Configuring Intel QuickAssist Technology in a Linux VM on Lenovo ThinkSystem Servers

Planning / Implementation

Intel® QuickAssist Technology (QAT) is a feature of Intel Xeon Scalable processors that is typically used to offload tasks from the server CPU, such as bulk crypto, compression, decompression and public key encryption.

For virtualization customers, QAT provides exclusive access to virtual machines using a method known as PCI device passthrough. This allows the PCIe QAT devices to be removed from the host and be assigned to individual guests.

However, with the large-scale deployment of virtual machines, cloud computing environment needs more and more QAT resources. Intel QuickAssist Technology 2.0 supports Single Root I/O Virtualization (SR-IOV) and Scalable I/O Virtualization (SIOV) to offer support for more guest operating systems. Starting with 4th Gen Intel Xeon Scalable processors, it is possible to divide a physical QAT device into multiple virtual devices and assign the virtual devices to multiple guest OSes. Conceptually, the guest OSes share the performance of a single physical QAT.

Hardware Architecture

The QAT hardware architecture in the Intel Xeon Scalable processor is illustrated in the figure below. The processor is comprised of multiple IO interfaces and CHA/core modules, and the modules are connected to a mesh interconnect. The mesh consists of horizontal and vertical interconnects. Each interconnect represents a bi-directional channel.

The 4th Gen Intel Xeon ("Sapphire Rapids") processors have 0, 1, 2 or 4 QAT accelerators, depending on the processor SKU. For the specifics, see the Processor features table in the SR630 V3 product guide: https://lenovopress.lenovo.com/lp1600-thinksystem-sr630-v3-server#processor-features

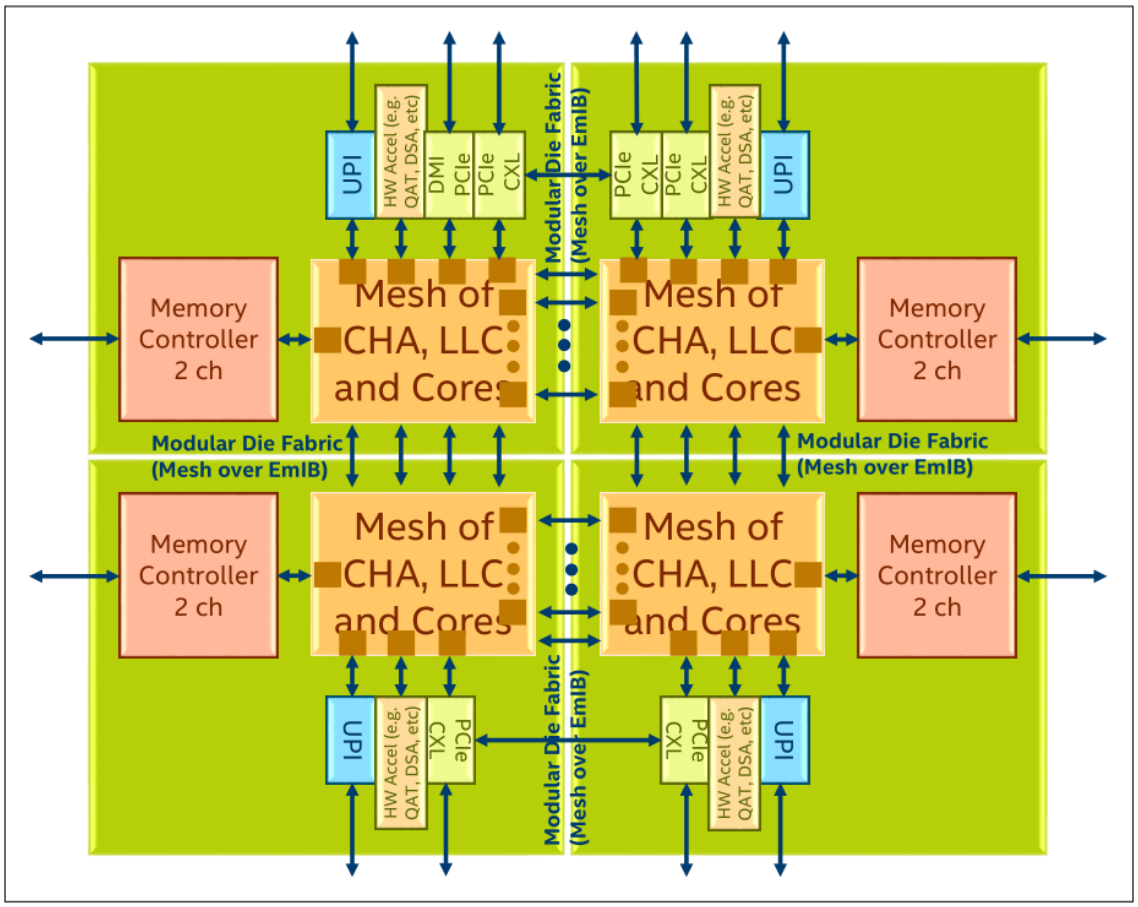


Figure 1. Intel QAT in Sapphire Rapids System Architecture (from Intel Document ID #611488: Sapphire Rapids Processor External Design Specification, Volume One: Architecture)

In the 4th Gen Intel Xeon Scalable processors ("Sapphire Rapids"), QAT is an accelerator integrated in root complex that is seen as a RCiEP (Root Complex Integrated Endpoint), which means the QAT is a PCIe device as well, the physical device ID is 0x4940, and the virtual function device ID is 0x4941.

