PCI passthrough via OVMF

Related articles

- Intel GVT-g
- PCI passthrough via OVMF/Examples
- QEMU/Guest graphics acceleration

The Open Virtual Machine Firmware (OVMF (https://github.com/tianocore/tianocore.github.io/wiki/OVMF)) is a project to enable UEFI support for virtual machines. Starting with Linux 3.9 and recent versions of QEMU, it is now possible to passthrough a graphics card, offering the virtual machine native graphics performance which is useful for graphic-intensive tasks.

Provided you have a desktop computer with a spare GPU you can dedicate to the host (be it an integrated GPU or an old OEM card, the brands do not even need to match) and that your hardware supports it (see #Prerequisites), it is possible to have a virtual machine of any OS with its own dedicated GPU and near-native performance. For more information on techniques see the background presentation (pdf) (https://www.linux-kvm.org/images/b/b3/01x09b-VFIOandYou-small.pdf).

1 Prerequisites

A VGA Passthrough relies on a number of technologies that are not ubiquitous as of today and might not be available on your hardware. You will not be able to do this on your machine unless the following requirements are met:

- Your CPU must support hardware virtualization (for kvm) and IOMMU (for the passthrough itself).
  - List of compatible Intel CPUs (Intel VT-x and Intel VT-d) (https://ark.intel.com/content/www/us/en/ark/search/featurefilter.html?productType=873&0_VTD=True&2_VTX=true).
  - All AMD CPUs from the Bulldozer generation and up (including Zen) should be compatible.
    - CPUs from the K10 generation (2007) do not have an IOMMU, so you **need** to have a motherboard with a 890FX (https://support.amd.com/TechDocs/43403.pdf#page=18) or 990FX (https://support.amd.com/TechDocs/48691.pdf#page=21) chipset to make it work, as those have their own IOMMU.
- Your motherboard must also support IOMMU.
  - Both the chipset and the BIOS must support it. It is not always easy to tell at a glance whether or not this is the case, but there is a fairly comprehensive list on the matter on the Xen wiki (https://wiki.xen.org/wiki/VTd_HowTo) as well as Wikipedia:List of IOMMU-supporting hardware.
- Your guest GPU ROM must support UEFI.
  - If you can find any ROM in this list (https://www.techpowerup.com/vgabios/) that applies to your specific GPU and is said to support UEFI, you are generally in the clear. All GPUs from 2012 and later should support this, as Microsoft made UEFI a requirement for devices to be marketed as compatible with Windows 8.

You will probably want to have a spare monitor or one with multiple input ports connected to different GPUs (the passthrough GPU will not display anything if there is no screen plugged in and using a VNC or Spice connection will not help your performance), as well as a mouse and a keyboard you can pass to your virtual machine. If anything goes wrong, you will at least have a way to control your host machine this way.

Tip: You can do steps 1-3 with gpu-passthrough-manager (https://aur.archlinux.org/packages/gpu-passthrough-manager/)AUR. This is a simple graphical interface that lets you choose what graphics drivers you want to load on your system. This can also set up IOMMU and adds VFIO modules, but is limited to GRUB only. Select the graphics and audio devices you want to passthrough and then select LOAD VFIO. Reboot when prompted and then continue to step 4.

2 Setting up IOMMU

Note:

- IOMMU is a generic name for Intel VT-d and AMD-Vi.
- 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 while VT-d improves security and reliability of the systems and also improves performance of I/O devices in virtualized environments.

Using IOMMU opens to features like PCI passthrough and memory protection from faulty or malicious devices, see Wikipedia:Input-output memory management unit#Advantages and Memory Management (computer programming): Could you explain IOMMU in plain English? (https://www.quora.com/Memory-Management-computer-programming/Could-you-explain-IOMMU-in-plain-English).

2.1 Enabling IOMMU

Ensure that AMD-Vi/Intel VT-d is supported by the CPU and enabled in the BIOS settings. Both normally show up alongside other CPU features (meaning they could be in an overclocking-related menu) either with their actual names ("VT-d" or "AMD-Vi") or in more ambiguous terms such as "Virtualization technology", which may or may not be explained in the manual.

Manually enable IOMMU support by setting the correct kernel parameter depending on the type of CPU in use:

- For Intel CPUs (VT-d) set `intel_iommu=on`. Since the kernel config option CONFIG_INTEL_IOMMU_DEFAULT_ON is not set in linux (https://archlinux.org/packages/?name=linux).
- For AMD CPUs (AMD-Vi), it is on if kernel detects IOMMU hardware support from BIOS.

You should also append the `iommu=pt` parameter. This will prevent Linux from touching devices which cannot be passed through.

After rebooting, check dmesg to confirm that IOMMU has been correctly enabled:

```
# dmesg | grep -i -e DMAR -e IOMMU
```

```
[    0.000000] ACPI: DMAR 0x00000000BDCB1CB0 0000B8 (v01 INTEL  BDW      00000001 INTL 00000001)
[    0.000000] Intel-IOMMU: enabled
[    0.028879] dmar: IOMMU 0: reg_base_addr fed90000 ver 1:0 cap c0000020660462 ecap f0101a
[    0.028883] dmar: IOMMU 1: reg_base_addr fed91000 ver 1:0 cap d2008c20660462 ecap f010da
[    0.028950] IOAPIC id 8 under DRHD base  0xfed91000 IOMMU 1
[    0.536212] DMAR: No ATSR found
[    0.536229] IOMMU 0 0xfed90000: using Queued invalidation
[    0.536230] IOMMU 1 0xfed91000: using Queued invalidation
[    0.536231] IOMMU: Setting RMRR:
[    0.536241] IOMMU: Setting identity map for device 0000:00:02.0 [0xbf000000 - 0xcf1fffff]
[    0.537490] IOMMU: Setting identity map for device 0000:00:14.0 [0xbdea8000 - 0xbdeb6fff]
[    0.537512] IOMMU: Setting identity map for device 0000:00:1a.0 [0xbdea8000 - 0xbdeb6fff]
[    0.537530] IOMMU: Setting identity map for device 0000:00:1d.0 [0xbdea8000 - 0xbdeb6fff]
[    0.537543] IOMMU: Prepare 0-16MiB unity mapping for LPC
[    0.537549] IOMMU: Setting identity map for device 0000:00:1f.0 [0x0 - 0xffffff]
[    2.182790] [drm] DMAR active, disabling use of stolen memory
```

2.2 Ensuring that the groups are valid

The following script should allow you to see how your various PCI devices are mapped to IOMMU groups. If it does not return anything, you either have not enabled IOMMU support properly or your hardware does not support it.

```
#!/bin/bash
shopt -s nullglob
for g in $(find /sys/kernel/iommu_groups/* -maxdepth 0 -type d | sort -V); do
    echo "IOMMU Group ${g##*/}:"
    for d in $g/devices/*; do
        echo -e "\t$(lspci -nns ${d##*/})"
    done;
done;
```

Example output:

```
IOMMU Group 1:
        00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI Express Root Por
t [8086:0151] (rev 09)
IOMMU Group 2:
        00:14.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Host Contr
oller [8086:0e31] (rev 04)
IOMMU Group 4:
        00:1a.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced Host C
ontroller #2 [8086:0e2d] (rev 04)
IOMMU Group 10:
        00:1d.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced Host C
ontroller #1 [8086:0e26] (rev 04)
IOMMU Group 13:
        06:00.0 VGA compatible controller: NVIDIA Corporation GM204 [GeForce GTX 970] [10de:13c2] (rev a
1)
        06:00.1 Audio device: NVIDIA Corporation GM204 High Definition Audio Controller [10de:0fbb] (rev
a1)
```

An IOMMU group is the smallest set of physical devices that can be passed to a virtual machine. For instance, in the example above, both the GPU in 06:00.0 and its audio controller in 6:00.1 belong to IOMMU group 13 and can only be passed together. The frontal USB controller, however, has its own group (group 2) which is separate from both the USB expansion controller (group 10) and the rear USB controller (group 4), meaning that **any of them could be passed to a virtual machine without affecting the others**.

2.3 Gotchas

2.3.1 Plugging your guest GPU in an unisolated CPU-based PCIe slot

Not all PCI-E slots are the same. Most motherboards have PCIe slots provided by both the CPU and the PCH. Depending on your CPU, it is possible that your processor-based PCIe slot does not support isolation properly, in which case the PCI slot itself will appear to be grouped with the device that is connected to it.

```
IOMMU Group 1:
        00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI Express Root Po
rt (rev 09)
        01:00.0 VGA compatible controller: NVIDIA Corporation GM107 [GeForce GTX 750] (rev a2)
        01:00.1 Audio device: NVIDIA Corporation Device 0fbc (rev a1)
```

This is fine so long as only your guest GPU is included in here, such as above. Depending on what is plugged in to your other PCIe slots and whether they are allocated to your CPU or your PCH, you may find yourself with additional devices within the same group, which would force you to pass those as well. If you are ok with passing everything that is in there to your virtual machine, you are free to continue. Otherwise, you will either need to try and plug your GPU in your other PCIe slots (if you have any) and see if those provide isolation from the rest or to install the ACS override patch, which comes with its own drawbacks. See **#Bypassing the IOMMU groups (ACS override patch)** for more information.

Note: If they are grouped with other devices in this manner, pci root ports and bridges should neither be bound to vfio at boot, nor be added to the virtual machine.

3 Isolating the GPU

In order to assign a device to a virtual machine, this device and all those sharing the same IOMMU group must have their driver replaced by a stub driver or a VFIO driver in order to prevent the host machine from interacting with them. In the case of most devices, this can be done on the fly right before the virtual machine starts.

However, due to their size and complexity, GPU drivers do not tend to support dynamic rebinding very well, so you cannot just have some GPU you use on the host be transparently passed to a virtual machine without having both drivers conflict with each other. Because of this, it is generally advised to bind those placeholder drivers manually before starting the virtual machine, in order to stop other drivers from attempting to claim it.

The following section details how to configure a GPU so those placeholder drivers are bound early during the boot process, which makes said device inactive until a virtual machine claims it or the driver is switched back. This is the preferred method, considering it has less caveats than switching drivers once the system is fully online.

Warning: Once you reboot after this procedure, whatever GPU you have configured will no longer be usable on the host until you reverse the manipulation. Make sure the GPU you intend to use on the host is properly configured before doing this - your motherboard should be set to display using the host GPU.

Starting with Linux 4.1, the kernel includes vfio-pci. This is a VFIO driver, meaning it fulfills the same role as pci-stub did, but it can also control devices to an extent, such as by switching them into their D3 state when they are not in use.

3.1 Binding vfio-pci via device ID

Vfio-pci normally targets PCI devices by ID, meaning you only need to specify the IDs of the devices you intend to passthrough. For the following IOMMU group, you would want to bind vfio-pci with `10de:13c2` and `10de:0fbb`, which will be used as example values for the rest of this section.

```
IOMMU Group 13:
        06:00.0 VGA compatible controller: NVIDIA Corporation GM204 [GeForce GTX 970] [10de:13c2] (rev a
1)
```

```
        06:00.1 Audio device: NVIDIA Corporation GM204 High Definition Audio Controller [10de:0fbb] (rev
a1)}}
```

Note:

- You cannot specify which device to isolate using vendor-device ID pairs if the host GPU and the guest GPU share the same pair (i.e : if both are the same model). If this is your case, read **#Using identical guest and host GPUs** instead.
- If, as noted in **#Plugging your guest GPU in an unisolated CPU-based PCIe slot**, your pci root port is part of your IOMMU group, you **must not** pass its ID to `vfio-pci`, as it needs to remain attached to the host to function properly. Any other device within that group, however, should be left for `vfio-pci` to bind with.
- Binding the audio device (`10de:0fbb` in above's example) is optional. Libvirt is able to unbind it from the `snd_hda_intel` driver on its own.

Two methods exist for providing the device IDs. Specifying them via **kernel parameters** has the advantage of being able to easily edit, remove, or undo any breaking changes via your boot loader:

```
vfio-pci.ids=10de:13c2,10de:0fbb
```

Alternatively, the IDs may be added to a modprobe conf file. Since these conf files are embedded in the initramfs image, any changes require regenerating a new image each time:

```
/etc/modprobe.d/vfio.conf
```

```
options vfio-pci ids=10de:13c2,10de:0fbb
```

3.2 Loading vfio-pci early

Since Arch's **linux (https://archlinux.org/packages/?name=linux)** has vfio-pci built as a module, we need to force it to load before the graphics drivers have a chance to bind to the card. There are two methods: modprobe configuration, or adding the modules to initramfs.

Note: When troubleshooting this step, to make sure all other conditions are fulfilled, verify the initramfs method works, since it is the stronger guarantee for isolating the card.

3.2.1 modprobe.d

This method loads `vfio` when udev loads GPU drivers. This avoids bloating initramfs and slowing boot times unnecessarily.

```
/etc/modprobe.d/vfio.conf
```

```
softdep drm pre: vfio-pci
```

3.2.2 initramfs

Warning: Since kernel 6.0, framebuffer output will freeze after loading VFIO modules and until GPU drivers are loaded. When using disk encryption, this will prevent the password prompt from showing.

