

GPUs for Machine Learning on VMware vSphere

Insights and step-by-step setup instructions
for VMware vSphere administrators

Table of Contents

Introduction	3
1. VMware vSphere Overview	4
2. Why Virtualized GPUs Work Best for Machine Learning	4
Neural Networks: Why GPUs and Machine Learning Are a Good Match	8
3. Getting Started Using GPUs with VMs on vSphere	10
4. Choosing Methods to Configure GPUs With VMs	11
5. GPU Method 1: VMDirectPath I/O (Passthrough)	12
5.1. How to Enable a GPU Device in Passthrough Mode on vSphere	13
5.2. Virtual Machine Setup for DirectPath I/O Use of GPUs	14
5.3. Multiple GPUs Assigned to One VM	18
6. GPU Method 2: The NVIDIA GRID	19
6.1 NVIDIA GRID vGPU Setup on the vSphere Host Server	20
6.2 Choosing the vGPU Profile for the VM	24
6.3 VM Guest OS NVIDIA GRID Driver Installation	24
6.4 Install and Test the CUDA Libraries	26
7. GPU Method 3: Bitfusion FlexDirect	27
7.1 The Bitfusion FlexDirect Architecture	28
7.2 Bitfusion FlexDirect: Installation and Setup	28
8. Performance of Machine Learning Workloads Using GPUs	34
Resource Repository	39
About the Authors	42

ABOUT GPU USAGE

There are many forms of machine learning, and not all of them require accelerator technology such as GPUs. For example, logistic regression and gradient-boosted machine techniques perform very well on CPUs. Deep learning uses deep neural networks (DNNs) as the learning technology, which require GPUs or other accelerators to improve their performance. GPUs can also be applied to techniques outside of DNNs for acceleration, but their prime usage today is in accelerating the training phase when using DNNs.

SUCCESS STORY

How The University of Groningen Implemented a Robust, Shared Virtual Infrastructure for Machine Learning Workloads.

[>Read the blog post.](#)

Introduction

Machine learning is a subset of the broader field of artificial intelligence, which uses statistical techniques to allow programs to learn from experiences or from existing data.

Deep learning is a machine learning technique that enables computers to learn from example data. It's what supports autonomous vehicles or medical image recognition, for example. The recent rise of deep learning platforms has led to an exponential growth in these workloads, in both data centers and in cloud environments. GPUs provide the computing power needed to run deep learning and other machine learning programs efficiently, reliably, and quickly. These GPU-based workloads are even more versatile, flexible, and efficient when they run in virtual machines on the VMware ESXi™ hypervisor.

Administrators have options to consider and decisions to make when it comes to how to configure GPUs through virtual machines (VMs). This guide covers three common approaches: VMware VMDirectPath I/O (Passthrough), NVIDIA GRID vGPU setup, and Bitfusion FlexDirect.

Read on for a detailed description of each approach, step-by-step setup instructions, and links to other helpful resources to help you succeed using GPUs for machine learning on VMware vSphere*.

1. VMware vSphere Overview

Machine learning technologies are innovating at an extremely rapid pace, and that means the supporting compute platforms they work with must be flexible enough to accommodate the speed of change.

VMware enables administrators to build, run, and quickly update machine learning environments and associated data infrastructure using the following products:

VMware *vSphere*

VMware vSphere is the industry-leading server virtualization software and the heart of a modern *software-defined data center* (SDDC), helping you run, manage, connect, and secure your applications in a common operating environment across clouds.

VMware *vSAN*

VMware vSAN™ powers industry-leading hyperconverged infrastructure (HCI) solutions with vSphere-native, flash-optimized storage for private and public cloud deployments.

VMware *NSX*

VMware NSX® Data Center delivers virtualized networking and security entirely in software, completing a key pillar of the SDDC, and enabling the *virtual cloud network* to connect and protect across data centers, clouds, and applications.

2. Why Virtualized GPUs Work Best for Machine Learning

A GPU is most commonly associated with graphics-intensive applications such as 3D modeling software or virtual desktop infrastructure (VDI). General-purpose GPUs (GPGPUs), rather than CPUs, are used to accelerate computational workloads in modern high-performance computing (HPC) and machine learning or deep learning landscapes.

However, there are many different data types being used in machine learning, and it is important for you to understand what types your organization is dealing with. Large quantities of business data are contained in structured or semi-structured form (called IID data) such as database tables, spreadsheets, or tabular data in general.

These forms of data lend themselves well to being analyzed using machine learning algorithms such as logistic regression, random forest, XGBoost, and KMeans clustering techniques. VMware has done a lot of work in this area. These particular use cases may not benefit a great deal from using GPU performance to accelerate the training time taken. Structured data exercises like these can be done effectively with the traditional CPU-based computing.

You should ask your data scientists what types of data they intend to use for training and what types of ML models will be used the most. If they follow the above pattern, then GPUs may be a luxury for those cases.

If however, the data science team responds that they are making use of deep neural network structures to train on image, voice, text, or generally unstructured data, then GPUs will become a necessity.

Here are the key reasons why:

1. Latency vs. Throughput

A CPU is optimized to finish a task as quickly as possible—at a latency that is as low as possible—while also quickly switching between operations. Its focus is on processing tasks in a serialized way.

A GPU is optimized for throughput, and also focuses on pushing through as many operations as possible at the same time. It does so by parallel processing each task.

Figure 1 shows an exemplary ‘core’ count of both a CPU and a GPU. Note that the key difference between the two is the number of cores in the GPU, which can be used for parallel processing a task.

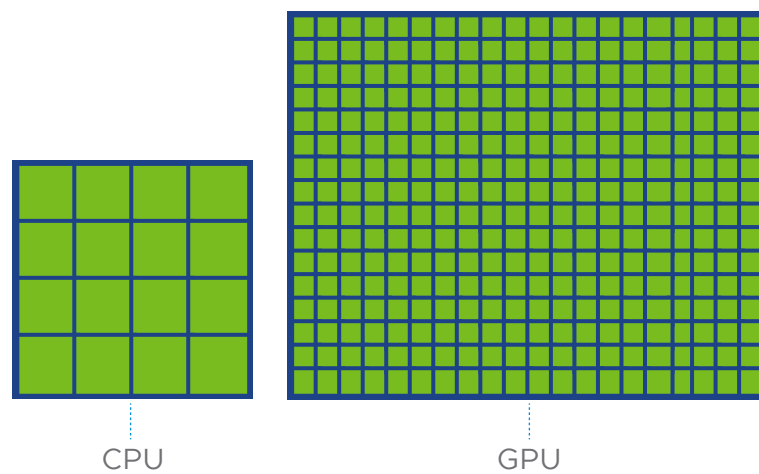


Figure 1: A GPU has significantly more cores than a CPU, which can be used for processing tasks in parallel.

2. Architecture

While CPUs and GPUs both use the memory constructs of cache layers, memory controller, and global memory, CPU architectures use significant cache memory levels for low-latency memory access.

CPU

In this diagram of a generic, memory-focused, modern CPU package (note that the precise layout strongly depends on vendor and model), a single CPU package consists of cores that contain separate data and instruction level-1 caches, supported by the level-2 cache. The level-3 cache, or last-level cache, is shared across multiple cores.

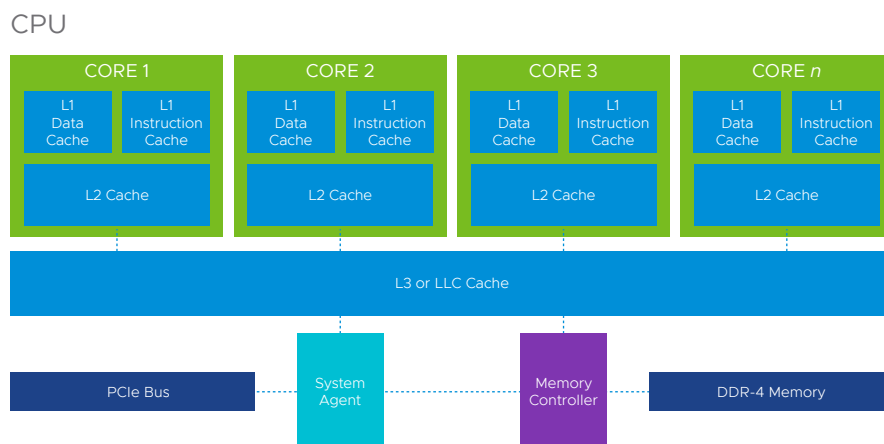


Figure 2: Diagram of a generic CPU architecture.

If data is not residing in the cache layers, the CPU will fetch the data from the global DDR-4 memory. In modern CPU packages, the number of cores per CPU can go up to 28 or 32, and run up to 2.5 GHz or 3.8 GHz in turbo mode (depending on make and model). Cache sizes range up to 2 MB L2 cache per core.

GPU

In this diagram of a generic GPU architecture (again, the layout is dependent on vendor and model), the focus is more on core availability than low-latency cache memory access. It's optimized for data parallel throughput computations.

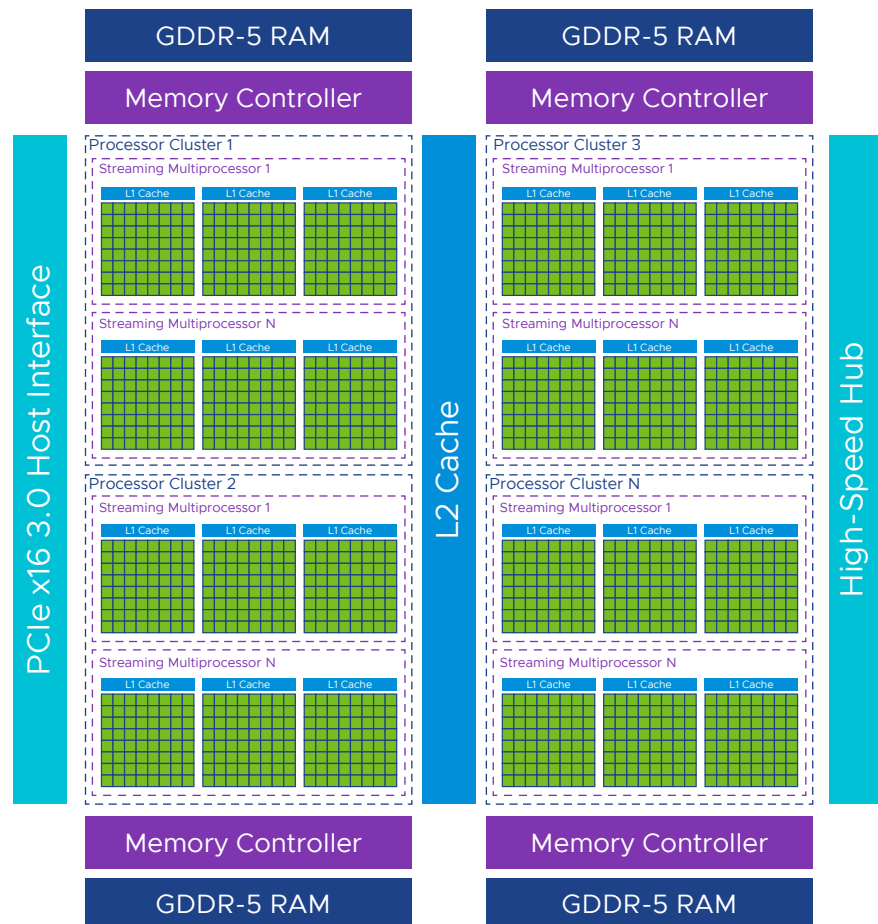


Figure 3: Diagram of a generic GPU architecture.

A single GPU device consists of multiple processor clusters (PCs) that contain multiple streaming multiprocessors (SMs). Each SM accommodates an L1 instruction cache layer with its associated cores. Typically, one SM uses a dedicated L1 cache and a shared L2 cache before pulling data from global GDDR-5 memory. The GPU architecture is tolerant of memory latency as it is more designed for higher throughput.

