Implementing the Intel DLB and Intel QAT Accelerators on ThinkSystem Servers Running VMware ESXi

Planning / Implementation

Compute architecture and workloads are evolving and digital traffic has grown tremendously, and this has pushed data centers to be larger than ever, driving an explosive demand for compute. Modern workloads place increased demands on compute, storage and network resources. To deal with the ever-increasing need for compute power, many customers deploy power-efficient accelerators to offload specialized functions and reserve compute cores for general-purpose tasks. Offloading specialized tasks to AI, security, HPC, networking and analytics can result in power savings and faster time to results.

The 4th Gen and 5th Gen Intel Xeon Scalable processors include a broad set of integrated accelerators, which speed up data movement, encryption, and compression for faster networking and storage, boost query throughput for more responsive analytics, and offload scheduling and queue management to dynamically balance loads across multiple cores.

Applications from storage, database, networking, and big data can greatly benefit from the higher performance, reduced latency, lower data footprint and processing efficiencies these accelerator engines will deliver.

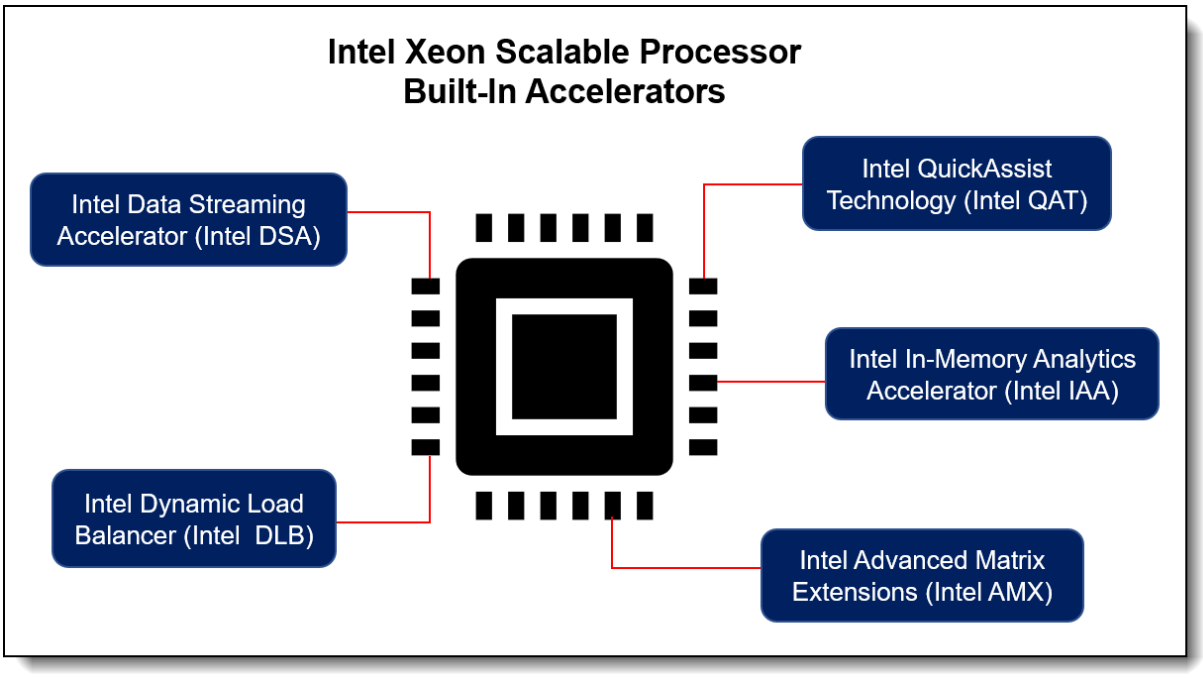

Figure 1. Built-in accelerators of the 4th Gen and 5th Gen Intel Xeon Scalable processor

As the above figure shows, the 4th Gen and 5th Gen Intel Xeon Scalable processor include a set of dedicated accelerator engines for the most common processing functions in the data center:

- Intel Advanced Matrix Extension (Intel AMX) to speed up low-precision math and accelerate AI/ML.
- Intel Data Streaming Accelerator (Intel DSA) to copy and move data faster and assist PDK.

- Intel QuickAssist Technology (Intel QAT) to accelerate compression, encryption, and decryption.
- Intel In-memory Analytics Accelerator (Intel IAA) to speed up query processing performance.
- Intel Dynamic Load Balancer (Intel DLB) to help speed up data queues.

The availability of accelerators varies depending on the processor SKU. For details which processor SKUs include each accelerator, see the Lenovo Press processor comparison reference:

https://lenovopress.lenovo.com/lp1262-intel-xeon-scalable-processor-comparison

In this paper, we demonstrate two of the accelerators, Intel DLB and Intel QAT, with VMware ESXi 7.0 U3. Installing Intel accelerator driver for enables single root I/O virtualization (SR-IOV) to create the virtual function (VF) from a single physical function (PF) to support hardware acceleration for guest virtual machine.

Intel Dynamic Load Balancer

Intel Dynamic Load Balancer (Intel DLB) is a hardware managed system of queues and arbiters connecting producers and consumers. It supports high queuing rates, load balancing across consumers, multi-priority queuing arbitration, multiple scheduling types, and efficient queue notification. Intel DLB appears to software as a PCIe device that provides load-balanced, prioritized scheduling of events (packets) across CPU cores/threads enabling efficient core-to-core communication.

Traditionally, the queues in the memory model have fundamental performance and algorithmic limitations. Some examples include: impact of lock latency, lock contention, memory latency, cache and snooping behaviors, and polling of multiple queues. This can lead to insufficient core compute cycles being available to meet real-time requirements for more complicated queue configurations and/or more complicated scheduling decisions such as: multiple queues, priority between queues, and consumer load balancing.

With the queues and the associated pointers implemented in the Intel DLB accelerator, the limitations of locking, memory/cache latencies, and snooping behaviors are addressed by storing the pointers and the queue itself in Intel DLB local memory. The compute and polling limitations are solved by using purpose-built logics in the Intel DLB.