Make sure to add GPU drivers to initramfs when choosing this method. See this forum thread (https://bbs.archlinux.org/viewtopic.php?pid=2070655#p2070655).

3.2.2.1 mkinitcpio

Add `vfio_pci`, `vfio`, and `vfio_iommu_type1` to mkinitcpio:

```
/etc/mkinitcpio.conf
```
```
MODULES=(... vfio_pci vfio vfio_iommu_type1 ...)
```

Note:

- As of kernel 6.2, the `vfio_virqfd` functionality has been folded into the base `vfio` module.
- If you also have another driver loaded this way for **early modesetting** (such as `nouveau`, `radeon`, `amdgpu`, `i915`, etc.), all of the aforementioned VFIO modules must precede it.
- If you are modesetting the `nvidia` driver, the `vfio-pci.ids` must be embedded in the initramfs image. If given via kernel arguments, they will be read too late to take effect. Follow the instructions in **#Binding vfio-pci via device ID** for adding the ids to a modprobe conf file.

Also, ensure that the modconf hook is included in the HOOKS list of `mkinitcpio.conf`:

```
/etc/mkinitcpio.conf
```
```
HOOKS=(... modconf ...)
```

Since new modules have been added to the initramfs configuration, you must regenerate the initramfs.

3.2.2.2 booster

Similar to mkinitcpio you need to specify modules to load early:

```
/etc/booster.yaml
```
```
modules_force_load: vfio_pci,vfio,vfio_iommu_type1
```

and then regenerate the initramfs.

3.2.2.3 dracut

Following the same idea, we need to ensure all vfio drivers are in the initramfs. Add the following file to `/etc/dracut.conf.d`:

```
10-vfio.conf
```
```
force_drivers+=" vfio_pci vfio vfio_iommu_type1 "
```

Note that we used `force_drivers` instead the usual `add_drivers` option, which will ensure that the drivers are tried to be loaded early via modprobe (Dracut#Early kernel module loading).

As with mkinitcpio, you must regenerate the initramfs afterwards. See dracut for more details.

3.3 Verifying that the configuration worked

Reboot and verify that vfio-pci has loaded properly and that it is now bound to the right devices.

```
# dmesg | grep -i vfio

[    0.329224] VFIO - User Level meta-driver version: 0.3
[    0.341372] vfio_pci: add [10de:13c2[ffff:ffff]] class 0x000000/00000000
[    0.354704] vfio_pci: add [10de:0fbb[ffff:ffff]] class 0x000000/00000000
[    2.061326] vfio-pci 0000:06:00.0: enabling device (0100 -> 0103)
```

It is not necessary for all devices (or even expected device) from `vfio.conf` to be in *dmesg* output. Even if a device does not appear, it might still be visible and usable in the guest virtual machine.

```
$ lspci -nnk -d 10de:13c2

06:00.0 VGA compatible controller: NVIDIA Corporation GM204 [GeForce GTX 970] [10de:13c2] (rev a1)
        Kernel driver in use: vfio-pci
        Kernel modules: nouveau nvidia
```

```
$ lspci -nnk -d 10de:0fbb

06:00.1 Audio device: NVIDIA Corporation GM204 High Definition Audio Controller [10de:0fbb] (rev a1)
        Kernel driver in use: vfio-pci
        Kernel modules: snd_hda_intel
```

4 Setting up an OVMF-based guest virtual machine

OVMF is an open-source UEFI firmware for QEMU virtual machines. While it is possible to use SeaBIOS to get similar results to an actual PCI passthrough, the setup process is different and it is generally preferable to use the EFI method if your hardware supports it.

4.1 Configuring libvirt

Libvirt is a wrapper for a number of virtualization utilities that greatly simplifies the configuration and deployment process of virtual machines. In the case of KVM and QEMU, the frontend it provides allows us to avoid dealing with the permissions for QEMU and make it easier to add and remove various devices on a live virtual machine. Its status as a wrapper, however, means that it might not always support all of the latest qemu features, which could end up requiring the use of a wrapper script to provide some extra arguments to QEMU.

Install **qemu-desktop (https://archlinux.org/packages/?name=qemu-desktop)**, **libvirt (https://archlinux.org/packages/?name=libvirt)**, **edk2-ovmf (https://archlinux.org/packages/?name=edk2-ovmf)**, and **virt-manager (https://archlinux.org/packages/?name=virt-manager)**. For the default network connection **dnsmasq (https://archlinux.org/packages/?name=dnsmasq)** is required.

Follow Libvirt#Configuration to configure libvirt for use.

You may also need to activate the default libvirt network (https://wiki.libvirt.org/page/Networking#NAT_forwarding_.28aka_.22virtual_networks.22.29):

```
# virsh net-autostart default
# virsh net-start default
```

Note: The default libvirt network will only be listed if the virsh command is run as root.

4.2 Setting up the guest OS

The process of setting up a virtual machine using `virt-manager` is mostly self-explanatory, as most of the process comes with fairly comprehensive on-screen instructions.

However, you should pay special attention to the following steps:

- When the virtual machine creation wizard asks you to name your virtual machine (final step before clicking "Finish"), check the "Customize before install" checkbox.
- In the "Overview" section, set your firmware to "UEFI" (https://i.imgur.com/73r2ctM.png). If the option is grayed out, make sure that:
  - Your hypervisor is running as a system session and not a user session. This can be verified by clicking, then hovering (https://i.ibb.co/N1XZCdp/Deepin-Screenshot-select-area-20190125113216.png) over the session in virt-manager. If you are accidentally running it as a user session, you must open a new connection by clicking "File" > "Add Connection..", then select the option from the drop-down menu station "QEMU/KVM" and not "QEMU/KVM user session".
- In the "CPUs" section, change your CPU model to "host-passthrough". If it is not in the list, you will have to either type it by hand or by using `virt-xml vmname --edit --cpu host-passthrough`. This will ensure that your CPU is detected properly, since it causes libvirt to expose your CPU capabilities exactly as they are instead of only those it recognizes (which is the preferred default behavior to make CPU behavior easier to reproduce). Without it, some applications may complain about your CPU being of an unknown model.
- If you want to minimize IO overhead, it is easier to setup #Virtio disk before installing

The rest of the installation process will take place as normal using a standard QXL video adapter running in a window. At this point, there is no need to install additional drivers for the rest of the virtual devices, since most of them will be removed later on. Once the guest OS is done installing, simply turn off the virtual machine. It is possible you will be dropped into the UEFI menu instead of starting the installation upon powering your virtual machine for the first time. Sometimes the correct ISO file was not automatically detected and you will need to manually specify the drive to boot. By typing exit and navigating to "boot manager" you will enter a menu that allows you to choose between devices.

4.3 Attaching the PCI devices

With the installation done, it is now possible to edit the hardware details in libvirt and remove virtual integration devices, such as the spice channel and virtual display, the QXL video adapter, the emulated mouse and keyboard and the USB tablet device. For example, remove the following sections from your XML file:

```
<channel type="spicevmc">
  ...
</channel>
<input type="tablet" bus="usb">
  ...
</input>
<input type="mouse" bus="ps2"/>
```

```
    <input type="keyboard" bus="ps2"/>
    <graphics type="spice" autoport="yes">
      ...
    </graphics>
    <video>
      <model type="qxl" .../>
      ...
    </video>
```

Since that leaves you with no input devices, you may want to bind a few USB host devices to your virtual machine as well, but remember to **leave at least one mouse and/or keyboard assigned to your host** in case something goes wrong with the guest. This may be done by using `Add Hardware > USB Host Device`.

At this point, it also becomes possible to attach the PCI device that was isolated earlier; simply click on "Add Hardware" and select the PCI Host Devices you want to passthrough. If everything went well, the screen plugged into your GPU should show the OVMF splash screen and your virtual machine should start up normally. From there, you can setup the drivers for the rest of your virtual machine.

4.4 Video card driver virtualisation detection

Video card drivers by AMD incorporate very basic virtual machine detection targeting Hyper-V extensions. Should this detection mechanism trigger, the drivers will refuse to run, resulting in a black screen.

If this is the case, it is required to modify the reported Hyper-V vendor ID:

```
$ virsh edit vmname
```

```
...
<features>
  ...
  <hyperv>
    ...
    <vendor_id state='on' value='randomid'/>
    ...
  </hyperv>
  ...
</features>
...
```

Nvidia guest drivers prior to version 465 exhibited a similar behaviour which resulted in a generic error 43 in the card's device manager status. Systems using these older drivers therefore also need the above modification. In addition, they also require hiding the KVM CPU leaf:

```
$ virsh edit vmname
```

```
...
<features>
  ...
  <kvm>
    <hidden state='on'/>
  </kvm>
  ...
</features>
...
```

Note that the above steps do not equate 'hiding' the virtual machine from Windows or any drivers/programs running in the virtual machine. Also, various other issues not related to any detection mechanism referred to here can also trigger error 43.

4.5 Passing keyboard/mouse via Evdev

If you do not have a spare mouse or keyboard to dedicate to your guest, and you do not want to suffer from the video overhead of Spice, you can setup evdev to share them between your Linux host and your virtual machine.

Note: By default, press both left and right `Ctrl` keys at the same time to swap control between the host and the guest.

You can change this hotkeys. You need to set `grabToggle` variable to one of available combination `Ctrl+Ctrl`, `Alt+Alt`, `Shift+Shift`, `Meta+Meta`, `ScrollLock` or `Ctrl+ScrollLock` for your keyboard. More information: https://github.com/libvirt/libvirt/blob/master/docs/formatdomain.rst#input-devices

First, find your keyboard and mouse devices in `/dev/input/by-id/`. Only devices with `event` in their name are valid. You may find multiple devices associated to your mouse or keyboard, so try `cat /dev/input/by-id/device_id` and either hit some keys on the keyboard or wiggle your mouse to see if input comes through, if so you have got the right device. Now add those devices to your configuration:

```
$ virsh edit vmname
```

```
...
  <devices>
    ...
    <input type='evdev'>
      <source dev='/dev/input/by-id/MOUSE_NAME'/>
    </input>
    <input type='evdev'>
      <source dev='/dev/input/by-id/KEYBOARD_NAME' grab='all' repeat='on' grabToggle='ctrl-ctrl'/>
    </input>
    ...
  </devices>
```

Replace `MOUSE_NAME` and `KEYBOARD_NAME` with your device path. Now you can startup the guest OS and test swapping control of your mouse and keyboard between the host and guest by pressing both the left and right control keys at the same time.

You may also consider switching from PS/2 to Virtio inputs in your configurations. Add these two devices:

```
$ virsh edit vmname
```

```
...
<input type='mouse' bus='virtio'/>
<input type='keyboard' bus='virtio'/>
...
```

The virtio input devices will not actually be used until the guest drivers are installed. QEMU will continue to send key events to the PS2 devices until it detects the virtio input driver initialization. Note that the PS2 devices cannot be removed as they are an internal function of the emulated Q35/440FX chipsets.

4.6 Gotchas

4.6.1 Using a non-EFI image on an OVMF-based virtual machine

The OVMF firmware does not support booting off non-EFI mediums. If the installation process drops you in a UEFI shell right after booting, you may have an invalid EFI boot media. Try using an alternate Linux/Windows image to determine if you have an invalid media.

5 Performance tuning

Most use cases for PCI passthroughs relate to performance-intensive domains such as video games and GPU-accelerated tasks. While a PCI passthrough on its own is a step towards reaching native performance, there are still a few ajustments on the host and guest to get the most out of your virtual machine.

5.1 CPU pinning

The default behavior for KVM guests is to run operations coming from the guest as a number of threads representing virtual processors. Those threads are managed by the Linux scheduler like any other thread and are dispatched to any available CPU cores based on niceness and priority queues. As such, the local CPU cache benefits (L1/L2/L3) are lost each time the host scheduler reschedules the virtual CPU thread on a different physical CPU. This can noticeably harm performance on the guest. CPU pinning aims to resolve this by limiting which physical CPUs the virtual CPUs are allowed to run on. The ideal setup is a one to one mapping such that the virtual CPU cores match physical CPU cores while taking hyperthreading/SMT into account.

In addition, in some modern CPUs, groups of cores often share a common L3 cache. In such cases, care should be taken to pin exactly those physical cores that share a particular L3. Failing to do so might lead to cache evictions which could result in microstutters.

Note: For certain users enabling CPU pinning may introduce stuttering and short hangs, especially with the MuQSS scheduler (present in linux-ck and linux-zen kernels). You might want to try disabling pinning first if you experience similar issues, which effectively trades maximum performance for responsiveness at all times.

5.1.1 CPU topology

Most modern CPUs support hardware multitasking, also known as hyper-threading on Intel CPUs or SMT on AMD CPUs. Hyper-threading/SMT is simply a very efficient way of running two threads on one CPU core at any given time. You will want to take into consideration that the CPU pinning you choose will greatly depend on what you do with your host while your virtual machine is running.

To find the topology for your CPU run `lscpu -e`:

Note: Pay special attention to the 4th column **"CORE"** as this shows the association of the Physical/Logical CPU cores as well as the 8th column **"L3"** which shows which cores are connected to which L3 cache.

