Configuring a Passthrough GPU in a Linux VM on a Lenovo ThinkSystem Server

Explains how to make a GPU available to a virtual machine

Provides step-by-step instructions on how to configure Passthrough in Linux

Covers both AMD and Intel platforms

Provides examples using an NVIDIA GPU in guest OS

Xiaochun Li

LENOVO PRESS

Abstract

Graphics Processing Units (GPUs) on Lenovo® ThinkSystem™ servers are typically used to offload tasks from the server CPU, such as AI, VDI, and rendering tasks. Customers who use a Linux virtual environment on their ThinkSystem server may want to assign the GPU to a virtual machine (VM), and thus allow the GPU to appear as if it was physically attached to the guest OS running in the VM. This functionality is called *GPU passthrough*.

This paper provides guidance on enabling GPU passthrough to a VM running in a Kernel Virtual Machine (KVM)-based OS. The paper is for Linux administrators wishing to use a GPU in a ThinkSystem server to pass through to a VM.

At Lenovo Press, we bring together experts to produce technical publications around topics of importance to you, providing information and best practices for using Lenovo products and solutions to solve IT challenges.

See a list of our most recent publications at the Lenovo Press web site:

http://lenovopress.com

Do you have the latest version? We update our papers from time to time, so check whether you have the latest version of this document by clicking the **Check for Updates** button on the front page of the PDF. Pressing this button will take you to a web page that will tell you if you are reading the latest version of the document and give you a link to the latest if needed. While you’re there, you can also sign up to get notified via email whenever we make an update.

Contents

Introduction

Many virtual machine administrators want to make a GPU installed in a server available to a single machine. The method known as PCI device passthrough allows the GPU PCIe device to be removed from the host and instead assigned to a single guest VM for exclusive access.

The paper describes the steps needed to implement the passthrough GPU:

1. “Enabling IOMMU in UEFI”
2. “Enabling IOMMU host kernel support” on page 5
3. “Unbinding the GPU device from host physical machine driver” on page 6
4. “Getting the GPU IOMMU configuration” on page 7
5. “Attaching a GPU device with virsh” on page 10
6. “Installing and enabling the NVIDIA driver in the guest OS” on page 12

Enabling IOMMU in UEFI

I/O Memory Management Unit (IOMMU) is the common name for Intel VT-d and AMD-Vi technologies. PCI device passthrough is only available on hardware platforms supporting either Intel VT-d or AMD-Vi. The Intel VT-d and AMD-Vi specifications provide hardware support for directly assigning a physical device to a VM.

The first step is to enable IOMMU in the ThinkSystem UEFI. The steps required for Intel and AMD processor-based ThinkSystem servers are listed in the following subsections.

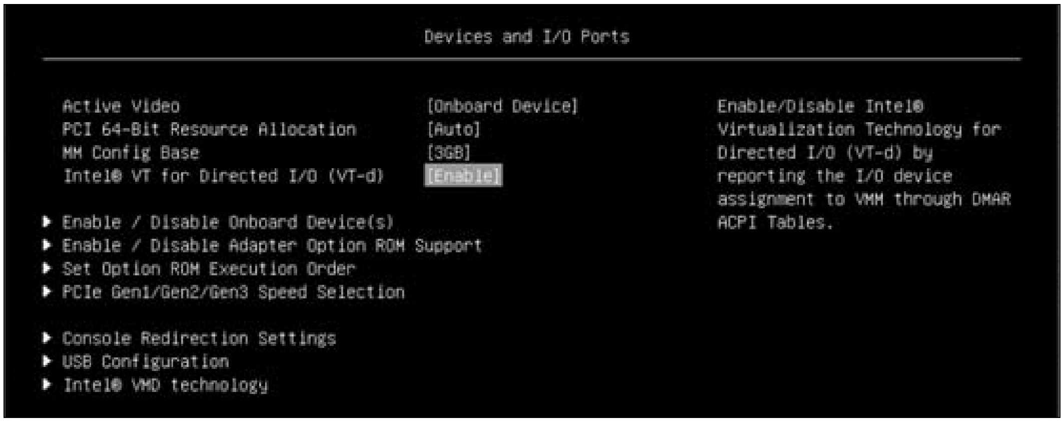
IOMMU settings on Intel system

VT-d stands for Intel Virtualization Technology for Directed I/O and 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 also improves performance of I/O devices in virtualized environments.

