Increasing Platform Determinism with Platform Quality of Service for the Data Plane Development Kit

White Paper

February 2016

Document Number: 333742-001US

Contents

Figures

Tables

Revision History

Date	Revision	Description
February 2016	1.0	Initial release.

§

1.0 Introduction

1.1 Abstract

Network Functions Virtualization (NFV) enables the consolidation of a wide variety of communication appliances. In order to meet the NFV input/output (I/O) performance requirements, it is likely that these virtual machines will utilize the Data Plane Development Kit (DPDK). With core count growth and increased number of (I/O) devices being integrated in Intel's processors, shared resource contention could lead to performance degradation.

For this reason, Intel has introduced Platform Quality of Service (PQoS) consisting of both Monitoring and Allocation capabilities focusing on shared platform resources enabling system administrators and developers to regain control over the platform. The purpose of this paper is to cover the new capabilities that Intel processors integrate to deal with potential resource contention issues associated with NFV.

- The motivation for Cache Monitoring Technology (CMT), Cache Allocation Technology (CAT) and Memory Bandwidth will be covered in Section 1.0.
- For DPDK to take advantage of the PQoS capabilities certain libraries and Linux* kernel infrastructure has been developed, these are covered in Section 2.0.
- In Section 3.0, the paper reviews the usage of a DPDK based packet processing pipeline in both a contested and non-contested configuration to demonstrate the potential performance impact of shared resource contention.
- Section 4.0 covers the platform requirements to take advantage of PQoS and provides links to the various tools covered in this paper.
- Section 5.0 provides closing statements and the need for PQoS to provide both insight into and the ability to reconfigure the platform to eliminate the impact of shared resource contention.

1.2 Dependencies

All software tools, patches, and components referenced in this document should be the latest stable release, unless a specific version is provided. In rare cases, some modifications of the baseline versions are needed. Scope and description of such modifications are detailed in this document.

1.2.1 Acronyms and Abbreviations

Table 1 lists acronyms and abbreviations used in this document.

Table 1. Acronyms and Abbreviations

Name	Description
CAT	Cache Allocation Technology
CDP	Code Date Prioritization
cgroup	Control Group
CLI	Command Line Interface
CMT	Cache Monitoring Technology
COS	Class of Service
CPU	Central Processing Unit
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
I/O	Input/Output
IP	Internet Protocol
LLC	Last Level Cache
MBM	Memory Bandwidth Monitoring
Mpps	Million packets per second
NAT	Network Address Translation
NFV	Network Functions Virtualization
PID	Process Identifier
PQoS	Platform Quality of Service
QoS	Quality of Service
RMID	Resource Monitoring ID
SDN	Software-Defined networking
SKU	Stock Keeping Unit
TID	Task Identifier
VM	Virtual Machine
VNF	Virtual Network Function

1.2.2 Problem Statement

The next-generation networking and communications equipment industry is beginning to embrace NFV. NFV is an emerging approach which enables consolidation of network functions that typically reside on a fixed function platform using Virtualization and other technologies. NFVs' promise is to utilize off-the-shelf servers with general-purpose processors and maintain performance expectations.

Traditionally Intel® Architecture-Based Platforms have shared resources such as Last Level Cache (LLC) (L3), memory, and I/O controllers. VNFs competing for shared resources such as the LLC and Memory Bandwidth Monitoring (MBM) could lead to increased packet processing latency and jitter, resulting in non-deterministic platform performance. NFV simplifies consolidation of Virtual Network Function (VNFs) such as firewalls, network address translation (NAT), etc. These VNFs contend for shared resources and if any one of them utilizes more than its fair share of L3 cache, memory bandwidth, and more; the latency, jitter, and performance of the entire system may become unpredictable.

To avoid this situation and boost performance further while making NFV more reliable, Intel introduced Cache Monitoring Technology (CMT), Memory Bandwidth Monitoring (MBM), and Cache Allocation Technology (CAT). CAT provides a hardware framework to manage LLC shared resource. CMT enables tracking of the LLC occupancy, enabling detailed profiling and tracking of threads and applications or virtual machines (VMs). The MBM feature reports two type of events: local memory bandwidth and remote memory bandwidth. Local memory bandwidth reports bandwidth of a thread accessing memory associated with the local socket. In a dual socket system, the remote memory bandwidth reports the bandwidth of a thread accessing the remote socket. CAT allows an OS, hypervisor, or VMM to control allocation of a Central Processing Unit (CPU) shared LLC. A Class of Service (COS) constitutes LLC ways that the CPU can populate with data from RAM. Therefore a resource such as cores/logical threads associated with a COS can only populate its own cache ways and will only fill data from DRAM into cache ways specified in the COS.

1.2.3 CMT, MBM, and CAT Hardware Requirements

In order to run the DPDK use case with CMT, MBM, and CAT, these features must be supported in the hardware. There are currently a number of CPUs which have these features enabled.

