virtualized environments.

6.3.2 Prerequisites

- **Hardware Support**: Platform must support MBA, indicated by CPUID function 0x8000_0020, EBX bit 1 (ECX=0). Platform support for SMBA is indicated by CPUID function 0x8000_0020, EBX bit 2 (ECX=0).
- **BIOS/SMU Enablement**: MBA (or SMBA) must be enabled in BIOS or firmware.
- **Operating System/Driver**: The OS must support MSR programming and thread-to-COS assignment. The kernel must be built with the CONFIG_X86_CPU_RESCTRL flag.
- **QoS Domain Awareness**: Understanding the system's CCX/L3 topology and how threads are mapped to COS is required.

[Public]

6.3.3 Interfaces

6.3.3.1 CPUID Enumeration

- 0x8000_0020, EBX bit 1 (ECX=0): Indicates MBA support.
- 0x8000_0020, EAX (ECX=1): BW_LEN field gives the width of the bandwidth ceiling field for MBA.
- 0x8000_0020, EDX (ECX=1): COS_MAX field gives the maximum number of supported COS for MBA.
- 0x8000_0020, EBX bit 2 (ECX=0): Indicates SMBA support.
- 0x8000_0020, EAX (ECX=2): BW_LEN field gives the width of the bandwidth ceiling field for SMBA.
- 0x8000_0020, EDX (ECX=2): COS_MAX field gives the maximum number of supported COS for SMBA.

6.3.3.2 MSRs and Registers

- L3QOS_BW_CONTROL_n (MSR C000_0200h + n) for MBA and L3QOS_SMBW_CONTROL_n (MSR C000_0280h + n) for SMBA: Per-COS bandwidth ceiling register.
  - U: Set to 1 for unlimited bandwidth.
  - BW: Define bandwidth limit in 1/8 GB/s increments.
- PQR_ASSOC (MSR C8Fh): Assigns threads to COS.

6.3.4 Software Interfaces

MBA is exposed to user space via the Linux kernel and the resctrl filesystem. To enable MBA, the kernel must be built with `CONFIG_X86_CPU_RESCTRL`.

6.3.4.1 Key Files in resctrl Used for Bandwidth Monitoring

- `min_bandwidth`: Minimum bandwidth percentage per CPU model.
- `bandwidth_gran`: Allocation granularity; values round to nearest control step.
- `delay_linear:` Linear or non-linear delay scaling.
- `ctrl_hw_id:` Debug-only field showing hardware COS ID.

MBA and SMBA use absolute bandwidth values. Example ceiling values:

- 1 = 128 MB/s.
- 16 = 2 GB/s.
- 2048 = 256 GB/s.

To set MBA or SMBA bandwidth via `schemata` (using *MB* for MBA) with example values:

[Public]


```
echo "MB:1=16" > schemata
echo "SMBA:0=32" > schemata
```

The MBA example sets 2 GB/s for cache ID 1, while the SMBA example sets 4 GB/s for cache ID 0. Continuing with these examples, `cat schemata` might show (maximum value is 2048):

```
MB:0=2048;1=16;2=2048;3=2048
SMBA:0=32;1=2048;2=2048;3=2048
```

6.3.4.2 Resource Group Configuration

Resource groups are directories under `/sys/fs/resctrl`. The root group owns all tasks and CPUs initially.

- CTRL_MON groups: Created under root to define resource allocations.
- MON groups: Created under `mon_groups` to monitor subsets of tasks in a CTRL_MON group.

Removing a CTRL_MON group also removes its MON groups. MON groups can be moved between CTRL_MON parents (except those monitoring CPUs). Renaming is the only other supported move.

6.3.4.3 Group Files and Controls

- `tasks`: Assign PIDs to the group. Failures abort the operation but retain successful assignments.
- `cpus` / `cpus_list`: Assign logical CPUs using bitmasks or ranges. MON groups must inherit CPUs from their parent.
- `schemata`: Defines resource allocations. For memory bandwidth, the format is: `MB:<cache_id>=<bw_MiBps>` (e.g., `MB:1=16` for 2 GB/s).

