WHITE PAPER–OCTOBER 2019

FlexDirect On-Premise Evaluation Guide

Bitfusion Guide

vmware®

Table of Contents

Welcome

This document will help you get started with the evaluation of FlexDirect in your own environment. After finishing the tutorial presented in this document you will be able to do the following:

- Use FlexDirect to execute code on remotely attached virtual GPUs (vGPUs), implementing the Elastic GPU architecture, with no performance impact
- Use FlexDirect to partition a GPU into smaller virtual GPUs (vGPUs) to achieve higher utilization
- Evaluate FlexDirect performance across a variety of standard Machine Learning Benchmarks
- Exercise advanced FlexDirect features

Cluster Configuration and Nomenclature

Figure 1 below represents the most basic FlexDirect Architecture for exercising remotely attached virtual GPUs.

- A CPU server with no physical GPUs
- A GPU server with four NVIDIA GPUs. The figure shows four V100s, each with 16GB of memory. The two instances are expected to be connected via at-least 10Gb/s Ethernet. Bitfusion operates with many other NICs, GPUs and CPU server types. The configuration shown below is just one possible example.

Fig 1. Remote GPU Architecture with FlexDirect

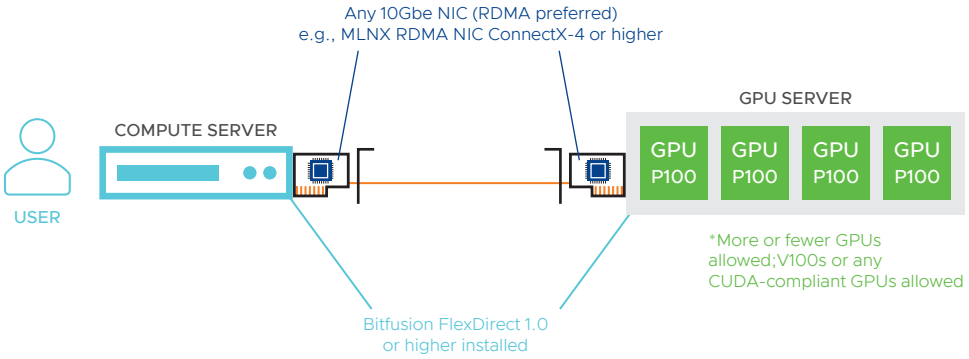
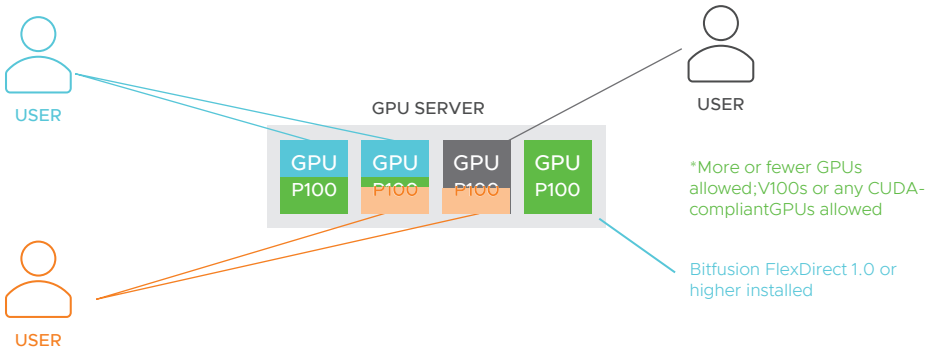


Figure 2 represents the FlexDirect Architecture if you would like to exercise just partial GPUs.

- A GPU server with four NVIDIA GPUs. The figure shows four V100s, each with 16GB of memory
- The memory is partitioned:
  - One user requests two GPUs at 50% partial size
  - Another user requests two GPUs at 33% partial size
  - A third user requests a GPU at 66% partial size

Fig 2. Partial GPU Architecture with FlexDirect



Cluster Instances & Software

The guide will have you run applications on both the CPU and the GPU servers, and for ease of doing this, it assumes that each bare metal server is running the exact same operating system and software stack. It assumes that you have installed a recent TensorFlow version (we recommend installing a prebuilt version of TensorFlow to avoid build issues—be sure to use one built to use GPUs) and CUDA prerequisites. Here is the list of the required stack:

- Ubuntu 16.04.02 LTS
- FlexDirect 1.9.0 or higher
- TensorFlow 1.5.0
- CUDA 9.0, CuDNN 7
- NVIDIA Drivers 390.41 or higher (GPU server only)

Installation of NVIDIA Drivers, CUDA, CUDNN and TensorFlow are all beyond the scope of this evaluation guide. Installing them may seem straightforward to you, if you have experience with these packages, or it may seem a bit tricky. You can search online for help installing them, or you can use the Bitfusion Unofficial Guide to CUDA and TensorFlow installation (these drivers and libraries are not Bitfusion products so the "Unofficial" designation is in earnest).