Specifically for the 4th Gen processors, each processor has up to four QAT PCIe Endpoints in a single CPU package, and the hardware exposes one Physical Function (PF) per QAT PCIe Endpoint to the host. When SR-IOV is enabled, each Physical Function (PF) supports up to 16 Virtual Functions (VFs), one or more VFs can be passed to different guests/VMs and one or more PFs may be passed to a single virtual machine.

Virtualization Mode

Three different methods of virtualization are supported as shown in the following figure, Physical Device direct assignment, Single Root IOV (SR-IOV) and Scalable IOV.

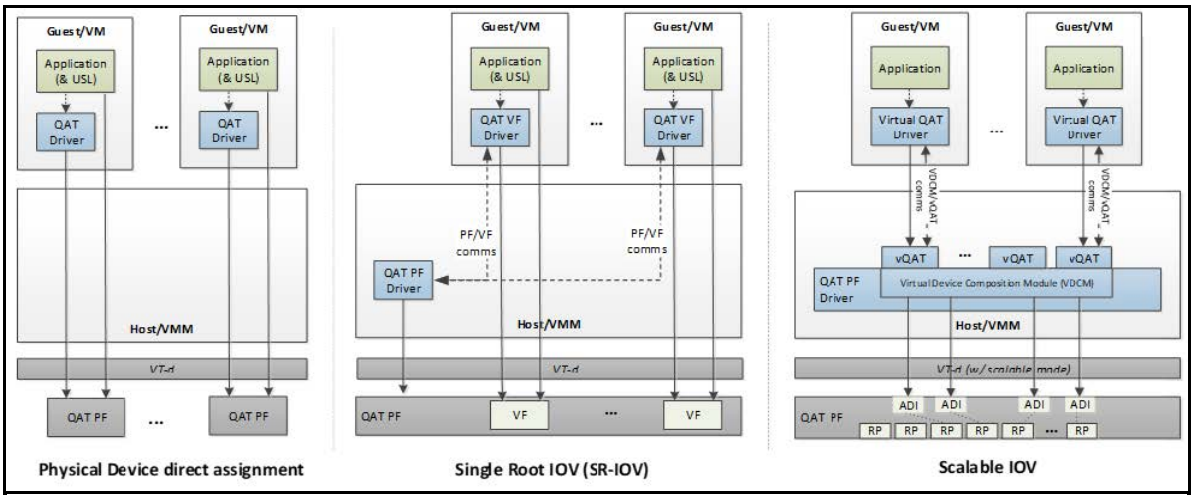


Figure 2. Virtualization Deployment Model for Intel® QAT 2.0 (from Intel Document ID #743912: Programmer's Guide Intel QuickAssist Technology Hardware Version 2.0)

With Physical Device direct assignment, shown in the left of Figure 2, QAT as PCIE device expose as Physical Function (PF), each QAT only has one PF, each PF can only be passed to one guest OS, one or more PFs may be passed to a single guest OS.

With Single Root IOV (SR-IOV), shown in the middle of Figure 2, each QAT device can exposes one PF and multiple Virtual Functions (VFs), the maximum VFs number is 16 in 4th Gen Intel Xeon Scalable processors. One or more VFs can be passed through to different guest OS.

Scalable IOV (SIOV), on the right, is under development. SIOV enables flexible composition of Virtual Functions by software from native hardware interfaces, rather than implementing a complete SR-IOV virtual function (VF) interface. SIOV devices expose light weight Assignable Device Interfaces (ADIs) that are optimized for fast-path (data-path) operations from the guest. SIOV uses PASID rather than BDF to identify unique address spaces, which supports better scalability. 4th Gen Intel Xeon Scalable processors have up to 64 ADIs per socket.

Software Architecture

The Intel QAT APIs establish the interface to access the QAT hardware. Thanks to the QAT APIs, the details of the hardware and software architecture are transparent to user applications. However, some knowledge of the underlying hardware and software is helpful for performance optimization and debugging purposes.

The brief view of software stack is shown in the figure below.