`lscpu -e` on a 6c/12t Ryzen 5 1600:

```
CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE MAXMHZ    MINMHZ
0   0    0      0    0:0:0:0       yes    3800.0000 1550.0000
1   0    0      0    0:0:0:0       yes    3800.0000 1550.0000
2   0    0      1    1:1:1:0       yes    3800.0000 1550.0000
3   0    0      1    1:1:1:0       yes    3800.0000 1550.0000
4   0    0      2    2:2:2:0       yes    3800.0000 1550.0000
5   0    0      2    2:2:2:0       yes    3800.0000 1550.0000
6   0    0      3    3:3:3:1       yes    3800.0000 1550.0000
7   0    0      3    3:3:3:1       yes    3800.0000 1550.0000
8   0    0      4    4:4:4:1       yes    3800.0000 1550.0000
9   0    0      4    4:4:4:1       yes    3800.0000 1550.0000
10  0    0      5    5:5:5:1       yes    3800.0000 1550.0000
11  0    0      5    5:5:5:1       yes    3800.0000 1550.0000
```

Considering the L3 mapping, it is recommended to pin and isolate CPUs 6–11. Pinning and isolating fewer than these (e.g. 8–11) would result in the host system making use of the L3 cache in core 6 and 7 which would eventually lead to cache evictions and therefore bad performance.

Note: Ryzen 3000 ComboPi AGESA changes topology to match Intel example, even on prior generation CPUs. Above valid only on older AGESA.

`lscpu -e` on a 6c/12t Intel 8700k:

```
CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE MAXMHZ    MINMHZ
0   0    0      0    0:0:0:0       yes    4600.0000 800.0000
1   0    0      1    1:1:1:0       yes    4600.0000 800.0000
2   0    0      2    2:2:2:0       yes    4600.0000 800.0000
3   0    0      3    3:3:3:0       yes    4600.0000 800.0000
4   0    0      4    4:4:4:0       yes    4600.0000 800.0000
5   0    0      5    5:5:5:0       yes    4600.0000 800.0000
6   0    0      0    0:0:0:0       yes    4600.0000 800.0000
7   0    0      1    1:1:1:0       yes    4600.0000 800.0000
8   0    0      2    2:2:2:0       yes    4600.0000 800.0000
9   0    0      3    3:3:3:0       yes    4600.0000 800.0000
10  0    0      4    4:4:4:0       yes    4600.0000 800.0000
11  0    0      5    5:5:5:0       yes    4600.0000 800.0000
```

Since all cores are connected to the same L3 in this example, it does not matter much how many CPUs you pin and isolate as long as you do it in the proper thread pairs. For instance, (0, 6), (1, 7), etc.

As we see above, with AMD **Core 0** is sequential with **CPU 0 & 1**, whereas Intel places **Core 0** on **CPU 0 & 6**.

Tip: You can view your systems topology in diagram form, which may help some users. If you have **hwloc (https://archlinux.org/packages/?name=hwloc)** installed, run `lstopo` to generate a helpful image of your CPU/Thread groupings.

If you do not need all cores for the guest, it would then be preferable to leave at the very least one core for the host. Choosing which cores one to use for the host or guest should be based on the specific hardware characteristics of your CPU, however **Core 0** is a good choice for the host in most cases. If any cores are reserved for the host, it is recommended to pin the emulator and iothreads, if used, to the host cores rather than the VCPUs. This may improve performance and reduce latency for the guest since those threads will not pollute the cache or contend for scheduling with the guest VCPU threads. If all cores are passed to the guest, there is no need or benefit to pinning the emulator or iothreads.

5.1.2 XML examples

Note: Do not use the **iothread** lines from the XML examples shown below if you have not added an **iothread** to your disk controller. **iothread**'s only work on **virtio-scsi** or **virtio-blk** devices.

5.1.2.1 4c/1t CPU w/o Hyperthreading Example

```
$ virsh edit vmname
```

```
...
<vcpu placement='static'>4</vcpu>
<cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
</cputune>
...
```

5.1.2.2 4c/2t Intel/AMD CPU example (after ComboPI AGESA bios update)

```
$ virsh edit vmname
```

```
...
<vcpu placement='static'>8</vcpu>
<iothreads>1</iothreads>
<cputune>
    <vcpupin vcpu='0' cpuset='2'/>
    <vcpupin vcpu='1' cpuset='8'/>
    <vcpupin vcpu='2' cpuset='3'/>
    <vcpupin vcpu='3' cpuset='9'/>
    <vcpupin vcpu='4' cpuset='4'/>
    <vcpupin vcpu='5' cpuset='10'/>
    <vcpupin vcpu='6' cpuset='5'/>
    <vcpupin vcpu='7' cpuset='11'/>
    <emulatorpin cpuset='0,6'/>
    <iothreadpin iothread='1' cpuset='0,6'/>
</cputune>
    ...
<cpu mode='host-passthrough'>
    <topology sockets='1' cores='4' threads='2'/>
</cpu>
    ...
```

5.1.2.3 4c/2t AMD CPU example (Before ComboPi AGESA bios update)

```
$ virsh edit vmname
```

```
...
<vcpu placement='static'>8</vcpu>
<iothreads>1</iothreads>
<cputune>
  <vcpupin vcpu='0' cpuset='2'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='4'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='6'/>
```

```
    <vcpupin vcpu='5' cpuset='7'/>
    <vcpupin vcpu='6' cpuset='8'/>
    <vcpupin vcpu='7' cpuset='9'/>
    <emulatorpin cpuset='0-1'/>
    <iothreadpin iothread='1' cpuset='0-1'/>
  </cputune>
      ...
  <cpu mode='host-passthrough'>
      <topology sockets='1' cores='4' threads='2'/>
  </cpu>
      ...
```

Note: If further CPU isolation is needed, consider using the **isolcpus** kernel command-line parameter on the unused physical/logical cores.

If you do not intend to be doing any computation-heavy work on the host (or even anything at all) at the same time as you would on the virtual machine, you may want to pin your virtual machine threads across all of your cores, so that the virtual machine can fully take advantage of the spare CPU time the host has available. Be aware that pinning all physical and logical cores of your CPU could induce latency in the guest virtual machine.

5.2 Huge memory pages

When dealing with applications that require large amounts of memory, memory latency can become a problem since the more memory pages are being used, the more likely it is that this application will attempt to access information across multiple memory "pages", which is the base unit for memory allocation. Resolving the actual address of the memory page takes multiple steps, and so CPUs normally cache information on recently used memory pages to make subsequent uses on the same pages faster. Applications using large amounts of memory run into a problem where, for instance, a virtual machine uses 4 GiB of memory divided into 4 KiB pages (which is the default size for normal pages) for a total of 1.04 million pages, meaning that such cache misses can become extremely frequent and greatly increase memory latency. Huge pages exist to mitigate this issue by giving larger individual pages to those applications, increasing the odds that multiple operations will target the same page in succession.

5.2.1 Transparent huge pages

QEMU will use 2MiB sized transparent huge pages automatically without any explicit configuration in QEMU or Libvirt, subject to some important caveats. When using VFIO the pages are locked in at boot time and transparent huge pages are allocated up front when the virtual machine first boots. If the kernel memory is highly fragmented, or the virtual machine is using a majority of the remaining free memory, it is likely that the kernel will not have enough 2MiB pages to fully satisfy the allocation. In such a case, it silently fails by using a mix of 2MiB and 4KiB pages. Since the pages are locked in VFIO mode, the kernel will not be able to convert those 4KiB pages to huge after the virtual machine starts either. The number of available 2MiB huge pages available to THP is the same as via the **#Dynamic huge pages** mechanism described in the following sections.

To check how much memory THP is using globally:

```
$ grep AnonHugePages /proc/meminfo
```

```
AnonHugePages:   8091648 kB
```

To check a specific QEMU instance. QEMU's PID must be substituted in the grep command:

```
$ grep -P 'AnonHugePages:\s+(?!0)\d+' /proc/[PID]/smaps
```

```
AnonHugePages:   8087552 kB
```

In this example, the virtual machine was allocated 8388608KiB of memory, but only 8087552KiB was available via THP. The remaining 301056KiB are allocated as 4KiB pages. Aside from manually checking, there is no indication when partial allocations occur. As such, THP's effectiveness is very much dependent on the host system's memory fragmentation at the time of virtual machine startup. If this trade off is unacceptable or strict guarantees are required, #Static huge pages is recommended.

Arch kernels have THP compiled in and enabled by default with `/sys/kernel/mm/transparent_hugepage/enabled` set to `madvise` mode.

5.2.2 Static huge pages

While transparent huge pages should work in the vast majority of cases, they can also be allocated statically during boot. This should only be needed to make use 1 GiB hugepages on machines that support it, since transparent huge pages normally only go up to 2 MiB.

Warning: Static huge pages lock down the allocated amount of memory, making it unavailable for applications that are not configured to use them. Allocating 4 GiBs worth of huge pages on a machine with 8 GiB of memory will only leave you with 4 GiB of available memory on the host **even when the virtual machine is not running**.

Note: The described procedures have some drawbacks but will not necessarily net you a great performance advantage. According to Red Hat benchmarks (https://developers.redhat.com/blog/2021/04/27/benchmarking-transparent-versus-1gib-static-huge-page-performance-in-linux-virtual-machines#benchmarks), you should not expect more than a 2% performance gain from this over #Transparent huge pages.

To allocate huge pages at boot, one must simply specify the desired amount on their kernel command line with `hugepages=x`. For instance, reserving 1024 pages with `hugepages=1024` and the default size of 2048 KiB per huge page creates 2 GiB worth of memory for the virtual machine to use.

If supported by CPU page size could be set manually. 1 GiB huge page support could be verified by `grep pdpe1gb /proc/cpuinfo`. Setting 1 GiB huge page size via kernel parameters : `default_hugepagesz=1G hugepagesz=1G hugepages=X`.

Also, since static huge pages can only be used by applications that specifically request it, you must add this section in your libvirt domain configuration to allow kvm to benefit from them :

```
$ virsh edit vmname
```

```
...
<memoryBacking>
        <hugepages/>
</memoryBacking>
...
```

5.2.3 Dynamic huge pages

Hugepages could be allocated manually via `vm.nr_overcommit_hugepages` sysctl parameter.

```
/etc/sysctl.d/10-kvm.conf
```

```
vm.nr_hugepages = 0
vm.nr_overcommit_hugepages = num
```

Where `num` - is the number of huge pages, which default size if 2 MiB. Pages will be automatically allocated, and freed after the virtual machine stops.

More manual way:

```
# echo num > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
# echo num > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

For 2 MiB and 1 GiB page size respectively. And they should be manually freed in the same way.

It is hardly recommended to drop caches, compact memory and wait a couple of seconds before starting the virtual machine, as there could be not enough free contiguous memory for required huge pages blocks. Especially after some uptime of the host system.

```
# echo 3 > /proc/sys/vm/drop_caches
# echo 1 > /proc/sys/vm/compact_memory
```

Theoretically, 1 GiB pages works as 2 MiB. But practically - no guaranteed way was found to get contiguous 1 GiB memory blocks. Each consequent request of 1 GiB blocks lead to lesser and lesser dynamically allocated count.

5.3 CPU frequency governor

Depending on the way your CPU governor is configured, the virtual machine threads may not hit the CPU load thresholds for the frequency to ramp up. Indeed, KVM cannot actually change the CPU frequency on its own, which can be a problem if it does not scale up with vCPU usage as it would result in underwhelming performance. An easy way to see if it behaves correctly is to check if the frequency reported by `watch lscpu` goes up when running a CPU-intensive task on the guest. If you are indeed experiencing stutter and the frequency does not go up to reach its reported maximum, it may be due to cpu scaling being controlled by the host OS (https://lime-technology.com/forum/index.php?topic=46664.msg447678#msg447678). In this case, try setting all cores to maximum frequency to see if this improves performance. Note that if you are using a modern intel chip with the default pstate driver, cpupower commands will be ineffective, so monitor `/proc/cpuinfo` to make sure your cpu is actually at max frequency.

Warning: Depending on your processor, it might actually be detrimental to performance to force the whole CPU to run at full frequency at all times. For instance, modern AMD processors (Zen 2 and Zen 3) depend heavily on being able to scale individual cores in separate core complexes for optimal thermal distribution. If the whole CPU is running at full frequency at all times, there is less headroom for invididual cores to clock high, resulting in worse performance for processes that are not heavily multithreaded such as games. This should be benchmarked in your virtual machine.

5.4 Isolating pinned CPUs

CPU pinning by itself will not prevent other host processes from running on the pinned CPUs. Properly isolating the pinned CPUs can reduce latency in the guest virtual machine.

5.4.1 With isolcpus kernel parameter

In this example, let us assume you are using CPUs 4-7. Use the kernel parameters `isolcpus nohz_full` to completely isolate the CPUs from the kernel. For example:

```
isolcpus=4-7 nohz_full=4-7
```

Then, run `qemu-system-x86_64` with taskset and chrt:

```
# chrt -r 1 taskset -c 4-7 qemu-system-x86_64 ...
```

The `chrt` command will ensure that the task scheduler will round-robin distribute work (otherwise it will all stay on the first cpu). For `taskset`, the CPU numbers can be comma- and/or dash-separated, like "0,1,2,3" or "0-4" or "1,7-8,10" etc.

See this Internet Archive copy of a Removeddit mirror of a Reddit thread (https://web.archive.org/web/20210520061110/https://www.removeddit.com/r/VFIO/comments/6vgtpx/high_dpc_latency_and_audio_stuttering_on_windows/dm0sfto/) for more info. (The original thread (https://www.reddit.com/r/VFIO/comments/6vgtpx/high_dpc_latency_and_audio_stuttering_on_windows/dm0sfto/) is worthless because of deleted comments, and Removeddit not longer works.)

5.4.2 Dynamically isolating CPUs

The isolcpus kernel parameter will permanently reserve CPU cores, even when the guest is not running. A more flexible alternative is to dynamically isolate CPUs when starting the guest. This can be achieved with the following alternatives:

- `cpuset-git (https://aur.archlinux.org/packages/cpuset-git/)`AUR (vfio-users post (https://www.redhat.com/archives/vfio-users/2016-September/msg00072.html), blog post (https://rokups.github.io/#!pages/gaming-vm-performance.md), example script (https://github.com/PassthroughPOST/VFIO-Tools/blob/master/libvirt_hooks/hooks/cset.sh))
- `vfio-isolate (https://aur.archlinux.org/packages/vfio-isolate/)`AUR
- systemd

5.4.2.1 Example with systemd

In this example, we assume a host with 12 CPUs, where CPUs 2-5 and 8-11 are pinned to the guest. Then run the following to isolate the host to CPUs 0, 1, 6, and 7:

```
# systemctl set-property --runtime -- user.slice AllowedCPUs=0,1,6,7
# systemctl set-property --runtime -- system.slice AllowedCPUs=0,1,6,7
# systemctl set-property --runtime -- init.scope AllowedCPUs=0,1,6,7
```

After shutting down the guest, run the following to reallocate all 12 CPUs back to the host:

```
# systemctl set-property --runtime -- user.slice AllowedCPUs=0-11
# systemctl set-property --runtime -- system.slice AllowedCPUs=0-11
# systemctl set-property --runtime -- init.scope AllowedCPUs=0-11
```

You can use a libvirt hook (https://libvirt.org/hooks.html) to automatically run the above at startup/shutdown of the guest like so:

Create or edit `/etc/libvirt/hooks/qemu` with the following content.

```
/etc/libvirt/hooks/qemu
```

```
#!/bin/sh

command=$2

if [ "$command" = "started" ]; then
    systemctl set-property --runtime -- system.slice AllowedCPUs=0,1,6,7
    systemctl set-property --runtime -- user.slice AllowedCPUs=0,1,6,7
    systemctl set-property --runtime -- init.scope AllowedCPUs=0,1,6,7
elif [ "$command" = "release" ]; then
    systemctl set-property --runtime -- system.slice AllowedCPUs=0-11
    systemctl set-property --runtime -- user.slice AllowedCPUs=0-11
    systemctl set-property --runtime -- init.scope AllowedCPUs=0-11
fi
```

Afterwards make it executable.

Restart `libvirtd.service` and then start your virtual machine. If you create some heavily multithreaded load on your host now, you should see that it keeps your chosen CPUs free from load while the virtual machine can still make use of it. You should also see those CPUs automatically getting fully used by your host once you terminate the virtual machine.

More examples are contained in the following reddit threads: [1] (https://www.reddit.com/r/VFIO/comments/ebe3l5/deprecated_isolcpus_workaround/fem8jgk) [2] (https://www.reddit.com/r/VFIO/comments/gyem88/noob_question_how_to_isolate_cpu_cores/ftaqno1/) [3] (https://www.reddit.com/r/VFIO/comments/ij25rg/splitting_ht_cores_between_host_and_vm/)

Note that this requires systemd 244 or higher, and cgroups v2, which is now enabled by default.

5.5 Improving performance on AMD CPUs

Starting with QEMU 3.1 the TOPOEXT cpuid flag is disabled by default. In order to use hyperthreading (SMT) on AMD CPUs you need to manually enable it:

```
 <cpu mode='host-passthrough' check='none'>
 <topology sockets='1' cores='4' threads='2'/>
 <feature policy='require' name='topoext'/>
 </cpu>
```

commit: https://git.qemu.org/?p=qemu.git;a=commit;h=7210a02c58572b2686a3a8d610c6628f87864aed

5.6 Virtio disk

The default disk types are SATA or IDE emulation out of the box. These controllers offer maximum compatibility but are not suited for efficient virtualization. Two accelerated models exist: `virtio-scsi` for SCSI emulation and passthrough, or `virtio-blk` for a more basic block device emulation.

5.6.1 Drivers

- Linux guests should support these out of the box on any modern kernel
- macOS has `virtio-blk` support starting in Mojave via `AppleVirtIO.kext`
- Windows needs the **Windows virtio drivers (https://docs.fedoraproject.org/en-US/quick-docs/creating-windows-virtual-machines-using-virtio-drivers/)**. `virtio-scsi` uses the `vioscsi` driver. `virtio-blk` uses the `viostor` driver
- Windows can be installed directly onto these disks by selecting 'load driver' on the installer disk selection menu. The windows iso and virtio driver iso should both be attached as regular SATA/IDE cdroms during the installation process
- To switch boot disks to virtio on an existing Windows installation:
  - `virtio-blk` : Add a temporary disk with bus `virtio` , boot windows & load the driver for the disk, then shutdown and switch the boot disk disk bus to `virtio`
  - `virtio-scsi` : Add a scsi controller with model `virtio` , boot windows & load the driver for the controller, then shutdown and switch the boot disk bus to `scsi` (not virtio)

5.6.2 Considerations

- `virtio-scsi` TRIM support is mature, all versions should support it. Traditionally, `virtio-scsi` has been the preferred approach for this reason
- `virtio-blk` TRIM support is new, this requires requires qemu 4.0+, guest linux kernel 5.0+, guest windows drivers 0.1.173+
- Thin provisioning works by enabling TRIM on a sparse image file: `discard='unmap'` . Unused blocks will be freed and the disk usage will drop (works on both raw and qcow2). Actual on-disk size of a sparse image file may be checked with `du /path/to/disk.img`
- Thin provisioning can also work with block storage such as zfs zvols or thin lvm
- Virt queue count will influence the number of threads inside the guest kernel used for IO processing, suggest using `queues='4'` or more
- Native mode (`io='native'`) uses a single threaded model based on linux AIO, is a bit more CPU efficient but may have lower peak performance and does not allow host side caching to be used
- Threaded mode (`io='threads'`) will spawn dozens of threads on demand as the disk is used. This is less efficient but may perform better if there are enough host cores available to run them, and allows for host side caching to be used
- Modern versions of libvirt will group the dynamic worker threads created when using threaded mode in with the iothread=1 cgroup for pinning purposes. Very old versions of libvirt left these in the emulator cgroup

5.6.3 IO threads

An IO thread is a dedicated thread for processing disk events, rather than using the main qemu emulator loop. This should not be confused with the worker threads spawned on demand with `io='threads'` .

- You can only use one iothread per disk controller. The thread must be assigned to a specific controller with `iothread='X'` in the `<driver>` tag. Furthermore, extra & unassigned iothreads will not be used and do nothing
- In the case of `virtio-scsi`, there is one controller for multiple scsi disks. The iothread is assigned on the controller: `<controller><driver iothread='X'>`
- In the case of `virtio-blk`, each disk has its own controller. The iothread is assigned in the driver tag under the disk itself: `<disk><driver iothread='X'>`
- Since emulated disks incur a significant amount of CPU overhead, that can lead to vcpu stuttering under high disk load (especially high random IOPS). In this case it helps to pin the IO to different core(s) than your vcpus with `<iothreadpin>`

5.6.4 Examples with libvirt

virtio-scsi + iothread + worker threads + host side writeback caching + full disk block device backend:

```
<domain>
  <devices>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='writeback' io='threads' discard='unmap'/>
      <source dev='/dev/disk/by-id/ata-Samsung_SSD_840_EVO_1TB_S1D9NSAF206396F'/>
      <target dev='sda' bus='scsi'/>
    </disk>
    <controller type='scsi' index='0' model='virtio-scsi'>
      <driver iothread='1' queues='8'/>
    </controller>
```

virtio-blk + iothread + native aio + no host caching + raw sparse image backend:

```
<domain>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' discard='unmap' iothread='1' queues='8'/>
      <source file='/var/lib/libvirt/images/pool/win10.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
```

Creating the iothreads:

```
<domain>
  <iothreads>1</iothreads>
```

Pinning iothreads:

```
<domain>
  <cputune>
    <iothreadpin iothread='1' cpuset='0-1,6-7'/>
```

5.6.5 Example with virt-manager

This will create a `virtio-blk` device:

1. Open the virtual machine preferences
2. Go to `Add Hardware > Storage`

3. Create or choose a storage file
4. Select `Device Type: Disk device` and `Bus type: VirtIO`
5. Click Finish

5.7 Virtio network

The default NIC models rtl8139 or e1000 can be a bottleneck for gigabit+ speeds and have a significant amount of CPU overhead compared to `virtio-net`.

- Select `virtio` as the model for the NIC with libvirt or use the `virtio-net-pci` device in bare qemu
- Windows needs the `NetKVM` driver from **Windows virtio drivers (https://docs.fedoraproject.org/en-US/quick-docs/creating-windows-virtual-machines-using-virtio-drivers/)**
- Virtio uses vhost-net by default for in-kernel packet processing without exiting to userspace
- Multiqueue can enabled for a further speedup with multiple connections but typically will not boost single stream speeds. For libvirt add `<driver queues='8'/>` under the interface tag
- Zero copy transmit may also be enabled on macvtap by setting the module parameter `vhost_net.experimental_zcopytx=1` but this may actually have worse performance, see **commit (https://github.com/torvalds/linux/commit/098eadce3c622c07b328d0a43dda379b38cf7c5e)**

Libvirt example with a bridge:

```
<interface type='bridge'>
  <mac address="52:54:00:6d:6e:2e"/>
  <source bridge='br0'/>
  <model type='virtio'/>
  <driver queues='8'/>
</interface>
```

MACVTAP example with a bridge:

```
<interface type="direct">
 <source dev="eno1" mode="vepa"/>
 <target dev="macvtap0"/>
 <model type="virtio"/>
 <alias name="net0"/>
</interface>
```

Possible options for mode are 'vepa', 'bridge', 'private', and 'passthrough'. A guide with decriptions of the differences is available from redhat[4] (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-virtual_networking-directly_attaching_to_physical_interface).

Replace the source `/dev` device with your own device address. You can get your local address with the following command:

```
$ ip link
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default ql
en 1000
```

```
    link/ether 30:9c:23:ac:51:d0 brd ff:ff:ff:ff:ff:ff
    altname enp0s31f6
```

5.8 Further tuning

More specialized virtual machine tuning tips are available at Red Hat's Virtualization Tuning and Optimization Guide (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/virtualization_tuning_and_optimization_guide/index).

6 Special procedures

Certain setups require specific configuration tweaks in order to work properly. If you are having problems getting your host or your virtual machine to work properly, see if your system matches one of the cases below and try adjusting your configuration accordingly.

6.1 Using identical guest and host GPUs

Due to how vfio-pci uses your vendor and device id pair to identify which device they need to bind to at boot, if you have two GPUs sharing such an ID pair you will not be able to get your passthrough driver to bind with just one of them. This sort of setup makes it necessary to use a script, so that whichever driver you are using is instead assigned by pci bus address using the `driver_override` mechanism.

6.1.1 Script variants

6.1.1.1 Passthrough all GPUs but the boot GPU

Here, we will make a script to bind vfio-pci to all GPUs but the boot gpu. Create the script `/usr/local/bin/vfio-pci-override.sh`:

```
#!/bin/sh

for i in /sys/bus/pci/devices/*/boot_vga; do
    if [ $(cat "$i") -eq 0 ]; then
        GPU="${i%/boot_vga}"
        AUDIO="$(echo "$GPU" | sed -e "s/0$/1/")"
        USB="$(echo "$GPU" | sed -e "s/0$/2/")"
        echo "vfio-pci" > "$GPU/driver_override"
        if [ -d "$AUDIO" ]; then
            echo "vfio-pci" > "$AUDIO/driver_override"
        fi
        if [ -d "$USB" ]; then
            echo "vfio-pci" > "$USB/driver_override"
        fi
    fi
done

modprobe -i vfio-pci
```

6.1.1.2 Passthrough selected GPU

In this case we manually specify the GPU to bind.

```
#!/bin/sh
```

```
DEVS="0000:03:00.0 0000:03:00.1"

if [ ! -z "$(ls -A /sys/class/iommu)" ]; then
    for DEV in $DEVS; do
        echo "vfio-pci" > /sys/bus/pci/devices/$DEV/driver_override
    done
fi

modprobe -i vfio-pci
```

6.1.1.3 Passthrough IOMMU Group based of GPU

Simplifying the passthrough of other necessary devices from selected GPU(s). Things like the graphicscard's onboard Audio, USB and RGB controllers.