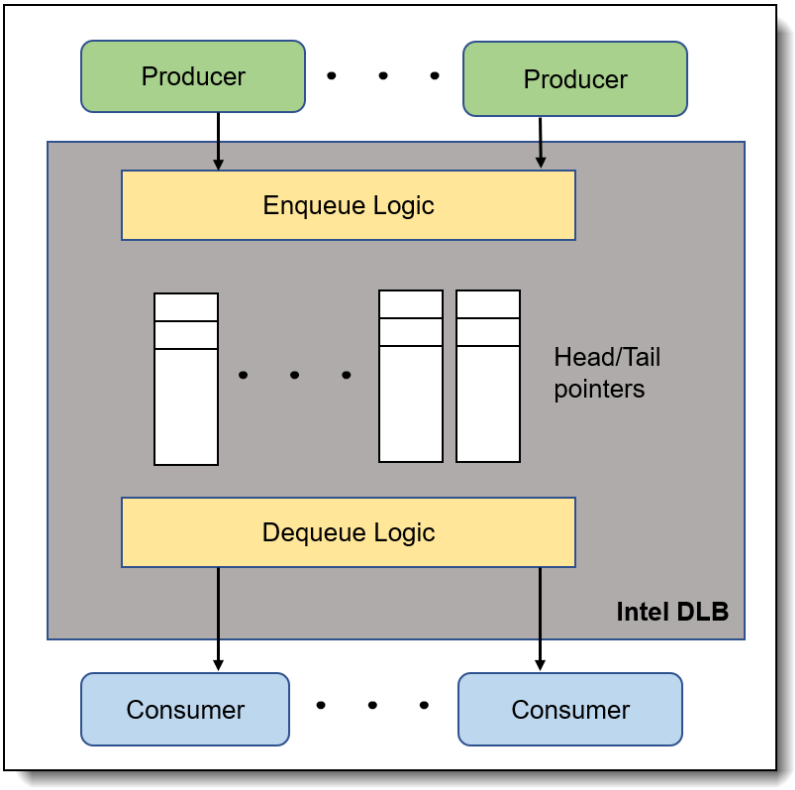

Figure 2. Intel DLB with producer/consumer model

Intel DLB supports a producer/consumer model as the above figure shows. A producer is an agent that has a type of message to place onto a queue. A consumer is an agent that removes the message from the queue. These messages typically describe work for the consumer to execute.

The high-level Intel DLB data flow is as follows:

1. Software threads interact with the hardware by enqueuing and dequeuing Queue Elements (QEs).
2. QEs are sent through a Producer Port (PP) to the Intel DLB internal QE storage (internal queues), optionally being reordered along the way.
3. The Intel DLB schedules QEs from internal queues to a consumer according to a two-stage priority arbiter.
4. Once scheduled, the Intel DLB writes the QE to a main-memory-based Consumer Queue (CQ), which the software thread reads and processes.

The supported processors have 0, 1, 2 or 4 DLB accelerators depending on the processor SKU. For specifics, refer to Intel Xeon Scalable processor comparison reference:
https://lenovopress.lenovo.com/lp1262-intel-xeon-scalable-processor-comparison#term=dlb

Implementing Intel DLB

This section describes the steps we took to configure and enable Intel DLB with VMware ESXi 7.0 U3 using SR-IOV technology, to create virtual function from physical function for providing acceleration in the virtual machine.

The test configuration of ThinkSystem SR630 V3 is listed in the following table.

Table 1. ThinkSystem SR630 V3 server configuration

Component	Configuration
Server	ThinkSystem SR630 V3 Server
CPU	2x Intel Xeon Platinum 8480+ Processors
Memory	16x DDR5 4800MHz 16GB DIMMs
HDD	1.0 TB SATA HDD
Host OS	ESXi 7.0 U3 Custom Image for Lenovo ThinkSystem
Guest VM OS	RHEL 8.7

Important restrictions related to Intel DLB:

- The DLB driver release doesn't support VMware vSphere vMotion
- The number of PCI passthrough devices per VM is limited. Refer to the configuration maximum guide for more information: https://kb.vmware.com/s/article/1003497

The steps to implement Intel DLB are as follows:

1. Power up the server and boot to the UEFI setup menu. Ensure that Intel VT for Directed I/O (VT-d) and SRIOV options are enabled as shown in the following figure.

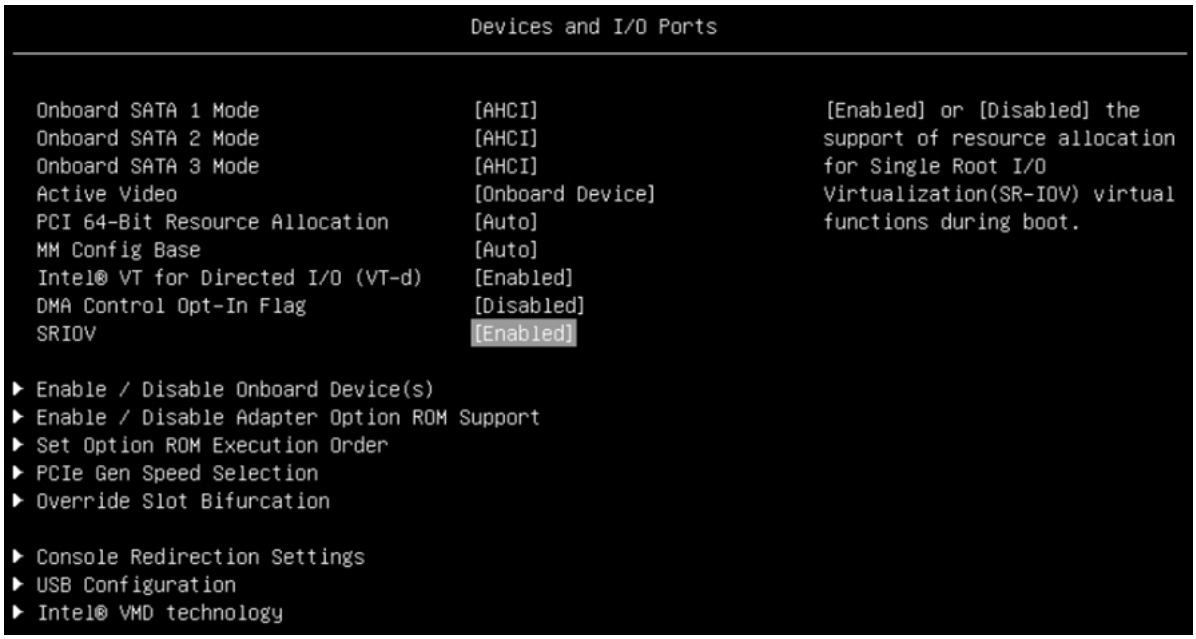

Figure 3. Intel VT-d and SRIOV options in the UEFI setup menu

2. Go to the Intel website and download the driver for Intel DLB hardware version 2.0 for VMware ESXi:

   https://www.intel.com/content/www/us/en/download/757792/intel-dynamic-load-balancer-driver-for-vmware-esxi.html?wapkw=DLB

3. Install VMware ESXi 7.0 U3 on the server and then install the Intel DLB driver component as shown below.

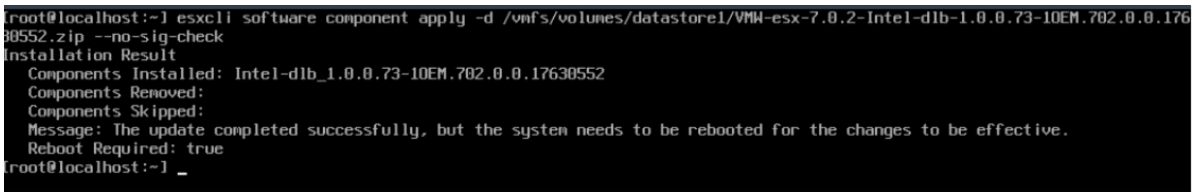

Figure 4. VMware DLB driver installation

4. Reboot the system to complete driver installation.
5. Verify the Intel DLB driver is loaded in OS after reboot. The following figure shows there are two DLB devices with PCI ID 8086:2710 in the system.