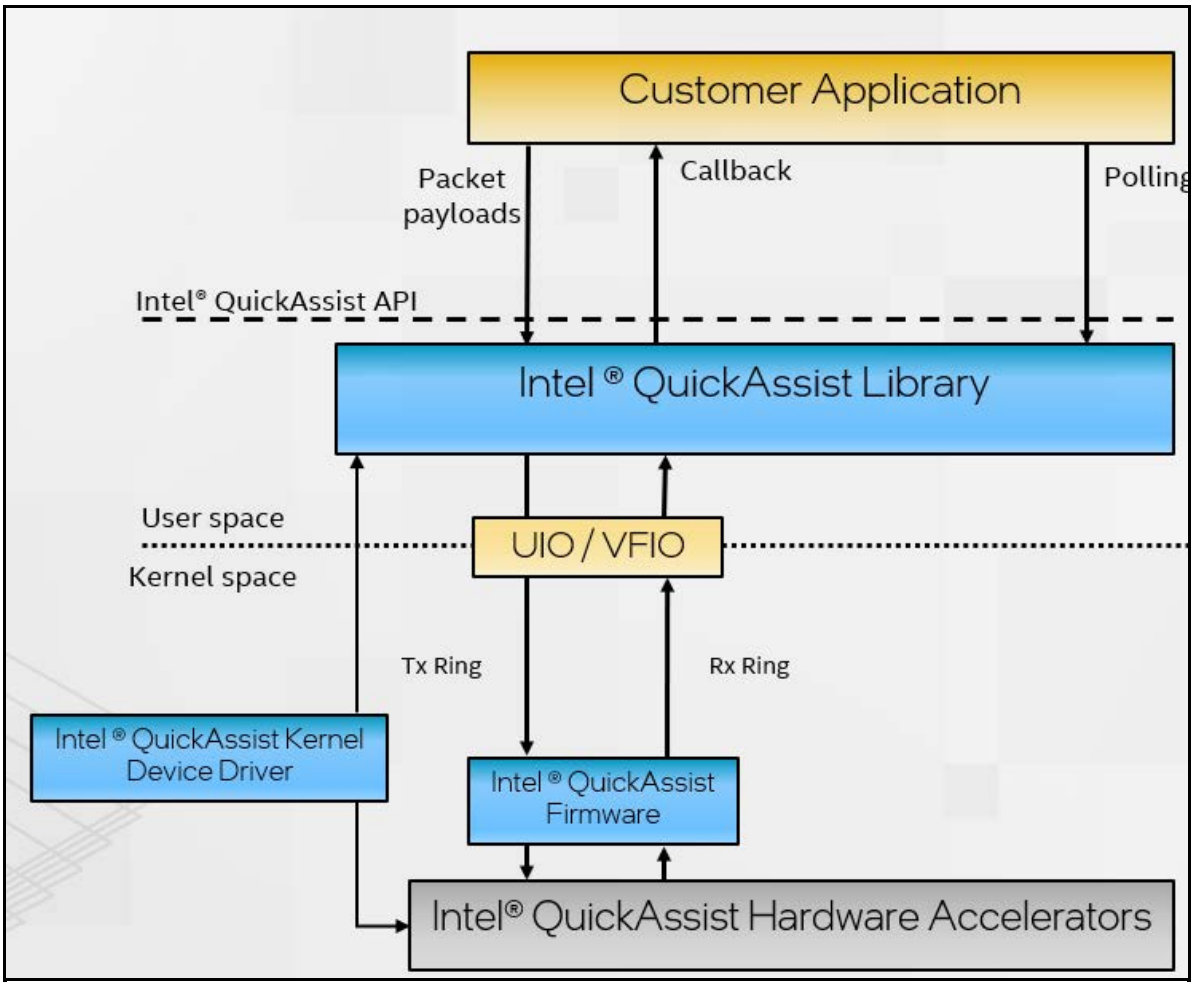


Figure 3. Intel QAT Software Architecture (from Intel Document ID #743912: Programmer's Guide Intel QuickAssist Technology Hardware Version 2.0)

In a nutshell, there are three processes to handle requests.

1. Applications submit payloads by the QuickAssist API as the part of the request. The QuickAssist Library in the user space helps convert these requests into descriptors and place them in the Tx Ring (Transmit Ring, also referred to as the hardware-assisted queue) for communication between CPU and QAT hardware.
2. QAT firmware parses the descriptors and configures the accelerators accordingly. Upon a job is completed, the QAT firmware returns the processed payload (encrypted or compressed, or both) and generates a response message that is inserted into the Rx Ring (Receive Ring).
3. There are two ways for an application to get results:
    - Application polls to query the Rx Ring via Intel QAT library
    - [Non-blocking application] Intel QAT library issues a callback to applications to inform that the operation is finished.

Setup QAT in UEFI

If you are using Virtual Function (VF) based on Intel QAT Physical Function (PF) device, make sure that the following BIOS settings are enabled on your server platform.

1. Enable QAT devices in BIOS setting.
2. Enable I/O Memory Management Unit (IOMMU) in BIOS setting

3. Enable Single Root I/O Virtualization (SR-IOV) in BIOS setting. The QAT supports SR-IOV residing in a PF, perform command `lspci -vn -d :4940|grep -i SR-IOV` to make sure the QAT supports SR-IOV.

IOMMU and SRIOV and QAT settings

I/O Memory Management Unit (IOMMU) is the generic name for Intel Virtualization Technology for Directed I/O (Intel VT-d). Intel QAT VFs are only available on hardware platforms supporting Intel VT-d. The Intel VT-d allows QAT VF to directly be assigned to a VM. The IOMMU must be enabled in the host UEFI.

Tip: Intel VT-d should not be confused with VT-x Intel Virtualization Technology. VT-x allows one hardware platform to function as multiple “virtual” platforms. However, VT-d improves security and reliability of the systems and improves performance of I/O devices in virtualized environments.

The steps to activate IOMMU are as follows:

1. In System Setup (F1 at boot), enter the UEFI System Configuration and Boot Management to enable the Intel VT for Directed I/O (VT-d).
2. From the BIOS setup menu path, select **System Settings → Devices and I/O ports → Intel VT for Directed I/O (VT-d)** to enable the IOMMU and SRIOV as shown in the following two figures.