1.2.3.1 Compatible CPU Models

The Intel® Xeon® processor E5 v3 generation supports four classes of service and a set of predefined classes of service that should not be changed at run time. Intel® Xeon® processor D generation supports 16 classes of service. There are no pre-defined classes of service and they can be changed at run time. Intel® Xeon® processor E5 v3 and Intel® Xeon® processor D generations support Core/Logical thread association with a COS that can be changed dynamically. CMT is supported on all Intel® Xeon® processor E5 v3 and Intel® Xeon® Processor D stock keeping units (SKUs). CAT is supported on the following six SKUs for Intel® Xeon® Processor E5 v3 Family shown in Table 2 and all Intel® Xeon® Processor D SKUs.

Table 2. CAT-compatible CPU Models

Model Name
Intel® Xeon® E5-2658 V3
Intel® Xeon® E5-2648L V3
Intel® Xeon® E5-2628L V3
Intel® Xeon® E5-2618L V3
Intel® Xeon® E5-2608L V3
Intel® Xeon® E5-2658A v3
Intel® Xeon® E3-1258L v4
Intel® Xeon® E3-1278L v4
all Intel® Xeon® processor D SKUs

Further compatibility information is available on the following website:

https://www-ssl.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html

§

2.0 CMT, MBM, and CAT Software Support and Tools

2.1.1 CMT, MBM, and CAT Software Requirements

The CMT, CAT, and MBM functionalities are programmed through Model Specific Register (MSR) interface. The detection process is carried out using the CPUID instruction.

2.1.1.1 CMT, MBM, and CAT Library and Utility Software Support

Standalone Approach (CMT, MBM, and CAT library) looks at the LLC usage from a Core or Logical Thread (referred to as CPU hereafter) perspective, regardless of what task is executing. For CMT, a Resource Monitoring ID (RMID) is statically assigned to a CPU and periodically the occupancy is read back. For CAT, the command line utility provides the necessary functionality to set up the CAT capabilities. The software provides flags to configure and associate cores/logical threads with a COS. If the platform has been statically configured and applications have been pinned to resources then this method will yield appropriate results. If system administrators are interested in whether the platform is suitably balanced and there are no misbehaving applications, this approach is quite reasonable.

CMT, MBM, and CAT user space software enables access to CMT, MBM, and CAT CPU hardware technologies on Linux*. The software consists of two components:

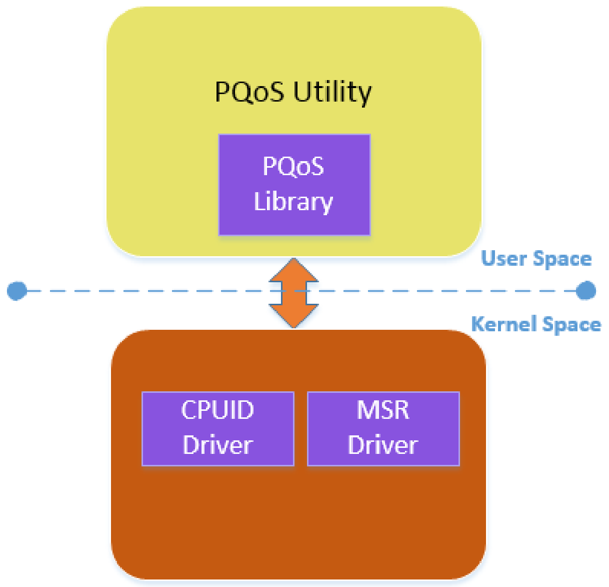
- Software library
- Utility that links against the library

The software library provides simple C API to access CMT, MBM, and CAT technologies. It only relies on C and pthread libraries and uses MSR and CPUID Linux* kernel drivers through their standard interfaces. Consequently an application using the library needs an adequate privilege level to operate. Scheduler-integrated *performance* (Kernel Version 4.1 or later) LLC occupancy monitoring per application process identifier (PID) and all associated task identifiers (TIDs) is integrated in the standalone library/utility.

The utility provides a simple command line interface to library features. The aim is to quickly enable customers to evaluate the technologies.

Figure 1 shows CMT, MBM, and CAT library implementation.

Figure 1. CMT, MBM, and CAT Library and Utility



This CMT, MBM, and CAT library/Utility is available for download on http://01.org which supports the CMT, MBM, CAT, and Code Date Prioritization (CDP).

> https://01.org/packet-processing/cache-monitoring-technology-memory-bandwidth-monitoring-cache-allocation-technology

For additional CMT, MBM, CAT, and CDP details see the *Intel® Architecture Software Development Manuals* available at:

> http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Specific information with regard to CMT, MBM, CAT and CDP can be found in chapters 17.14 and 17.15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manuals*.