Compared to a CPU, a GPU works with fewer, relatively small memory cache layers because it has more components dedicated to computation. As a result, it's less concerned with how long it takes to retrieve data from memory. The potential memory access 'latency' is masked as long as the GPU has enough computations at hand, keeping it busy.

The number of cores reveals the parallelism capabilities. In a current NVIDIA flagship offering, the Tesla V100, one GPU device contains 80 SMs, each containing 64 cores making a total of 5,120 cores. Tasks are scheduled to processor clusters and SMs rather than individual cores, which is how it is able to process in parallel. Combining this powerful hardware device with a programming framework such as TensorFlow/Keras or PyTorch enables applications to fully utilize the computing power of a GPU.

Neural Networks: Why GPUs and Machine Learning Are a Good Match

The machine learning programming frameworks, such as TensorFlow, PyTorch, Keras, and others, hide the complexity of the detailed GPU CUDA instructions from the developer, and present a higher-level API for access to GPUs. Another key benefit of those programming frameworks is that they simplify the creation and training of a neural network model. Neural networks are an important software component of machine learning that improve model performance in the training phase, especially when the data being used for training and inference is made up of images, video, voice, or textual data.

Deep learning is a technique within the broader machine learning field that uses these neural networks extensively. The training datasets used with neural network models differ in structure from that of more traditional tabular data, often seen in relational databases, CSV files, and spreadsheets. In those structured tables, there are rows and columns where every column has an identifiable feature name, and features may have dependencies between them. Graphics, voice, or text data, on the other hand, do not have that same structure to begin with. The data also often has higher dimensionality. GPUs for executing neural networks can accelerate the training phase for such non-tabular data by an order of magnitude, which is a big benefit to the data scientist, who can then do more experiments with models in any one period of time.

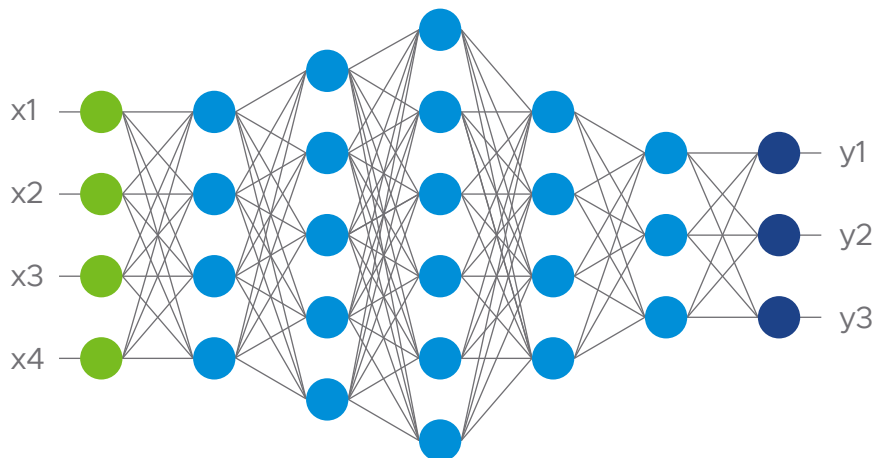


Figure 4: Neural networks are a component of machine learning that improve application performance in the training phase, when the data is made up of images, video, voice, or text.

As illustrated in Figure 4, neural networks are comprised of layers of nodes (the colored circles) arranged into connected layers, which are represented by the columns. The lines between the nodes show how data flows between them. When there are many hidden layers (blue circles) operating between the input (green) and output (dark blue) layers, we say we have a “deep neural network.”

A significant amount of computation is done repeatedly on each network node—several million times in one training run. The input data, which is made up of images, voice, or text that is converted to numeric representations, is fed into the green layer of neurons and flows from left to right through the neural network repeatedly. Next, further derived data is propagated backwards through the layers from right to left in a process called back-propagation.

ADDITIONAL RESOURCES

Check out these blog posts for a deeper dive into GPUs for machine learning.

- [Machine Learning with GPUs on vSphere](#)
- [Why the Data Scientist and Data Engineer Need to Understand Virtualization in the Cloud](#)
- [Running Common Machine Learning Use Cases on vSphere Leveraging NVIDIA GPU](#)
- [Machine Learning with H2O – the Benefits of VMware](#)

This computation, as shown in Figure 5, occurs within each node in parallel. Note how multiple values of “x” (the input data coming from a previous node) are multiplied by different values of “w” (the weight values). The neural network is adjusting those weight values (w) over time in the training process. A sum, identified by the sigma symbol, is done of all the multiplied values, so there is a matrix multiplication followed by a sum. Finally, an activation function (f) is applied to that sum, giving non-linearity in the network.

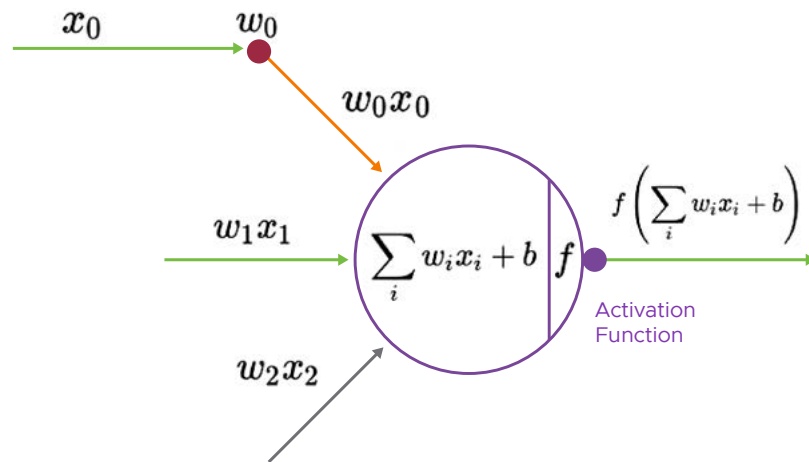


Figure 5: Multiple values of “x” (the input data coming from a previous node) are multiplied by different values of “w” (the weight values).

The fact that each of these operations can be done in parallel, on many cores that live on the GPU, is what makes it ideal for doing thousands or millions of these calculations in parallel. It’s a core reason why GPUs are so useful for neural networks to operate at speed and so powerful in deep learning, and the reason why GPUs are such a desirable hardware accelerator for this type of workload.

Q.

Will an application using a GPU within a vSphere VM perform at the same level as one running on bare metal?

A.

An application using a GPU within a vSphere VM can deliver near bare-metal performance, with overheads in the low single digits, if any.

3. Getting Started Using GPUs with VMs on vSphere

Increasingly, data scientists and machine learning developers are asking their systems administrators to provide them with a GPU-capable machine setup so they can execute workloads that need GPU power. The data scientist often describes these workloads as machine “training,” “inference,” or “development.”

The reason they need GPU capability is simply for faster time to results. Machine learning models, especially deep neural networks, involve a very large number of matrix multiplications that can be done in parallel, and GPUs can compute these operations much faster than CPUs.

GPUs are commonly used today for highly graphical applications on the desktop. Organizations already using desktop VMs on vSphere can also use their existing GPUs with vSphere for applications other than this virtual desktop infrastructure (VDI) scenario. This non-graphical use case is known as a “Compute” workload in vSphere, and it enables end users to consume GPUs in VMs in the same way they do in any GPU-enabled public cloud instance or on bare metal, but with more flexibility. Through collaboration with VMware technology partners, vSphere allows flexible consumption and multiple GPU utilization models that can increase the ROI of this infrastructure, while providing end users with exactly what they need.

4. Choosing Methods to Configure GPUs With VMs

There are different ways of configuring GPUs with VMs, and deciding which method to choose largely depends on who will be using the GPUs, and what type of applications they'll be supporting.

Following is a chart of GPU configurations and corresponding use cases:

GPU Configuration	Use Cases
A full GPU is dedicated to 1 VM	<ul style="list-style-type: none"> • Data science workstation for development and training of machine models
Multiple GPUs used by 1 VM	<ul style="list-style-type: none"> • High-end machine model training • High-performance computing (genomic sequencing, Monte-Carlo analysis) • GPU-enabled databases
Shared GPUs across multiple VMs (including partial use of a GPU)	<ul style="list-style-type: none"> • Development and testing of machine learning applications • Small data science workstations • The inference phase of machine learning

For each configuration, there are also implementation technology options to consider. Note that select use cases are enabled by third-party VMware partner technology providers.

Compute Workloads: Decision Tree for Different GPU Use Cases

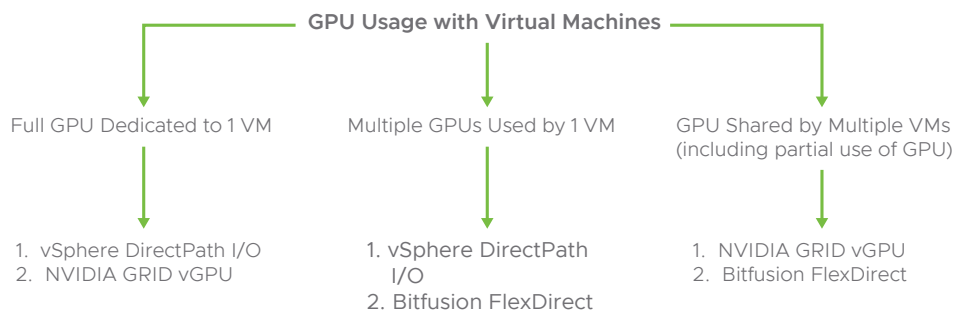


Figure 6: The high-level decisions to be made in adopting GPUs with VMs, showing possible implementation technologies.

Note: These three methods are not always mutually exclusive. For example, you can combine the Passthrough method with the Bitfusion method in a single design.

In passthrough mode, a single VM can make use of multiple physical GPUs. Check out the blog post [“Machine Learning using Virtualized GPUs on VMware vSphere”](#) for more information.

5. GPU Method 1: VMDirectPath I/O (Passthrough)

The VMDirectPath I/O mode of operation—also called passthrough—allows the GPU device to be accessed directly by the guest operating system (OS), bypassing the ESXi hypervisor.

The top three reasons for choosing the passthrough configuration approach are to

- Begin the process of exposing GPUs in VMs to move end users away from storing data and executing workloads on physical workstations
- Dedicate a full physical GPU to one virtual machine’s use exclusively (a common request from data scientists)
- Replicate a public cloud instance of an application, but using a private cloud setup

The passthrough option works without any third-party software driver being loaded into the ESXi hypervisor.

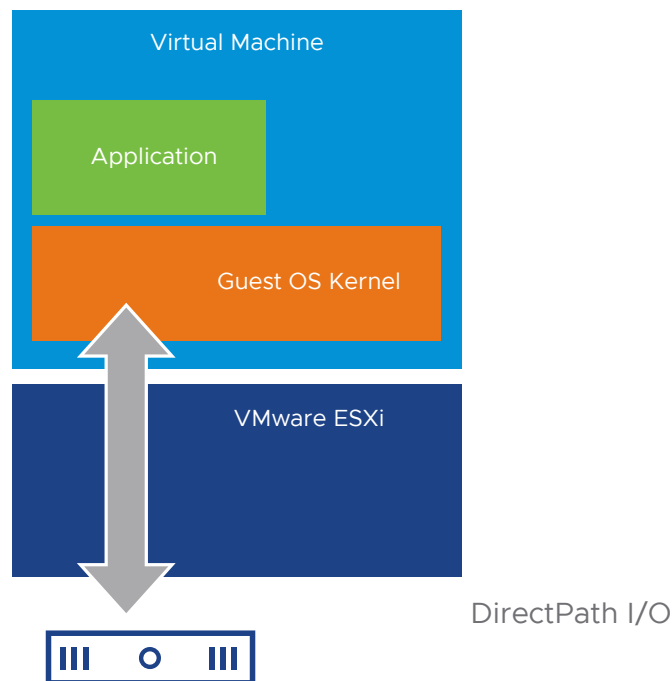


Figure 7: Outline architecture of VMDirectPath I/O or “passthrough” mode.