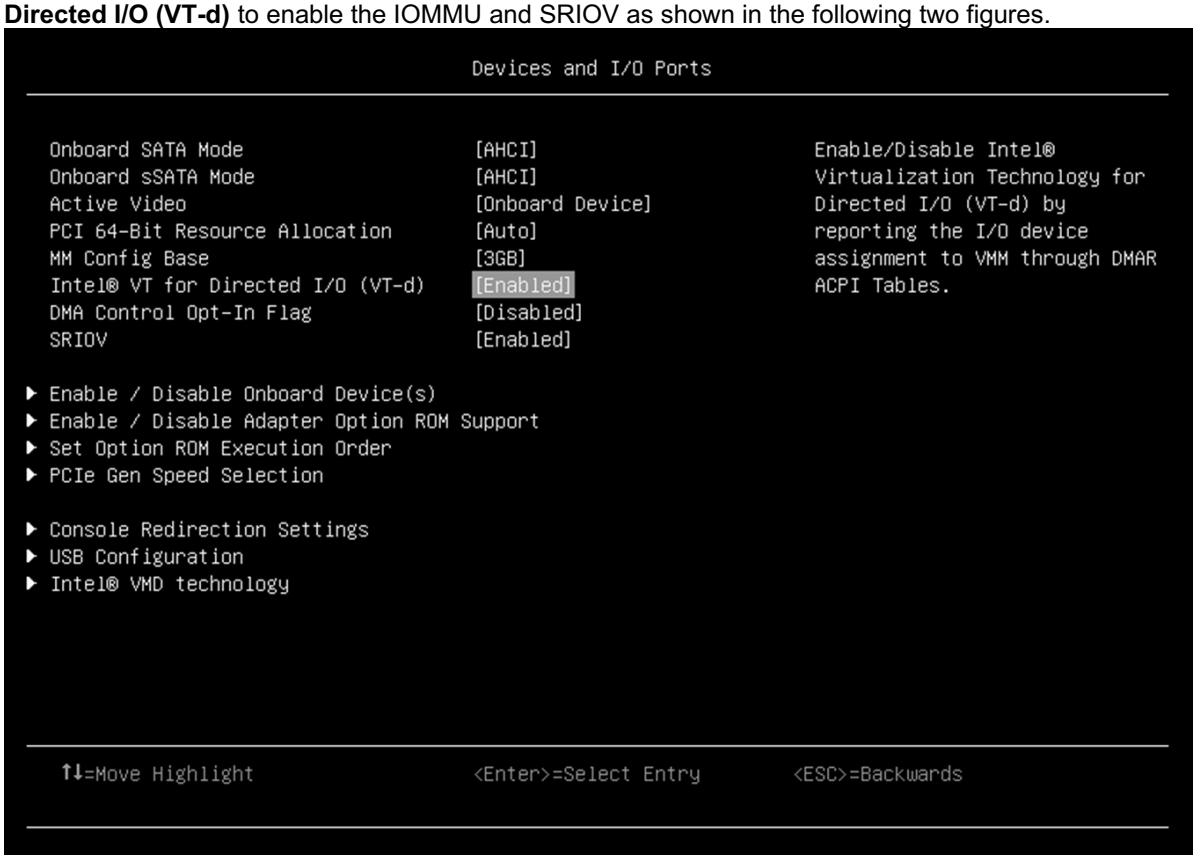


Figure 4. IOMMU in Devices and I/O Ports in System Setup (UEFI)