```
#!/bin/sh

DEVS="0000:03:00.0"

if [ ! -z "$(ls -A /sys/class/iommu)" ]; then
    for DEV in $DEVS; do
        for IOMMUDEV in $(ls /sys/bus/pci/devices/$DEV/iommu_group/devices) ; do
            echo "vfio-pci" > /sys/bus/pci/devices/$IOMMUDEV/driver_override
        done
    done
fi

modprobe -i vfio-pci
```

6.1.2 Script installation

Edit `/etc/mkinitcpio.conf`:

1. Add `modconf` to the **HOOKS** array and `/usr/local/bin/vfio-pci-override.sh` to the **FILES** array.

Edit `/etc/modprobe.d/vfio.conf`:

1. Add the following line: `install vfio-pci /usr/local/bin/vfio-pci-override.sh`
2. Regenerate the initramfs and reboot.

6.2 Passing the boot GPU to the guest

The GPU marked as `boot_vga` is a special case when it comes to doing PCI passthroughs, since the BIOS needs to use it in order to display things like boot messages or the BIOS configuration menu. To do that, it makes **a copy of the VGA boot ROM which can then be freely modified (https://www.redhat.com/archives/vfio-users/2016-May/msg00224.html)**. This modified copy is the version the system gets to see, which the passthrough driver may reject as invalid. As such, it is generally recommended to change the boot GPU in the BIOS configuration so the host GPU is used instead or, if that is not possible, to swap the host and guest cards in the machine itself.

6.3 Using Looking Glass to stream guest screen to the host

It is possible to make a virtual machine share the monitor, and optionally a keyboard and a mouse with a help of Looking Glass (https://looking-glass.io/).

6.3.1 Adding IVSHMEM Device to virtual machines

Looking glass works by creating a shared memory buffer between a host and a guest. This is a lot faster than streaming frames via localhost, but requires additional setup.

With your virtual machine turned off open the machine configuration

```
$ virsh edit vmname
```

```
...
<devices>
    ...
  <shmem name='looking-glass'>
    <model type='ivshmem-plain'/>
    <size unit='M'>32</size>
  </shmem>
</devices>
...
```

You should replace 32 with your own calculated value based on what resolution you are going to pass through. It can be calculated like this:

```
width x height x 4 x 2 = total bytes
total bytes / 1024 / 1024 = total mebibytes + 10
```

For example, in case of 1920x1080

```
1920 x 1080 x 4 x 2 = 16,588,800 bytes
16,588,800 / 1024 / 1024 = 15.82 MiB + 10 = 25.82
```

The result must be **rounded up** to the nearest power of two, and since 25.82 is bigger than 16 we should choose 32.

Next create a configuration file to create the shared memory file on boot

```
/etc/tmpfiles.d/10-looking-glass.conf
```

```
f       /dev/shm/looking-glass  0660    user    kvm     -
```

Replace user with your username.

Ask systemd-tmpfiles to create the shared memory file now without waiting to next boot

```
# systemd-tmpfiles --create /etc/tmpfiles.d/10-looking-glass.conf
```

6.3.2 Installing the IVSHMEM Host to Windows guest

Currently Windows would not notify users about a new IVSHMEM device, it would silently install a dummy driver. To actually enable the device you have to go into device manager and update the driver for the device under the "System Devices" node for **"PCI standard RAM Controller"**. Download the signed driver from Red Hat (https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/upstream-virtio/).

Once the driver is installed you must download a matching looking-glass-host (https://looking-glass.io/downloads) package that matches the client you will install from AUR, and install it on your guest. In order to run it you would also need to install Microsoft Visual C++ Redistributable from Microsoft (https://www.visualstudio.com/downloads/). The recent version will automatically install a service that starts the daemon on boot. The logs of the host daemon are located at `%ProgramData%\Looking Glass (host)\looking-glass-host.txt` on the guest system.

6.3.3 Setting up the null video device

(Retrieved from: https://looking-glass.io/docs/stable/install/#spice-server)

If you would like to use Spice to give you keyboard and mouse input along with clipboard sync support, make sure you have a `<graphics type='spice'>` device, then:

- Find your `<video>` device, and set `<model type='none'/>`
- If you cannot find it, make sure you have a `<graphics>` device, save and edit again

6.3.4 Getting a client

Looking glass client can be installed from AUR using `looking-glass` (https://aur.archlinux.org/packages/looking-glass/)AUR or `looking-glass-git` (https://aur.archlinux.org/packages/looking-glass-git/)AUR packages.

You can start it once the virtual machine is set up and running

```
$ looking-glass-client
```

If you do not want to use Spice to control the guest mouse and keyboard you can disable the Spice server.

```
$ looking-glass-client -s
```

Additionally you may want to start Looking Glass Client in full screen, otherwise the image may be scaled down resulting in poor image fidelity.

```
$ looking-glass-client -F
```

To inhibit your host's screensaver start Looking Glass with the `-S` flag.

```
$ looking-glass-client -S
```

This does not work on some DEs, including KDE. The issue with KDE is likely related to this behaviour (https://bugs.kde.org/show_bug.cgi?id=383575). To temporarily disable the host's screensaver while using Looking Glass and KDE you can wrap Looking Glass in the following script:

```
#!/bin/sh
kwriteconfig5 --file kscreenlockerrc --group Daemon --key Autolock false
qdbus org.freedesktop.ScreenSaver /ScreenSaver configure
looking-glass-client
```

```
kwriteconfig5 --file kscreenlockerrc --group Daemon --key Autolock true
qdbus org.freedesktop.ScreenSaver /ScreenSaver configure
```

Launch with the `--help` option for further information.

6.3.5 Additional information

Refer to the [upstream documentation (https://looking-glass.io/docs/)](https://looking-glass.io/docs/) for further details.

6.4 Swap peripherals to and from the Host

Looking Glass includes a Spice client in order to control mouse movement on the Windows guest. However this may have too much latency for certain applications, such as gaming. An alternative method is passing through specific USB devices for minimal latency. This allows for switching the devices between host and guest.

First create a .xml file for the device(s) you wish to pass-through, which libvirt will use to identify the device.

~/.VFIOinput/input_1.xml

```
<hostdev mode='subsystem' type='usb' managed='no'>
<source>
<vendor id='0x[Before Colon]'/>
<product id='0x[After Colon]'/>
</source>
</hostdev>
```

Replace [Before/After Colon] with the contents of the 'lsusb' command, specific to the device you want to pass-through.

For instance my mouse is `Bus 005 Device 002: ID 1532:0037 Razer USA, Ltd` so I would replace `vendor id` with 1532, and `product id` with 0037.

Repeat this process for any additional USB devices you want to pass-through. If your mouse / keyboard has multiple entries in `lsusb`, perhaps if it is wireless, then create additional xml files for each.

Note: Do not forget to change the path & name of the script(s) above and below to match your user and specific system.

Next a bash script file is needed to tell libvirt what to attach/detach the USB devices to the guest.

~/.VFIOinput/input_attach.sh

```
#!/bin/sh

virsh attach-device [VirtualMachine-Name] [USBdevice]
```

Replace [VirtualMachine-Name] with the name of your virtual machine, which can be seen under virt-manager. Additionally replace [USBdevice] with the **full** path to the .xml file for the device you wish to pass-through. Add additional lines for more than 1 device. For example here is my script:

~/.VFIOinput/input_attach.sh

```
#!/bin/sh

virsh attach-device win10 /home/$USER/.VFIOinput/input_mouse.xml
virsh attach-device win10 /home/$USER/.VFIOinput/input_keyboard.xml
```

Next duplicate the script file and replace `attach-device` with `detach-device`. Ensure both scripts are **executable**.

This 2 script files can now be executed to attach or detach your USB devices from the host to the guest virtual machine. It is important to note that they may need to be executed as root. To run the script from the Windows virtual machine, one possibility is using **PuTTY** to **SSH** into the host, and execute the script. On Windows PuTTY comes with plink.exe which can execute singular commands over SSH before then logging out, instead of opening a SSH terminal, all in the background.

detach_devices.bat

```
"C:\Program Files\PuTTY\plink.exe" root@$HOST_IP -pw $ROOTPASSWORD /home/$USER/.VFIOinput/input_detach.sh
```

Replace `$HOST_IP` with the Host **IP Address** and $ROOTPASSWORD with the root password.

Warning: This method is insecure if somebody has access to your virtual machine, since they could open the file and read your password. It is advisable to use **SSH keys** instead!

You may also want to execute the script files using key binds. On Windows one option is **Autohotkey (https://autohotkey.com/)**, and on the Host **Xbindkeys**. Because of the need to run the scripts as root, you may also need to use **Polkit** or **Sudo** which can both be used to authenticate specific executables as able to run as root without needing a password.

6.5 Bypassing the IOMMU groups (ACS override patch)

If you find your PCI devices grouped among others that you do not wish to pass through, you may be able to separate them using Alex Williamson's ACS override patch. Make sure you understand **the potential risk (https://vfio.blogspot.com/2014/08/iommu-groups-inside-and-out.html)** of doing so.

You will need a kernel with the patch applied. The easiest method to acquiring this is through the **`linux-zen (https://archlinux.org/packages/?name=linux-zen)`** or **`linux-vfio (https://aur.archlinux.org/packages/linux-vfio/)`**AUR package.

In addition, the ACS override patch needs to be enabled with kernel command line options. The patch file adds the following documentation:

```
pcie_acs_override =
        [PCIE] Override missing PCIe ACS support for:
    downstream
        All downstream ports - full ACS capabilties
    multifunction
        All multifunction devices - multifunction ACS subset
    id:nnnn:nnnn
        Specfic device - full ACS capabilities
        Specified as vid:did (vendor/device ID) in hex
```

The option `pcie_acs_override=downstream,multifunction` should break up as many devices as possible.

After installation and configuration, reconfigure your kernel parameters to load the new kernel with the `pcie_acs_override=` option enabled.

7 Plain QEMU without libvirt

Instead of setting up a virtual machine with the help of libvirt, plain QEMU commands with custom parameters can be used for running the virtual machine intended to be used with PCI passthrough. This is desirable for some use cases like scripted setups, where the flexibility for usage with other scripts is needed.

To achieve this after **#Setting up IOMMU** and **#Isolating the GPU**, follow the QEMU article to setup the virtualized environment, **enable KVM** on it and use the flag `-device vfio-pci,host=07:00.0` replacing the identifier (07:00.0) with your actual device's ID that you used for the GPU isolation earlier.

For utilizing the OVMF firmware, make sure the **edk2-ovmf (https://archlinux.org/packages/?name=edk2-ovmf)** package is installed, copy the UEFI variables from `/usr/share/edk2-ovmf/x64/OVMF_VARS.fd` to temporary location like `/tmp/MY_VARS.fd` and finally specify the OVMF paths by appending the following parameters to the QEMU command (order matters):

- `-drive if=pflash,format=raw,readonly=on,file=/usr/share/edk2-ovmf/x64/OVMF_CODE.fd`
  for the actual OVMF firmware binary, note the readonly option
- `-drive if=pflash,format=raw,file=/tmp/MY_VARS.fd` for the variables

Note:

- Make sure that `OVMF_CODE.fd` is given as a command line parameter before `MY_VARS.fd`. The boot sequence will fail otherwise.
- QEMU's default SeaBIOS can be used instead of OVMF, but it is not recommended as it can cause issues with passthrough setups.

It is recommended to study the QEMU article for ways to enhance the performance by using the **virtio drivers** and other further configurations for the setup.

You also might have to use the `-cpu host,kvm=off` parameter to forward the host's CPU model info to the virtual machine and fool the virtualization detection used by Nvidia's and possibly other manufacturers' device drivers trying to block the full hardware usage inside a virtualized system.

8 Passing through other devices

8.1 USB controller

If your motherboard has multiple USB controllers mapped to multiple groups, it is possible to pass those instead of USB devices. Passing an actual controller over an individual USB device provides the following advantages :

- If a device disconnects or changes ID over the course of an given operation (such as a phone undergoing an update), the virtual machine will not suddenly stop seeing it.
- Any USB port managed by this controller is directly handled by the virtual machine and can

have its devices unplugged, replugged and changed without having to notify the hypervisor.
- Libvirt will not complain if one of the USB devices you usually pass to the guest is missing when starting the virtual machine.

Unlike with GPUs, drivers for most USB controllers do not require any specific configuration to work on a virtual machine and control can normally be passed back and forth between the host and guest systems with no side effects.

Warning: Make sure your USB controller supports resetting: **#Passing through a device that does not support resetting**

You can find out which USB devices correspond to which controller and how various ports and devices are assigned to each one of them using this command:

```
$ for usb_ctrl in /sys/bus/pci/devices/*/usb*; do pci_path=${usb_ctrl%/*}; iommu_group=$(readlink $pci_pa
th/iommu_group); echo "Bus $(cat $usb_ctrl/busnum) --> ${pci_path##*/} (IOMMU group ${iommu_group##*/})";
lsusb -s ${usb_ctrl#*/usb}:; echo; done
```

```
Bus 1 --> 0000:00:1a.0 (IOMMU group 4)
Bus 001 Device 004: ID 04f2:b217 Chicony Electronics Co., Ltd Lenovo Integrated Camera (0.3MP)
Bus 001 Device 007: ID 0a5c:21e6 Broadcom Corp. BCM20702 Bluetooth 4.0 [ThinkPad]
Bus 001 Device 008: ID 0781:5530 SanDisk Corp. Cruzer
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 2 --> 0000:00:1d.0 (IOMMU group 9)
Bus 002 Device 006: ID 0451:e012 Texas Instruments, Inc. TI-Nspire Calculator
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

This laptop has 3 USB ports managed by 2 USB controllers, each with their own IOMMU group. In this example, Bus 001 manages a single USB port (with a SanDisk USB pendrive plugged into it so it appears on the list), but also a number of internal devices, such as the internal webcam and the bluetooth card. Bus 002, on the other hand, does not apprear to manage anything except for the calculator that is plugged into it. The third port is empty, which is why it does not show up on the list, but is actually managed by Bus 002.