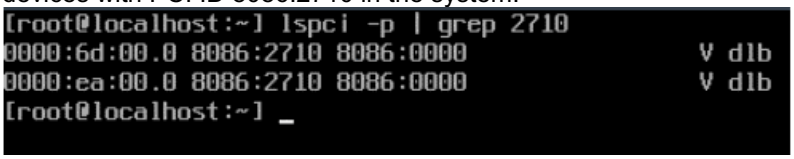

Figure 5. Display DLB device in VMware

6. Login to vSphere client.
7. From the left-hand navigation menu, select **Manage > Hardware > PCI Devices** . Select **Intel Corporation DLB PF v2.0 > Configure SR-IOV**, set the Enabled option to **Yes** and input the desired number of virtual function devices in the range between 1 and maximum indicated in the window, as shown in the two figures below.

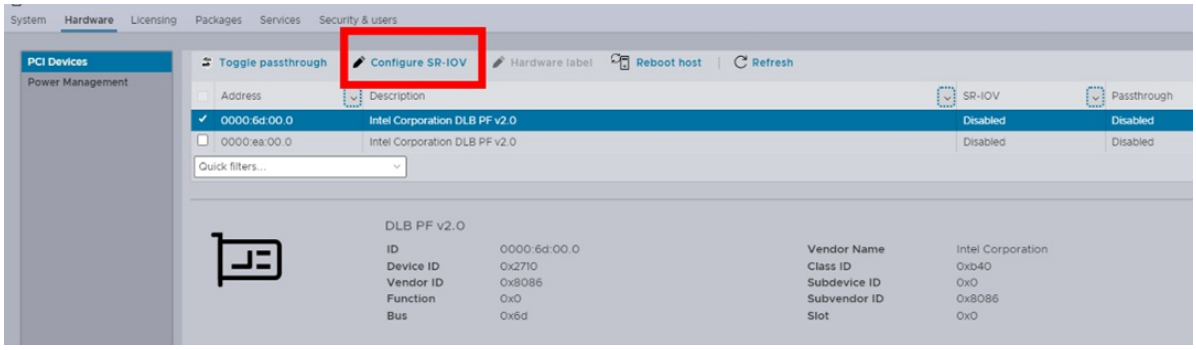

Figure 6. Configure SR-IOV in vSphere client

Figure 7. Configure DLB virtual function number

8. Save the settings and reboot system to make changes take effect.
9. Login to the vSphere client again to verify the two DLB virtual functions (PCI ID 8086:2711) have been enabled with the description text “Intel Corporation DLB VF v2.0”, as shown below.

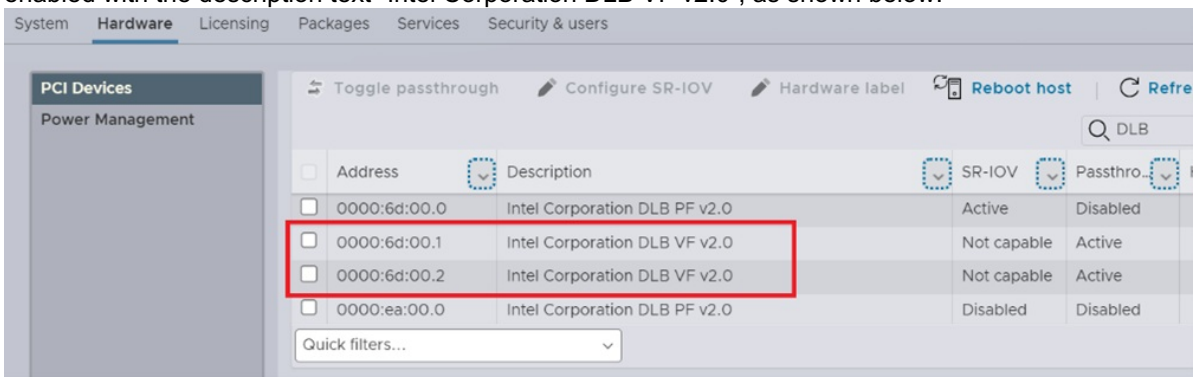

Figure 8. DLB virtual function display in vSphere Client

10. Create a new VM and install RHEL 8.7 guest OS in the VM.
11. Before powering on the VM, click the **Edit** button to configure Memory RAM of desired size, and set **Reserve all guest memory (All locked)** checkbox, as shown below.

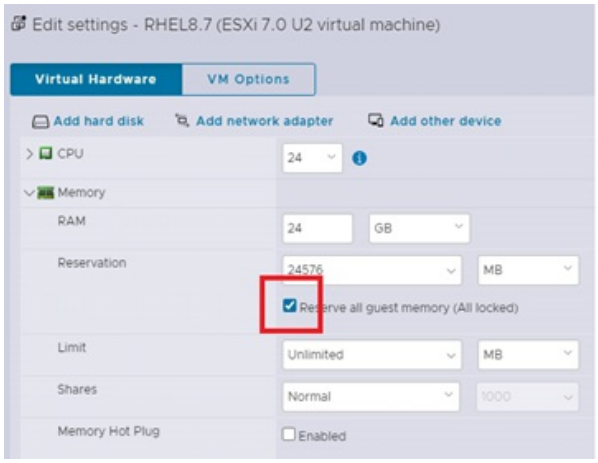

Figure 9. Reserved all memory in a VM

12. Click **Add other device > PCI device** and add “DLB VF v2.0 – 0000:6d:00.1” and “DLB VF v2.0 – 0000:6d:00.2” as new PCI devices, as shown below.

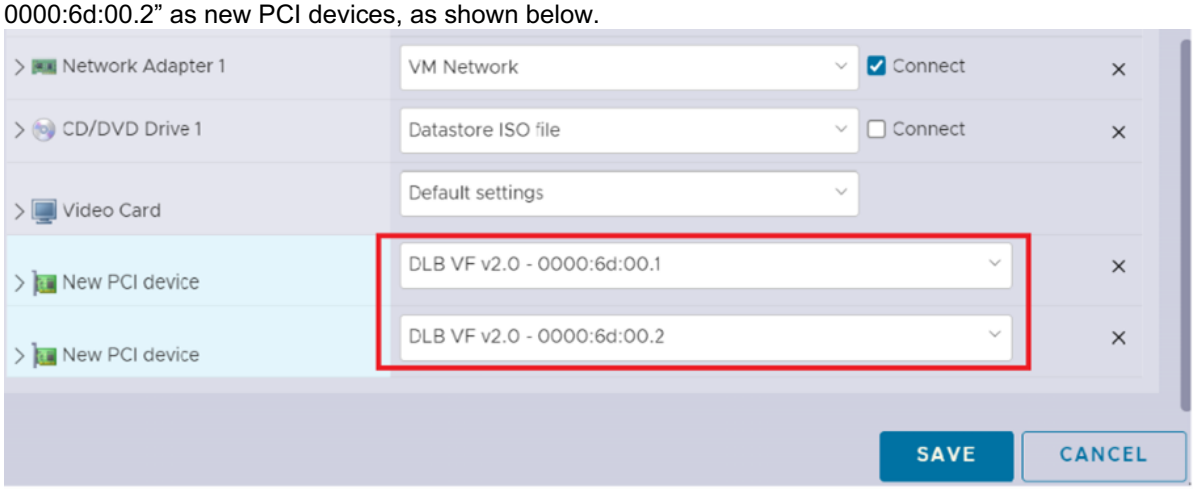

Figure 10. Add DLB device to a VM

13. Click **Save** to finish the setting and power on the VM to install guest OS.
14. When guest OS installation completes, login to the guest OS and check there are two DLB devices (PCI ID 8086:2711) in the virtual machine, as shown in the following figure.

```
[root@localhost ~]# lspci -v | grep 2711
0b:00.0 Co-processor: Intel Corporation Device 2711
13:00.0 Co-processor: Intel Corporation Device 2711
[root@localhost ~]#
```