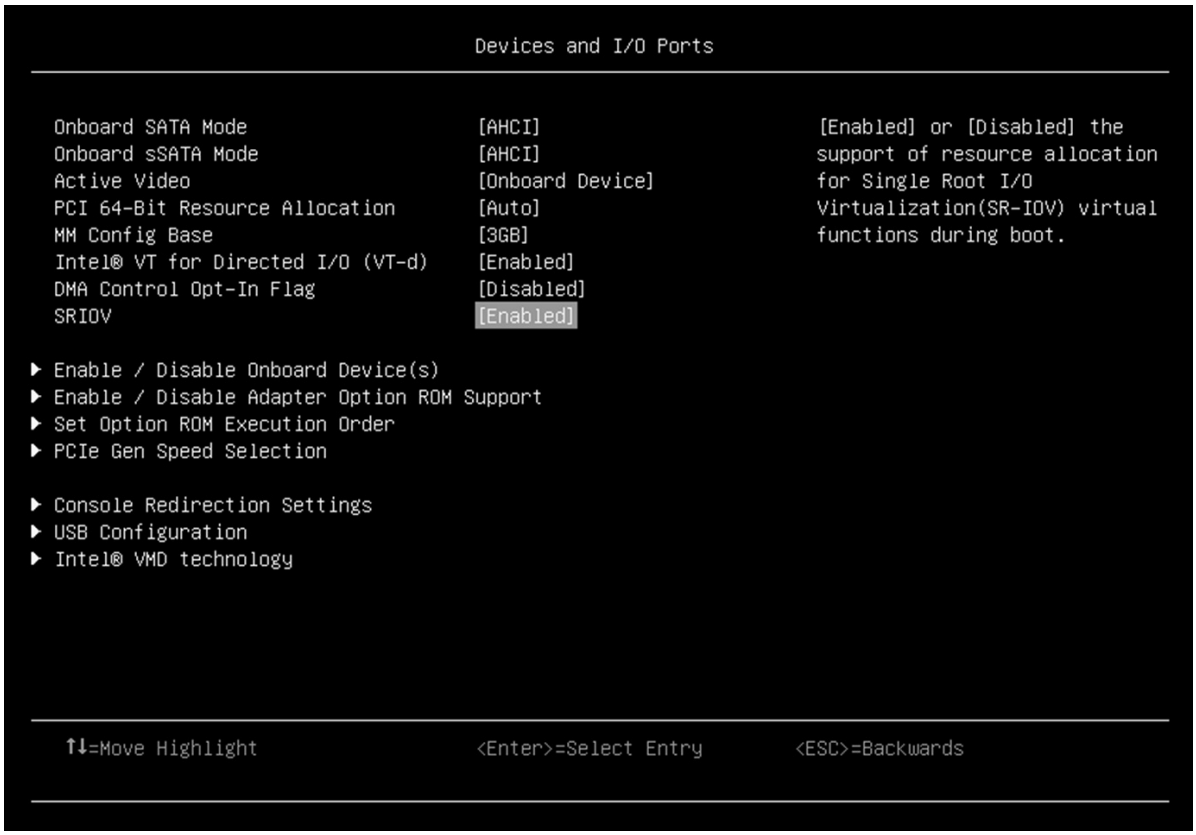
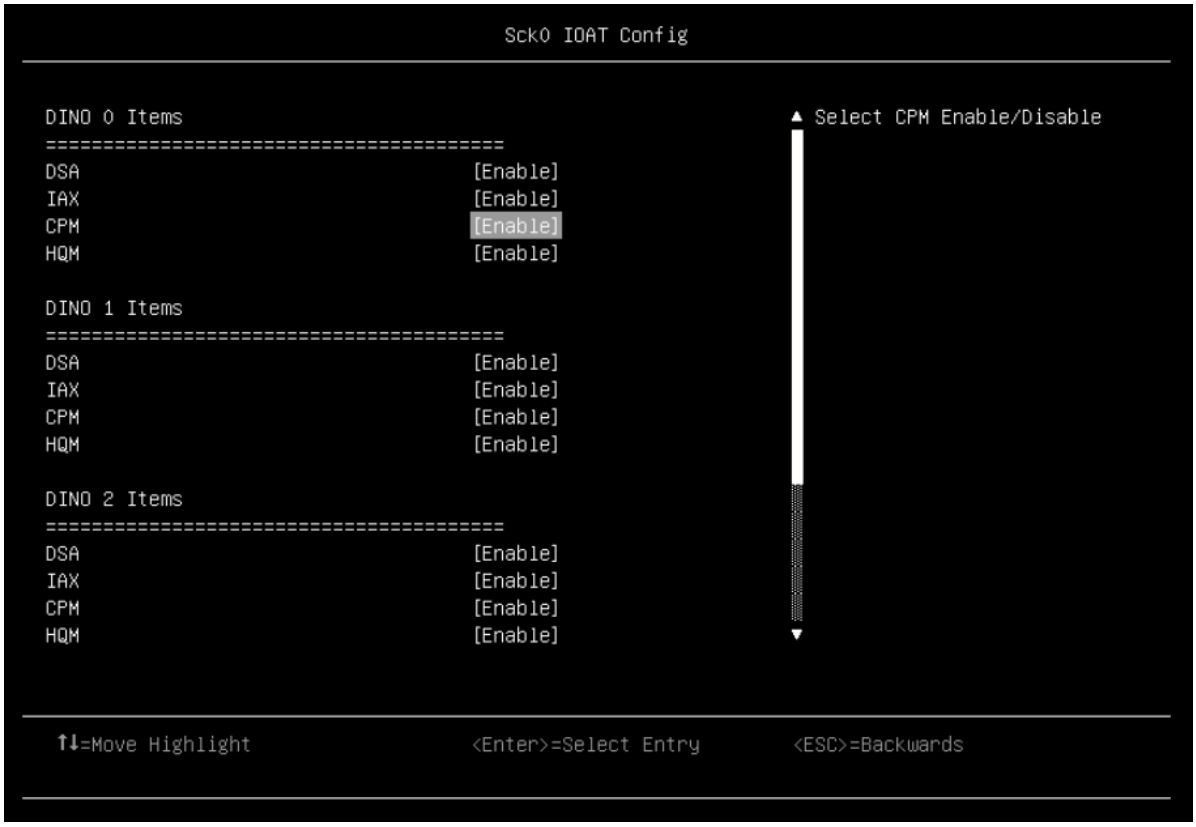


Figure 5. SRIOV in Devices and I/O Ports in System Setup (UEFI)

3. Enable QAT device in UEFI by navigating the following menu path in System Setup: **System Configuration and Boot Management > System Information > Socket Configuration > IIO Configuration > IOAT Configuration > Sock0 IOAT Config > CPM** as shown in the following figure.



4. After enabling Intel IOMMU, SRIOV, QAT, save and exit System Setup menu, and then boot the Linux OS.
5. Check that IOMMU is enabled. Boot up to OS and ensure that the IOMMU is enabled by the following command
   `dmesg|grep DMAR`

   If the output includes **DMAR: IOMMU enabled**, it means that system has enabled VT-d (Intel Virtualization Technology for Direct I/O) by reporting the I/O device assignment to VM through DMAR (DMA Remapping) ACPI table.
6. Check if system has QAT PF device using the following command:
   `lspci -d :4941`
7. Verify SR-IOV hardware capabilities. The QAT supports SR-IOV residing in a PF. Perform the following command to make sure the QAT supports SR-IOV capability:
   `lspci -vn -d :4940|grep -i SR-IOV`

   For the 4th Gen Intel Xeon Scalable processors (Sapphire Rapids), then QAT PF device ID is `0x4940`, the VF device ID is `0x4941`. The command should display one of the capabilities as shown below:

```
# lspci -vn -d :4940|grep -i SR-IOV
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [150] Single Root I/O Virtualization (SR-IOV)
```

Enable IOMMU host kernel support

Currently, most released operating systems, such as RHEL/SLES OSes, set IOMMU default to off (CONFIG_INTEL_IOMMU_DEFAULT_ON is not set). You will need to append the kernel parameter `intel_iommu=on` to the affected OSes.