2.1.1.2 CMT and CAT Kernel Support

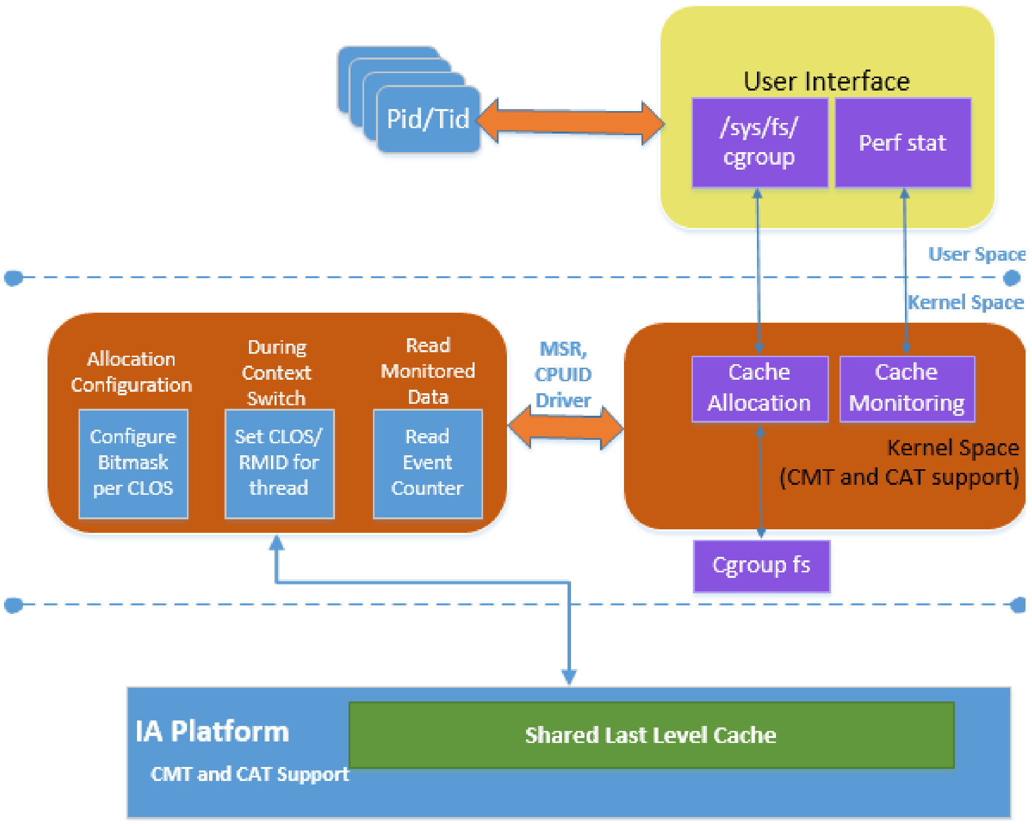
The scheduler-based approach extends the operating system task scheduler with CMT and CAT support, in this case via Linux* perf and cgroup frameworks. This results in the ability to monitor and manage LLC resources on task basis as opposed to the previously described CPU granularity method. The scheduler-based approach is the preferred method in cases when the CPU is shared by multiple applications or an application migrates between CPUs.

To enable standalone and scheduler based monitoring, several software development initiatives are in progress that are described in subsequent sections.

Scheduler-based Cache Monitoring ensures that the application of interest is tracked with the appropriate core and RMID association. Likewise, scheduler-based Cache Allocation ensures that the application of interest is associated with appropriate cache partition. Scheduler-based Cache Monitoring ensures monitoring of the application LLC occupancy. Under Linux* this is achieved by integrating CMT into *perf* and integrating CAT into a control group (cgroup). Kernel support for both CMT and CAT are tightly bound to the Linux* scheduler's functionality.

Figure 2 shows a brief diagram of the CMT/CAT kernel implementation.

Figure 2. CMT/CAT Kernel Implementation



cgroup CAT Usage involves two steps:

1. Assigning cache bitmask to cgroup
```
/bin/echo 0x0000f >
/sys/fs/cgroup/intel_rdt/cat_cgroup1/intel_rdt.cache_mask
```

2. Allocating cache per thread through cache bitmask to *cgroup*
```
/bin/echo 3530 > /sys/fs/cgroup/intel_rdt/cat_cgroup1/tasks
```

perf CMT Usage for monitoring per thread cache occupancy in bytes:

```
perf stat -e intel_cqm/llc_occupancy/  -p  <pid>
perf stat -e intel_cqm/llc_occupancy/  -t  <tid>
```

Output:

```
Performance counter stats for process id '120673':
30,867,456.00 Bytes intel_cqm/llc_occupancy/
1.355717105 seconds time elapsed
```

2.1.1.3 CAT Kernel Patch Details

Download the two sets of patches from:

http://marc.info/?l=linux-kernel&m=142620227328406

http://marc.info/?l=linux-kernel&m=142203876105241

2.1.1.4 Steps to Build a Patched Kernel

1. Get Kernel sources:
   a. Kernel version 4.1.0, stable version
   b. Fedora 21 (`cat /etc/`**`issue`** to see)
2. Configuration: Old:
   a. Check the number of cores (`cat /proc/`**`cpuinfo`**)
   b. Type `Make oldconfig`
   c. Use defaults, EXCEPT when prompted on number of cores: enter the correct value (not the default)
3. Patch the kernel:
   a. Download and apply these nine patches:

      http://marc.info/?l=linux-kernel&m=142203876105241
   b. Download and apply these seven patches:

      http://marc.info/?l=linux-kernel&m=142620227328406
4. Configuration: RDT cgroup:
   a. Type `Make menuconfig`
   b. Enable the following:

```
General setup -> Control Group Support -> Resource
Director Technology cgroup subsystem
```

5. Build the kernel:

   Edit the `Makefile`, and set custom `EXTRAVERSION`. For example:
   `EXTRAVERSION = -mybuildDDMMYYwithCAT`
   a. Type `Make`.