6.3.4.4 Bandwidth Monitoring Configuration

To control the bandwidth that is counted for throttling, set `mbm_total_bytes_config` in `info/L3_MON`. It has a 7-bit value:

- Bit 0: Reads to local NUMA.
- Bit 1: Reads to non-local NUMA.
- Bits 2–6: Non-temporal writes, reads to slow memory, and dirty victims.

Examples:

- 0x7F: Monitor all traffic.

[Public]

- 0x01: Monitor only local NUMA reads.

6.3.5 Software Setup and Workflow

1. **Enumerate MBA Support**: Use CPUID to confirm MBA (or SMBA) availability and determine COS_MAX and BW_LEN.
2. **Assign Threads to COS**: Use the PQR_ASSOC MSR to assign each thread to a COS.
3. **Program Bandwidth Ceilings**: For each COS, write the desired bandwidth limit to L3QOS_BW_CONTROL_n for MBA or L3QOS_SMBW_CONTROL_n for SMBA. Example: To set a 32 GB/s limit, write BW = 256 (256 × 1/8 GB/s = 32 GB/s).
4. **Monitor and Adjust**: Use performance counters and QoS monitoring features to observe bandwidth usage and adjust ceilings as needed.

6.3.6 Feature Operation

MBA (and similarly SMBA) operates as a closed-loop control system that tracks memory bandwidth usage per COS and dynamically adjusts throttle levels to enforce bandwidth ceilings. It ensures predictable performance by limiting memory access when usage exceeds configured thresholds.

- **Bandwidth Tracking:** Hardware counters monitor memory bandwidth usage per COS, including reads/writes to DRAM, non-temporal writes, and victim traffic.
- **Ceiling Programming:** Each COS is assigned a bandwidth ceiling via L3QOS_BW_CONTROL_n for MBA or L3QOS_SMBW_CONTROL_n MSR for SMBA, specified in 1/8 GB/s units (e.g., 256 = 32 GB/s).
- **Periodic Evaluation:** At the end of each feedback period (e.g., every 128 µs), the system compares actual usage against the ceiling.
- **Throttle Level Adjustment:** If usage exceeds the ceiling, throttle levels are increased—limiting outstanding memory operations, delaying token returns, and controlling write-back context release. If usage is below the ceiling, throttling is relaxed.
- **L2 Enforcement:** Throttle levels are enforced at the L2 cache by restricting memory instruction buffers and delaying memory-related actions.
- **Unlimited Mode:** Setting the U bit in the ceiling register disables enforcement for that COS, allowing unrestricted bandwidth.
- **Dynamic Adjustment:** Ceilings and COS assignments can be modified at runtime to adapt to changing workload demands.
- **Monitoring:** Bandwidth usage and throttle levels can be observed via performance counters and QoS MSRs.