To enable the I/O Memory Management Unit (IOMMU) on the host OS, follow these procedures:

1. Edit the host kernel boot command line

   For an Intel VT-d system, IOMMU is activated by adding the `intel_iommu=on` parameters to the kernel command line.

   Qemu parameter `aw-bit` determines the address width of IOVA address space. The address space has 39 bits width for 3-level IOMMU page tables, and 48 bits for 4-level IOMMU page tables, thus we also need to add `aw-bit=48` to kernel boot parameters.

   To enable these options, edit or add the GRUB_CMDLINX_LINUX line to the /etc/default/grub configuration file as follows:

   ```
   # cat /etc/default/grub
   GRUB_TIMEOUT=5
   GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
   GRUB_DEFAULT=saved
   GRUB_DISABLE_SUBMENU=true
   GRUB_TERMINAL_OUTPUT="console"
   GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.l
   v=rhel/root rd.lvm.lv=rhel/swap rhgb quiet intel_iommu=on aw-bits=48"
   GRUB_DISABLE_RECOVERY="true"
   GRUB_ENABLE_BLSCFG=true
   ```

2. Regenerate the grub2 config file To apply changes to the kernel command line, regenerate the boot loader configuration using the following command:
   `grub2-mkconfig`

   Check whether the changes are effective by using the following command:

   `grubby --info=0`

3. Reboot the host OS For the changes to take effect to kernel driver, reboot the host machine
4. Use the following command to confirm that IOMMU is enabled:
   `dmesg|grep iommu`

If successfull, you should see one of the following messages:

```
Adding to iommu group 0
iommu: Default domain type: Translated (set via kernel command line)
```

Download and install QAT firmware

In order to use QAT Virtual Function (VF) on a VM, make sure that the following are running in your host and guest OS:

- QAT Firmware (qat_4xxx.bin, qat_4xxx_mmp.bin)
- QAT kernel modules (intel_qat, 4xxx)
- User space library (qatlib)

The steps to download and install QAT firmware are as follows:

1. Download the firmware. At the time of writing, currently only available through early-access agreement with Intel.

   Once generally available, you will be able to download the QAT firmware at the following page (filename qat_4xxx.bin, qat_4xxx_mmp.bin)
   https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git/tree/

   Future RHEL OS releases will include it in package named “linux-firmware”.

2. On the host, copy the two firmware files to the /lib/firmware/ directory:
   - qat_4xxx.bin
   - qat_4xxx_mmp.bin
3. Reboot the system
4. Confirm the operation by running the following command
   `lspci -d :4940 -k`

   It should show the kernel module “qat_4xxx” is loaded and driver name “4xxx” is in use as following.

```
# lspci -d :4940 -k
6b:00.0 Co-processor: Intel Corporation Device 4940 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: 4xxx
        Kernel modules: qat_4xxx
70:00.0 Co-processor: Intel Corporation Device 4940 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: 4xxx
        Kernel modules: qat_4xxx
75:00.0 Co-processor: Intel Corporation Device 4940 (rev 40)
        Subsystem: Intel Corporation Device 0000
```

Installing and configuring Intel QAT software (qatlib)

The following sections detail the steps to use the virsh command line interface (CLI). The approach of libvirt* Virtual Machine Manager GUI has the similar steps to achieve it.

Topics in this section:

- Installing Intel qatlib on the host
- Enabling and Verifying VF on the host

- Attach QAT VF Device to the guest OS
- Installing qatlib on the guest OS

Installing Intel qatlib on the host

Intel QAT software can help enable QAT VF for all QAT PF devices in your system. In addition, QAT software also binds vfio-pci driver to every VF.

The steps to install qatlib are as follows:

1. Download the latest qatlib main source code from https://github.com/intel/qatlib, and copy it to your host.
2. Compile the installer as follows:

```
# cd qatlib-main
# ./autogen.sh
# ./configure –enable-service
# make -j4
# sudo make install
```

3. Use the following commands to ensure that the services is activated automatically for each boot:

```
# sudo systemctl enable qat
# sudo systemctl start qat
```

4. If the service is not configured to start automatically after rebooting the system, add the argument `--enable-service` when executing the command `./configure`. qat.service requires "qat" group when a non-root user starts it. If you don't use the root user to start qat.service, the command `configure --enable-service` can also automatically add a qat group in your OS.
5. If the above command prompt results in an error, you may be missing some of the required packages, including:
   - tool package: gcc, make, automake, autoconf, libtool, systemd-devel, yasm
   - library package: openssl-devel, zlib-devel

   If some packages aren't installed, you will receive an error. Use the error messages you get to determine the missing package(s) and use the following for additional information:

   - yasm not installed: Error "configure: error: yasm required".
   - automake not installed: The error is as following two lines:
     "Can't exec "aclocal": No such file or directory at /usr/share/autoconf/Autom4te/FileUtils.pm line 326.
     autoreconf: failed to run aclocal: No such file or directory"