6. Install the kernel:
   a. `Type sudo make modules_install`
   b. `Type sudo make install`
   c. `Type sudo reboot`

Also see:

- Uninstall or remove old kernels in Fedora*:
  http://fsdportal.com/remove-old-kernel-fedora/
- Build the Linux* kernel:
  http://kernelnewbies.org/KernelBuild
- Obtain the kernel config from the current system:
  http://superuser.com/questions/287371/obtain-kernel-config-from-currently-running-linux-system
- Update options in the **boot** menu:
  https://ask.fedoraproject.org/en/question/8885/how-can-i-change-default-operating-system-in-start-up-boot-menu/

2.1.1.5 Shell Command to Check CAT Status

In order to check if CAT was enabled on the system, you must be able to see the following messages:

```
$ dmesg | egrep -i 'rdt'
[    2.587504] intel_rdt: cbmlength:20,Closs: 4
[    0.134496] Initializing cgroup subsys rdt
```

The **`cbmlength`** shown above represents your bitmask size (20 bits). Your LLC can be split (a granularity of up to 20 partitions).

2.1.2 Software Requirements DPDK with PQoS Usage

- DPDK 2.0.0
- CMT, MBM, and CAT software support

Refer to Section 4.1.

§

3.0 Measurement and Analysis

3.1 DPDK IP-pipeline and Noisy Neighbor Overview

3.1.1 Stress-ng Tool

The following command to start the `stress-ng` tool is aimed at utilizing the cache as aggressively as possible. When three such instances are running in parallel, they effectively act on one another as a cache-intensive noisy neighbor.

```
stress-ng-0.03.20/stress-ng \
--cache 90 \
--cache-flush \
--cache-prefetch \
--aggressive \
--cpu 2 \
--cpu-method matrixprod \
--timeout ${timeout}
```

The timeout on the tests ranges from 10 seconds up to 12 hours (10 s – 12 h).

For testing purposes three stress-ng instances were run with this cache configuration for various periods of time.

Refer to http://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf

3.1.2 DPDK IP-pipeline Application

The benefits of CMT and CAT can be seen in a native as well as a virtualized packet processing application. The sample application used in this example is the Internet Protocol (IP) pipeline included in the DPDK. The IP pipeline application is intended to be a vehicle for rapid development of packet processing applications running on multi-core CPUs.

The application provides a library of reusable functional blocks called pipelines. These pipelines can be seen as prefabricated blocks that can be instantiated and inter-connected through packet queues (rings) to create complete applications (super-pipelines).

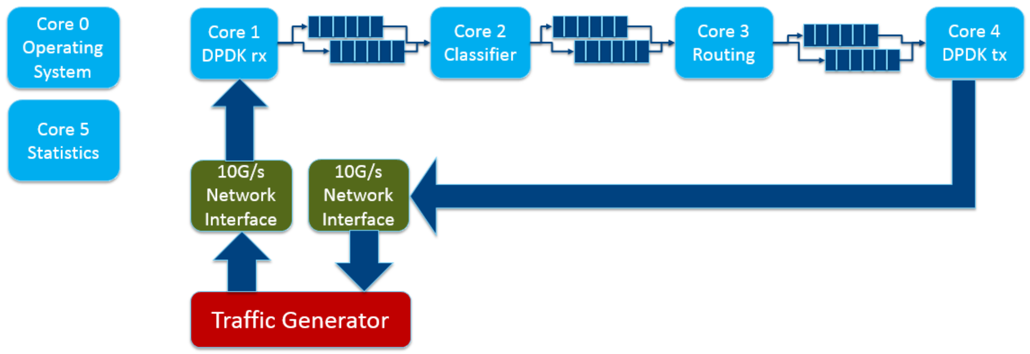
Pipelines are created and inter-connected through the application configuration file. By using different configuration files, different applications are effectively created, therefore this application can be seen as an application generator. The configuration of each pipeline can be updated at run-time through the application's Command Line Interface (CLI).

The application uses six CPU cores:

- Core 1 (RX core) receives traffic from the NIC ports and feeds core 2 with traffic through software queues.
- Core 2 (Flow Classification core) searches through the current set of flows (for example, list all flows with a specific source IP address). This pipeline functional block performs exact matching based on five tuples (source IP address, destination IP address, source port number, destination port number, and protocol). Core 2 receives traffic from Core 1 through software queues, processes it according to the actions configured in the table entries that are hit by the input packets and feeds it to Core 3 through another set of software queues.
- Core 3 (Routing core) receives traffic from Core 2 through software queues and routes it to the ports based on routing table rules/configuration.
- Core 4 (TX core) receives traffic from Core 3 through software queues and sends it to the NIC ports for transmission.
- Core 0 and Core 5 are unused by the IP Pipeline application. They are left to run OS and monitoring tasks.