The passthrough mode of operation has been shown to produce performance that is within 4-5 percent of an equivalent native setup. Each GPU device in passthrough mode is dedicated to a VM, meaning there is no sharing of GPUs across other VMs. However, it’s important to note that select vSphere features including VMware vSphere® vMotion®, VMware vSphere® Distributed Resource Scheduler™ (DRS), and VM Snapshots are not compatible with this option.

5.1. How to Enable a GPU Device in Passthrough Mode on vSphere

Start by confirming that your GPU device is supported by your host server vendor, and that it can be used in “passthrough” mode on the server. In general, most GPU devices can be used in this mode.

Next, confirm that your PCI GPU device can map memory regions greater than 16 GB. High-end GPU cards—such as the Tesla P100—typically require at least this level of memory mapping. Memory mappings are specified in the PCI BARs (Base Address Registers) for the device.

Check out the article [“How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads”](#) for more information.

Note: If your GPU card does NOT need PCI MMIO regions that are larger than 16 GB, then skip section 5.1.1 (Host BIOS Setting), as well as sections 5.2.1 (Configuring EFI or UEFI Boot Mode) and 5.2.2 (Adjusting the Memory Mapped I/O Settings for the VM) of this section.

5.1.1 Host BIOS Setting

If you have a GPU device that requires 16 GB or above of memory mapping, then find and enable the vSphere host server’s BIOS setting for “above 4G decoding” or “memory mapped I/O above 4 GB” or “PCI 64-bit resource handling above 4G.” This setting is typically found in the PCI section of the BIOS menu for the server, but note that the terminology varies by vendor.

5.1.2 Editing the PCI Device Availability on the Host Server

The vSphere hypervisor will recognize an installed PCI-compatible GPU hardware device at server boot-up time without having any specific drivers installed into the hypervisor.

VMware vSphere recognizes all PCI devices in this way, and you can see the list of PCI devices found in the vSphere Client by choosing the host server you are interested in, and then following the menu choices **Configure > Hardware > PCI Devices > Edit** to see the list, as seen in an example in Figure 8.

If the particular GPU device has not been previously enabled for DirectPath I/O, then you can place the GPU device in Direct Path I/O Passthrough mode by clicking the check-box on the device entry as seen in this NVIDIA device example in Figure 8:

Edit PCI Device Availability | **sc2esx01.vslab.local** ✕

ID	Status	Vendor Name	Device Name	ESX/ESXi Device
0000:80:02.0	Not Configurable	Intel Corporation	Xeon E7 v4/Xeon E5 ...	
<input checked="" type="checkbox"/> 0000:82...	Available	NVIDIA Corporation	GP100GL [Tesla P100...	
0000:80:03.0	Not Configurable	Intel Corporation	Xeon E7 v4/Xeon E5 ...	

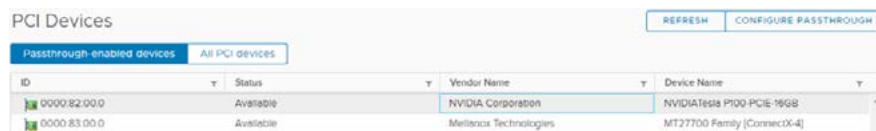
Figure 8: Edit the PCI device for DirectPath I/O availability.

Click **OK** to save this edit, then reboot your host server.

After the server is rebooted, you use the menu sequence:

Configure > Hardware > PCI Devices in the vSphere Client to access the Passthrough-enabled devices tab.

You should now see the devices that are enabled for DirectPath I/O, as shown in Figure 9:



The screenshot shows the 'PCI Devices' configuration window in the vSphere Client. It has two tabs: 'Passthrough-enabled devices' (selected) and 'All PCI devices'. There are two buttons at the top right: 'REFRESH' and 'CONFIGURE PASSTHROUGH'. Below the tabs is a table with the following data:

ID	Status	Vendor Name	Device Name
0000:82:00.0	Available	NVIDIA Corporation	NVIDIA Tesla P100-PCIe 16GB
0000:83:00.0	Available	Mellanox Technologies	MT27700 Family (ConnectX-4)

Figure 9: The DirectPath I/O enabled devices in the “Configure->Hardware->PCI Devices” screen in the vSphere Client.

5.2. Virtual Machine Setup for DirectPath I/O Use of GPUs

Start by creating the new VM in the vSphere Client following standard protocol. First turn your attention to the boot options for the VM.

5.2.1 Configuring EFI or UEFI Boot Mode

Before installing the guest OS into the VM, verify the Boot Options configuration for the Firmware field shown in Figure 10:

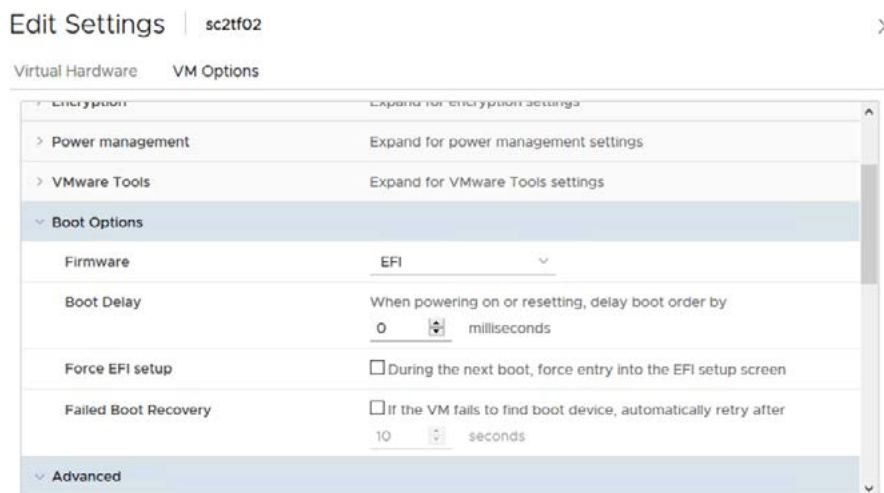


Figure 10: Enabling EFI in the Boot Options for the VM in the vSphere Client.

Your guest OS in the VM must boot in EFI or UEFI mode for correct GPU use. To access the setting for this, highlight your VM in the vSphere Client and use the menu items **Edit Settings > VM Options > Boot Options** in order to get to the Firmware parameter.

You may see **EFI in legacy compatibility mode** in the Firmware section.

More information can be found on page 3, table 7 of the [NVIDIA P100 GPU product brief](#).

5.2.2 Adjusting the Memory Mapped I/O Settings for the VM

PCI GPU devices that map memory regions greater than 16 GB in size require that two specific configuration parameters be set in the vSphere Client.

These memory requirements can be different for the various GPU models. Consult your GPU vendor's documentation to determine the PCI Base Address Registers' (BARs) memory requirements before proceeding.

If you cannot determine from the vendor documentation how much memory your GPU device maps, then follow this process:

1. Enable the GPU device in passthrough mode in the VM (without the settings in this section).
2. Boot up the VM. (This process will fail due to the GPU hardware being present.)
3. Examine the VM's `vmware.log` file to find those entries that resemble the data shown in Figure 11.

```
2017-03-07T07:40:38.467Z| vmx| I125: PCIPassthru: Device
0000:09:00.0 barIndex 0 type 2 realaddr 0xc6000000 size
16777216 flags

02017-03-07T07:40:38.467Z| vmx| I125: PCIPassthru: Device
0000:09:00.0 barIndex 1 type 3 realaddr 0x3b800000000 size
17179869184 flags 12

2017-03-07T07:40:38.467Z| vmx| I125: PCIPassthru: Device
0000:09:00.0 barIndex 3 type 3 realaddr 0x3bc00000000 size
33554432 flags 12
```

Figure 11: Entries from the `vmware.log` file for a VM that has attempted to use a GPU (in passthrough mode).

All three entries from the VM's log file shown refer to the same PCI GPU device, which is located at device address `0000:09:00.0`. Here, the GPU device has requested to map a total of just over 16 GB—the sum of the three sizes shown before the word `flags` (in bytes).

The following instructions are only for GPU cards that map more than 16 GB of memory (referred to as “high-end” cards).

In the vSphere Client again, choose the VM and use the options **Edit Settings > VM Options > Advanced > Configuration Parameters > Edit Configuration** to get to the list of PCI-related options. This is shown in Figure 12.

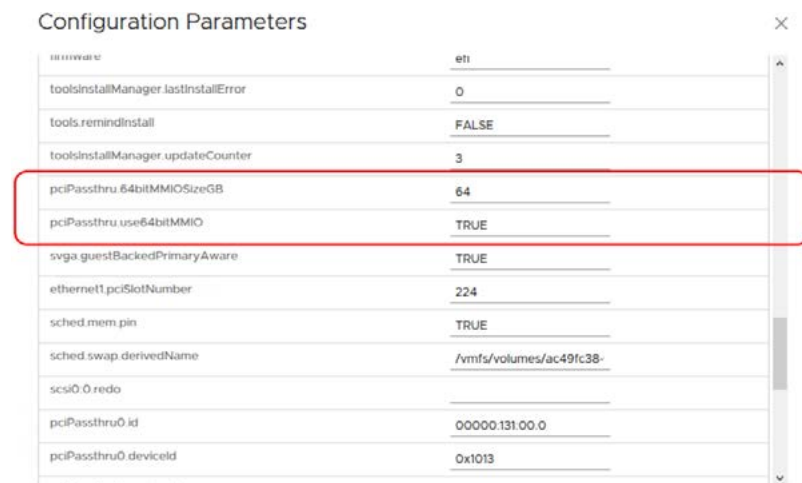


Figure 12: vSphere Client window for setting the “pciPassthru” configuration parameters for a VM.

Add the following two parameters, with the value of the first parameter, as shown, set to TRUE.

```
pciPassthru.use64bitMMIO="TRUE"
```

The value of the second parameter is adjusted to suit your specific GPU’s requirements:

```
pciPassthru.64bitMMIOSizeGB=<n>
```

Calculate the value of the second parameter by counting the number of high-end PCI GPU devices that you intend to pass into this VM. This can be one or more GPU cards. Multiply that number by 16 and round it up to the next power of two. For example, to use passthrough mode with two GPU devices in one VM, the value would be:

$2 * 16 = 32$, rounded up to the next power of two to give 64.

The largest NVIDIA V100 device’s BAR is 32 GB. For a single V100 device being used in passthrough mode in a VM, the 64bitMMIOSizeGB parameter is 64 (rounding up to the next power of two).

5.2.3 Installing the Guest OS

Install the guest OS into the VM. This should be an EFI or UEFI-capable operating system, if your GPU card has a requirement for large PCI MMIO regions.

The vendor’s GPU driver must be installed within the guest OS. Consult your GPU vendor’s documentation for this step.

For more information on how to assign a GPU device to a VM, check out the article [“Configuring VMDirectPath I/O pass-through devices on a VMware ESX or VMware ESXi host.”](#)

5.2.4 Assigning a GPU Device to a VM

Power off the VM before assigning the GPU device to it.

To enable a VM to have access to a PCI device, in the vSphere Client, highlight the VM, use the **Edit Settings** option and scroll down to the PCI Device list. If your device is not already listed there, then use the **Add New Device** button to add it to the list. Once added, your VM settings should look similar to those shown in Figure 13. In this example, the relevant entry is PCI Device 0.