   A good way to check which package is not installed on your OS by running the following shell script.

```
# export target_tools="gcc\nmake\nautomake\nautoconf\nlibtool\nsystemd-dev
el\nyasm\nopenssl-devel\nzlib-devel"
# export check_item="make|gcc|automake|autoconf|libtool|systemd-devel|open
ssl-devel|zlib-devel"
# echo -e $target_tools|grep -vE `rpm -qa|grep -Eowi $check_item |xargs |s
ed "s/ /|/g"`
# unset target_tools check_item
```

6. RHEL 9.1 does not include the yasm package, so manual installation is required, as follows:

```
# 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
```

7. Check “qat.service” as follows to see if it is activated. If the service is started, the text "active (running)" is displayed. If it is not started, you wil see "down" or "inactive".

```
# systemctl status qat
● qat.service - QAT service
     Loaded: loaded (/usr/lib/systemd/system/qat.service; enabled; vendor
preset: disabled)
     Active: active (running) since Tue 2023-02-14 21:19:28 EST; 12s ago
   Main PID: 105788 (qatmgr)
      Tasks: 1 (limit: 201748)
     Memory: 5.6M
        CPU: 5.477s
     CGroup: /system.slice/qat.service
             └─105788 /usr/local/sbin/qatmgr --policy=0

Feb 14 21:19:14 RHEL9.1 systemd[1]: Starting QAT service...
Feb 14 21:19:19 RHEL9.1 qatmgr[105788]: Open failed on /dev/vfio/358
Feb 14 21:19:19 RHEL9.1 qatmgr[105788]: Open failed on /dev/vfio/354
Feb 14 21:19:28 RHEL9.1 systemd[1]: Started QAT service.
```

8. Use the following commands to enable qat service and make persistent after reboot:
   `systemctl enable qat`
   `systemctl start qat`

Notes:

- It is not necessary to install qatlib in host OS if you don’t want to run sample code in host OS. It just takes advantage of the qatlib to enable VFs and bind vfio-pci driver here. Of course, you can also complete the options manually.
- The qat configuration script in the qatlib software package will automatically take care of certain build environment details, including copying the correct sample configuration files. If you are not using an included script to build and install the software, you must perform these operations yourself. For more information, refer to the INSTALL file in the source code you’ve downloaded.

Enabling and Verifying VF on the host

The QAT service creates VFs for all the QAT PFs and automatically binds the vfio-pci driver to all VFs.

Verify the VFs and the vfio-pci driver in use by VFs via running the following command in the host OS. The output below shows a 4th Gen Intel Xeon system and the output will have 16 or more VF devices, as shown, where the VF device ID is 4941.

```
# lspci -d :4941 -k
6b:00.1 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
6b:00.2 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
6b:00.3 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
6b:00.4 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
6b:00.5 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
```

If there aren't any VFs after performing the above command, run the following script to enable the VF devices. Note that the script must be run on every boot.

```
#for i in `lspci -D -d :4940| awk '{print $1}'`; do echo 16|sudo tee /sys/bus/p
ci/devices/$i/sriov_numvfs; done
```

Attach QAT VF Device to the guest OS

QAT supports the assignment of multiple VFs to a single VM. For those CPU-intensive workloads, it is recommended to add multiple VFs to a single VM for better performance.

To add a VF to a VM, make sure that the following prerequisites are met.

1. The Guest OS to which you want to add VFs is shut down. Use the command `virsh list --all` to check the domain status and command `virsh start/shutdown <domain name>` to start/shutdown a domain.

   ```
   # virsh list --all
    Id   Name      State
   --------------------------
    -    rhel9.1   shut off
   ```

2. The minimum version of libvirt needed is libvirt-8.10.0 for RHEL9. Older versions have boot VM bugs. The inbox libvirt-8.5.0-7 of the RHEL 9.1 GA must be updated higher than libvirt-8.10.0. The command listed below upgrades to libvirt-9.0.0-3 for example:

   `yum install libvirt-9.0.0-3.el9.x86_64`

   If you receive the error shown below when you boot a domain by adding two QAT VFs, you can try to update the libvirt version on your host OS.

```
# virsh start rhel9.1
internal error: qemu unexpectedly closed the monitor: 2023-02-13T03:21:40.
208989Z qemu-kvm: -device {"driver":"vfio-pci","host":"0000:6b:00.7","id":
"hostdev1","bus":"pci.9","addr":"0x0"}: VFIO_MAP_DMA failed: Cannot alloca
te memory#0122023-02-13T03:21:40.260532Z qemu-kvm: -device {"driver":"vfio
-pci","host":"0000:6b:00.7","id":"hostdev1","bus":"pci.9","addr":"0x0"}: V
FIO_MAP_DMA failed: Cannot allocate memory#0122023-02-13T03:21:40.260711Z
qemu-kvm: -device {"driver":"vfio-pci","host":"0000:6b:00.7","id":"hostdev
1","bus":"pci.9","addr":"0x0"}: vfio 0000:6b:00.7: failed to setup contain
er for group 358: memory listener initialization failed: Region pc.ram: vf
io_dma_map(0x5580b5255850, 0xc0000, 0x7ff40000, 0x7f88080bf000) = -12 (Can
not allocate memory)
```

To update the libvert version, perform the following command (to upgrade to libvirt-9.0.0-3):

```
yum install libvirt-9.0.0-3.el9.x86_64
```

To attach VFs to a guest OS, follow the instructions as follows:

1. Run the following command to edit the domain XML configuration file.
   `vrish edit`

   The domain name can be retrieved from the command `virsh list --all`, where the column name shows the domain name of your Guest OS.