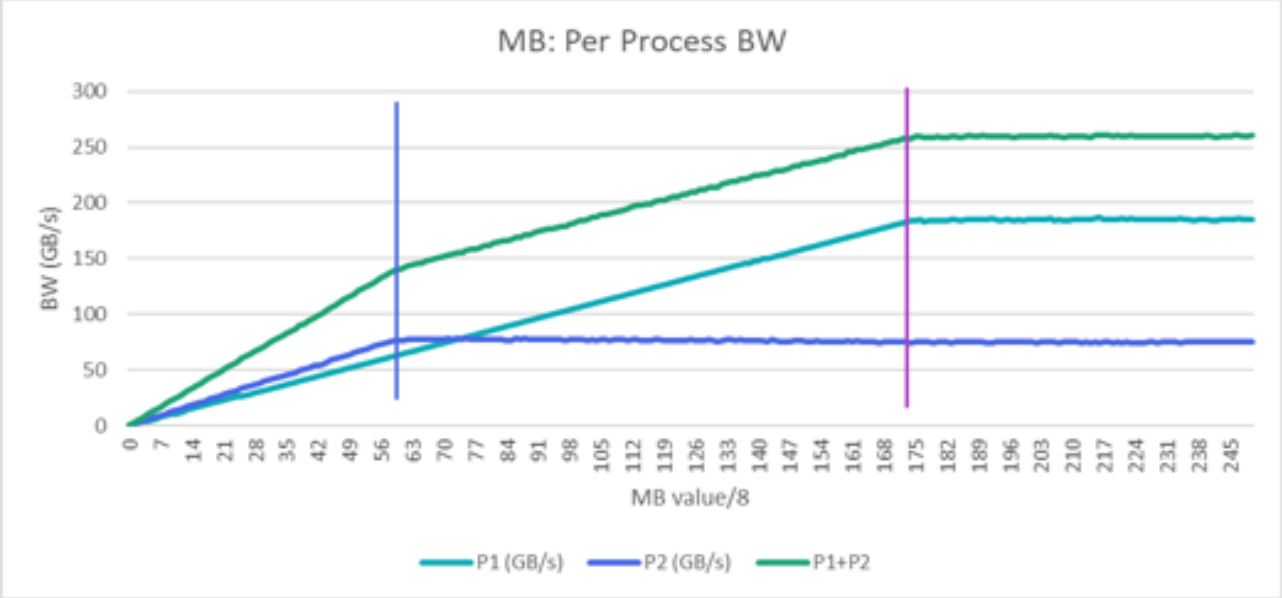
6.3.7 Performance Profiling and Trade-Offs

This test evaluates bandwidth scaling behavior under MBA enforcement.

Test Setup:

- Two processes:
  - P1 runs on 6 cores from CCX0.
  - P2 runs on 2 cores from CCX1.
- Both processes are assigned to the same COS.
- PIDs are added to `/sys/fs/resctrl/COSn/tasks` and cores to `/sys/fs/resctrl/COSn/cpus_list`.
- Bandwidth ceilings vary along the X-axis.

Plotting:



Observations and Inferences:

- **P1 (teal line)**
  Bandwidth increases steadily with MBA value until around x = 168, after which it plateaus. This plateau does not indicate memory saturation, but rather that P1 becomes CCX-bound—limited by the bandwidth capacity of its CCX interface. Further increases in MBA do not improve throughput beyond this architectural ceiling.
- **P2 (blue line)**
  Bandwidth increases slightly until x = 63, then flattens, confirming that P2 is core-bound. Its performance is constrained by compute resources rather than memory bandwidth, and additional bandwidth allocation has no effect past this point.
- **P1 + P2 (green line)**
  Combined bandwidth scales similarly to P1 but remains consistently higher due to P2's contribution. However, the curve does not represent the system's maximum bandwidth

[Public]

capacity. Instead, it reflects the sum of P1's CCX bandwidth limit and P2's core throughput. If additional cores or CCXs were active, the system could achieve higher aggregate bandwidth. This highlights that bandwidth ceilings are shared within a QoS domain (CCX), but not across different CCXs.

- **Vertical lines**
  - **x = 63**: Marks the inflection point where P2 stops responding to increased bandwidth, indicating core-bound behavior.
  - **x = 168**: Marks the saturation point for P1's CCX interface, not the system-wide bandwidth limit.

Conclusions:

- The plot illustrates effective MBA scaling up to architectural limits specific to each QoS domain.
- It distinguishes between core-bound and CCX-bound workloads.
- It emphasizes that bandwidth ceilings are enforced per CCX and that combining workloads across CCXs can increase total bandwidth without contention.
- This behavior guides optimal bandwidth allocation strategies and validates system-level QoS enforcement.

6.4 Code and Data Prioritization (CDP)

CDP is a hardware-enabled QoS feature that allows the system to distinguish between instruction (code) and data cache lines in the shared L3 cache and allocate cache resources accordingly. By enabling CDP, the system can assign separate cache allocation masks (CBMs) for code and data, enabling differentiated cache usage policies.