To install FlexDirect, please refer to our System Requirements and our Installation guides at *https://www-review.vmware.com/solutions/business-critical-apps/hardwareaccelerators-virtualization.html*.

Section 1: Baseline GPU Examples

The objective of this section is for you to run several TensorFlow examples on the GPU server directly. First, access a shell on the GPU server, perhaps as follows:

Shell

```
$ ssh username@ip_address_gpu_server
```

Download the FlexDirect benchmark scripts from the following public repository on your CPU and GPU systems. These scripts use FlexDirect, accessing virtualized GPUs, to run TensorFlow benchmarks, which you will download in a moment.

Shell

```
$ git clone https://github.com/bitfusionio/batch-scripts.git
```

The TensorFlow benchmarks you will run are available on GitHub and are not modified by Bitfusion. You should move into the directory created above: `cd batch-scripts`. Then, from the table below, clone the code repositories (and checkout the branches with the given hashes), or extract the tar files as shown:

URL (BENCHMARK, MODEL, DATA)	GIT HASH OR INSTRUCTIONS
https://github.com/tensorflow/benchmarks	`git checkout 9815f5f`
https://github.com/tensorflow/models	`git checkout f505cec`
wget *http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz*	In the batch-scripts directory `mkdir data`, `cd data`, and `tar -xvzf simple-examples.tgz`

This guide will use the scripts in the `cpu` and the `gpu` sub-directories. You will need to move the set of scripts you wish to use (either the `cpu` or `gpu` set) up one level in the directory tree (this will be shown below). The other scripts are more advanced and are documented in the accompanying `README.md` file.

The wrapper scripts to be used are listed below and fall into two categories of benchmarks:

Convolutional Neural Network Benchmarks (GPU Wrapper Scripts):

- gpu_instance_alexnet_0.25gpu.sh
- gpu_instance_alexnet_0.5gpu.sh
- gpu_instance_alexnet_1gpu.sh
- gpu_instance_alexnet_2gpu.sh
- gpu_instance_inceptionv3_0.25gpu.sh
- gpu_instance_inceptionv3_0.5gpu.sh
- gpu_instance_inceptionv3_1gpu.sh
- gpu_instance_inceptionv3_2gpu.sh
- gpu_instance_resnet50_0.25gpu.sh
- gpu_instance_resnet50_0.5gpu.sh
- gpu_instance_resnet50_1gpu.sh
- gpu_instance_resnet50_2gpu.sh
- gpu_instance_vgg16_0.5gpu.sh
- gpu_instance_vgg16_1gpu.sh
- gpu_instance_vgg16_2gpu.sh

Recurrent Neural Network Benchmarks (GPU Wrapper Scripts):

- gpu_instance_rnn_ptb_0.25gpu_small.sh
- gpu_instance_rnn_ptb_0.5gpu_small.sh
- gpu_instance_rnn_ptb_1gpu.sh
- gpu_instance_rnn_ptb_1gpu_small.sh
- gpu_instance_rnn_ptb_2gpu.sh

Note: There are similarly named scripts on the CPU instance that we will be running later.

You can execute any of the above benchmarks from the terminal as follows:

Shell

```
$ # Move the GPU scripts up one directory...
$ # You only need to do this once on the GPU server, of course.
$ mv ./gpu/* .
$
$ # Now, you can run most scripts
$ ./gpu_<benchmark_wrapper_name>.sh
$ # NOTE: You cannot run scripts that use FlexDirect yet because you
$ # have not launched the resource scheduler at this point. These
$ # scripts have fractional numbers in the name, such as
$ # gpu_instance_alexnet_0.5gpu.sh
```

Assuming your GPU instance features 4 x NVIDIA P100 cards, you can verify their presence on the instance using the following command:

Shell

```
$ nvidia-smi
```

You will see the following output:

Shell

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.41                 Driver Version: 390.41                    |
|-------------------------------+----------------------+----------------------+
| 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-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   29C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   30C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  On   | 00000000:84:00.0 Off |                    0 |
| N/A   28C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
| N/A   27C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

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

The provided wrapper scripts can execute the benchmarks using one or two of the GPUs. Each script is named to show how many GPUs it is using – for example script gpu_instance_resnet50_1gpu.sh uses one GPU and gpu_instance_resnet50_2gpu.sh uses two GPUs.