The steps to activate the Intel IOMMU on a server with an Intel processor are as follows:

1. Boot the server and when prompted, press F1 to enter System Setup.

2. From the UEFI menu, select **System Settings** → **Devices and I/O ports**, select **Intel VT for Directed I/O (VT-d)** and press Enter to enable the Intel IOMMU as shown in Figure 1.



```
                              Devices and I/O Ports

   Active Video                        [Onboard Device]      Enable/Disable Intel®
   PCI 64-Bit Resource Allocation      [Auto]                Virtualization Technology for
   MM Config Base                      [3GB]                 Directed I/O (VT-d) by
   Intel® VT for Directed I/O (VT-d)   [Enable]              reporting the I/O device
                                                             assignment to VMM through DMAR
 ▶ Enable / Disable Onboard Device(s)                        ACPI Tables.
 ▶ Enable / Disable Adapter Option ROM Support
 ▶ Set Option ROM Execution Order
 ▶ PCIe Gen1/Gen2/Gen3 Speed Selection

 ▶ Console Redirection Settings
 ▶ USB Configuration
 ▶ Intel® VMD technology
```

Figure 1   Devices and I/O Ports in Intel System Setup

3. Save and exit the BIOS setup menu, and then enter the Linux OS.
4. Boot up the OS and ensure the IOMMU is enabled with the following command

```
# dmesg|grep DMAR
DMAR: IOMMU enabled
```

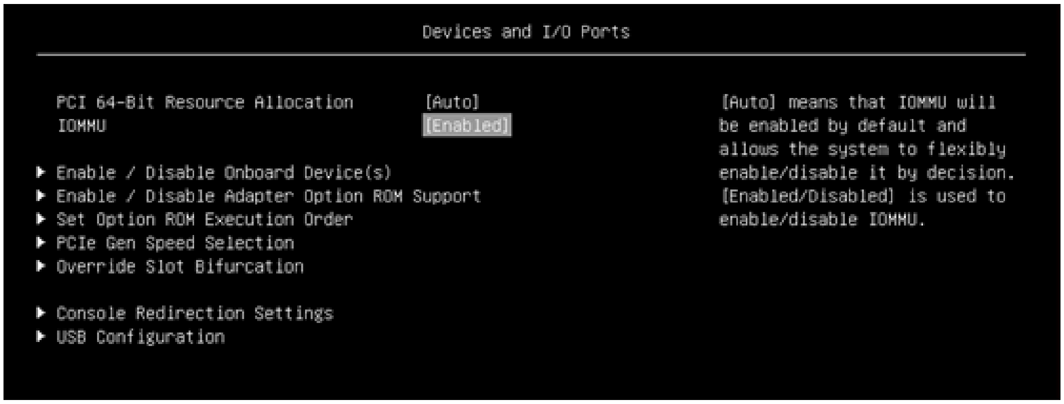
5. If you see `DMAR: IOMMU enabled`, it means that VT-d has been enabled by reporting the I/O device assignment to VMM through the DMAR (DMA Remapping) ACPI table.

IOMMU settings on AMD system

The AMD IOMMU specifications are required to use PCI device assignment in Linux OS. These specifications must be enabled in the BIOS.