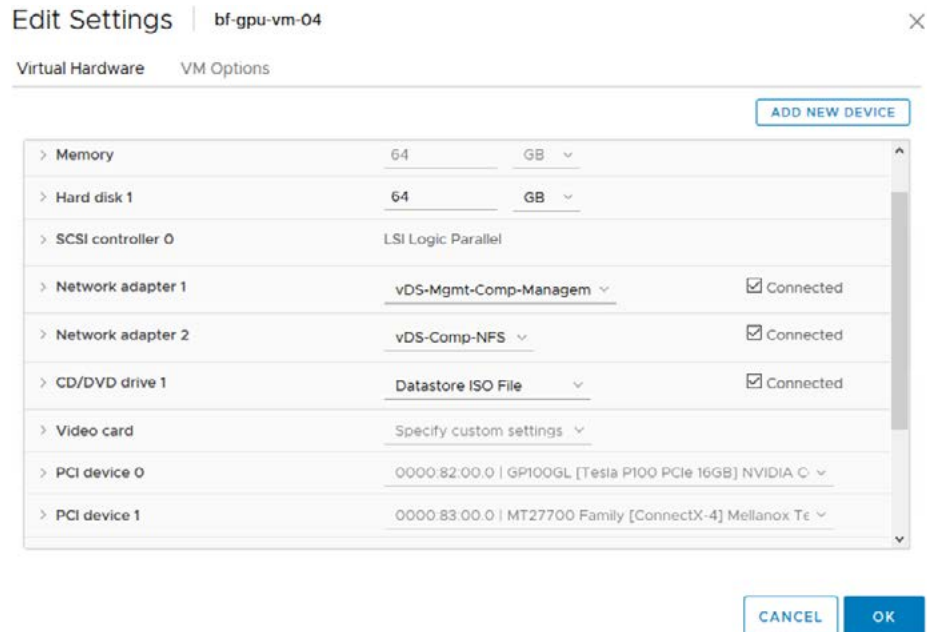


Figure 13: PCI devices that are enabled in a VM using “Edit Settings.”

5.2.5 Memory Reservation

Note that when the PCI device is assigned to a VM, the VM must have a memory reservation for the full configured memory size for the VM. This is done in the vSphere Client by choosing the VM, using **Edit Settings > Virtual Hardware > Memory** and changing the value in the Reservation area.

You are now ready to power on the VM. Next, log in to the guest OS and check that the GPU card is present using one of the following methods.

- (a) On Linux, use the command

```
"lspci | grep nvidia"
```
- (b) On a Windows operating system, use the Device Manager from the Control Panel to check the available GPU devices.

The GPU is now ready for application use in passthrough mode.

5.2.6 How to Set Up Multiple VMs Using Passthrough Mode

The recommended approach for creating multiple identical VMs using GPUs is to first clone a VM without the PCI assignments covered in section 2.5, then assign specific device(s) to each VM.

ADDITIONAL RESOURCES

Check out these resources for a deeper dive into VMDirectPath I/O (Passthrough) setup:

- Blog Post: [How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads](#)
- Article: [Configuring VMDirectPath I/O pass-through devices on a VMware ESX or VMware ESXi Host](#)
- Article: [VMware vSphere VMDirectPath I/O: Requirements for Platforms and Devices](#)

5.3. Multiple GPUs Assigned to One VM

You can use this same process multiple times in order to dedicate more than one GPU to one VM if needed, as shown in Figure 14.

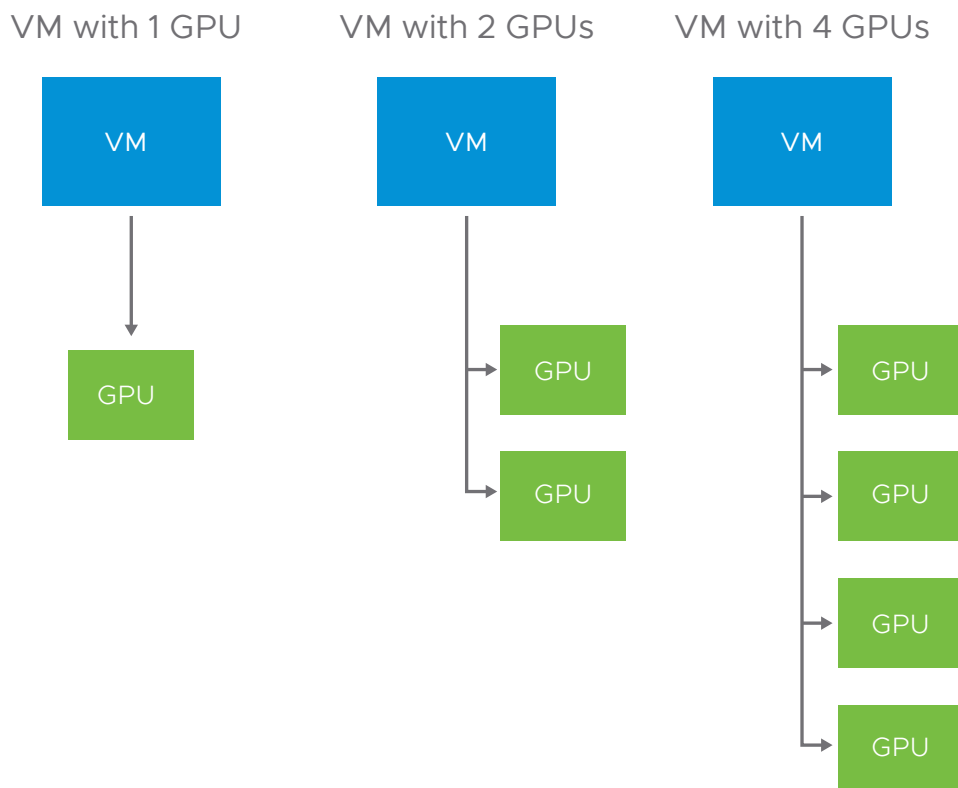


Figure 14: One or more GPUs allocated fully to one VM in passthrough mode.

Using this DirectPath I/O method, one or more full GPUs is allocated to only one VM. Multiple GPUs can be fully allocated to one VM in this way.

6. GPU Method 2: The NVIDIA GRID vGPU

The NVIDIA Quadro Virtual Data Center Workstation (vDWS) product—one of the NVIDIA GRID vGPU family of offerings—is another widely used configuration option for using GPUs on vSphere, that is separate from the DirectPath I/O method.

This guide focuses on the use of GPUs for compute workloads (such as for machine learning, deep learning, and high-performance computing applications). Although vGPUs may also be used for graphical applications that run on desktops and workstations, such as in virtual desktop infrastructure (VDI), that use case is not covered in this guide.

The NVIDIA GRID vGPU software includes two separate components:

- The NVIDIA management server (the NVIDIA Virtual GPU Manager) that is loaded as a VMware Installation Bundle (VIB) into the VMware vSphere hypervisor.
- A separate guest OS NVIDIA driver that is installed within the guest OS of your VM (the “guest VM driver”).

Figure 15 illustrates the relationship between these two parts of the NVIDIA GRID vGPU product in the overall vSphere and VM architecture.

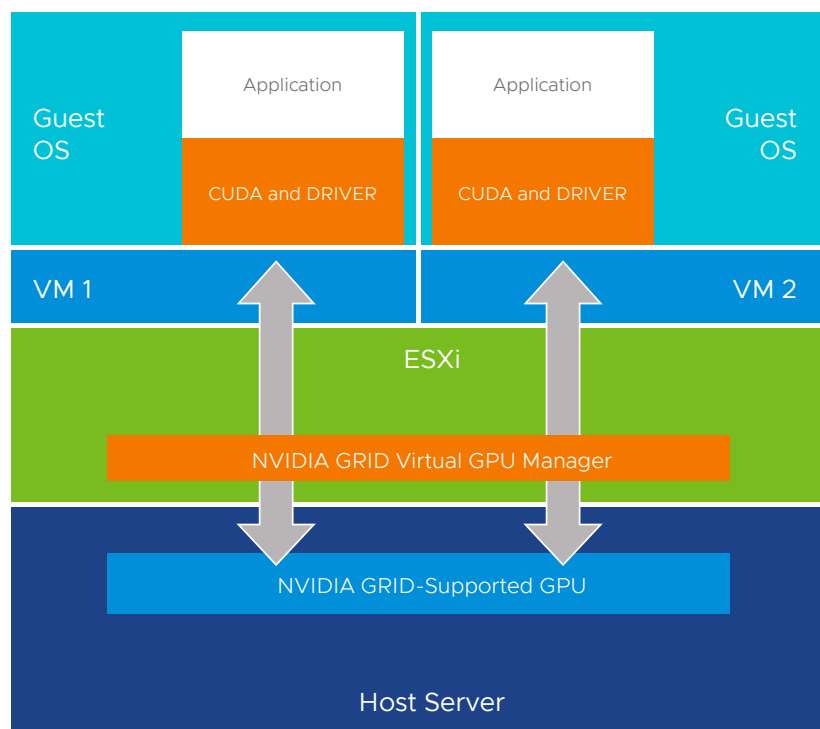


Figure 15: Diagram of the NVIDIA components in the vSphere and VM architecture.

By using the NVIDIA GRID vGPU technology with vSphere, you either can dedicate one full GPU device to one VM, or share a GPU device across multiple VMs, by using vGPU profiles.

VMware recommends using vSphere 6.7 or later. If you are running [vSphere 6.7 Update 1](#), then you will be able to leverage vMotion with your GPU-enabled VMs. If you choose to use vSphere 6.5, then ensure you are on update 1 before proceeding. Carefully review the prerequisites and other details in the [NVIDIA GRID vGPU Software User Guide](#).

There are three reasons for choosing the NVIDIA GRID vGPU option for GPU support:

- The applications in your VMs do not need the power of full GPU and can share a GPU.
- Multiple teams need to use a limited number of physical GPU devices at the same time.
- You want the flexibility to switch between dedicating a full GPU device to one VM and allowing partial use of a GPU to a VM.

Consult the [NVIDIA Release Notes](#) to be sure that the versions of the vGPU Manager and guest VM drivers that you install are compatible.

6.1 NVIDIA GRID vGPU Setup on the vSphere Host Server

To set up the NVIDIA GRID vGPU environment, you will need the licensed NVIDIA GRID vGPU product (including the VIB for vSphere and the guest OS driver), and administrator login access to the console of your vSphere ESXi machine in order to perform the following steps.

The host server part of the NVIDIA GRID vGPU installation process makes use of a [VIB install](#) technique that is used in vSphere for installing third-party drivers into the ESXi hypervisor itself. The NVIDIA GRID Virtual GPU manager (contained in the VIB) that is to be installed into your ESXi server must be downloaded from NVIDIA. The NVIDIA GRID vGPU software can be found by searching for the NVIDIA Quadro Virtual Data Center Workstation (vDWS) product.

Follow this process to install the NVIDIA GRID Virtual GPU Manager software into the vSphere ESXi hypervisor.

6.1.1 Set the GPU Device to vGPU Mode

While a GPU card can be configured in one of two modes (vSGA or vGPU), the NVIDIA GPU card should be configured with vGPU mode. This is specifically for use of the GPU in machine learning or high-performance computing applications.

Access the ESXi host server either using the ESXi shell or through SSH. You will need to enable SSH access using the ESXi management console, as SSH is disabled by default.

6.1.2 Host Graphics Setting

To enable vGPU mode on the ESXi host, use the command line to execute this command:

```
# esxcli graphics host set --default-type SharedPassthru
```

You may also get to this setting through the vSphere Client by choosing your host server and selecting **Configure > Hardware > Graphics > Host Graphics** tab > **Edit**.

An ESXi server reboot is now required. The settings should appear as shown in Figure 16.