Figure 11. Display DLB device in guest VM

15. Download Intel DLB Linux software package which contains the Intel DLB kernel driver and the libdlb client library for non-dpdk application, and copy the software package to the RHEL 8.7 guest OS. https://www.intel.com/content/www/us/en/download/686372/intel-dynamic-load-balancer.html
Libdlb is a POSIX based client library for building Intel DLB based application and provides sample code for directed and load balanced traffic tests to demonstrate features supported by the Intel DLB. The sample code is located in the dlb/libdlb/examples/ directory.
16. The Intel DLB Linux software package can be extracted using the following command:

```
~# tar xfJ dlb_linux_src_release_8.4.0.txz
```

17. Go to dlb/driver/dlb2/ directory, simply run the command `make` to build out-of-tree driver from source code and get the dlb2.ko driver module, as shown in the following figure.

```
[root@localhost dlb2]# make
make -C /lib/modules/`uname -r`/build M=/root/dlb/dlb/driver/dlb2 modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-425.3.1.el8.x86_64'
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_main.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_file.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_ioctl.o
  CC [M]  /root/dlb/dlb/driver/dlb2/base/dlb2_resource.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_pf_ops.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_intr.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_dp.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_sriov.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_vf_ops.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_vdcm.o
  CC [M]  /root/dlb/dlb/driver/dlb2/dlb2_perf.o
  LD [M]  /root/dlb/dlb/driver/dlb2/dlb2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/dlb/dlb/driver/dlb2/dlb2.mod.o
  LD [M]  /root/dlb/dlb/driver/dlb2/dlb2.ko
make[1]: Leaving directory '/usr/src/kernels/4.18.0-425.3.1.el8.x86_64'
[root@localhost dlb2]#
```

Figure 12. Compile the Linux DLB driver

18. Go to dlb/libdlb/ directory and run the command `make` to build libdlb library, as shown in the following figure. The sample source code is located at dlb/libdlb/examples/ directory.

```
[root@localhost libdlb]# make
Compiling build/dlb.o
Compiling build/dlb2_ioctl.o
Linking libdlb.so
Generating libdlb.a
make[1]: Entering directory '/root/dlb/dlb/libdlb/examples'
cc -I/root/dlb/dlb/libdlb/examples/../ -g -fPIC  -pthread  -L/root/dlb/dlb/libdlb/examples/.
. -pthread  ldb_traffic.c  -ldlb -lrt -o ldb_traffic
cc -I/root/dlb/dlb/libdlb/examples/../ -g -fPIC  -pthread  -L/root/dlb/dlb/libdlb/examples/.
. -pthread  dir_traffic.c  -ldlb -lrt -o dir_traffic
make[1]: Leaving directory '/root/dlb/dlb/libdlb/examples'
make[1]: Entering directory '/root/dlb/dlb/libdlb/cli'
Compiling /root/dlb/dlb/libdlb/cli/open_dlb
Compiling /root/dlb/dlb/libdlb/cli/dump_dlb_regs
Compiling /root/dlb/dlb/libdlb/cli/dlb_monitor_sec
make[1]: Leaving directory '/root/dlb/dlb/libdlb/cli'
make[1]: Entering directory '/root/dlb/dlb/libdlb/doc'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/root/dlb/dlb/libdlb/doc'
[root@localhost libdlb]#
```

Figure 13. Compile libdlb library and sample code

19. Use the following commands to load the DLB driver module before running the sample application:

```
~# modprobe mdev; modprobe vfio_mdev
~# insmod dlb/driver/dlb2/dlb2.ko ; cd libdlb
```

Now that the drivers are installed, you can perform some tests.

The following figure shows OS log message when the DLB driver modules is loaded in Linux OS.

```
[  216.581140] dlb2: loading out-of-tree module taints kernel.
[  216.581635] dlb2: module verification failed: signature and/or required key missing - tainting kernel
[  216.591633] dlb2 0000:0b:00.0: enabling device (0000 -> 0002)
[  216.592778] dlb2 0000:0b:00.0: [dlb2_probe()] AER is not supported
[  218.660536] dlb2 0000:0b:00.0: perf pmu not supported. Skipping perf init
[  218.660771] dlb2 0000:13:00.0: enabling device (0000 -> 0002)
[  218.662580] dlb2 0000:13:00.0: [dlb2_probe()] AER is not supported
[root@localhost ~]#
```

Figure 14. Linux OS message

For Directed Traffic test, run the following command:

```
~# LD_LIBRARY_PATH=$PWD ./examples/dir_traffic -w poll -n 128 -f 4
```

Where:

- `-w` option to specify wait mode,
- `-n` option to specify number of looped events
- `-f` to specify number of worker threads that forward events

As the following figure shows, the 128 events are evenly distributed to the 4 workers (each worker received 32 events) in the directed traffic test.

```
[root@localhost libdlb]# ./examples/dir_traffic -w poll -n 128 -f 4
DLB's available resources:
        Domains:           32
        LDB queues:        32
        LDB ports:         64
        DIR ports:         64
        SN slots:          4,4
        ES entries:        2048
        Contig ES entries: 2048
        LDB credits:       8192
        Contig LDB cred:   8192
        DIR credits:       4096
        Contig DIR cred:   4096
        LDB credit pls:    64
        DIR credit pls:    64

[tx_traffic()] Sent 128 events
[rx_traffic()] Received 128 events
[worker_fn()] Received 32 events
[worker_fn()] Received 32 events
[worker_fn()] Received 32 events
[worker_fn()] Received 32 events
[root@localhost libdlb]#
```

Figure 15. Sample code – directed traffic test

For Load Balanced Traffic test, run the following command:

```
~# LD_LIBRARY_PATH=$PWD ./examples/ldb_traffic -w poll -n 128 -f 4
```

As the following figure shows, the 128 events are dynamic distributed to the 4 workers in the load balanced traffic test.

```
[root@localhost libdlb]# ./examples/ldb_traffic -w poll -n 128 -f 4
DLB's available resources:
        Domains:           32
        LDB queues:        32
        LDB ports:         64
        DIR ports:         64
        SN slots:          4,4
        ES entries:        2048
        Contig ES entries: 2048
        LDB credits:       8192
        Contig LDB cred:   8192
        DIR credits:       4096
        Contig DIR cred:   4096
        LDB credit pls:    64
        DIR credit pls:    64

[tx_traffic()] Sent 128 events
[rx_traffic()] Received 128 events, num_mismatch: 0
[worker_fn()] Received 20 events
[worker_fn()] Received 76 events
[worker_fn()] Received 20 events
[worker_fn()] Received 12 events
[root@localhost libdlb]#
```

Figure 16. Sample code – load balanced traffic test

Intel QuickAssist Technology

Intel QuickAssist Technology (Intel QAT) is an integrated accelerator for compute intensive workloads, Intel QAT offloads bulk cryptography, compression/decompression, and Public Key Encryption (PKE). It has support for additional standards and features, including Key Protection Technology, Shared Virtual Memory (SVM), Scalable IO Virtualization (SIOV), Extended RAS (uncorrectable and fatal error support), as the following figure shows.