The steps to activate the Intel IOMMU on a server with an AMD processor are as follows:

1. Boot the server and when prompted, press F1 to enter System Setup.
2. From the UEFI menu, select **System Settings** → **Devices and I/O ports**, highlight IOMMU and press Enter to enable the AMD IOMMU as shown in Figure 2.



```
                              Devices and I/O Ports

   PCI 64-Bit Resource Allocation      [Auto]                [Auto] means that IOMMU will
   IOMMU                               [Enabled]             be enabled by default and
                                                             allows the system to flexibly
 ▶ Enable / Disable Onboard Device(s)                        enable/disable it by decision.
 ▶ Enable / Disable Adapter Option ROM Support               [Enabled/Disabled] is used to
 ▶ Set Option ROM Execution Order                            enable/disable IOMMU.
 ▶ PCIe Gen Speed Selection
 ▶ Override Slot Bifurcation

 ▶ Console Redirection Settings
 ▶ USB Configuration
```

Figure 2   Devices and I/O Ports in AMD System Setup

3. Save and exit the BIOS setup menu, and then enter the Linux OS.

4. Boot up the OS and ensure the IOMMU is enabled by entering the following command:

```
# dmesg|grep AMD-Vi
AMD-Vi: Interrupt remapping enabled
```

5. If you see `AMD-Vi: Interrupt remapping enabled`, it means system has enabled AMD IOMMU.

Enabling IOMMU host kernel support

Currently, up to two GPUs may be attached to the virtual machine not including the standard emulated VGA interfaces. The emulated VGA is used for pre-boot and installation only; the NVIDIA GPU takes over once the NVIDIA graphics drivers are loaded.

To assign a GPU to a guest virtual machine, you must enable the IOMMU on the host machine, as described in the following procedure:

1. Edit the host kernel boot command line. For an Intel VT-d system, IOMMU is activated by adding the following parameters to the kernel command line:

   ```
   intel_iommu=on
   iommu=pt
   ```

   For an AMD-Vi system, the parameters needed are

   ```
   amd_iommu=on
   iommu=pt
   ```

   To enable this option, edit or add the GRUB_CMDLINX_LINUX line to the /etc/default/grub configuration file as shown in Figure 3 (Intel example).

```
# 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.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet intel_iommu=on iommu=pt"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
#
```

Figure 3 Changes to the grub configuration file (Intel example)

2. Regenerate the grub2 config file

   For the changes to the kernel command line to be applied, regenerate the boot loader configuration using the following command:

   ```
   # grub2-mkconfig
   ```

   You can verify the changes are effective with the following command:

   ```
   # grubby --info=0
   ```

3. Reboot the host OS

   For the changes to take effect to kernel driver, reboot the host machine and use the following command:

   ```
   # dmesg|grep iommu
   ```

   Look for one of the following lines in the output:

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

Unbinding the GPU device from host physical machine driver

For GPU passthrough, it is recommended to unbind the GPU device from host driver, as these drivers often do not support dynamic unbinding of the device. When using the Virtual Machine Manager interface to attach a GPU device, these steps also need to be performed if the GPU driver does not support dynamic unbinding.\

Steps to unbind the GPU device from the host driver are as follows:

1. Identify the GPU PCI bus address

   To identify the GPU PCI bus address and IDs of the device, run the command as listed in Figure 4. In our lab configuration, our server has the NVIDIA Tesla V100 GPU installed.

   ```
   # lspci -Dnn|grep -i NVIDIA
   0000:5b:00.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100
   PCIe 16GB] [10de:1db4] (rev a1)
   ```

   Figure 4 Identify the GPU PCI bus address

   The command reveals that the PCI bus address of this device is `0000:5b:00.0` and the PCI ID for the device is `10de:1db4`. The PCI bus address and device ID will be used in the following steps.