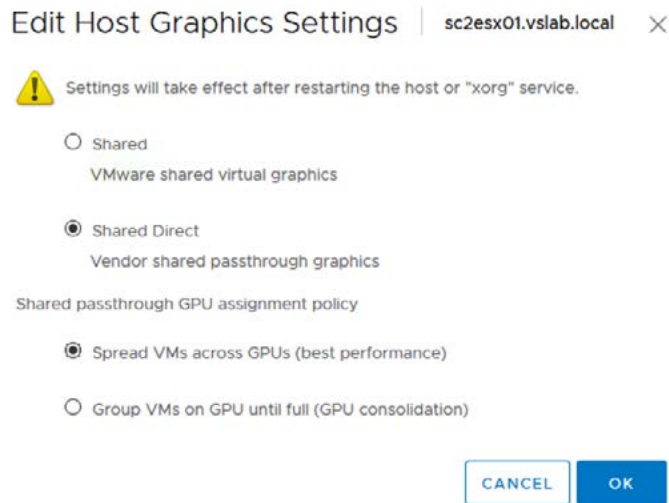


Figure 16: Change host server settings via the Edit Host Graphics Settings screen.

Using the ESXi command line, verify the settings by typing:

```
# esxcli graphics host get
Default Graphics Type: SharedPassthru
Shared Passthru Assignment Policy: Performance
```

This command shows the results given in the vSphere Client as seen in Figure 16. Note that a reboot of the host system is required after these settings have been applied.

6.1.3 Install the VIB

Before installing the VIB, place the ESXi host server into maintenance mode (that is, all VMs are moved away or quiesced).

```
# esxcli system maintenanceMode set --enable true
```

Be sure that you install the VIB *after* enabling the vGPU mode (see section 6.1.2) on your ESXi host. Otherwise you can't enable Shared Direct in the vSphere Client UI for this device once the VIB is installed.

To install the VIB, use a command similar to the following (where the path to your VIB may differ):

```
# esxcli software vib install -v /vmfs/volumes/ARL-ESX14-DS1/NVIDIA/
NVIDIA-VMware_ESXi_6.7_Host_Driver_390.42-10EM.670.0.0.7535516.vib
Installation Result
Message: Operation finished successfully.
Reboot Required: false
VIBs Installed: NVIDIA_bootbank_NVIDIA-VMware_ESXi_6.7_Host_
Driver_390.42-10EM.670.0.0.7535516
VIBs Removed:
VIBs Skipped:
```

Take the ESXi host server out of Maintenance Mode, using this command:

```
# esxcli system maintenanceMode set --enable false
```

6.1.4 List the VIB as installed into the ESXi Hypervisor

To list the VIBs installed on the ESXi host and ensure that the NVIDIA one was done correctly, use the command:

```
# esxcli software vib list |grep -i nvidia
```

NVIDIA-VMware ESXi 6.5 Host Driver 384.43-10EM.650.0.0.4598673

6.1.5 Checking the NVIDIA Driver Operation after VIB Installation

To confirm that the GPU card and ESXi are working together correctly, use the command:

```
# nvidia-smi
```

```
+-----+
| NVIDIA-SMI 390.42                  Driver Version:390.42                |
+-----+-----+-----+-----+-----+
| GPU   Name               Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|     Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+
|    0   Tesla P100-PCIE...    On      |00000000:02:00.0 Off |                    Off |
| N/A   34C    P0      28W / 250W |16363MiB / 16383MiB |           0%      Default |
+-----+-----+-----+-----+-----+
```

6.1.6 Check GPU Virtualization Mode

To check the GPU Virtualization Mode, use the command

```
# nvidia-smi -q | grep -I virtualization
GPU Virtualization Mode
Virtualization mode           : Host VGPU
```

6.1.7 Disabling ECC

This section only applies to versions of NVIDIA Grid vGPU **before** release 9.0 (which shipped in June 2019). In release 9.0, ECC is supported. If you are using NVIDIA GRID vGPU release 9.0 or later, then you may skip this section (6.1.7).

NVIDIA GPU cards that use the Pascal architecture, such as Tesla V100, P100, P40, as well as the Tesla M6 and M60 GPUs, support error correcting code (ECC) memory for improved data integrity. However, the NVIDIA GRID vGPU software does not support ECC. You must therefore ensure that ECC memory is disabled on all GPUs when using NVIDIA GRID vGPU software. Once the NVIDIA GRID vGPU Manager is installed into vSphere ESXi, issue the following command to disable ECC on ESXi:

```
# nvidia-smi -e 0
Disabled ECC support for GPU 0000....
All done.
```

Now check that the ECC mode is disabled:

```
# nvidia-smi -q
ECC Mode
Current           : Disabled
Pending          : Disabled
```

6.2 Choosing the vGPU Profile for the VM

Once the NVIDIA GRID vGPU Manager is operating correctly at the ESXi level along with the guest OS driver, you can choose a vGPU Profile for a VM to assign your GPU solely to one VM or to be used in a shared mode with other VMs on the same host server. When there is one physical GPU on a host server, then all VMs on that server will use the same vGPU profile.

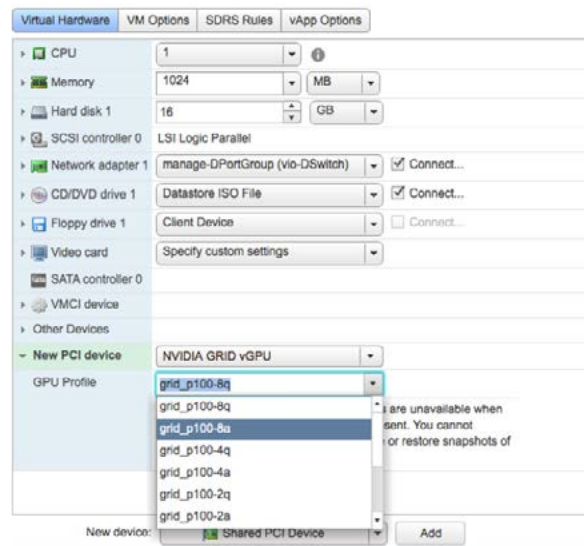


Figure 17: Choosing the vGPU Profile for a VM in the vSphere Client.

One type of vGPU profile we can choose from the list for the VM in Figure 17 is **grid_P100-8q**. The “q” at the end of the profile name indicates “Quadro.” This profile allows the VM to use at most 8 GB of the physical GPU’s memory (which totals 16 GB). Two VMs with this profile may therefore share the same physical GPU.

6.3 VM Guest OS NVIDIA GRID Driver Installation

Before beginning, ensure that the NVIDIA GRID vGPU Manager version that was installed into the ESXi hypervisor is compatible with the driver version you are installing into the guest OS for your VM. Consult the NVIDIA GRID Software User Guide for more information.

The following example installation steps up to section 3.4 are detailed for an Ubuntu VM. The process is very similar for another Linux flavor of guest OS, such as CentOS, though the individual commands may differ.

6.3.1 Developer Tools Installation

Ensure that the developer tools such as “gcc” are installed using the commands as follows:

```
# apt update
# apt upgrade
# sudo apt install build-essential
```


6.3.2 Preparing the Installation File

Download the `.run` file for the NVIDIA GRID Linux Guest VM driver from the NVIDIA site.

Note: This is a special driver that comes with the NVIDIA GRID software. It is not a stock NVIDIA driver that is found outside of that product.

Copy the NVIDIA GRID Linux driver package (for example the `NVIDIA-Linux-x86_64-390.42-grid.run` file) into the Linux VM's file system.

6.3.3 Exit from X-Windows Server

Before running the installer program for the driver, ensure that you have exited from the X-windows server in the VM and terminated all OpenGL applications, if present.

On Red Hat Enterprise Linux and CentOS systems, exit the X server by taking the guest OS to level 3:

```
# sudo init 3
```

On Ubuntu systems, first switch to a console login prompt using CTRL-ALT-F1. Then log in and shut down the display manager:

```
# sudo services lightdm stop
```

6.3.4 Install the NVIDIA Linux GRID Driver

Install the NVIDIA Linux GRID Driver from a console shell:

```
# chmod +x NVIDIA-Linux-x86_64-390.42-grid.run
# sudo sh ./NVIDIA-Linux-x86_64-390.42-grid.run
```

This may produce messages related to X-windows issues that you can ignore. Accept the license agreement to continue.

Confirm the setup with the following command:

```
# nvidia-smi
```

6.3.5 Applying the License

Add your license server's address to the `/etc/nvidia/gridd.conf` file (which is copied from `/etc/nvidia/grid.conf.template`). An example entry is:

```
ServerAddress=10.1.2.3
```

Set the `FeatureType` entry to 1.

This is for GRID vGPU. For more details, see the section on [“Licensing GRID vGPU on Linux” in the NVIDIA GRID User Guide](#).

Save your changes to the `/etc/nvidia/gridd.conf` file.

Restart the `nvidia-gridd` service in the VM:

```
# sudo service nvidia-gridd restart
```

Use the following command to check that the `/var/log/syslog` file has appropriate messages in it that indicate that the license was acquired and that the system is licensed correctly:

```
# sudo grep grid /var/log/messages
```

There should be entries in that file that are similar to the following:

```
The license was acquired successfully from the correct server URL
The system is licensed for GRID vGPU
```

Read the white paper [Enabling Machine Learning as a Service with GPU Acceleration using VMware vRealize Automation](#) for more insights on this approach.

ADDITIONAL RESOURCES

Check out these resources for a deeper dive into NVIDIA GRID vGPU setup.

- Guide: [Virtual Machine Graphics Acceleration](#)
- User Guide: [NVIDIA GRID Software](#)
- Release Notes: [NVIDIA Virtual GPU Software for VMware vSphere](#)
- List: [NVIDIA list of certified servers with GPU types supported](#)
- Support Site: [NVIDIA Virtual vGPU Support Site \(for Evaluations\)](#)
- Guide: [NVIDIA Virtual GPU Packaging, Pricing and Licensing](#)
- Guide: [NVIDIA Virtual GPU Software Quick Start Guide](#)

6.4 Install and Test the CUDA Libraries

6.4.1 Using Containers

The CUDA and machine learning frameworks can be installed using Docker containers. This is available to allow the administrator to avoid the complexity of installing each component one by one, as described in the next section.

The approach is to use Docker via the `nvidia-docker` tool and with the appropriate CUDA/ML/HPC containers.

6.4.2 Installing the CUDA Libraries Manually

This section describes the installation of the CUDA libraries without the use of containers.

6.4.2.1 Download the Libraries

Prior to beginning, be sure that the CUDA library versions are compatible with the guest OS driver. As an example of version compatibility, version 9.1 of the CUDA libraries is compatible with the “390” version of the drivers.

Download the appropriate packages (using a `wget` command, for example).

Following are file name examples for CUDA version 9.1. They may not apply to your installation as subsequent versions become available.

CUDA Version 9.1 File Names:

- https://developer.nvidia.com/compute/cuda/9.1/Prod/local_installers/cuda_9.1.85_387.26linux
- https://developer.nvidia.com/compute/cuda/9.1/Prod/patches/1/cuda_9.1.85.1_linux
- https://developer.nvidia.com/compute/cuda/9.1/Prod/patches/2/cuda_9.1.85.2_linux
- https://developer.nvidia.com/compute/cuda/9.1/Prod/patches/3/cuda_9.1.85.3_linux

6.4.2.2 Library Installation

To install the CUDA libraries, run the command:

```
# sudo sh cuda_9.1.85_387.26_linux
```

Accept the EULA conditions.

When prompted to install the driver, answer `No`.

Respond `yes` to the questions about libraries, symbolic links, and samples.

6.4.3 CUDA Application Code Testing

Compile and run the `deviceQuery` program. This should work correctly.

Compile and run the `vectorAdd` program. Ensure that the GPU is working.

You may now proceed to installation of cuDNN and the TensorFlow platform for machine learning application development—or you may use the container method for installation as an alternative.

7. GPU Method 3: Bitfusion FlexDirect

Bitfusion is a VMware company that produces and markets the FlexDirect software product for optimizing the use of GPUs across multiple VMs.