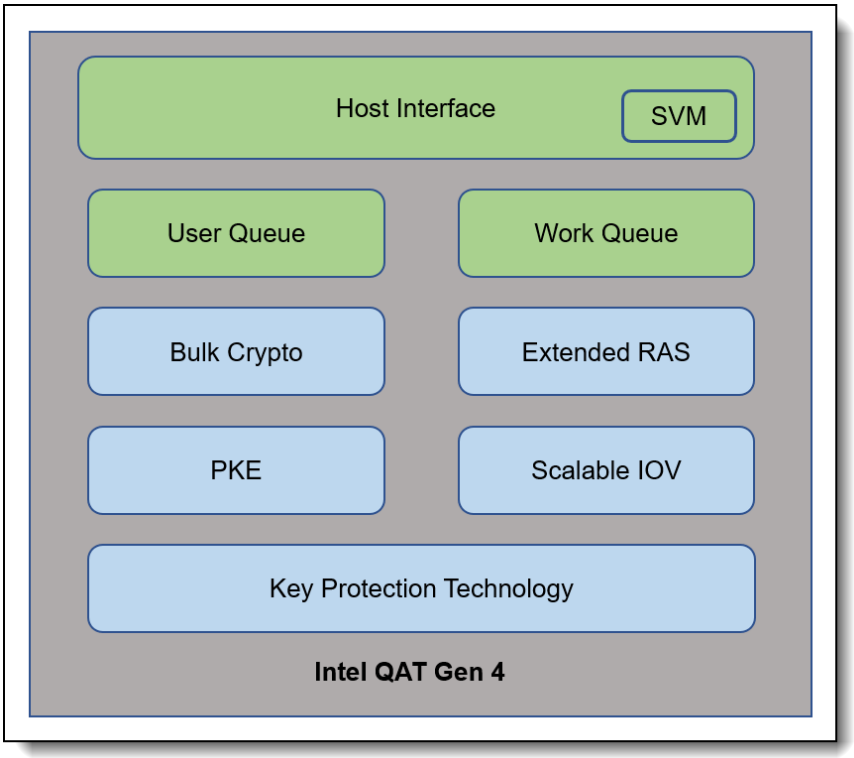

Figure 17. Intel QAT components

Intel Xeon Scalable processors have 0, 1, 2 or 4 QAT accelerators depending on the processor SKU. For specifics, refer to Intel Xeon Scalable processor comparison reference:
https://lenovopress.lenovo.com/lp1262-intel-xeon-scalable-processor-comparison#term=qat

The CPU cores access the QAT acceleration services via a standard PCIe interface. Application developers can use the feature through the QAT API, which is the top-level API for Intel QAT and enables easy interfacing between the customer application and the Intel QAT acceleration driver.

For more information about Intel QAT technology, refer to the Intel website:
https://www.intel.com/quickassist

Implementing Intel QAT

This section describes the steps we took to configure and enable Intel QAT with VMware ESXi 7.0 U3 using SR-IOV technology, to create virtual function from physical function for providing cryptographic and compression acceleration capabilities in the virtual machine.

The test configuration of the ThinkSystem SR650 V3 is listed in the following table.

Table 2. ThinkSystem SR630 V3 server configuration

Component	Configuration
Server	ThinkSystem SR650 V3 Server
CPU	2x Intel Xeon Platinum 8490H Processors
Memory	2x DDR5 4800MHz 16GB DIMMs
HDD	1.0 TB SATA HDD
Host OS	ESXi 7.0 U3 Custom Image for Lenovo ThinkSystem
Guest VM OS	RHEL 8.7

Important notes regarding Intel QAT:

- The QAT driver release doesn't support VMware vSphere vMotion
- Number of PCI passthrough devices per VM is limited. Please refer to the configuration maximum guide for more details: https://kb.vmware.com/s/article/1003497
- VMKernel supports 1024 interrupt cookies by default. On system with large number of accelerators, interrupt cookies could be exhausted, which may lead to various issues and accelerator HW will be not available. Use below command to check the current interrupt cookie allocations of PCI devices in OS:

```
~# cat /var/run/log/vmkernel.log | grep "allocate.*interrupts"
```

  Increase interrupt cookies number to the desired value (up to 4096) to support more devices via the following command:

```
~# esxcli system settings kernel set -s maxIntrCookies -v 4096
```

The steps to implement Intel DLB are as follows:

1. Power up the server and boot to the UEFI setup menu. Ensure that Intel VT for Directed I/O (VT-d) and SRIOV options are enabled as shown in the following figure.