Figure 3 shows the four-stage IP pipeline function block diagram. Core 0 and Core 5 are unused by the IP Pipeline application; they are left to run the OS and to monitor tasks.

Figure 3. Four Stage IP Pipeline Application



Refer to http://dpdk.org/doc/guides/sample_app_ug/ip_pipeline.html for specific information with regard to IP pipeline app details, packet profile configuration, etc.

3.1.3 Test Environment Configuration

The four stage IP pipeline above has been implemented in a guest VM. Figure 4 shows an example of a setup with noisy neighbor detection as well as a mechanism to tune the DPDK-IP pipeline VM for a performance boost using CMT and CAT.

DPDK IP Pipeline VM Configuration:

1. DPDK VM is brought up and connected to the NIC with PCI passthrough.
2. Data Path:
   - The packet generator creates flows based on RFC 2544.
   - Flows arrived at the first vPort of the VM.
   - The VM receives the 16 million flows and the IP pipeline functional blocks does packet processing and forwards them out though the vPort.
   - The flow is sent back to the packet generator.
3. VM is pinned to hardware threads (i.e., cores) 2, 3, 4, 5, 6, and 7.
4. DPDK IP pipeline command line example:

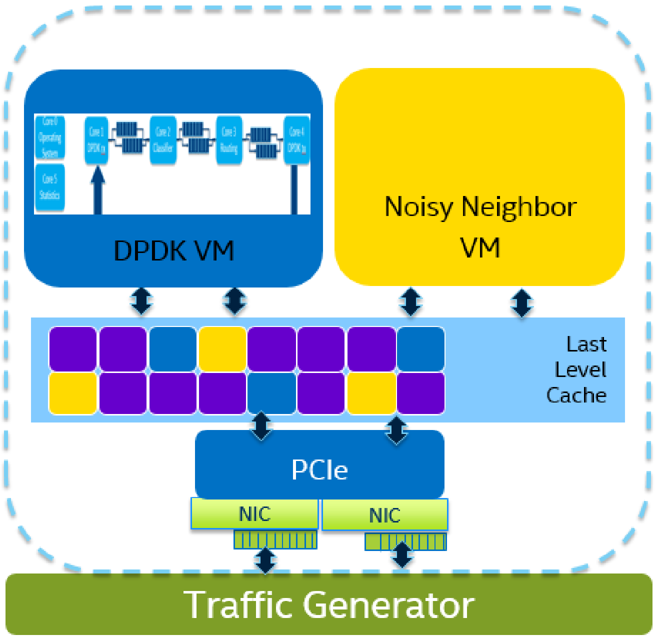
```
./build/ip-pipeline -c 7e -n 4 -- -p 3 -f ip_pipeline.cfg
```

5. Throughput reported is bidirectional.

Noisy Neighbor (Stress-ng) VM Configuration:

1. VM image with stress-ng tool. For details about the stress-ng command line, see Section 3.1.1.
2. VM is pinned to hardware threads (i.e., cores) 8, 9, and 10.

Figure 4. DPDK Performance Scaling using CMT and CAT



3.1.4 System Configuration

There are three test cases to compare the packet performance of DPDK IP-pipeline when run with and without CAT enabled and configured to prioritize the LLC for a sensitive IP-pipeline application.

- Running IP-pipeline in a VM (see Section 3.1.5).
- Running IP-pipeline in a VM with and three instances of stress-ng in a VM without CAT (see Section 3.1.6).
- Running IP pipeline in a VM with three instances of stress-ng in a VM with CAT (refer to Section 3.1.7.)

Refer to Section 4.0.

3.1.5 DPDK IP-pipeline Idle Platform

The purpose of running this test is to see what the "default" or "idle" packet performance of the system really is. It provides a reference packet performance to see if it degrades or improves compared to Step 2 and 3. This will be called the "ideal" packet performance.

In this 2 x 10 Gbps port configuration, the platform delivers 16 million packets per second (Mpps) of 64-byte packets. LLC occupancy to sustain this performance is 23 MB (out of 25 MB available in this CPU model) as shown in Table 3.

Table 3. DPDK IP-pipeline Packet Performance and LLC Occupancy Idle Platform

CAT	Noisy Neighbor	DPDK IP Pipeline Application			
		Packet Size (Bytes)	Flows (Millions)	Throughput (Mpps)	LLC Occupancy (MB)
Not Present	Not Present	64	16	16	23

Note: LLC occupancy can be determined via Linux* perf utility, when using the scheduler-based approach (see Section 2.1.1.2 for information about Linux* perf usage monitoring LLC occupancy). The CMT, MBM, and CAT user space utility (pqos) can be used in CPU based approach. A simple "./pqos" command can be used to monitor LLC occupancy on the CPU core basis. Refer to the utility help page for more information about tool functions and usage.