Bitfusion's FlexDirect provides better flexibility when utilizing GPUs on vSphere in multiple ways. Physical GPUs can be allocated in part or as a whole to applications running in VMs, and the consumer VMs can be hosted on servers that don't have physical GPUs attached to them. Bitfusion FlexDirect uses techniques for remoting of the CUDA instructions to other servers in order to achieve this.

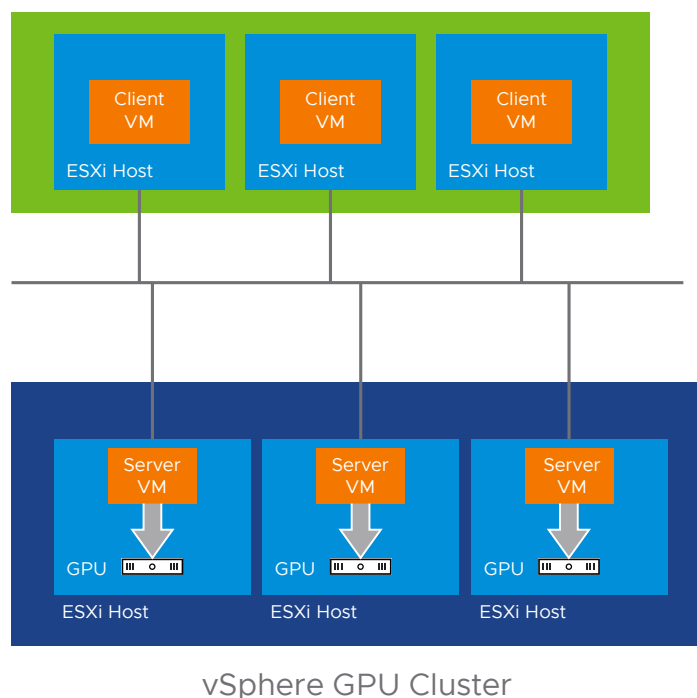


Figure 18: Example outline architecture for a Bitfusion FlexDirect Setup on VMs, with one VM per ESXi client-side host server. Multiple VMs of this type can live on the same client or server-side host server.

Bitfusion FlexDirect may be used to dedicate one or more full GPUs to a VM, or to allow the sharing of a single physical GPU across multiple VMs. The VMs that are sharing a physical GPU in this case need not be taking equal shares in it—their share sizes can be different. The share of the GPU is specified by the application invoker at startup time. Bitfusion allows the set of consumer VMs to use multiple physical GPUs at once from one VM.

7.1 The Bitfusion FlexDirect Architecture

Bitfusion FlexDirect uses a client-server architecture, as seen in Figure 18, where the server-side VMs provide the GPU resources, while the client-side VMs provide the locations for end-user applications to run. The server-side GPU-enabled VMs are referred to as the “GPU Cluster.” An individual node or VM may play both roles and have client and server-side execution capability locally, if required.

The client-side and server-side VMs are often hosted on different physical servers, and can be configured to communicate over a range of different types of network protocols including TCP/IP and RDMA. RDMA may be implemented using RoCE (RDMA over Converged Ethernet) on vSphere. These different forms of networking have been tested by VMware and Bitfusion engineers working together on vSphere, and the results of those tests are published in the blog post [Machine Learning leveraging NVIDIA GPUs with Bitfusion on VMware vSphere](#).

The Bitfusion FlexDirect software eliminates the need for physical locality of the GPU device to the consuming application—the GPU can be remotely accessed on the network. This approach allows for pooling of your GPUs on a set of servers. GPU-based applications can then run on any node or VM in the cluster, whether it has a physical GPU attached to it or not.

7.2 Bitfusion FlexDirect: Installation and Setup

Before getting started, make sure that any GPU involved in the Bitfusion installation has been correctly configured. The GPU cards can be configured for the server-side VMs that run on GPU-bearing hosts using the Direct Path I/O (Passthrough) or the NVIDIA vGPU methods, discussed in previous sections.

The product that is installed for Bitfusion functionality is called “FlexDirect.” Bitfusion has a client-side and a server side FlexDirect process and certain helper processes that are installed as described in this section. This software operates in user mode within the guest OS of the VMs and needs no special drivers.

You can also find these installation steps on the [Bitfusion website](#).

7.2.1 License Key

Ensure you have the appropriate Bitfusion License Key. If you do not have a current license key, contact Bitfusion to acquire one.

7.2.2 FlexDirect CLI

Install the FlexDirect CLI program in your Ubuntu or CentOS Linux VM (you will need Internet access for this command):

```
wget -O - getflex.bitfusion.io | sudo bash
```

This command downloads the shell script to install the FlexDirect material and passes that script to be executed in a shell by the root user.

7.2.3 Check the FlexDirect Location

Determine where the FlexDirect CLI program has been installed with the command:

```
which flexdirect
```

Output

```
/usr/bin/flexdirect
```

7.2.4 Initialization

Initialize the FlexDirect system using the following command:

```
sudo flexdirect init
```

Output

```
License has been initialized Attempting to refresh...
Refresh successful. License is ready for use.
Flexdirect is licensed and is ready for use.
```

You may be asked to enter the software license key you have previously acquired at this point. If you have a license key, enter it when prompted. If you don't yet have a license key, then you may email support@bitfusion.io to request access.

7.2.5 Launching the FlexDirect Resource Scheduler on the GPU-Enabled Servers

Launch the FlexDirect Resource Scheduler daemon on the GPU servers in your cluster using the following command on all the server-side VMs that have a GPU on their host:

```
nohup flexdirect resource_scheduler &> /dev/null &
```

Example Output

```
Flexdirect resource
Running resource scheduler on 0.0.0.0:56001 using 1 GPUs (0)
with 1024 MiB of memory each
```

The FlexDirect Resource Scheduler daemon provides the resource scheduling function for GPU users on your cluster and is known also as SRS.

The syntax of the various parameters to the `flexdirect` command are given in the [Bitfusion Usage page](#).

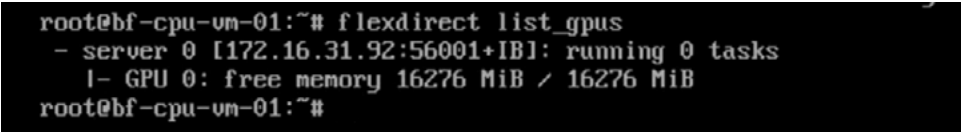
You may also get help on the `flexdirect` command on its own as follows to see the various parameters and options available by typing:

```
flexdirect
```

To see the GPUs that are available, either from a suitably configured client-side VM or server-side VM, type the command:

```
flexdirect list_gpus
```

An example output for a single server VM with one GPU enabled is shown in Figure 19.



```
root@bf-cpu-vm-01:~# flexdirect list_gpus
- server 0 [172.16.31.92:56001+IB]: running 0 tasks
  | GPU 0: free memory 16276 MiB / 16276 MiB
root@bf-cpu-vm-01:~#
```

Figure 19: Example output from `flexdirect` to show the available GPUs.

7.2.6 Testing an Application on the Bitfusion FlexDirect Server-Side

Execute the FlexDirect program on the GPU-enabled VM with a named application as a client would, in order to test it. Figure 20 shows an example of such a test command, using the `nvidia-smi` test program:

```
flexdirect run -n 1 -m 2048 nvidia-smi |more
```

```
root@bf-gpu-vm-02:~# flexdirect run -n 1 -m 2048 nvidia-smi |more
Choosing GPUs from server list [172.16.31.92]
Requesting GPUs [0] with 2048 MiB of memory from server 0...
Locked 1 GPUs with partial memory 1, configuration saved to '/tmp/flexdirect088628902'
Running client command 'nvidia-smi' on 1 GPUs, with the following servers:
172.16.31.92 55001 ff415ac1-c36d-11e8-9bdc-00505691fbb9 56001

Fri Sep 28 18:29:43 2018

+-----+
| NVIDIA-SMI 384.81                  Driver Version: 384.81                  |
+-----+-----+
| GPU Name Persistence-MI Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0 Tesla P100-PCIE... Off | 00000000:13:00.0 Off |           |       Off           |
| N/A   33C    P0   26W / 250W |    0MiB / 2048MiB |           |         Default    |
+-----+-----+

+-----+
| Processes:                         GPU Memory |
|  GPU       PID    Type    Process name      Usage    |
|=====+=====+
| No running processes found          |
+-----+

Releasing GPUs from config file '/tmp/flexdirect088628902'...
Released GPUs on 1 servers and removed generated config file '/tmp/flexdirect088628902'
```

Figure 20: The flexdirect run command showing output from a health check run.

You would replace the `nvidia-smi` string in the above command with your own application's executable name in order to have it run on the appropriate number of GPUs (the `-n` parameter) using the suitable GPU memory allocation (`-m` parameter). You can also choose to use a fraction of a GPU by issuing a command such as

```
flexdirect local -p 0.5 -m 1024 nvidia-smi |more
```

where the parameter `-p 0.5` indicates a partial share of one half of the GPU power for this application.

7.2.7 Execute Health Checks

Gauge the health of the server-side process using the command shown in Figure 21:

```
flexdirect localhealth
```

```
root@bf-gpu-vm-02:~# flexdirect localhealth
[PASS] Check flexdirect install
[PASS] Check library dependency
[PASS] Check Network Errors/Drops: found no errors or packet drops
[PASS] Check external connectivity to Bitfusion license server. Ignore for on-premise installations.
[PASS] Check PCIe Width:: GPUs are all set to use their maximum PCIe lane capacity
[PASS] Check OFED Version: 4.3
[PASS] Check IB physically up
[PASS] Check IB SM up
[PASS] Check MAD agent registration errors
[PASS] Check GPU API mismatch
[PASS] Check CUDA version >= 7050: Current version: 9000
[PASS] Check GPU Xid errors
[PASS] Check GPU driver version >= 367.0: Current version: 384.81
[PASS] Check temperature <= 100.00: current temp: 34.00
[PASS] Check ECC errors
[PASS] Check shadow memory 50244MB and total gpu mem 16276MB
[PASS] Check mem ops
[PASS] Check ulimit -n >= 4096: 4096
[MARGINAL] Check MTU Size: 10000Mbps interface ens160 MTU 1500 < 4K: performance hit
: 10000Mbps interface ens256 MTU 1500 < 4K: performance hit
[MARGINAL] Check RDMA: command 'ethtool -a ens160' fails: cannot check pause parameters
: command 'ethtool -a ens192' fails: cannot check pause parameters
[MARGINAL] Check multinode support: GPU direct not supported
[MARGINAL] Check nv_peer_mem: nv_peer_mem module not loaded

PASS: 18
MARGINAL: 4
FATAL: 0
SKIPPED: 0
```

Figure 21: flexdirect localhealth command output.

The `flexdirect health` command may also be executed on a client-side non-GPU attached VM in the same way, producing a more concise output. Note the hostname of the VM here includes `cpu`, indicating it does not have a GPU attached to it in Figure 22.

```
root@bf-cpu-vm-01:/etc/bitfusionio# flexdirect localhealth
[PASS] Check flexdirect install
[PASS] Check library dependency
[PASS] Check Network Errors/Drops: found no errors or packet drops
[PASS] Check external connectivity to Bitfusion license server. Ignore for on-premise installations.
[PASS] Check CUDA version >= 7050: Current version: 9000
[MARGINAL] Check MTU Size: 10000Mbps interface ens160 MTU 1500 < 8K: performance hit

PASS: 5
MARGINAL: 1
FATAL: 0
```

Figure 22: Executing flexdirect health command on the client-side VM.