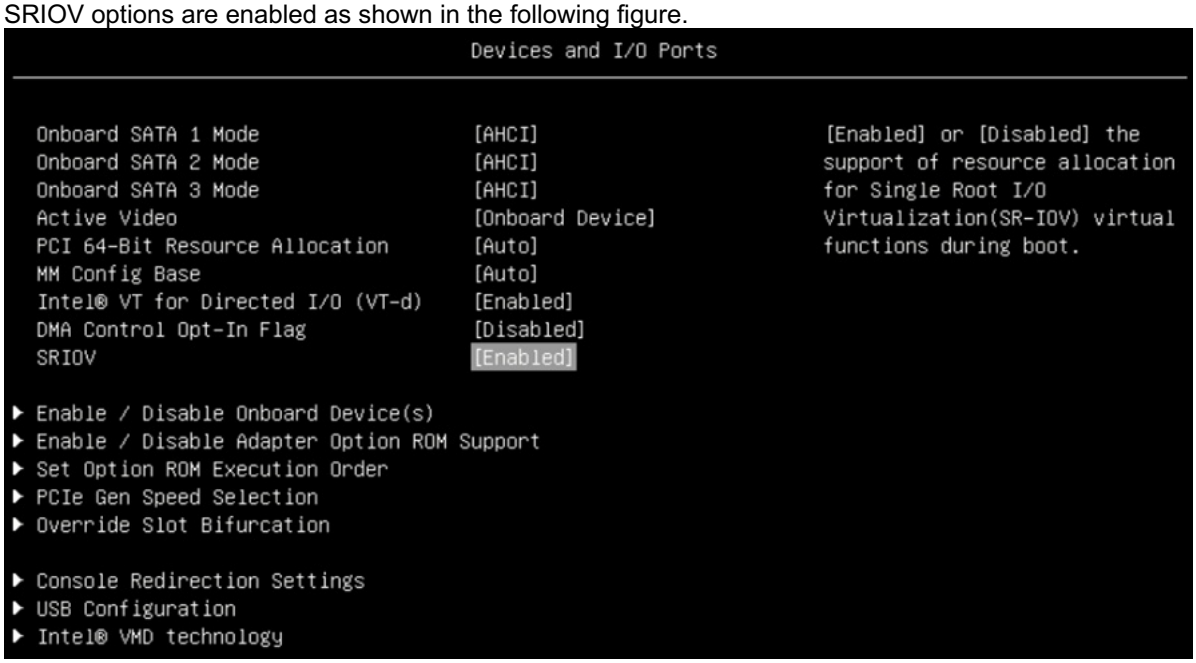

   Figure 18. Intel VT-d and SRIOV options in the UEFI setup menu

2. Go to the Intel website and download the driver for Intel QAT hardware version 2.0 for VMware ESXi:

https://www.intel.com/content/www/us/en/download/761741/intel-quickassist-technology-driver-for-vmware-esxi-hardware-version-2-0.html

3. Install VMware ESXi 7.0 U3 on the server and then install the Intel QAT driver component.

```
[root@localhost:~] esxcli software component apply -d /vmfs/volumes/datastore1/Intel-qat_2.4.0.72-10EM.700.1.0.15843807_2096223
.zip
Installation Result
   Components Installed: Intel-qat_2.4.0.72-10EM.700.1.0.15843807
   Components Removed:
   Components Skipped:
   Message: Operation finished successfully.
   Reboot Required: false
[root@localhost:~] _
```

Figure 19. VMware QAT driver installation

4. Reboot the system to complete driver installation.
5. Verify the QAT driver is loaded in OS after reboot. The following figure shows there are eight QAT devices with PCI ID 8086:4940 in the system.

```
[root@localhost:~] lspci -p |grep 4940
0000:6b:00.0 8086:4940 8086:0000                V qat
0000:70:00.0 8086:4940 8086:0000                V qat
0000:75:00.0 8086:4940 8086:0000                V qat
0000:7a:00.0 8086:4940 8086:0000                V qat
0000:e8:00.0 8086:4940 8086:0000                V qat
0000:ed:00.0 8086:4940 8086:0000                V qat
0000:f2:00.0 8086:4940 8086:0000                V qat
0000:f7:00.0 8086:4940 8086:0000                V qat
[root@localhost:~]
```

Figure 20. Display QAT device in VMware host

6. Login to vSphere client.
7. From the left-hand navigation menu, select **Manage > Hardware > PCI Devices** , then select **Intel Corporation 4xxx QAT > Configure SR-IOV**, set the Enabled option to **Yes** and input the desired number of virtual function devices, as shown below.

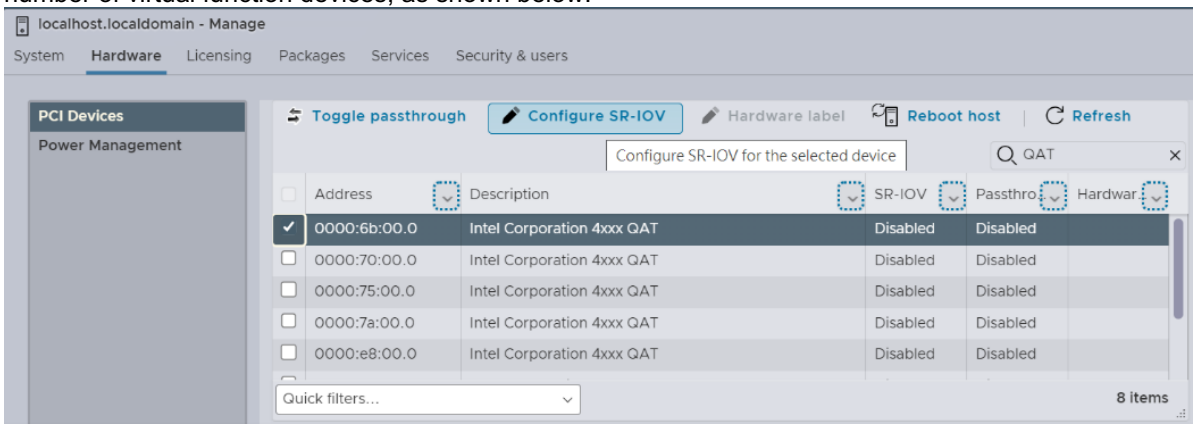

Figure 21. Configure SR-IOV function in vSphere client

8. Save the settings and reboot system to make changes take effect.
9. Login to the vSphere client and verify the QAT virtual function (PCI ID 8086:4941) has enabled with the description text “Intel Corporation 4xxx QAT VF”, as shown in the following figure.

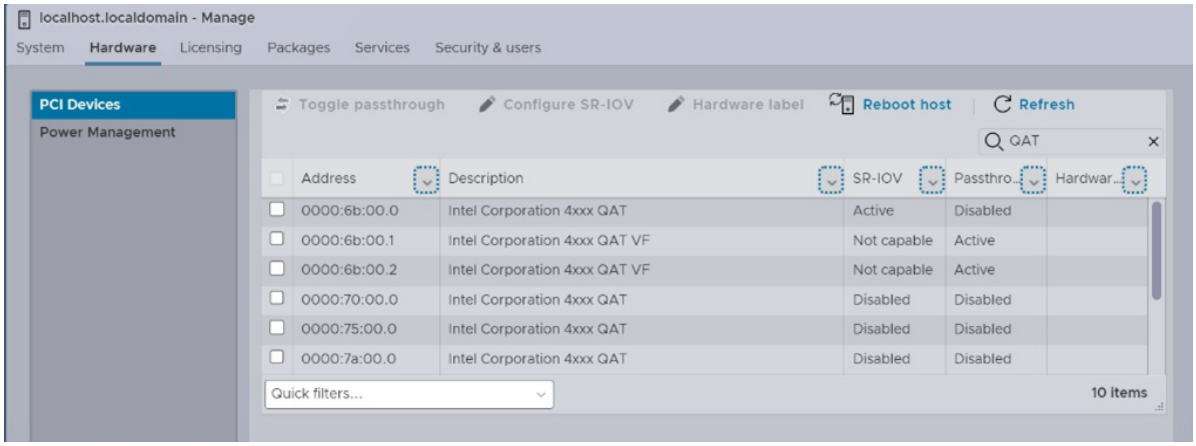

Figure 22. QAT virtual function display in vSphere Client

10. Create a new VM and install RHEL 8.7 guest OS in the VM.
11. Before powering on the VM, click the **Edit** button to configure Memory RAM of desired size, check the box **Reserve all guest memory (All locked)** checkbox. Click **Add other device > PCI device** and add "4xxx QAT VF – 0000:6b:00.1" and "4xxx QAT VF – 0000:6b:00.2" as new PCI devices, as shown in the following figure.