Once you have identified which controller manages which ports by plugging various devices into them and decided which one you want to passthrough, simply add it to the list of PCI host devices controlled by the virtual machine in your guest configuration. No other configuration should be needed.

Note: If your USB controller does not support resetting, is not in an isolated group, or is otherwise unable to be passed through then it may still be possible to accomplish similar results through udev rules. See [5] (https://github.com/olavmrk/usb-libvirt-hotplug) which allows any device connected to specified USB ports to be automatically attached to a virtual machine.

8.2 Passing audio from virtual machine to host via PulseAudio

It is possible to route the virtual machine's audio to the host as an application using libvirt. This has the advantage of multiple audio streams being routable to one host output, and working with audio output devices that do not support passthrough. This requires PulseAudio to be running on the host system.

First, install `qemu-audio-pa (https://archlinux.org/packages/?name=qemu-audio-pa)`.

Then, remove the comment from the `#user = ""` line. Then add your username in the quotations. This tells QEMU which user's pulseaudio stream to route through.

```
/etc/libvirt/qemu.conf

user = "example"
```

An emulated audio setup consists of two components: An emulated sound device exposed to the guest and an audio backend connecting the sound device to the host's PulseAudio.

Of the emulated sound devices available, two are of main interest: ICH9 and usb-audio. ICH9 features both output and input but is limited to stereo. usb-audio only features audio output but supports up to 6 channels in 5.1 configuration. For ICH9 remove any pre-existing audio backend in the `<devices>` section and add:

```
$ virsh edit vmname

    <sound model='ich9'>
      <codec type='micro'/>
      <audio id='1'/>
    </sound>
    <audio id='1' type='pulseaudio' serverName='/run/user/1000/pulse/native'/>
```

Note the matching `id` elements. The example above assumes a single-user system with user ID 1000. Use the `id` command to find the correct ID. You can also use the `/tmp` directory if you have multiple users accessing PulseAudio:

```
$ virsh edit vmname

    <audio id='1' type='pulseaudio' serverName='unix:/tmp/pulse-socket'/>
```

If you get crackling or distorted sound, try experimenting with some latency settings. The following example uses 20000 microseconds:

```
$ virsh edit vmname

    <audio id="1" type="pulseaudio" serverName="/run/user/1000/pulse/native">
      <input latency="20000"/>
      <output latency="20000"/>
    </audio>
```

You can also try disabling the software mixer included in QEMU. This should, in theory, be more efficient and allow for lower latencies since mixing will then take place on your host only:

```
$ virsh edit vmname

    <audio id="1" type="pulseaudio" serverName="/run/user/1000/pulse/native">
      <input mixingEngine="no"/>
      <output mixingEngine="no"/>
    </audio>
```

For usb-audio, the corresponding elements read

```
$ virsh edit vmname
```

```
    <sound model='usb'>
      <audio id='1'/>
    </sound>
    <audio id='1' type='pulseaudio' serverName='/run/user/1000/pulse/native'/>
```

However, if a 5.1 configuration is required the sound device needs to be configured via QEMU command line arguments:

```
$ virsh edit vmname
```

```
  </devices>
  <qemu:commandline>
    <qemu:arg value='-device'/>
    <qemu:arg value='usb-audio,id=sound0,audiodev=audio1,multi=on'/>
  </qemu:commandline>
</domain>
```

The `audiodev` tag has to be set to match the audio backend's `id` element. `id='1'` corresponds to `audio1` and so on.

Note:

- You can have multiple audio backends, by simply specifying `<audio>` / `-audiodev` multiple times in your XML and by assigning them different ids. This can be useful for a use case of having two identical backends. With PulseAudio each backend is a separate stream and can be routed to different output devices on the host (using a pulse mixer like **pavucontrol (https://archlinux.org/packages/?name=pavucontrol)** or **pulsemixer (https://archlinux.org/packages/?name=pulsemixer)**).
- USB 3 emulation is needed in Libvirt/QEMU to enable the usb-audio.
- It is recommended to enable MSI interrupts with a tool such as [6] (https://forums.guru3d.com/threads/windows-line-based-vs-message-signaled-based-interrupts-msi-tool.378044/) on the ICH9 audio device to mitigate any crackling, stuttering, speedup, or no audio at all after virtual machine restart.
- If audio is crackling/stuttering/speedup etc. is still present you may want to adjust parameters such as `buffer-length` and `timer-period`, more information on these parameters and more can be found in the **qemu(1) (https://man.archlinux.org/man/qemu.1)** manual.
- Some audio chipsets such as Realtek alc1220 (https://bugzilla.kernel.org/show_bug.cgi?id=195303) may also have issues out of the box so do consider this when using any audio emulation with QEMU.
- Improper pinning or heavy host usage without using isolcpus can also influence sound bugs, especially while gaming in a virtual machine.

8.3 Passing audio from virtual machine to host via JACK and PipeWire

It is also possible to pass the virtual machine's audio to the host via JACK and PipeWire.

First, make sure you have a working PipeWire setup with JACK support.

Next, you will need to tell libvirt to run QEMU as your user:

```
/etc/libvirt/qemu.conf
```

```
user = "example"
```

Do not forget to restart `libvirtd.service`.

As a final preparation, the XML scheme has to be extended to allow passing of environment variables. For this, modify the virtual machine domain configuration

```
$ virsh edit vmname
```
```
<domain type='kvm'>
```

to

```
$ virsh edit vmname
```
```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

Then, you can add the actual audio config to your virtual machine:

```
$ virsh edit vmname
```
```
    <devices>
    ...
      <audio id="1" type="jack">
        <input clientName="vm-win10" connectPorts="your-input"/>
        <output clientName="vm-win10" connectPorts="your-output"/>
      </audio>
    </devices>
      <qemu:commandline>
        <qemu:env name="PIPEWIRE_RUNTIME_DIR" value="/run/user/1000"/>
        <qemu:env name="PIPEWIRE_LATENCY" value="512/48000"/>
      </qemu:commandline>
 </domain>
```

Note: Use a tool like carla (https://archlinux.org/packages/?name=carla) to figure out which input and outputs you want.

Note the matching `id` elements. Above's example assumes a single-user system with user ID 1000. Use the `id` command to find the correct ID.

You might have to play with the `PIPEWIRE_LATENCY` values to get to the desired latency without crackling.

8.4 Passing audio from virtual machine to host via Scream

It is possible to pass the virtual machine's audio through a bridged network such as the one provided by Libvirt or by adding a IVSHMEM device to the host by using a application called Scream (https://github.com/duncanthrax/scream). This section will only cover using PulseAudio as a receiver on the host. See the project page for more details and instructions on other methods.

8.4.1 Using Scream with a bridged network

Note:

- This is the *preferred* way to use this, although results may vary per user

- It is recommend to use the #Virtio network adapter while using Scream, other virtual adapters provided by QEMU such as **e1000e** may lead to poor performance

To use scream via your network you will want to find your bridge name via `ip a`, in most cases it will be called **br0** or **virbr0**. Below is a example of the command needed to start the Scream application:

```
$ scream -o pulse -i virbr0 &
```

Warning: This will not work with a **macvtap bridge** as that does not allow host to guest communication, also make sure you have the proper firewall ports open for it to communicate with the virtual machine

8.4.2 Adding the IVSHMEM device to use Scream with IVSHMEM

With the virtual machine turned off, edit the machine configuration

```
$ virsh edit vmname
```

```
...
<devices>
    ...
  <shmem name='scream-ivshmem'>
    <model type='ivshmem-plain'/>
    <size unit='M'>2</size>
  </shmem>
</devices>
...
```

In the above configuration, the size of the IVSHMEM device is 2MB (the recommended amount). Change this as needed.

Now refer to **#Adding IVSHMEM Device to virtual machines** to configure the host to create the shared memory file on boot, replacing `looking-glass` with `scream-ivshmem`.

8.4.2.1 Configuring the Windows guest for IVSHMEM

The correct driver must be installed for the IVSHMEM device on the guest. See **#Installing the IVSHMEM Host to Windows guest**. Ignore the part about `looking-glass-host`.

Install the **Scream (https://github.com/duncanthrax/scream/releases)** virtual audio driver on the guest. If you have secure boot enabled for your virtual machine, you may need to disable it.

Using the registry editor, set the DWORD `HKLM\SYSTEM\CurrentControlSet\Services\Scream\Options\UseIVSHMEM` to the size of the IVSHMEM device in MB. Note that scream identifies its IVSHMEM device using its size, so make sure there is only one device of that size (the suggested default is `2` for 2MB).

Use the following command in an admin CMD shell to create both key and DWORD: `REG ADD HKLM\SYSTEM\CurrentControlSet\Services\Scream\Options /v UseIVSHMEM /t REG_DWORD /d 2`
(**sourced from scream on Github (https://github.com/duncanthrax/scream)**)

8.4.2.1.1 Configuring the host

Install scream (https://aur.archlinux.org/packages/scream/)AUR.

Create a **systemd user service** to control the receiver:

```
~/.config/systemd/user/scream-ivshmem-pulse.service
```

```
[Unit]
Description=Scream IVSHMEM pulse receiver
After=pulseaudio.service
Wants=pulseaudio.service

[Service]
Type=simple
ExecStartPre=/usr/bin/truncate -s 0 /dev/shm/scream-ivshmem
ExecStartPre=/usr/bin/dd if=/dev/zero of=/dev/shm/scream-ivshmem bs=1M count=2
ExecStart=/usr/bin/scream -m /dev/shm/scream-ivshmem

[Install]
WantedBy=default.target
```

Edit `count=2` with the size of the IVSHMEM device in MiB.

Tip: If you are using **PipeWire**, replace `pulseaudio.service` with `pipewire-pulse.service` and add `After=wireplumber.service`.

Now **start** the `scream-ivshmem-pulse.service` **user unit**.

To have it automatically start on next login, **enable** the **user unit**.

8.5 Physical disk/partition

Raw and qcow2 especially can have noticeable overhead for heavy IO. A whole disk or a partition may be used directly to bypass the filesystem and improve I/O performance. If you wish to dual boot the guest OS natively you would need to pass the entire disk without any partitioning. It is suggested to use /dev/disk/by- paths to refer to the disk since /dev/sdX entries can change between boots. To find out which disk/partition is associated with the one you would like to pass:

```
$ ls -l /dev/disk/by-id/*
```

```
/dev/disk/by-id/ata-ST1000LM002-9VQ14L_Z0501SZ9 -> ../../sdd
```

See **#Virtio disk** on how to add these with libvirt XML. You can also add the disk with Virt-Manager's **Add Hardware** menu and then type the disk you want in the **Select or create custom storage** box, e.g. **/dev/disk/by-id/ata-ST1000LM002-9VQ14L_Z0501SZ9**

8.6 Gotchas

8.6.1 Passing through a device that does not support resetting

When the virtual machine shuts down, all devices used by the guest are deinitialized by its OS in preparation for shutdown. In this state, those devices are no longer functional and must then be power-cycled before they can resume normal operation. Linux can handle this power-cycling on its own, but when a device has no known reset methods, it remains in this disabled state and becomes unavailable. Since Libvirt and Qemu both expect all host PCI devices to be ready to reattach to the host before completely stopping the virtual machine, when encountering a device that will not reset, they will hang in a "Shutting down" state where they will not be able to be restarted until the host system has been rebooted. It is therefore recommended to only pass through PCI devices which the kernel is able to reset, as evidenced by the presence of a `reset` file in the PCI device sysfs node, such as `/sys/bus/pci/devices/0000:00:1a.0/reset`.

The following bash command shows which devices can and cannot be reset.