3.1.6 DPDK IP-pipeline VM with Aggressor (stress-ng) VM without CAT

In this system there are two VMs set up: one VM runs the DPDK IP Pipeline application and the other VM runs the stress-ng aggressor application concurrently.

In this 2 x 10 Gbps port configuration, the platform delivers 9.8 million packets per second (Mpps) of 64 byte packet. DPDK IP pipeline application performance drops by 6 Mpps because of noisy neighbor (stress-ng) occupying substantial LLC. Table 4 shows LLC occupancy of the IP Pipeline as reported by CMT is 4.5 MB.

Table 4. DPDK IP-pipeline Packet Performance and LLC Occupancy without CAT

CAT	Noisy Neighbor	DPDK IP Pipeline Application			
		Packet Size (Bytes)	Flows (Millions)	Throughput (Mpps)	LLC Occupancy (MB)
Not Present	Present	64	16	9.8	4.5

3.1.7 DPDK IP-pipeline VM with Aggressor (stress-ng) VM with CAT

This is the most important part of the testing: running the same DPDK IP pipeline with an aggressor workload (stress-ng). We are creating two COS as shown below.

Note: All commands are executed in a host shell.

- For DPDK IP pipeline VM:
  - CAT using *cgroup* (scheduler model):
    1. Create cgroup for DPDK IP pipeline VM: (Size 13.75 MB)
       ```
       /bin/echo 0xffe00>
       /sys/fs/cgroup/intel_rdt/cat_cgroup1/intel_rdt.cache_mask
       ```
    2. Allocating cache per thread through cache bitmask to cgroup:
       ```
       /bin/echo "PID of DPDK VM" >
       /sys/fs/cgroup/intel_rdt/cat_cgroup1/tasks
       ```
  - CAT using CMT, MBM, and CAT library/utility (standalone model):
    1. Set COS 1 to bitmask (13.75 MB of 25 MB with 20 ways, 1.25 MB per way)
       ```
       ./pqos -e "llc:1=0xffe00"
       ```
    2. Associate DPDK VM logical cores to COS 1
       ```
       ./pqos -a "llc:1=2,3,4,5,6,7"
       ```
- For stress-ng VM:
  1. Create cgroup for stress-ng VM: (Size 2.5 MB)
     ```
     /bin/echo 0x00003>
     /sys/fs/cgroup/intel_rdt/cat_cgroup2/intel_rdt.cache_mask
     ```
     a. Allocating cache per thread through cache bitmask to cgroup
     ```
     /bin/echo "PID of stress-ng VM" >
     /sys/fs/cgroup/intel_rdt/cat_cgroup2/tasks
     ```
  2. CAT using CMT, MBM, and CAT library/utility:
     a. Set COS2 to bitmask value 3 (2.5 MB, 25 MB LLC, 20 cache ways, 1.25 MB per cache way):
     ```
     ./pqos -e "llc:2=0x00003"
     ```

b. Associate stress-ng VM logical cores to COS 2:

```
./pqos -a "llc:2=8,9,10"
```

Stress-ng VM: should no longer be cache hungry, by allocating this limit, and noisy neighbor behavior should not occur. The performance improvement should be clearly visible by seeing increased packet performance. Figure 5 shows a general flowchart for this test run.