2. Enable iommu and intel VT-d on the guest VM as follows:

   a. Edit the Guest VM config file

      `sudo virsh edit`

   b. Add the following entries
      - Under features heading add `<ioapic driver='qemu' />`. After update, it should look as follows:

        ```
        <features>
            <acpi/>
            <apic/>
            <ioapic driver='qemu'/>
        </features>
        ```

      - At the end of the devices heading, add the following (before the </devices> end tag):

        ```
        <iommu model='intel'>
          <driver intremap='on' caching_mode='on'/>
        </iommu>
        ```

3. Add the appropriate device XML entry to the <devices> sections in the XML configurations of guests that you want to get the VF resources. Add the following lines inside `<device> </device>` section.

```
# virsh edit rhel9.1
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
        <address domain='0x0000' bus='0x6b' slot='0x00' function='0x3'/>
      </source>
      <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function=
'0x0'/>
    </hostdev>
```

The form of an address element inside the source element should be included the hexadecimal digits of domain, bus, slot and function of the VF. The following command on the host OS will help you get hexadecimal digits:

```
lspci -Dd :4941
```

The line under </source> specifies the QAT VF domain, bus, slot, function number on your guest OS. Use the following command to get the last bus number in your guest OS, and then get the bus number plus one as your VF bus number in your guest OS:

```
lspci -D|tail -n 1
```

In our example, the last bus number gotten by `lspci -D|tail -n 1` is 0x07 before adding VFs to guest OS, so the VF attached to guest OS should be 0x07+1 (0x08).

The following is an example for adding two VFs to guest OS. It just adds another section same as the first one.

```
# virsh edit rhel9.1
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
        <address domain='0x0000' bus='0x6b' slot='0x00' function='0x3'/>
      </source>
      <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function=
'0x0'/>
    </hostdev>
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
        <address domain='0x0000' bus='0x6b' slot='0x00' function='0x7'/>
      </source>
      <address type='pci' domain='0x0000' bus='0x09' slot='0x00' function=
'0x0'/>
    </hostdev>
```

Installing qatlib on the guest OS

The installation of qatlib on a guest is similar to the process of installing it on the host. Refer to the Installing Intel qatlib on Host section for detailed steps.

After installing qatlib in your guest OS and starting qat service, you can issue the command `lspci -kd :4941` to check if the driver is used by QAT VFs. The output from the command should be as follows:

```
# lspci -kd :4941
08:00.0 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
09:00.0 Co-processor: Intel Corporation Device 4941 (rev 40)
        Subsystem: Intel Corporation Device 0000
        Kernel driver in use: vfio-pci
```

If the VFs can be activated by “vfio-pci” driver, it means that everything is OK.

The following example shows that two qat devices (“08:00:00” and “09:00:00”) are accelerating crypto/compression operations while running the sample code in qatlib.

```
# ./cpa_sample_code
qaeMemInit started
icp_sal_userStartMultiProcess("SSL") started
*** QA version information ***
device ID               = 0
software                = 22.7.0
*** END QA version information ***
*** QA version information ***
device ID               = 2
software                = 22.7.0
*** END QA version information **
Inst 0, Affin: 1, Dev: 0, Accel 0, EE 0, BDF 08:00:00
Inst 1, Affin: 3, Dev: 0, Accel 0, EE 0, BDF 08:00:00
Inst 2, Affin: 5, Dev: 1, Accel 0, EE 0, BDF 09:00:00
Inst 3, Affin: 7, Dev: 1, Accel 0, EE 0, BDF 09:00:00
```

For more information

For more information, see these resources:

- Github repository for QATlib:
  https://github.com/intel/qatlib/blob/main/INSTALL
- Red Hat article: Ensuring that Intel QuickAssist Technology stack is working correctly on RHEL
  https://access.redhat.com/articles/6376901
- Intel Document ID #611488: Sapphire Rapids Processor External Design Specification, Volume One: Architecture (login required):
  https://edc.intel.com/content/www/us/en/secure/design/confidential/products-and-solutions/processors-and-chipsets/eagle-stream/sapphire-rapids-server-processor-external-design-specification-volume-one-arch/2.0/overview-features-and-topologies/
- Intel Document ID #743912: Programmer’s Guide Intel QuickAssist Technology Hardware Version 2.0
  https://www.intel.com/content/www/us/en/content-details/743912/intel-quickassist-technology-intel-qat-software-for-linux-programmers-guide-hardware-version-2-0.html?wapkw=743912

Author

Xiaochun Li is a Linux engineer in the Lenovo Infrastructure Solutions Group based in Beijing, China. He specializes in development related to Linux kernel storage and memory management, as well as kernel DRM. Before joining Lenovo, he was an operating system engineer for INSPUR. With eight years of industry experience, he now focuses on Linux kernel RAS, storage, security and virtualization.

Related product families

Product families related to this document are the following:

- Processors
- Red Hat Enterprise Linux
- SUSE Linux Enterprise Server

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, LP1709, was created or updated on March 23, 2023.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP1709
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP1709.

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.