The Neural Network Benchmarks report results in Images per Second, while the Recurrent Neural Network Benchmarks report results in Words per Second. Feel free to execute one or more of the Benchmarks. Take into account that some of the benchmarks may take some time to run—progress, however, is always displayed on the screen. The expected approximate results for the runs on the GPU Instances are shown in the tables below:

CONVOLUTIONAL NEURAL NETWORK BENCHMARKS	IMAGES PER SECOND (P100 GPU)
gpu_instance_alexnet_1gpu.sh	~ 690
gpu_instance_alexnet_2gpu.sh	~ 1050
gpu_instance_inceptionv3_1gpu.sh	~ 45
gpu_instance_inceptionv3_2gpu.sh	~ 90
gpu_instance_resnet50_1gpu.sh	~ 70
gpu_instance_resnet50_2gpu.sh	~ 130
gpu_instance_vgg16_1gpu.sh	~ 45
gpu_instance_vgg16_2gpu.sh	~ 80

RECURRENT NEURAL NETWORK BENCHMARKS	WORDS PER SECOND (P100 GPU)
gpu_instance_rnn_ptb_1gpu.sh	~ 2700
gpu_instance_rnn_ptb_2gpu.sh	~ 4600

Now that you are familiar with how to run these Benchmarks directly on a GPU instance, let's see how FlexDirect allows you to execute these exact same Benchmarks from a CPU instance which has no physical GPUs.

But before you switch to using the CPU server in the next section, you need to launch the resource scheduler (SRS) on the GPU server. Note that the process continues to run listening to port 56001.

Shell

```
$ flexdirect resource_scheduler
[INFO ] 2018/09/25 00:27:28 Using 2 GPUs and 7618 MiB of GPU memory per GPU, as reported
by nvidia-smi. To override, specify --devices (-d), --num_gpus (-n) and/or --gpu_memory
(-m).
[INFO ] 2018/09/25 00:27:28 Running resource scheduler on 0.0.0.0:56001 using 2 GPUs
(0,1) with 7618 MiB of memory each
[INFO ] 2018/09/25 00:27:32 Server is now listening on 56001
```

Section 2: FlexDirect—Remote GPU Examples

The objective of this section is help you get familiar with FlexDirect and to show you how easy it is to run several TensorFlow examples on a CPU instance using remote virtual GPUs (vGPUs).

First, access the CPU instance:

Shell

```
$ ssh username@ip_address_cpu_server
```

Next, verify that this instance has no physical GPU by executing the following command (note that you may need to copy the `nvidia-smi` executable over from the GPU server):

Shell

```
$ nvidia-smi
```

You will see the following error message as there are no GPUs on this system:

Shell

```
NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver. Make sure
that the latest NVIDIA driver is installed and running.
```

Next, create a configuration file to point to the GPU server and test FlexDirect by issuing a command to list the available GPUs:

Text

```
$ # Assmuing the GPU server has IP address 192.168.10.10
$ sudo su -
$ echo 192.168.10.10 > /etc/bitfusionio/servers.conf
$ exit
$
$ flexdirect list_gpus
```

You will see output similar to the following:

Shell

```
- server 0 [10.0.0.4:56001+IB]: running 0 tasks
   |- GPU 0: free memory 16280 MiB / 16280 MiB
   |- GPU 1: free memory 16280 MiB / 16280 MiB
   |- GPU 2: free memory 16280 MiB / 16280 MiB
   |- GPU 3: free memory 16280 MiB / 16280 MiB
```

The output confirms that the CPU instance is properly connected to the GPU instance, and that FlexDirect is able to detect four GPUs on the GPU instance. Note that the first line shows the IP address, which is the IP address of the GPU instance. If the cluster were to contain multiple GPU instances, you would see an entry for each GPU instance available to FlexDirect, and the output would look similar to the following:

Shell

```
- server 0 [10.0.1.180:56001]: running 0 tasks
   |- GPU 0: free memory 16280 MiB / 16280 MiB
   |- GPU 1: free memory 16280 MiB / 16280 MiB
   |- GPU 2: free memory 16280 MiB / 16280 MiB
   |- GPU 3: free memory 16280 MiB / 16280 MiB

- server 1 [10.0.1.181:56001]: running 0 tasks
   |- GPU 0: free memory 16280 MiB / 16280 MiB
   |- GPU 1: free memory 16280 MiB / 16280 MiB
   |- GPU 2: free memory 16280 MiB / 16280 MiB
   |- GPU 3: free memory 16280 MiB / 16280 MiB
```

The "hello world" equivalent for FlexDirect is to run nvidia-smi on a remote vGPU. Please execute the following command.