2. Prevent the native host machine driver from using the GPU device

   To prevent the native host machine driver from using the GPU device, you can use PCI ID with the pci-stub driver. To do this, append the following option to the GRUB_CMDLINX_LINUX line in the /etc/default/grub configuration file:

   ```
   pci-stub.ids=10de:1db4
   ```

where `10de:1db4` is the PCI ID for our GPU, as shown in Figure 5. To add additional PCI IDs for pci-stub, separate them with a comma.

```
# 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.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet intel_iommu=on iommu=pt
pci-stub.ids=10de:1db4"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
#
```

Figure 5 Add pci-stub.ids to the grub configuration file

3. Regenerate the grub2 config file

   For the changes to the kernel command line to be applied, regenerate the boot loader configuration using the following command:

   ```
   # grub2-mkconfig
   ```

   You can verify the changes are effective with the following command:

   ```
   # grubby --info=0
   ```

4. Reboot the host OS for the changes to take effect, using the following command

   ```
   # init 6
   ```

5. After the OS boot, run the command in Figure 6 to check if the GPU device is using vfio-pci driver instead of standard inbox (nouveau) driver.

```
# lspci -vvvnnn -s 0000:5b:00.0|grep -i "kernel driver in use"
        Kernel driver in use: vfio-pci
#
```

Figure 6 Verify the driver

Getting the GPU IOMMU configuration

Before attaching the GPU device, editing its IOMMU configuration is needed for the GPU to work properly on the guest. The steps are as follows.

1. List all PCI devices in the host machine

   Using the following command to list all devices of a particular type that are attached to the host machine.

   ```
   # virsh nodedev-list --cap pci
   ```

The output of the command is shown in Figure 7. Review the output for the string that maps to the GPU device you wish to enable for passthrough.

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_04_0
pci_0000_00_04_1
pci_0000_00_04_2
pci_0000_00_04_3
pci_0000_00_04_4
pci_0000_00_04_5
pci_0000_00_04_6
pci_0000_00_04_7
pci_0000_00_05_0
…
pci_0000_5b_00_0
pci_0000_ad_02_0
pci_0000_ad_05_0
pci_0000_ad_05_2
pci_0000_ad_05_4
…
#
```

Figure 7 Output from virsh command (partial output)

This example shows partial output info. The string that maps to the GPU with the `0000:5b:00.0` is `pci_0000_5b_00_0` (bolded in Figure 7). Note that the ‘:’ and ‘.’ characters are replaced with underscores in the libvirt-compatible identifier.

Record the PCI device number that maps to the GPU device you want to pass through to VM; this is required in the next steps.

2. Display the XML information of the GPU

   To display the settings of the GPU in XML form, it needs to use libvirt-compatible format PCI bus address. In this example, the GPU PCI device identifier is pci_0000_5b_00_0. Use the libvirt-compatible address of the GPU device with the **`virsh nodedev-dumpxml`** command to display its XML configuration as shown in Figure 8.

```
# virsh nodedev-dumpxml pci_0000_5b_00_0
<device>
  <name>pci_0000_5b_00_0</name>
  <path>/sys/devices/pci0000:5a/0000:5a:00.0/0000:5b:00.0</path>
  <parent>pci_0000_5a_00_0</parent>
  <driver>
    <name>vfio-pci</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>91</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x1db4'>GV100GL [Tesla V100 PCIe 16GB]</product>
    <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <iommuGroup number='31'>
      <address domain='0x0000' bus='0x5b' slot='0x00' function='0x0'/>
    </iommuGroup>
    <numa node='0'/>
    <pci-express>
      <link validity='cap' port='0' speed='8' width='16'/>
      <link validity='sta' speed='8' width='16'/>
    </pci-express>
  </capability>
</device>
#
```

Figure 8 Output from virsh nodedev-dumpxml command

   Note the `<iommuGroup>` element is an entry of the XML configuration (bolded in Figure 8). The iommuGroup indicates a set of devices that are considered isolated from other devices due to IOMMU capabilities and PCI bus topologies. All of the endpoint devices within the iommuGroup (meaning devices that are not PCIe root ports, bridges, or switch ports) need to be unbound from the native host drivers in order to be assigned to a guest OS. In the example above, the group is composed of the GPU device (`0000:5b:00.0`), and some GPU cards might have a companion audio device, such as (`0000:5b:00.1`).

3. Adjust IOMMU settings (optional)

   Note each IOMMU group may contain one or more devices. When multiple devices are present, all endpoints within the IOMMU group must be claimed for any device within the group to be assigned to a guest. This can be accomplished either by also assigning the extra endpoints to the guest or by detaching them from the host driver using the **`virsh nodedev-detach`** command.

Devices within an IOMMU group can be determined using the iommuGroup section of the **virsh nodedev-dumpxml** output. Each member of the group is provided in a separate `address` field. This information may also be found in sysfs using the command listed in Figure 9.