Figure 5. Standalone Model Feedback Loop

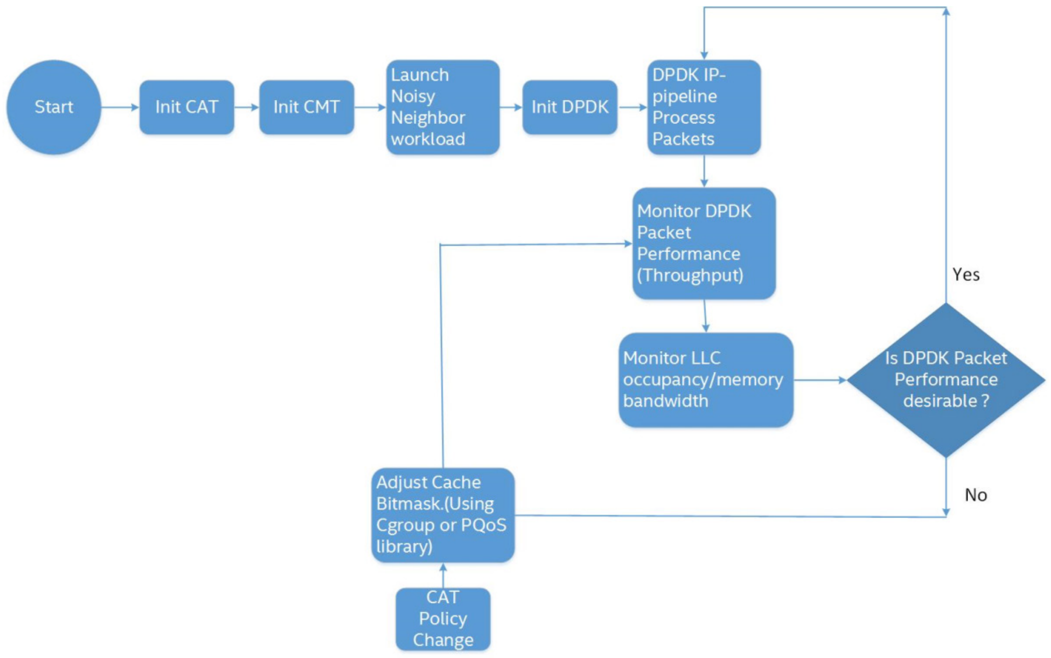

Start
Init CAT
Init CMT
Launch Noisy Neighbor workload
Init DPDK
DPDK IP-pipeline Process Packets
Monitor DPDK Packet Performance (Throughput)
Monitor LLC occupancy/memory bandwidth
Is DPDK Packet Performance desirable ?
Yes
No
Adjust Cache Bitmask.(Using Cgroup or PQoS library)
CAT Policy Change

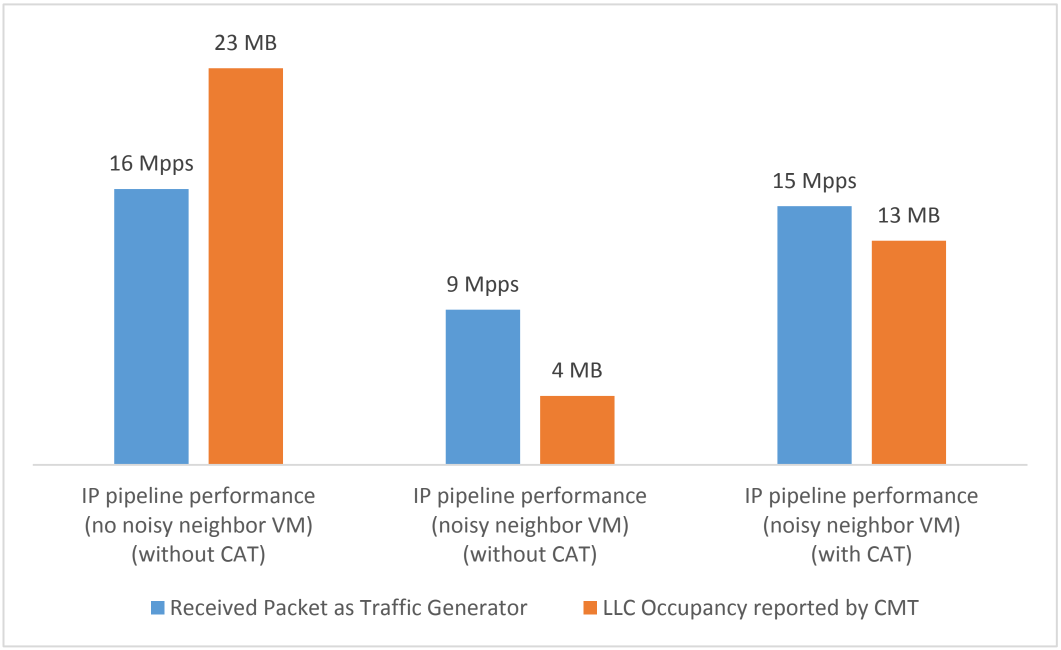
Table 5. DPDK IP Pipeline Packet Performance and LLC Occupancy with CAT

CAT	Noisy Neighbor	DPDK IP Pipeline Application			
		Packet Size (Bytes)	Flows (Millions)	Throughput (Mpps)	LLC Occupancy (MB)
Present	Present	64	16	15	13.75

3.1.8 Performance Improvement

In this 2 x 10 Gbps port configuration, the platform delivers 16 million packets per second (Mpps) of 64 byte packet throughput as depicted in the leftmost pane of Figure 6 . In this scenario, there is only one active application in the system and it is the DPDK IP pipeline. Consequently, the application's cache occupancy reaches 23 MB out of 25 MB available in this CPU model. The middle pane depicts performance of the DPDK application with active aggressor VM. The noisy neighbor application occupies about 20 MB of cache leaving only 4 MB to the DPDK VM. As the result, DPDK IP pipeline performance drops by 6 Mpps. The pane on the right depicts performance after applying CAT settings to limit the aggressor VM access to last level cache and isolate it from the DPDK VM. This results in an almost complete recovery of DPDK IP pipeline performance to 15 Mpps with the aggressor VM being active at the same time. In this case, the DPDK VM needs only 13 MB of exclusive cache in order to sustain 15 Mpps DPDK IP pipeline performance. This is far less than 23 MB occupied on the leftmost pane but in that case the aggressor VM was not present and the default CAT configuration was in place. Consequently, the DPDK VM occupied almost all of the last level cache.

Figure 6. IP Pipeline Performance Improvement using CMT and CAT



§

4.0 Platform/Software Details

4.1 Software Packages

Table 6 lists the software packages that were used.