```
for iommu_group in $(find /sys/kernel/iommu_groups/ -maxdepth 1 -mindepth 1 -type d);do echo "IOMMU group
$(basename "$iommu_group")"; for device in $(\ls -1 "$iommu_group"/devices/); do if [[ -e "$iommu_group"/
devices/"$device"/reset ]]; then echo -n "[RESET]"; fi; echo -n $'\t';lspci -nns "$device"; done; done
```

```
IOMMU group 0
        00:00.0 Host bridge [0600]: Intel Corporation Xeon E3-1200 v2/Ivy Bridge DRAM Controller [8086:01
58] (rev 09)
IOMMU group 1
        00:01.0 PCI bridge [0604]: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI Express R
oot Port [8086:0151] (rev 09)
        01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK208 [GeForce GT 720] [10de:1288]
(rev a1)
        01:00.1 Audio device [0403]: NVIDIA Corporation GK208 HDMI/DP Audio Controller [10de:0e0f] (rev a
1)
IOMMU group 2
        00:14.0 USB controller [0c03]: Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Hos
t Controller [8086:1e31] (rev 04)
IOMMU group 4
[RESET] 00:1a.0 USB controller [0c03]: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced
Host Controller #2 [8086:1e2d] (rev 04)
IOMMU group 5
[RESET] 00:1b.0 Audio device [0403]: Intel Corporation 7 Series/C210 Series Chipset Family High Definitio
n Audio Controller [8086:1e20] (rev 04)
IOMMU group 10
[RESET] 00:1d.0 USB controller [0c03]: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced
Host Controller #1 [8086:1e26] (rev 04)
IOMMU group 13
        06:00.0 VGA compatible controller [0300]: NVIDIA Corporation GM204 [GeForce GTX 970] [10de:13c2]
(rev a1)
        06:00.1 Audio device [0403]: NVIDIA Corporation GM204 High Definition Audio Controller [10de:0fb
b] (rev a1)
```

This signals that the xHCI USB controller in 00:14.0 cannot be reset and will therefore stop the virtual machine from shutting down properly, while the integrated sound card in 00:1b.0 and the other two controllers in 00:1a.0 and 00:1d.0 do not share this problem and can be passed without issue.

9 Complete setups and examples

For many reasons users may seek to see **complete passthrough setup examples**.

These examples offer a supplement to existing hardware compatibility lists. Additionally, if you have trouble configuring a certain mechanism in your setup, you might find these examples very valuable. Users there have described their setups in detail, and some have provided examples of their configuration files as well.

We encourage those who successfully build their system from this resource to help improve it by contributing their builds. Due to the many different hardware manufacturers involved, the seemingly significant lack of sufficient documentation, as well as other issues due to the nature of this process, community contributions are necessary.

10 Troubleshooting

If your issue is not mentioned below, you may want to browse QEMU#Troubleshooting.

10.1 QEMU 4.0: Unable to load graphics drivers/BSOD/Graphics stutter after driver install using Q35

Starting with QEMU 4.0, the Q35 machine type changes the default `kernel_irqchip` from `off` to `split` which breaks some guest devices, such as nVidia graphics (the driver fails to load / black screen / code 43 / graphics stutters, usually when mouse moving). Switch to full KVM mode instead by adding `<ioapic driver='kvm'/>` under libvirt's `<features>` tag in your virtual machine configuration or by adding `kernel_irqchip=on` in the `-machine` QEMU arg.

10.2 QEMU 5.0: host-passthrough with kernel version 5.5 to 5.8.1 when using Zen 2 processors: Windows 10 BSOD loop 'KERNEL SECURITY CHECK FAILURE'

Note: As of kernel version 5.8.2, disabling STIBP is not required anymore.

Starting with QEMU 5.0 virtual machines running on Zen 2 and newer kernels than 5.4 will cause a BSOD loop of: 'KERNEL SECURITY CHECK FAILURE'. This can be fixed by either updating to kernel version 5.8.2 or higher, or disabling STIBP:

```
<cpu mode='host-passthrough' ...>
  ...
  <feature policy='disable' name='amd-stibp'/>
  ...
</cpu>
```

This requires libvirt 6.5 or higher. On older versions, several workarounds exist:

- Switch CPU mode from `host-passthrough` to `host-model`. This only works on libvirt 6.4 or lower.
- Manually patch qemu-desktop (https://archlinux.org/packages/?name=qemu-desktop) in order to revert this (https://github.com/qemu/qemu/commit/143c30d4d346831a09e59e9af45afdca0331e819) commit.
- On qemu commandline, add `amd-stibp=off` to the cpu flags string. This can also be invoked through libvirt via a `<qemu:commandline>` entry.

10.3 "Error 43: Driver failed to load" with mobile (Optimus/max-q) nvidia GPUs

This error occurs because the Nvidia driver wants to check the status of the power supply. If no battery is present, the driver does not work. Whether Libvirt or QEMU, by default none of them provide the possibility to simulate a battery. This might also result in a reduced screen resolution and the Nvidia Desktop Manager refusing to load when right-clicking the desktop, saying it requires Windows 10, a compatible GPU and the Nvidia graphics driver.

You can however create and add a custom acpi table file to the virtual machine which will do the work.

First you have to create the custom acpi table file by saving the following file as SSDT1.dat (base64 encoded here):

```
echo 'U1NEVKEAAAAB9EJPQ0hTAEJYUENTU0RUAQAAAElOVEwYEBkgoA8AFVwuX1NCX1BDSTAGABBMBi5f
U0JfUENJMFuCTwVCQVQwCF9ISUQMQdAMCghfVUlEABQJX1NUQQCkCh8UK19CSUYApBIjDQELcBcL
cBcBC9A5C1gCCywBCjwKPA0ADQANTElPTgANABQSX0JTVACkEgoEAAALcBcL0Dk=' | base64 -d > SSDT1.dat
```

Next you must add the processed file to the main domain of the virtual machine:

```
<domain xmlns:qemu="http://libvirt.org/schemas/domain/qemu/1.0" type="kvm">
  ...
  <qemu:commandline>
    <qemu:arg value="-acpitable"/>
    <qemu:arg value="file=/path/to/your/SSDT1.dat"/>
  </qemu:commandline>
</domain>
```

Make sure your XML file has the correct namespace in the `<domain>` tag as visible above, otherwise the XML verification will fail.

Source (https://www.reddit.com/r/VFIO/comments/ebo2uk/nvidia_geforce_rtx_2060_mobile_success_qemu_ovmf/)

10.4 "BAR 3: cannot reserve [mem]" error in dmesg after starting virtual machine

With respect to this article (https://www.linuxquestions.org/questions/linux-kernel-70/kernel-fails-to-assign-memory-to-pcie-device-4175487043/):

If you still have code 43 check *dmesg* for memory reservation errors after starting up your virtual machine, if you have similar it could be the case:

```
vfio-pci 0000:09:00.0: BAR 3: cannot reserve [mem 0xf0000000-0xf1ffffff 64bit pref]
```

Find out a PCI Bridge your graphic card is connected to. This will give actual hierarchy of devices:

```
$ lspci -t
```

Before starting the virtual machine run the following lines, replacing IDs with actual values from previous output.

```
# echo 1 > /sys/bus/pci/devices/0000\:00\:03.1/remove
# echo 1 > /sys/bus/pci/rescan
```

Note: Probably setting kernel parameter `video=efifb:off` is required as well. Source (https://pve.proxmox.com/wiki/Pci_passthrough#BAR_3:_can.27t_reserve_.5Bmem.5D_error)

In addition try adding kernel parameter `pci=realloc` which also helps with hotplugging issues (https://github.com/Dunedan/mbp-2016-linux/issues/60#issuecomment-396311301).

10.5 UEFI (OVMF) compatibility in VBIOS

With respect to this article (https://pve.proxmox.com/wiki/Pci_passthrough#How_to_known_if_card_is_UEFI_.28ovmf.29_compatible):

Error 43 can be caused by the GPU's VBIOS without UEFI support. To check whenever your VBIOS supports it, you will have to use `rom-parser`:

```
$ git clone https://github.com/awilliam/rom-parser
$ cd rom-parser && make
```

Dump the GPU VBIOS:

```
# echo 1 > /sys/bus/pci/devices/0000:01:00.0/rom
# cat /sys/bus/pci/devices/0000:01:00.0/rom > /tmp/image.rom
# echo 0 > /sys/bus/pci/devices/0000:01:00.0/rom
```

And test it for compatibility:

```
$ ./rom-parser /tmp/image.rom
```

```
Valid ROM signature found @600h, PCIR offset 190h
        PCIR: type 0 (x86 PC-AT), vendor: 10de, device: 1184, class: 030000
        PCIR: revision 0, vendor revision: 1
Valid ROM signature found @fa00h, PCIR offset 1ch
        PCIR: type 3 (EFI), vendor: 10de, device: 1184, class: 030000
        PCIR: revision 3, vendor revision: 0
                EFI: Signature Valid, Subsystem: Boot, Machine: X64
        Last image
```

To be UEFI compatible, you need a "type 3 (EFI)" in the result. If it is not there, try updating your GPU VBIOS. GPU manufacturers often share VBIOS upgrades on their support pages. A large database of known compatible and working VBIOSes (along with their UEFI compatibility status!) is available on TechPowerUp (https://www.techpowerup.com/vgabios/).

Updated VBIOS can be used in the virtual machine without flashing. To load it in QEMU:

```
-device vfio-pci,host=07:00.0,......,romfile=/path/to/your/gpu/bios.bin \
```

And in libvirt:

```
<hostdev>
     ...
     <rom file='/path/to/your/gpu/bios.bin'/>
     ...
   </hostdev>
```

One should compare VBIOS versions between host and guest systems using **nvflash (https://www.techpowerup.com/download/nvidia-nvflash/)** (Linux versions under *Show more versions*) or **GPU-Z (https://www.techpowerup.com/download/techpowerup-gpu-z/)** (in Windows guest). To check the currently loaded VBIOS:

```
$ ./nvflash --version
```

```
...
Version                 : 80.04.XX.00.97
...
UEFI Support            : No
UEFI Version            : N/A
UEFI Variant Id         : N/A ( Unknown )
UEFI Signer(s)          : Unsigned
...
```

And to check a given VBIOS file:

```
$ ./nvflash --version NV299MH.rom
```
```
...
Version                 : 80.04.XX.00.95
...
UEFI Support            : Yes
UEFI Version            : 0x10022 (Jul  2 2013 @ 16377903 )
UEFI Variant Id         : 0x0000000000000004 ( GK1xx )
UEFI Signer(s)          : Microsoft Corporation UEFI CA 2011
...
```

If the external ROM did not work as it should in the guest, you will have to flash the newer VBIOS image to the GPU. In some cases it is possible to create your own VBIOS image with UEFI support using **GOPUpd (https://www.win-raid.com/t892f16-AMD-and-Nvidia-GOP-update-No-requests-DIY.html)** tool, however this is risky and may result in GPU brick.

Warning: Failure during flashing may "brick" your GPU - recovery may be possible, but rarely easy and often requires additional hardware. **DO NOT** flash VBIOS images for other GPU models (different boards may use different VBIOSes, clocks, fan configuration). If it breaks, you get to keep all the pieces.

In order to avoid the irreparable damage to your graphics adapter it is necessary to unload the NVIDIA kernel driver first:

```
# modprobe -r nvidia_modeset nvidia
```

Flashing the VBIOS can be done with:

```
# ./nvflash romfile.bin
```

Warning: DO NOT interrupt the flashing process, even if it looks like it is stuck. Flashing should take about a minute on most GPUs, but may take longer.

10.6 Slowed down audio pumped through HDMI on the video card

For some users, the virtual machine's audio slows down/starts stuttering/becomes demonic after a while when it is pumped through HDMI on the video card. This usually also slows down graphics. A possible solution consists of enabling MSI (Message Signaled-Based Interrupts) instead of the default (Line-Based Interrupts).

In order to check whether MSI is supported or enabled, run the following command as root:

```
# lspci -vs $device | grep 'MSI:'
```

where `$device` is the card's address (e.g. `01:00.0`).

The output should be similar to:

```
Capabilities: [60] MSI: Enable- Count=1/1 Maskable- 64bit+
```

A `-` after `Enable` means MSI is supported, but not used by the virtual machine, while a `+` says that the virtual machine is using it.

The procedure to enable it is quite complex, instructions and an overview of the setting can be found here (https://forums.guru3d.com/showthread.php?t=378044).

On a linux guest you can use modinfo to see if there is option to enable MSI (for example: "modinfo snd_hda_intel |grep msi"). If there is, one can enable it by adding the relevant option to a custom omdprobe file - in "/etc/modprobe.d/snd-hda-intel.conf" inserting "options snd-hda-intel enable_msi=1"

Other hints can be found on the lime-technology's wiki (https://lime-technology.com/wiki/index.php/UnRAID_6/VM_Guest_Support#Enable_MSI_for_Interrupts_to_Fix_HDMI_Audio_Support) [dead link 2023-05-06 ⓘ], or on this article on VFIO tips and tricks (https://vfio.blogspot.it/2014/09/vfio-interrupts-and-how-to-coax-windows.html).

A UI tool called MSI Utility (FOSS Version 2) (https://forums.guru3d.com/threads/windows-line-based-vs-message-signaled-based-interrupts-msi-tool.378044/) works with Windows 10 64-bit and simplifies the process.

In order to fix the issues enabling MSI on the 0 function of a nVidia card (`01:00.0 VGA compatible controller: NVIDIA Corporation GM206 [GeForce GTX 960] (rev a1) (prog-if 00 [VGA controller])`) was not enough; it will also be required to enable it on the other function (`01:00.1 Audio device: NVIDIA Corporation Device 0fba (rev a1)`) to fix the issue.

10.7 No HDMI audio output on host when intel_iommu is enabled

If after enabling `intel_iommu` the HDMI output device of Intel GPU becomes unusable on the host then setting the option `igfx_off` (i.e. `intel_iommu=on,igfx_off`) might bring the audio back, please read iommu.html (https://docs.kernel.org/arch/x86/iommu.html#graphics-problems) for details about setting `igfx_off`.

10.8 X does not start after enabling vfio_pci

This is related to the host GPU being detected as a secondary GPU, which causes X to fail/crash when it tries to load a driver for the guest GPU. To circumvent this, a Xorg configuration file specifying the BusID for the host GPU is required. The correct BusID can be acquired from `lspci -n` or the Xorg log [7] (https://www.redhat.com/archives/vfio-users/2016-August/msg00025.html). Note that the value from the *lspci* output is hexadecimal and should be converted to decimal in the *.conf* file.

/etc/X11/xorg.conf.d/10-intel.conf