```
# ls /sys/bus/pci/devices/0000\:5b\:00.0/iommu_group/devices/
0000:5b:00.0 0000:5b:00.1
#
```

Figure 9 Viewing the information in sysfs

If a GPU card has a companion audio device (0000:5b:00.1), to assign only 0000.5b.00.0 to the guest, the unused endpoint device (0000:5b:00.1) should be detached from the host before starting the guest. The following two steps need to be performed:

a. Detect the PCI ID for the device and append it to the pci-stub.ids option in the /etc/default/grub file, as described in “Unbinding the GPU device from host physical machine driver” on page 6.

b. Use the **virsh nodedev-detach** command with a libvirt-compatible address as a parameter, for example:

```
# virsh nodedev-detach pci_0000_5b_00_1
```

Attaching a GPU device with virsh

The GPU can be attached to the guest using either of the following methods:

- Using the Virtual Machine Manager interface

  If device assignment fails, there may be other endpoints in the same IOMMU group that are still attached to the host. There is no way to retrieve group information using virt-manager, but virsh commands can be used to analyze the bounds of the IOMMU group.

- Creating XML configuration for the GPU and attaching it with the **virsh attach-device** command

The steps using the latter method, using **virsh attach-device**, are as follows:

1. From the output of step 2 on page 9, obtain the device values required for the configuration file. In our example, the device has the following values:

   – doman = `0x0000`
   – bus = `0x5b`
   – slot = `0x00`
   – function = `0x0`

   The configuration uses these three values.

2. Create an XML file for the GPU device. In the example, a file named GPU.xml is created and its content is as shown in Figure 10.

```
# cat GPU.xml
<hostdev mode='subsystem' type='pci' managed='yes'>
        <driver name='vfio'/>
        <source>
               <address domain='0x0000' bus='0x5b' slot='0x00' function='0x0'/>
        </source>
</hostdev>
#
```

Figure 10   Contents of file GPU.xml

3. Run the following command specifying the domain name you wish assign to and the XML file name you have created above.

   `virsh attach-device <domain name> <xml filename>`

   In the example in Figure 11, the domain name is `rhel8.2` and the XML filename is `GPU.xml`.

```
# virsh attach-device rhel8.2 GPU.xml
Device attached successfully

#
```

Figure 11   Attaching the GPU device

   The domain must be running before issuing the virsh attach-device command. Use the following commands to check the domain status or to start or shutdown the domain:

```
virsh list
virsh start <domain name>
virsh shutdown <domain name>
```

   The PCI device should now be successfully assigned to the virtual machine, and accessible to the guest operating system.

4. Login guest OS and run the command in Figure 12 to check GPU device in the guest OS. The GPU's PCI bus address on the guest will be different than on the host. In this example, the bus address is 07:00.0.

```
# lspci |grep -i nvidia
07:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 PCIe 16GB]
(rev a1)
#
```

Figure 12   Checking the GPU in the guest OS

Running `virsh attach-device <domain name> <xml file name>` just assigns GPU device to VM temporarily. After a reboot, the GPU is no longer attached. Appending the parameter `--persistent` persistently attaches to a guest OS. For example:

`virsh attach-device rhel8.2 GPU.xml --persistent`

In order to persistently attach GPU device to a guest OS, follow these steps:

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

   ```
   vrish edit <domain_name>’
   ```

2. Specify the domain name you wish to assign to.
3. Add the appropriate device XML entry in the `<source>` section to assign the PCI device to the guest manually.

The result is shown in Figure 13.