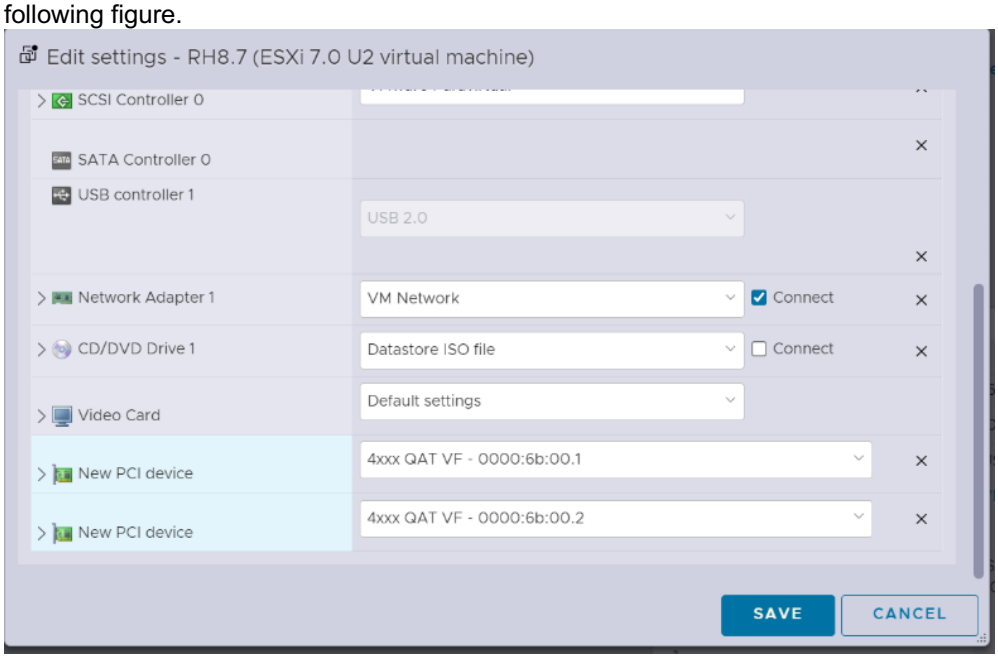

Figure 23. Add QAT device to a VM

12. When RHEL8 U7 guest OS installation completes, a number of libraries must be installed to resolve QAT package dependencies. Install these using the OS package management tool:
    - "Development Tools" group
    - boost-devel
    - systemd-devel
    - openssl-devel
    - yasm

    RHEL 8.7 does not include the yasm package, so it will need to be manually downloaded and installed, using the following commands:

```
~# wget http://www.tortall.net/projects/yasm/releases/yasm-1.3.0.tar.gz
~# tar zxvf yasm-1.3.0.tar.gz
~# cd yasm-1.3.0/
~# ./configure
~# make && make install
```

13. Download from the following URL, the out-of-tree Intel QAT software package which contains Intel QAT hardware 2.0 driver for Linux OS and copy the package to the RHEL8 U7 guest OS. https://www.intel.com/content/www/us/en/download/765501/intel-quickassist-technology-driver-for-linux-hw-version-2-0.html
14. Extract the Intel QAT software package using the following command:

```
~# tar zxvf QAT20.L.1.0.50-00003.tar.gz
```

15. Use the following commands to build and install the QAT driver and sample application, and then reboot:

```
~# ./configure
~# make && make install
~# make samples && make samples-install
~# reboot
```

16. Login to the guest OS and check that there are two QAT devices (PCI ID 8086:4941) with kernel module qat_4xxxvf loaded in the virtual machine, as shown in the following figure.

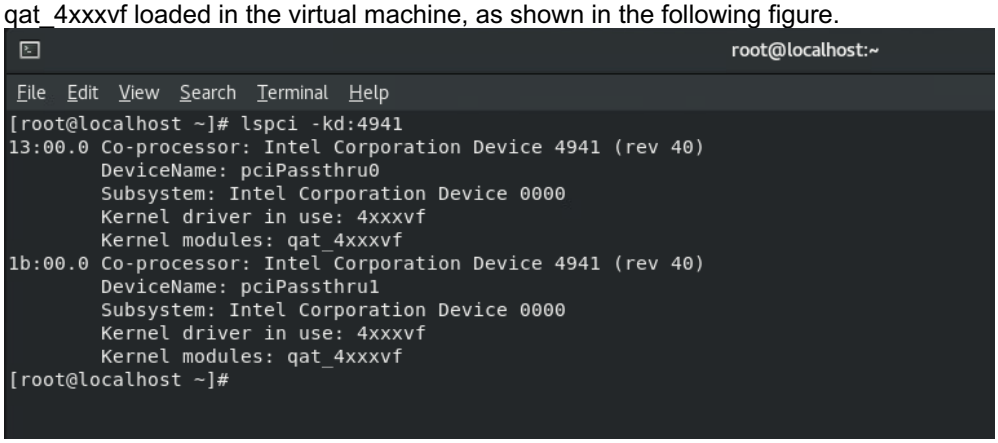

Figure 24. Display QAT device in guest VM

17. Check status of the two QAT endpoints (qat_dev0, qat_dev1) are “up” as shown below.

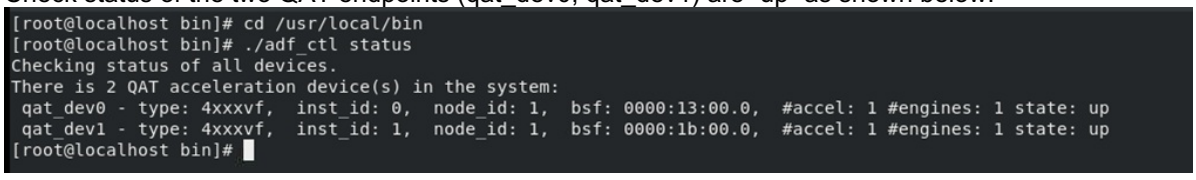

Figure 25. List QAT devices

18. Check that the QAT service is activated.

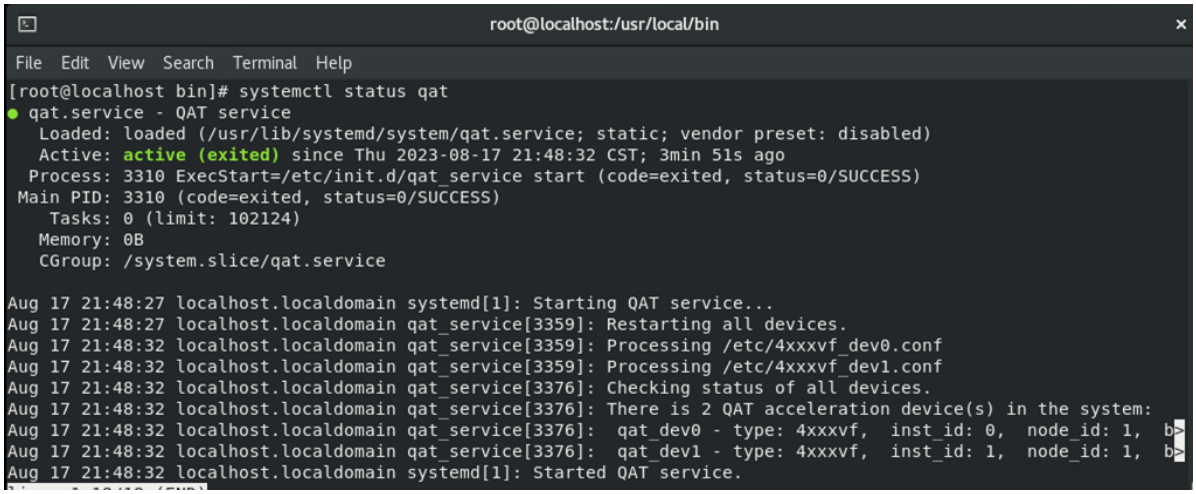

Figure 26. Check status of QAT service

Now that the drivers are installed, you can perform some tests.