7.2.8 Getting Concise Data on the GPU and Driver

To get a very concise view of the health and the driver and the GPU state, use the command shown in Figure 23:

```
flexdirect smi
```

```
root@bf-gpu-vm-02:/etc/bitfusionio/lic# flexdirect smi
+-----+-----+-----+-----+-----+-----+-----+-----+
| 172.16.31.92:56001 | Driver Version: 384.81 |
+-----+-----+-----+-----+-----+-----+-----+
| GPU Name | Persistence-M | Virt Mem | Alloc / All | BusId | Vol | Uncorr | ECC |
| Fan | Temp | Perf | Pur:Usage/Cap | Phy Mem | Used / All | GPU-Util | Compute M. |
+-----+-----+-----+-----+-----+-----+-----+
| 0 | Tesla P100-PCIE-16GB | Disabled | 0 | MB / 16276 | MB | 00000000:13:00.0 | N/A |
| 0 % | 36C | P0 | 29M / 250M | 0 | MB / 16276 | MB | 0% | Default |
+-----+-----+-----+-----+-----+-----+-----+

```

Figure 23: Flexdirect smi command output.

7.2.9 Installing FlexDirect on a Client-Side VM

Install the client side of the Bitfusion FlexDirect product using the same commands as for the FlexDirect server-side VMs:

```
wget -O - getflex.bitfusion.io | sudo bash
sudo flexdirect init
```

7.2.10 Configuring the Client-Side VM for Access to the Server-Side VMs

Once the FlexDirect software is installed, ensure the license is accepted as before (using the commands above as on the server VM side). Next, configure an entry in the `/etc/bitfusionio/servers.conf` file on the client side to contain the IP addresses or hostnames of the server-side VMs that this client VM will talk to. This configuration file should have one IP address or hostname per line as shown in Figure 24. There may be several of these server VMs mentioned in that file, one per line.



```
root@bf-cpu-vm-01:/etc/bitfusionio# cat servers.conf
172.16.31.92
root@bf-cpu-vm-01:/etc/bitfusionio# _
```

Figure 24: The `/etc/bitfusionio/servers.conf` file on the client VM side to enable it to talk the GPU-enabled server VM.

ADDITIONAL RESOURCES

Check out these resources for a deeper dive into Bitfusion FlexDirect setup.

- Guide: [Bitfusion Documentation](#)
- Blog Post: [Machine Learning leveraging NVIDIA GPUs with Bitfusion on VMware vSphere](#)

7.2.11 Testing the Connection Between the Client and Server VMs

Run your own application using the FlexDirect program to execute the GPU-specific parts on the server side, or use the command we tried previously on the server side to also execute the `nvidia-smi` tool on the client VM, as shown in Figure 25, to produce the same result:

```
flexdirect run -n 1 -m 2048 nvidia-smi
```

```
root@bf-cpu-vm-01:/etc/bitfusionio# flexdirect run -n 1 -m 2048 nvidia-smi
Choosing GPUs from server list [172.16.31.92]
Requesting GPUs (0) with 2048 MiB of memory from server 0...
Locked 1 GPUs with partial memory 1, configuration saved to '/tmp/flexdirect934103448'
Running client command 'nvidia-smi' on 1 GPUs, with the following servers:
172.16.31.92 55001 1506e61b-c379-11e8-9bdc-00505691fbb9 56001

Fri Sep 28 19:49:04 2018

+-----+
| NVIDIA-SMI 384.81                  Driver Version: 384.81                  |
+-----+
| GPU Name      Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|  Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0 Tesla P100-PCIE...    Off | 00000000:13:00.0 Off |                    |
| N/A   33C    P0     26W / 250W |      0MiB / 2048MiB |      0%      Default |
+-----+-----+

+-----+
| Processes:                         GPU Memory |
|  GPU       PID    Type    Process name                        Usage |
|=====+=====+
| No running processes found          |
+-----+

Releasing GPUs from config file '/tmp/flexdirect934103448'...
Released GPUs on 1 servers and removed generated config file '/tmp/flexdirect934103448'
root@bf-cpu-vm-01:/etc/bitfusionio#
```

Figure 25: Client-VM health check output using both the `flexdirect` and `nvidia-smi` commands.

Use any portion of a GPU, or a set of GPUs, on the server VMs to execute your job. Execute the application from your client-side VMs as seen previously. The client-side VMs can be moved from one vSphere host server to another using vSphere vMotion and DRS.

If your server-side VMs are hosted on vSphere 6.7 u1 or later, and are configured using NVIDIA GRID vGPU for sharing, then they may also be moved across vSphere hosts using vMotion.

8. Performance of Machine Learning Workloads Using GPUs

To support enterprises that are deploying machine learning or deep learning workloads on VMware vSphere, VMware conducted a series of performance studies on machine learning-based workloads using GPUs. On VMware vSphere, we first used direct passthrough (VMware DirectPath I/O) and in separate tests, mediated passthrough (NVIDIA GRID vGPU) to deploy machine learning workloads that use GPUs.

This section summarizes the benefits of using both passthrough mode and GRID vGPU, as well as analyzing their performance impacts from multiple aspects including

- Performance comparisons
- Scalability
- The use of a container inside a VM for machine learning workloads
- Mixing workloads
- vGPU scheduling
- vGPU profile selection

Passthrough mode (VMware DirectPath I/O) allows direct access from the guest OS in a VM to the physical PCI or PCIe hardware devices of the server controlled by the vSphere hypervisor layer. This mode is depicted in Figure 26. Each VM is assigned one or more GPUs as PCI devices. Because the guest OS bypasses the virtualization layer to access the GPUs, the overhead of using passthrough mode is low. There is no GPU sharing among VMs when using this mode.

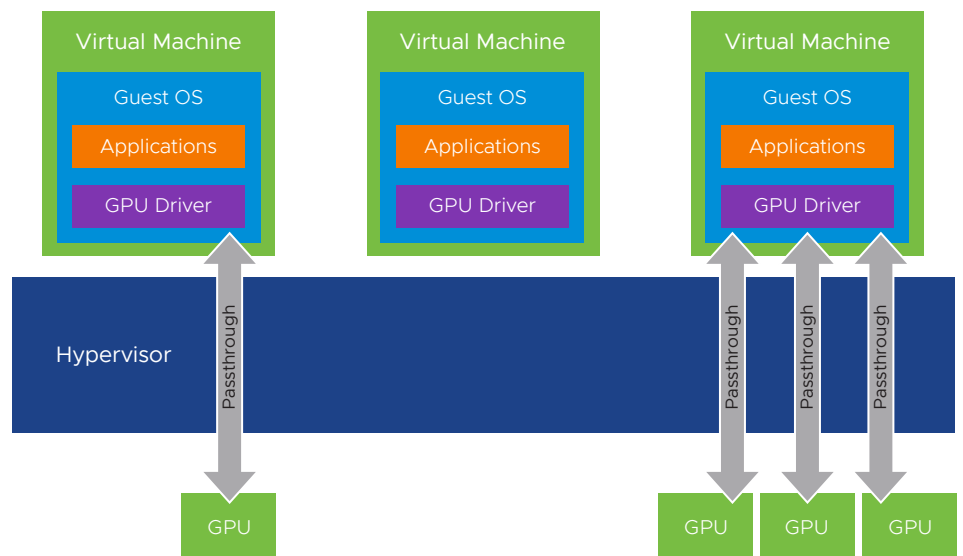


Figure 26: Passthrough mode (VMware DirectPath I/O).

Mediated passthrough (NVIDIA GRID vGPU). The NVIDIA GRID vGPU manager is installed into the hypervisor layer (that is, VMware ESXi) that virtualizes the underlying physical GPUs (Figure 27). The graphics memory of the physical GPU is divided into equal chunks and those chunks are given to each VM. The type of vGPU profile determines the amount of graphics memory each VM can have. This mode enables the virtualization benefits of (1) cloning a VM, (2) suspending and resuming a VM, and (3) migrating a VM from one host to another host.

Mediated Passthrough

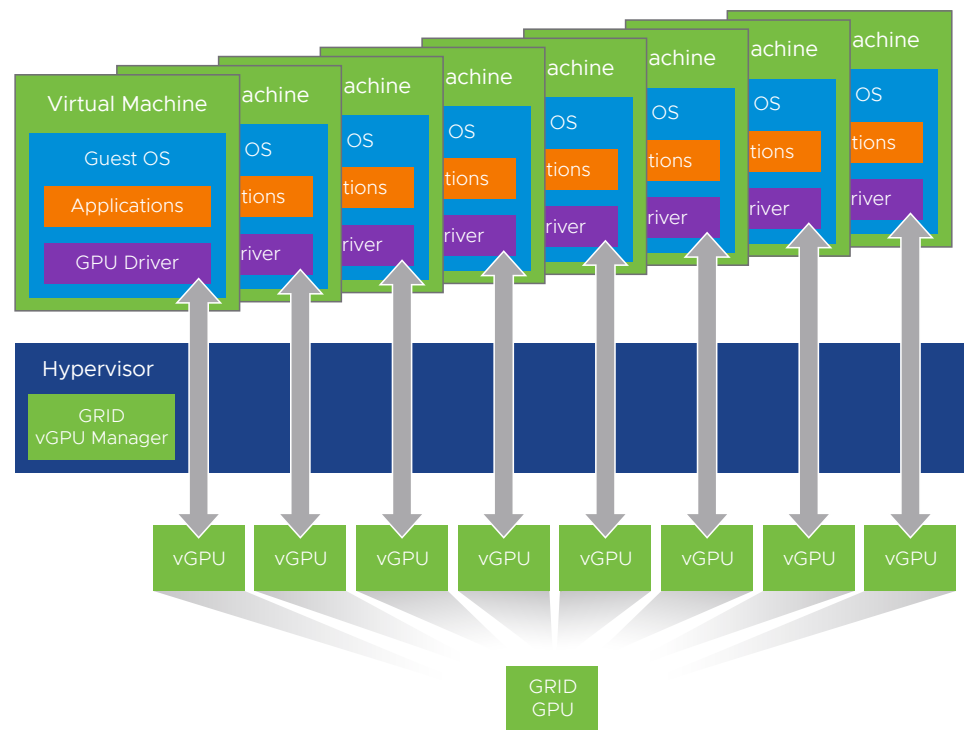


Figure 27: Mediated passthrough (NVIDIA GRID vGPU).

Following are some highlights of our exploration on virtualized GPUs on VMware vSphere.

Performance – To understand the performance impact of machine learning with GPUs using virtualization, we compared the performance of virtual GPU vs. physical GPU by benchmarking the same machine learning workload in three different cases: (1) GPU using DirectPath I/O on vSphere, (2) GRID vGPU on vSphere, and (3) native GPU on bare metal. Our results show that the virtualization layer (DirectPath I/O and GRID vGPU) introduced a 4 percent overhead for the tested machine learning application. For more information on this, read [this article](#).

Language Modeling with RNN on PTB

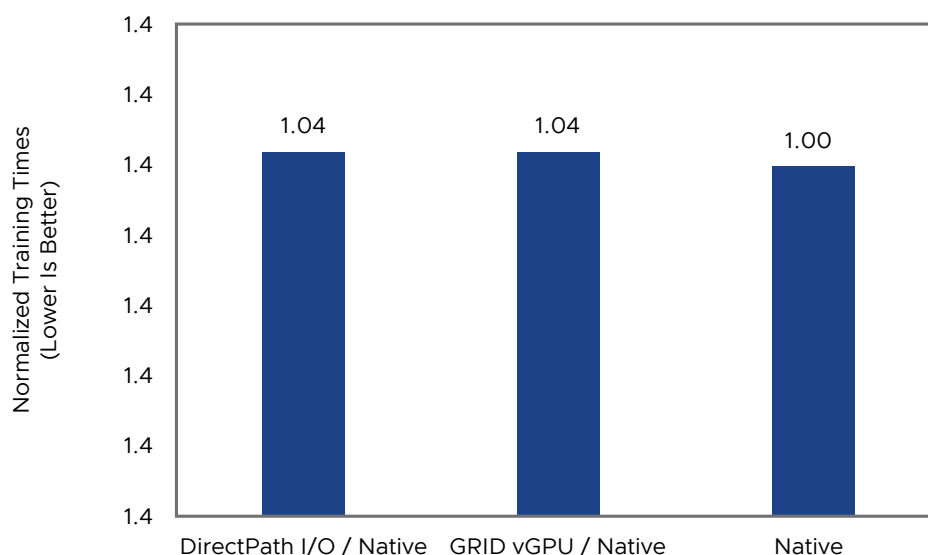


Figure 28: Performance Comparison of DirectPath I/O, NVIDIA GRID vGPU, and Native systems using the same hardware.