```
# virsh edit rhel8.2
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <driver name='vfio'/>
      <source>
        <address domain='0x0000' bus='0x5b' slot='0x00' function='0x0'/>
      </source>
      <address type='pci' domain='0x0000' bus='0x07' slot='0x00' function='0x0'/>
    </hostdev>
```

Figure 13 Editing the domain XML file

Installing and enabling the NVIDIA driver in the guest OS

This section describes how to enable a NVIDIA GPU from the Linux console. For GPU cards from other manufacturers, the steps may be slightly different. When using an assigned NVIDIA GPU in the guest OS, only the NVIDIA drivers are supported. Other drivers may not work well.

To install the NVIDIA driver based on RHEL7.x or RHEL8.x guest OS, perform the following steps:

1. Download the appropriate NVIDIA driver for your graphics controller from the NVIDIA web site, http://www.nvidia.com.
2. Ensure that this driver is saved in the local disk of the target system. Installing from an external device, such as a flash drive, will cause known issues such as an installation failure.
3. Run the commands listed below to install the NVIDIA driver. The driver cannot be installed if the X server is running on the system, so ensure that the system is started in text mode (runlevel 3).

   ```
   # init 3
   # sh nvidia_filename.run
   ```

4. Edit Grub 2 to blacklist the nouveau (inbox) driver. Edit /etc/default/grub and add the following parameter to the GRUB_CMDLINE_LINUX line.

   ```
   rd.driver.blacklist=nouveau nouveau.modeset=0
   ```

   This kernel parameter blacklists the nouveau driver module to disable it from getting loaded at boot from initramfs in guest OS.

5. Rebuild the grub.cfg file by running one of the following commands:

   ```
   # grub2-mkconfig
   ```

6. Edit the /etc/modprobe.d/blacklist.conf file and add the following line to it, so that the blacklist requirement is added into initramfs at rebuild:

   ```
   blacklist nouveau
   ```

7. Back up the current initramfs and build a new one as follows:

   ```
   # mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r)-nouveau.img
   # dracut /boot/initramfs-$(uname -r).img $(uname -r)
   ```

8. Restart the system.

The system should not load the nouveau module now at boot.

Before the above steps, the nouveau driver is in use as show in the command below:

```
# lspci -vvvnnn -s 07:00.0|grep -i "kernel driver in use"
        Kernel driver in use: nouveau
```

After the above steps, we can check the nvidia driver is in use by the following command:

```
# lspci -vvvnnn -s 07:00.0|grep -i "kernel driver in use"
        Kernel driver in use: nvidia
```

With this, the GPU is now available for exclusive use in the guest OS.

References

Use these references for more information

- X.org/XFree86 Video Timings HOWTO

  http://www.tldp.org/HOWTO/XFree86-Video-Timings-HOWTO/

- ArchWiki entry for PCI passthrough via OVMF

  https://wiki.archlinux.org/index.php/PCI_passthrough_via_OVMF

- RHEL 7 documentation: Virtualization Deployment and Administration Guide

  https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/

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.

Thanks to the following people for their contributions to this project:

- Yangyang Liang, Lenovo Test Engineer for Linux Enablement
- Adrian Huang, Lenovo OS Engineer
- Huaisheng Ye, Lenovo OS Engineer
- Gary Cudak, Lenovo OS Architect
- Paul Artman, Storage and I/O Architect
- JieJie Cheng, Technical Writer
- David Watts, Lenovo Press

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.
1009 Think Place - Building One
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 was created or updated on May 25, 2021.

Send us your comments via the **Rate & Provide Feedback** form found at
http://lenovopress.com/lp1234

Trademarks