```
Section "Device"
        Identifier "Intel GPU"
        Driver "modesetting"
        BusID  "PCI:0:2:0"
EndSection
```

10.9 Chromium ignores integrated graphics for acceleration

Chromium and friends will try to detect as many GPUs as they can in the system and pick which one is preferred (usually discrete NVIDIA/AMD graphics). It tries to pick a GPU by looking at PCI devices, not OpenGL renderers available in the system - the result is that Chromium may ignore the integrated GPU available for rendering and try to use the dedicated GPU bound to the `vfio-pci` driver, and unusable on the host system, regardless of whenever a guest virtual machine is running or not. This results in software rendering being used (leading to higher CPU load, which may also result in choppy video playback, scrolling and general un-smoothness).

This can be fixed by **explicitly telling Chromium which GPU you want to use**.

10.10 Virtual machine only uses one core

For some users, even if IOMMU is enabled and the core count is set to more than 1, the virtual machine still only uses one CPU core and thread. To solve this enable "Manually set CPU topology" in `virt-manager` and set it to the desirable amount of CPU sockets, cores and threads. Keep in mind that "Threads" refers to the thread count per CPU, not the total count.

10.11 Passthrough seems to work but no output is displayed

Make sure if you are using virt-manager that UEFI firmware is selected for your virtual machine. Also, make sure you have passed the correct device to the virtual machine.

10.12 Host lockup after virtual machine shutdown

This issue seems to primarily affect users running a Windows 10 guest and usually after the virtual machine has been run for a prolonged period of time: the host will experience multiple CPU core lockups (see **[8] (https://bbs.archlinux.org/viewtopic.php?id=206050&p=2)**). To fix this try enabling Message Signal Interrupts on the GPU passed through to the guest. A good guide for how to do this can be found in **[9] (https://forums.guru3d.com/threads/windows-line-based-vs-message-signaled-based-interrupts.378044/)**. You can also download this application for windows here **[10] (https://github.com/TechtonicSoftware/MSIInturruptEnabler)** that should make the process easier.

10.13 Host lockup if guest is left running during sleep

VFIO-enabled virtual machines tend to become unstable if left running through a sleep/wakeup cycle and have been known to cause the host machine to lockup when an attempt is then made to shut them down. In order to avoid this, one can simply prevent the host from going into sleep while the guest is running using the following libvirt hook script and systemd unit. The hook file needs executable permissions to work.

/etc/libvirt/hooks/qemu

```
#!/bin/sh

OBJECT="$1"
OPERATION="$2"
SUBOPERATION="$3"
EXTRA_ARG="$4"

case "$OPERATION" in
        "prepare")
                systemctl start libvirt-nosleep@"$OBJECT"
                ;;
        "release")
```

```
                systemctl stop libvirt-nosleep@"$OBJECT"
                ;;
esac
```

```
/etc/systemd/system/libvirt-nosleep@.service
```

```
[Unit]
Description=Preventing sleep while libvirt domain "%i" is running

[Service]
Type=simple
ExecStart=/usr/bin/systemd-inhibit --what=sleep --why="Libvirt domain \"%i\" is running" --who=%U --mode=
block sleep infinity
```

10.14 Cannot boot after upgrading ovmf

If you cannot boot after upgrading from **edk2-ovmf (https://archlinux.org/packages/?name=edk2-ovmf)** version 1:r23112.018432f0ce-1 then you need to remove the old `*VARS.fd` file in `/var/lib/libvirt/qemu/nvram/`:

```
# mv /var/lib/libvirt/qemu/nvram/vmname_VARS.fd /var/lib/libvirt/qemu/nvram/vmname_VARS.fd.old
```

See FS#57825 (https://bugs.archlinux.org/task/57825) for further details.

10.15 Bluescreen at boot since Windows 10 1803

Since Windows 10 1803 there is a problem when you are using "host-passthrough" as cpu model. The machine cannot boot and is either boot looping or you get a bluescreen. You can workaround this by:

```
# echo 1 > /sys/module/kvm/parameters/ignore_msrs
```

To make it permanently you can create a modprobe file `kvm.conf`:

```
options kvm ignore_msrs=1
```

To prevent clogging up *dmesg* with "ignored rdmsr" messages you can additionally add:

```
options kvm report_ignored_msrs=0
```

10.16 AMD Ryzen / BIOS updates (AGESA) yields "Error: internal error: Unknown PCI header type ‘127’"

AMD users have been experiencing breakage of their KVM setups after updating the BIOS on their motherboard. There is a kernel patch (https://clbin.com/VCiYJ), (see Kernel/Arch build system for instruction on compiling kernels with custom patches) that can resolve the issue as of now (7/28/19), but this is not the first time AMD has made an error of this very nature, so take this into account if you are considering updating your BIOS in the future as a VFIO user.

10.17 AMD GPU not resetting properly yielding "Error: internal error: Unknown PCI header type ‘127’" (Separate issue from the one above)

Passing through an AMD GPU may result into a problem known as the "AMD reset bug". Upon power cycling the guest, the GPU does not properly reset its state which causes the device to malfunction until the host is also rebooted. This is usually paired with a "code 43" driver error in a Windows guest, and the message "Error: internal error: Unknown PCI header type '127'" in the libvirt log on the host.

In the past, this meant having to use work-arounds to manually reset the GPU, or resorting to the use of kernel patches that were unlikely to land in upstream. Currently, the recommended solution that does not require patching of the kernel is to install `vendor-reset-git (https://aur.archlinux.org/packages/vendor-reset-git/)`AUR or `vendor-reset-dkms-git (https://aur.archlinux.org/packages/vendor-reset-dkms-git/)`AUR and making sure the 'vendor-reset' kernel module is loaded before booting the guest. For convenience, you can load the module automatically.

Note: Make sure you do not have any of the AMD reset bug kernel patches installed if you are using `vendor-reset-git (https://aur.archlinux.org/packages/vendor-reset-git/)`AUR or `vendor-reset-dkms-git (https://aur.archlinux.org/packages/vendor-reset-dkms-git/)`AUR.

10.18 Host crashes when hotplugging Nvidia card with USB

If attempting to hotplug an Nvidia card with a USB port, you may have to blacklist the `i2c_nvidia_gpu` driver. Do this by adding the line `blacklist i2c_nvidia_gpu` to `/etc/modprobe.d/blacklist.conf`.

10.19 Host unable to boot and stuck in black screen after enabling vfio

If debug kernel messages during boot are enabled by adding with the `debug ignore_loglevel` kernel parameters, you may see boot stuck with the last message similar to:

```
vfio-pci 0000:01:00.0: vgaarb: changed VGA decodes: olddecodes=io+mem,decodes=io+mem:owns=none
```

This can be mitigated by disconnecting the passed-through GPU from your monitor. You may reconnect the passed-through GPU to a monitor after the host has booted.

If you do not want to plug the cable in each time you boot the host. You can disable the framebuffer in your boot loader to bypass this message. For UEFI systems you can add `video=efifb:off` as a kernel parameter. For legacy support, use `video=vesafb:off` instead or in conjunction. Note that doing this may cause issues with Xorg.

If you encounter problems with Xorg, the following solution may help (remember to substitute with your own values if needed).

```
/etc/X11/xorg.conf.d/10-amd.conf

Section "Device"
        Identifier "AMD GPU"
        Driver "amdgpu"
        BusID  "PCI:0:2:0"
EndSection
```

10.20 AER errors when passing through PCIe USB hub

In some cases passing through a PCIe USB hub, such as one connected to the guest GPU, might fail with AER errors similar to the following:

```
kernel: pcieport 0000:00:01.1: AER: Uncorrected (Non-Fatal) error received: 0000:00:01.1
kernel: pcieport 0000:00:01.1: AER: PCIe Bus Error: severity=Uncorrected (Non-Fatal), type=Transaction La
yer, (Requester ID)
kernel: pcieport 0000:00:01.1: AER:   device [8086:1905] error status/mask=00100000/00000000
kernel: pcieport 0000:00:01.1: AER:    [20] UnsupReq               (First)
kernel: pcieport 0000:00:01.1: AER:   TLP Header: 00000000 00000000 00000000 00000000
kernel: pcieport 0000:00:01.1: AER: device recovery successful
```

10.21 Reserved Memory Region Reporting (RMRR) Conflict

If you run into an issue passing through a device because of the BIOS's usage of RMRR, like the error below.

```
vfio-pci 0000:01:00.1: Device is ineligible for IOMMU domain attach due to platform RMRR requirement. Con
tact your platform vendor.
```

You can try the patches here: **https://github.com/kiler129/relax-intel-rmrr**

10.22 Too-low frequency limit for AMD GPU passed-through to virtual machine

On some machines with AMD GPUs, binding the devices to vfio-pci may be insufficient to prevent interference from the host, since the amdgpu driver on the host may query global ATIF methods which can alter the behavior of the GPU. For example, a user with a Dell Precision 7540 laptop containing a Radon Pro WX 3200 AMD GPU reported that, with the AMD GPU bound to vfio-pci, the passed-through AMD GPU was limited to 501 MHz instead of the correct 1295 MHz limit. **Blacklisting** the amdgpu kernel module using the kernel command line was a workaround.

See **this kernel mailing list discussion (https://lore.kernel.org/regressions/092b825a-10ff-e197-18a1-d3e3a097b0e3@leemhuis.info/T/)** for further details.

10.23 Host clocksource is HPET rather than TSC

If the clocksource of your host machine is HPET you may see severely crippled performance in games. While this issue also occurs when running games outside of a virtual machine, unaware users will likely begin troubleshooting their virtual machine when the issue is actually with their host system.

To see your current clocksource, run this command on your host machine:

```
$ cat /sys/devices/system/clocksource/clocksource*/current_clocksource
```

If your clocksource is HPET you can try to force it to TSC by setting the kernel parameters `clocksource=tsc` and `tsc=reliable`. See this Reddit thread (https://www.reddit.com/r/linux_gaming/comments/rsvjqb/psa_if_your_clocksource_is_hpet_rather_than_tsc/) for more information.

10.24 Code 43 while Resizable Bar is turned on in the bios

If you own a GPU that supports Resizable BAR / SAM and have the corresponding BIOS option enabled, you might get code 43 because this feature is disabled in QEMU [11] (https://github.com/qemu/qemu/commit/3412d8ec9810b819f8b79e8e0c6b87217c876e32). Linux Kernel release 6.1 added option to manipulate PCIe Resizable BARs through sysfs, so you can try permanently resizing it using a udev rule:

```
/etc/udev/rules.d/01-amd.rules
```

```
ACTION=="add", SUBSYSTEM=="pci", ATTR{vendor}=="0x1002", ATTR{device}=="0x73bf", ATTR{resource0_resize}
="14"
ACTION=="add", SUBSYSTEM=="pci", ATTR{vendor}=="0x1002", ATTR{device}=="0x73bf", ATTR{resource2_resize}
="8"
```

Alternatively, to temporarily resize the PCIe BAR, you can write the new size to `/sys/bus/pci/devices/`*`device_address`*`/resource`*`number`*`_resize`, where *device_address* is the address of your GPU, such as `0000:03:00.0` (note that for changing this parameter, no driver can be loaded for the device).

You can confirm values using `lspci -vvvxxxx`.

10.25 Operating system hangs

If your virtualized OS (particularly Windows) hangs when the GPU is attached, you can also try opening your motherboard's settings and:

- disabling `Re-Size BAR Support`,
- disabling `Above 4G memory / Crypto Currency mining`,
- enabling `SR-IOV`,
- changing `Max TOLUD` from `dynamic` to a specific value,
- switching your `Initiate Graphic Adapter` from `PEG` to `IGD`.

At the very least, VEGA cards seem to be picky about those settings and won't actually work unless you change them.

11 See also

- **VFIO users mailing list (https://www.redhat.com/archives/vfio-users/)**
- **/r/VFIO: A subreddit focused on vfio (https://www.reddit.com/r/VFIO)**
- **GVT-d: passthrough of an entire integrated GPU (https://github.com/intel/gvt-linux/wiki/GVTd_Setup_Guide)**