Scalability – We evaluated the scalability of a GPU-based workload in a virtualized environment from different angles: scaling the number of users or VMs per server or per GPU and scaling the number of GPUs per VM. With DirectPath I/O, one or more GPUs can be assigned to a single VM. That assignment makes this solution a suitable choice for heavy machine learning workloads requiring one or multiple dedicated GPUs.

With the NVIDIA GRID vGPU software, the NVIDIA Pascal and later GPU devices support sharing their compute capability using different vGPU profiles. This allows multiple VMs running machine learning workloads to share a single physical GPU and thus helps increase the system consolidation and utilization. Using GRID vGPU is suitable for those use cases where more VMs or users need a GPU capability than there are available physical GPUs. Our study shows a good scalability of virtualized GPUs. More information about this is available on the [VMware VROOM! blog](#). We also present the scalability issue in detail [in the blog post](#) and there are more updates in [our recent GTC 2018](#).

Containerized Machine Learning Applications inside a VM – Containers are rapidly becoming a popular environment in which to run different applications, including those in machine learning. For GPU-based applications, we can use NVIDIA Docker (a wrapper around the Docker command line that understands CUDA and GPUs) to run applications that use NVIDIA GPUs. We compared the performance of a physical GPU with a native container against that of a virtual GPU associated with a container running inside a VM.

We trained a convolutional neural network (CNN) on MNIST, a handwriting recognition application, while running it in a container on CentOS executing natively. Then we also trained the CNN on MNIST while running in a container inside a CentOS VM on VMware vSphere. In our experiments, we demonstrated that running containers in a virtualized environment, such as in a CentOS VM on vSphere, incurred no performance penalty, while allowing the system to benefit from other virtualization features, especially in the security area. This use case is described in detail in a [Performance Comparison blog post](#).

Mixing GPU-based Workloads: Graphics, Machine Learning and Knowledge Worker – For machine learning workloads using GPUs, most of the computation is offloaded from the CPUs to the GPU and the CPU is underutilized. In such use cases, mixing workloads on vSphere can increase the consolidation of VMs on a single server and the efficiency of resource usage. We characterized the performance impact of running 3D CAD, machine learning, and knowledge worker (non-GPU) workloads concurrently on the same server. Our experiments showed that the machine learning training times increased by a factor of 2 to 18 percent as we scaled the number of knowledge worker VMs from 32 to 96. The impact on 3D CAD workloads ranged from 3 to 9 percent as we scaled the number of knowledge worker VMs from 32 to 96. The impact on the knowledge worker latencies was less than half a percent as we scaled from 32 to 96 VMs. For more information, see [our blog post](#) and our [recent tech talk](#).

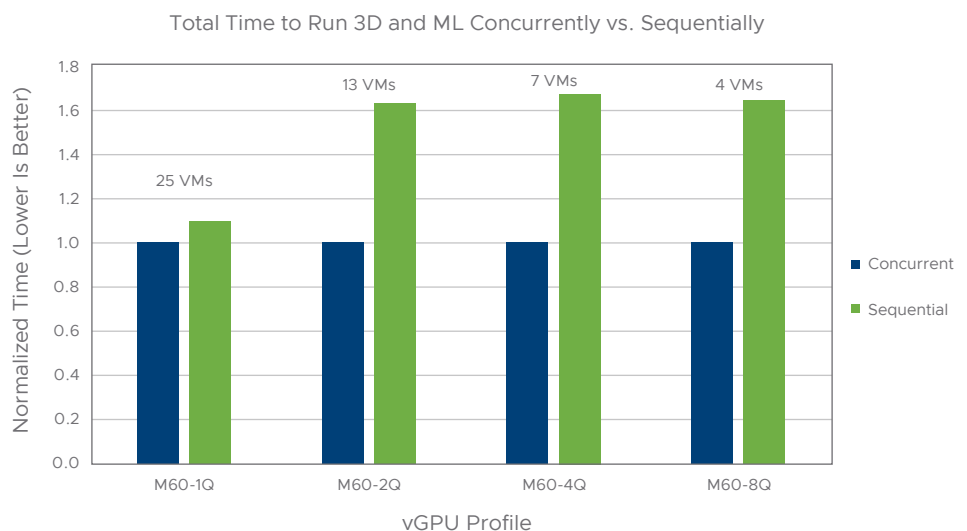


Figure 29: Performance comparisons of mixed workloads (GPU and non-GPU ones).

vGPU scheduling – NVIDIA GRID vGPU on vSphere provides three scheduling policies: Fixed Share, Equal Share, and Best Effort Scheduling that specify how VMs with machine learning workloads can share GPUs concurrently. The selection of scheduling policies depends on the business use cases. We characterize the performance impact of these scheduling options in the [Machine Learning on VMware vSphere using NVIDIA's Virtualized GPUs on-demand presentation](#).

vGPU profile selection – NVIDIA GRID vGPU on vSphere has multiple options for the vGPU profile and each profile specifies the amount of GPU memory each VM can use and the maximum number of VMs that can share a single GPU. Learn more about how to select the right vGPU profile and its performance impact in the [Machine Learning on VMware vSphere using NVIDIA's Virtualized GPUs on-demand presentation](#).

Check out our GTC 2018 tech talks for the latest updates on this topic.

- [Machine Learning on VMware vSphere using NVIDIA's Virtualized GPUs](#)
- [Maximizing The Power of GPU For Diverse Workloads of Enterprise Digital Workspaces On VMware vSphere](#)

Resource Repository

TITLE	SECTION
Applying Machine Learning Algorithms to Streaming IoT Data on VMware Cloud on AWS and vSphere	Additional Resources
Configuring VMDirectPath I/O pass-through devices on a VMware ESX or VMware ESXi host	GPU Method 1: VMDirectPath I/O (Passthrough)
VMware vSphere VMDirectPath I/O: Requirements for Platforms and Devices	Additional Resources
How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads	GPU Method 1: VMDirectPath I/O (Passthrough)
VMware vSphere VMDirectPath I/O: Requirements for Platforms and Devices	GPU Method 1: VMDirectPath I/O (Passthrough)
Machine Learning with GPUs on vSphere	Why Virtualized GPUs Work Best for Machine Learning
Why the Data Scientist and Data Engineer Need to Understand Virtualization in the Cloud	Why Virtualized GPUs Work Best for Machine Learning
Running Common Machine Learning Use Cases on vSphere Leveraging NVIDIA GPU	Why Virtualized GPUs Work Best for Machine Learning
Machine Learning with H2O – the Benefits of VMware	Why Virtualized GPUs Work Best for Machine Learning
Designing a Shared Virtualized Infrastructure for Business Critical, Machine Learning and High-Performance Computing Workloads	Introduction
Machine Learning on VMware vSphere 6 with NVIDIA GPUs – VMware VROOM!	GPU Method 1: VMDirectPath I/O (Passthrough)
How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads	Additional Resources
Sharing GPUs for Machine Learning/Deep Learning on vSphere with NVIDIA GRID – Performance Considerations	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Performance Results of Machine Learning with DirectPath I/O and NVIDIA GPUs	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Performance Impact of Mixed Workloads for Machine Learning with DirectPath I/O and NVIDIA GPUs	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Performance Comparison of Native GPU to Virtualized GPU and Scalability of Virtualized GPUs for Machine Learning	Additional Resources
Performance Comparison of Containerized Machine Learning Applications Running Natively with NVIDIA vGPUs vs. in a VM	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
What's New in vSphere 6.5 and 6.7 for High Performance Computing and Machine Learning/Deep Learning	Additional Resources
How to Enable Nvidia V100 GPU in Passthrough mode on vSphere for Machine Learning and Other HPC Workloads	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software

TITLE	SECTION
Using GPUs with Virtual Machines on vSphere – Part 1: Overview	GPU Method 1: VMDirectPath I/O (Passthrough)
Using GPUs with Virtual Machines on vSphere – Part 2: VMDirectPath I/O	GPU Method 1: VMDirectPath I/O (Passthrough)
Using GPUs with Virtual Machines on vSphere – Part 3: Installing the NVIDIA GRID Technology	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Using GPUs with Virtual Machines on vSphere – Part 4: Working with BitFusion FlexDirect	GPU Method 3: Bitfusion FlexDirect
How Virtualization Helps in the Data Science and Machine Learning Lab – Part 1	Additional Resources
How Virtualization Helps in the Data Science and Machine Learning Lab – Part 2	Additional Resources
Scaling HPC and ML with GPUDirect RDMA on vSphere 6.7 (Part 1 of 2)	Additional Resources
Scaling HPC and ML with GPUDirect RDMA on vSphere 6.7 (Part 2 of 2)	Additional Resources
Running Common Machine Learning Use Cases on vSphere leveraging NVIDIA GPU	Additional Resources
Sharing NVIDIA GPUs in Singularity HPC containers leveraging vSphere	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Machine Learning leveraging NVIDIA GPUs with Bitfusion on VMware vSphere (Part 1 of 2)	GPU Method 3: Bitfusion FlexDirect
How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads	GPU Method 1: VMDirectPath I/O (Passthrough)
Virtual Machine Graphics Acceleration Guide	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA GRID Software – User Guide, July 2018	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA Virtual GPU Software for VMware vSphere – Release Notes, June 2019	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA list of certified servers with GPU types supported	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA Virtual vGPU Support Site (for Evaluations)	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA Virtual GPU Packaging, Pricing and Licensing	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA Virtual GPU Software Quick Start Guide	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
NVIDIA P100 GPU Product Brief	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software

TITLE	SECTION
Bifusion FlexDirect Guide	GPU Method 3: Bitfusion FlexDirect
Maximizing The Power of GPU For Diverse Workloads of Enterprise Digital Workspaces On VMware vSphere	Additonal Resources
VMware for Machine Learning Workloads	Additonal Resources
Machine Learning on VMware vSphere using NVIDIA's Virtualized GPUs	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Standalone Spark on VMware vSphere with a Machine Learning Test Run	Additonal Resources
Virtualizing Apache Spark	Additonal Resources
Big Data Performance on VMware Cloud on AWS: Spark Machine Learning and IoT Analytics Performance On-premises and in the Cloud	Additonal Resources
Enabling Machine Learning as a Service (MLaaS) with GPU Acceleration Using VMware vRealize Automation	GPU Method 2: The NVIDIA GRID vGPU and Virtual Compute Server Software
Jackson National Life: Harvesting Free Virtual Desktop Infrastructure Compute Cycles for High Performance Computing Workloads Using VMware vSphere	Additonal Resources
New Architectures for Apache Spark and Big Data	Additonal Resources
VMware vSphere Product Overview	VMware vSphere Overview



About the Authors

Niels Hagoort is a Technical Marketing Architect and VCDX #212 working for VMware in the Cloud Platform business unit. In his role, he covers the vSphere ESXi architecture including hardware (accelerators) and compatibility, core storage, networking, and resource management. His writings can be found at <https://blogs.vmware.com/vsphere> and <https://nielshagoort.com>. Find Niels on Twitter via @NHagoort.



Justin Murray is a Technical Marketing Architect at VMware, working on the core vSphere hypervisor product and on VMware Cloud on AWS. He works in the machine learning and GPU ecosystems with VMware's partner companies to produce technical descriptions of their integrations with VMware vSphere. Justin has written several articles on these subjects at <http://blogs.vmware.com/apps>. Find Justin on Twitter @johjustinmurray.