Without CDP, both code and data share the same cache region, which can lead to cache contention. For example, data-heavy operations may evict frequently used instructions, increasing cache misses and degrading performance. CDP solves this problem by assigning two independent bitmasks per COS—one for code and one for data—to ensure that critical instructions remain resident in the L3 cache even during high data traffic.

6.4.1 Prerequisites

- **Hardware Support**: CDP must be supported, indicated by CPUID 0x0000_0010, ECX bit 2 (ECX=1).
- **BIOS/SMU Enablement**: CDP must be enabled in BIOS or firmware.
- **Operating System/Driver**: OS must support MSR programming and thread-to-COS assignment.

[Public]

- **QoS Domain Awareness**: Understanding CCX/L3 topology and thread mapping is required.

6.4.2 Interfaces

6.4.2.1 CPUID Enumeration

- 0x0000_0010, ECX bit 2 (ECX=1): Indicates CDP support.
- 0x0000_0010, EAX bits 4-0 (ECX=1): CBM_LEN indicates cache bitmask length (typically 15 for 16 ways).
- 0x0000_0010, EDX bits 15-0 (ECX=1): COS_MAX indicates maximum supported COS IDs (typically 16).

6.4.2.2 MSRs and Registers

- PQR_ASSOC (C8Fh): Assigns COS to logical thread.
- L3_MASK_n (C90h + n): Per-COS cache allocation mask; interpreted as code/data pair when CDP is enabled.
- L3_QOS_CFG1 (C81h): Enables CDP mode at L3 level.

6.4.3 Software Setup and Workflow

1. **Enumerate CDP Support**: Use CPUID to confirm CDP availability and determine cache way count.
2. **Enable CDP**: Set CDP bit in L3_QOS_CFG1 register.
3. **Assign Threads to COS**: Use PQR_ASSOC MSR to assign threads. With CDP enabled, COS values are shifted and concatenated with a code/data bit.
4. **Configure Cache Allocation Masks**: Set L3_MASK_n for each COS with separate masks for code and data.
5. **Monitor and Adjust**: Use QoS monitoring tools to observe cache usage and adjust masks as needed.
6. **Reassign Dynamically**: Threads and masks can be updated at runtime for workload-aware cache management.

6.4.4 Feature Operation

- **Code/Data Partitioning**: CDP interprets COS values as code/data pairs, allowing separate cache masks for each stream.
- **Competitive Sharing**: Overlapping masks result in shared cache ways among COS groups.
- **Isolation**: Exclusive cache ways ensure strong isolation between code and data streams.
- **Unlimited Mode**: Setting all bits in the mask allows full cache access.
- **Dynamic Adjustment**: Masks and COS assignments can be modified at runtime.

[Public]

- **Consistency Requirement**: CDP must be consistently enabled across L2 and L3 cache domains.

6.4.5 Using resctrl for CDP

```
umount /sys/fs/resctrl
mount -o cdp -t resctrl resctrl /sys/fs/resctrl

cd /sys/fs/resctrl
mkdir test
cd test
cat schemata
```

You should see `L3CODE` and `L3DATA` entries in the schemata file. To restrict data cache ways:

```
echo 0-3 > cpus_list
echo L3DATA:0=0001 > schemata
```

[Public]

Chapter 7 Conclusion

As computing environments grow increasingly complex and dynamic, AMD's commitment to advancing the PQOS extension remains central to enabling predictable and efficient system behavior. The current generation of PQOS features available on AMD EPYC 9004 and 9005 series processors provides robust mechanisms for resource control—including cache partitioning, memory bandwidth enforcement, and code/data prioritization—that are accessible through standard Linux interfaces.

Looking ahead, AMD continues to evolve its PQOS capabilities to meet the demands of increasingly complex and heterogeneous computing environments. Future generations of EPYC processors will introduce expanded feature sets designed to offer even finer-grained control, enhanced observability, and smarter resource orchestration. These advancements will further empower developers and system architects to optimize performance, enforce workload boundaries, and ensure consistent quality of service across diverse deployment scenarios.

AMD remains committed to enabling open, scalable, and intelligent platform management through PQOS and encourages early adoption and feedback to shape the next wave of innovation.