The QAT software package contains a sample application to demonstrate accelerating crypto/compression operation. To run the sample code, execute the following commands:

```
~# cd /usr/local/bin
~# ./cpa_sample_code
```

```
[root@localhost bin]# ./cpa_sample_code
qaeMemInit started
icp_sal_userStartMultiProcess("SSL") started
*** QA version information ***
device ID               = 0
software                = 1.0.50
*** END QA version information ***
*** QA version information ***
device ID               = 1
software                = 1.0.50
*** END QA version information ***
Warning! Skipping SYMMETRIC tests as they are not supported on Instance
runTests=126
*** QA version information ***
device ID               = 0
software                = 1.0.50
*** END QA version information ***
*** QA version information ***
device ID               = 1
software                = 1.0.50
*** END QA version information ***
Inst 0, Affin: 1, Dev: 0, Accel 0, EE 0, BDF 13:00:00
Inst 1, Affin: 2, Dev: 0, Accel 0, EE 0, BDF 13:00:00
Inst 2, Affin: 1, Dev: 1, Accel 0, EE 0, BDF 1B:00:00
Inst 3, Affin: 2, Dev: 1, Accel 0, EE 0, BDF 1B:00:00
--------------------------------------
RSA CRT DECRYPT
Modulus Size                1024
Number of Threads     4
Total Submissions     400000
Total Responses       400000
Total Retries         5675832
CPU Frequency(kHz)    1900474
Operations per second    68148
--------------------------------------
```

Figure 27. QAT sample code running

By default, the sample code will run all tests (RAS test, DSA test, compression test, etc). During the application running, the result of each test is printed to the terminal window, as these two figures show. After all the tests have been executed, it will display the message “Sample code completed successfully” as shown in the figure below.

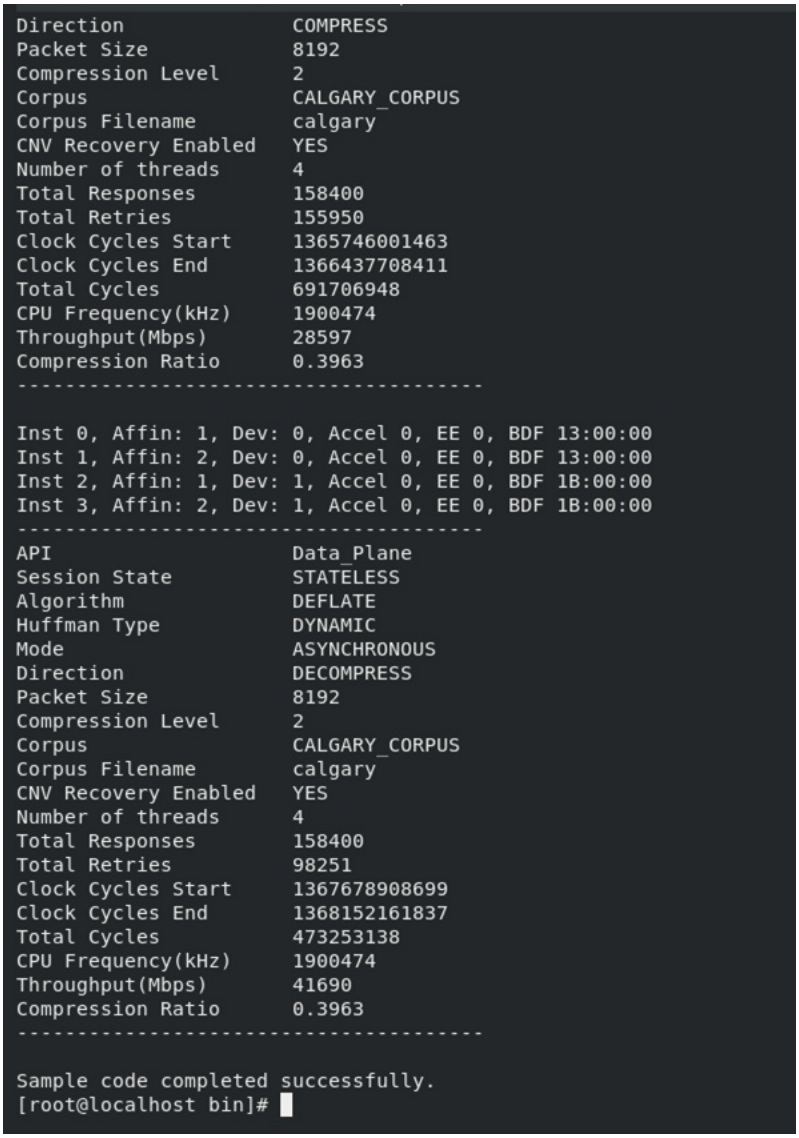

Figure 28. QAT sample code execution complete

References

For more information, see these resources:

- Intel Accelerator Engines overview
  https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/overview.html
- Intel DLB downloads
  https://www.intel.com/content/www/us/en/download/686372/intel-dynamic-load-balancer.html
- Intel QAT downloads
  https://www.intel.com/content/www/us/en/developer/topic-technology/open/quick-assist-technology/overview.html

Author

Alpus Chen is an OS Engineer at the Lenovo Infrastructure Solutions Group in Taipei, Taiwan. As a specialist in Linux and VMware technical support for several years, he is interested in operating system operation and recently focuses on VMware OS.

Thanks to the following specialists for their contributions and suggestions:

- Chengcheng Peng, Lenovo VMware Engineer
- Skyler Zhang, Lenovo VMware Engineer
- Gary Cudak, Lenovo OS Architect
- David Watts, Lenovo Press

Related product families

Product families related to this document are the following:

- Processors

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service. Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
8001 Development Drive
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary. Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk. Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document, LP1874, was created or updated on December 22, 2023.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP1874
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP1874.

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. A current list of Lenovo trademarks is available on the Web at https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:
Lenovo®
ThinkSystem®

The following terms are trademarks of other companies:

Intel® and Xeon® are trademarks of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the U.S. and other countries.

Other company, product, or service names may be trademarks or service marks of others.