Table 6. Table of Software Packages

Name	Version	Link	Comments
Host Kernel	4.1.0	https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.1.1.tar.gz	Refer to: CAT required Kernel patches Steps to build a patched Kernel
Kernel patch for CAT	v7	http://marc.info/?l=linux-kernel&m=142620227328406&w=2 http://marc.info/?l=linux-kernel&m=142203876105241	
CMT,MBM, and CAT library/ utility		https://01.org/packet-processing/cache-monitoring-technology-memory-bandwidth-monitoring-cache-allocation-technology-code-and-data	
DPDK 2.0.0		http://dpdk.org/browse/dpdk/snapshot/dpdk-2.0.0.tar.gz	
Stress-ng Benchmark		http://kernel.ubuntu.com/~cking/stress-ng/	

4.2 Hardware Requirements

Table 7 lists the hardware packages that were used.

Table 7. Table of Hardware Requirements

Name	Version	Link	Comments
Wildcat Pass	Intel® Xeon® CPU E5-2628 L v3 @ 2.00 GHz	http://mark.intel.com/products/81704/Intel-Xeon-Processor-E5-2628L-v3-25M-Cache-2_00-GHz?q=Intel%C2%AE%20Xeon%C2%AE%20Processor%20E5-2628L%20v3%20(25M%20Cache,%202.00%20GHz)	Refer to: CAT Hardware requirements
Ethernet Adapter	Intel® Ethernet Converged Network Adapter X520	http://ark.intel.com/products/codename/32659/Niantic	
Memory	DDR4 RAM		Size: 68 GB

4.3 Default BIOS Settings

BIOS settings unchanged, however, noticed that Hyper-Threading was enabled.

Table 8. BIOS Settings

Setting	State
Hyper-Threading	ENABLED
VT-d	ENABLED

§

5.0 Conclusion

Shared resource contention can cause performance degradation. For example, the performance of the DPDK-based packet processing pipeline introduced in Section 3.0 is impacted by 44% as a result of the execution of a cache-intensive application executing on a different core. With CMT we can detect subscription of the LLC, with CAT we can isolate the noisy neighbor and restore the performance up to 94%. The remaining 6% performance degradation is very likely due to resource contention at the memory controller level.

Various tools are introduced to take advantage of these new PQoS capabilities. Developers or system administrators can utilize a user library available on http://01.org that does not have kernel scheduler requirements to monitor Cache and/or Memory Bandwidth Monitoring, with the limitation that it provides data on a per core or thread basis. In addition, the user space tool provides the ability to setup the Cache Allocation functionality as well.

CMT and MBM scheduler integrated support is available as of Linux* kernel 4.1, the perf tool and the Linux* scheduler have been enhanced to track PID/TID occupancy. The CAT scheduler integrated support is available through a new cgroup introduced in Section 4.0.

To develop next generation virtual appliances based on DPDK to support both network functions virtualization (NFV) and Software-Defined Networking (SDN), it is crucial to provide PQoS. The new shared resource monitoring and allocation capabilities provide developers with the ability to regain control over the platform while executing in a consolidated environment. Stay tuned for further optimizations in the field of DPDK and PQoS.

§

6.0 References

Reference	URL
Intel® 64 and IA-32 Architectures Software Developer's Manuals	http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html (v055, Vol 3b. Chapter 17.15 and 17.16, covers CMT, CAT, MBM and CDP)
Intel, Cache Monitoring and Cache Allocation Technologies landing page	http://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html
CMT, MBM, CAT and CDP public software library/utility	https://01.org/packet-processing/cache-monitoring-technology-memory-bandwidth-monitoring-cache-allocation-technology-code-and-data
CMT, MBM, CAT and CDP public software library/utility GitHub Project	https://github.com/01org/intel-cmt-cat
Intel, "*Enabling NFV to Deliver on its Promise*"	http://www.intel.com/content/www/us/en/communications/nfv-packet-processing-brief.html
CAT cgroup kernel patches	http://marc.info/?l=linux-kernel&m=142620227328406&w=2
Christos Kozyrakis et al, "*Heracles: Improving Resource Efficiency at Scale*"	http://csl.stanford.edu/~christos/publications/2015.heracles.isca.pdf, 2015
Introduction to CMT Blog	https://software.intel.com/en-us/blogs/2014/06/18/benefit-of-cache-monitoring
Discussion of RMIDs and CMT Software Interfaces Blog	https://software.intel.com/en-us/blogs/2014/12/11/intel-s-cache-monitoring-technology-software-visible-interfaces
Use Models and Example Data using CMT Blog	https://software.intel.com/en-us/blogs/2014/12/11/intels-cache-monitoring-technology-use-models-and-data
Software Supports and Tools: Intel's CMT: Software Support and Tools	https://software.intel.com/en-us/blogs/2014/12/11/intels-cache-monitoring-technology-software-support-and-tools
Intel Platform Shared Resource Monitoring and CAT	http://smackerelofopinion.blogspot.com/2015/11/intel-platform-shared-resource.html

§