Shell

```
$ flexdirect run -n 1 nvidia-smi
$ # Or if nvidia-smi is a local, out-of-path program then you would run
$ # flexdirect run -n 1 ./nvidia-smi
```

As a result, you will be presented with the following output:

Shell

```
Requesting GPUs [0 1 2 3] with 16280 MiB of memory from server 0...
Locked four GPUs with partial memory 1, configuration saved to '/tmp/flexdirect004000124'
Running client command 'nvidia-smi' on four GPUs, with the following servers:
10.0.0.4 55001 d27165b5-4efa-11e8-b66e-0cc47adc9522 56001
Thu May  3 10:53:00 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.41                 Driver Version: 390.41                    |
|-------------------------------+----------------------+----------------------+
| 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-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   29C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   30C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  On   | 00000000:84:00.0 Off |                    0 |
| N/A   28C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
| N/A   27C    P0    32W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Releasing GPUs from config file '/tmp/flexdirect004000124'...
Released GPUs on 1 servers and removed generated config file '/tmp/flexdirect004000124'
```

Let's dissect the above command first:

Shell

```
flexdirect     # main FlexDirect command
run            # run a GPU application
-n 2           # using 2 vGPUs
nvidia-smi     # application command to be run
```

The output from the command should look very familiar, as it displays the exact same GPU information which you previously encountered when running directly on the GPU instance. It is worth examining the printed header before the GPU information, as it provides detailed information about the execution of the FlexDirect vGPU request:

Shell

```
Requesting GPUs [0 1 2 3] with 16280 MiB of memory from server 0...
Locked four GPUs with partial memory 1, configuration saved to '/tmp/flexdirect004000124'
Running client command 'nvidia-smi' on four GPUs, with the following servers:
10.0.0.4 55001 d27165b5-4efa-11e8-b66e-0cc47adc9522 56001
```

These lines inform you that four GPUs, each with 16280 MiB of memory, were allocated for the application request on server zero. `partial memory 1`, indicates that the complete memory for each GPU was allocated for the request (partial allocation of GPUs and memory will be discussed in more detail in the next Section 3, but the value, 1, means 100%). Finally, the IP address of the GPU server is shown, as this is where the CPU instance allocates the virtual GPUs (vGPUs) from. For you, this IP address will be the IP address of your GPU instance.

Now that the basic operation of FlexDirect is understood it is time to execute real workloads using remote vGPUs. Move into the working directory on your CPU instance the several wrapper scripts in the `cpu` subdirectory which execute the same benchmarks as before (and the benchmark repository should have also been cloned in this working directory), only this time they do it using remote vGPUs:

Convolutional Neural Network Benchmarks (CPU Wrapper Scripts):

cpu_instance_alexnet_0.25rgpu.sh
cpu_instance_alexnet_0.5rgpu.sh
cpu_instance_alexnet_1rgpu.sh
cpu_instance_alexnet_2rgpu.sh
cpu_instance_inceptionv3_0.25rgpu.sh
cpu_instance_inceptionv3_0.5rgpu.sh
cpu_instance_inceptionv3_1rgpu.sh
cpu_instance_inceptionv3_2rgpu.sh
cpu_instance_resnet50_0.25rgpu.sh
cpu_instance_resnet50_0.5rgpu.sh
cpu_instance_resnet50_1rgpu.sh
cpu_instance_resnet50_2rgpu.sh
cpu_instance_vgg16_0.5rgpu.sh
cpu_instance_vgg16_1rgpu.sh
cpu_instance_vgg16_2rgpu.sh

Recurrent Neural Network Benchmarks (CPU Wrapper Scripts):

cpu_instance_rnn_ptb_0.25rgpu_small.sh
cpu_instance_rnn_ptb_0.5rgpu_small.sh
cpu_instance_rnn_ptb_1rgpu.sh
cpu_instance_rnn_ptb_2rgpu.sh

You can execute any of the above benchmarks from the terminal as follows:

Shell

```
$ # Move the CPU scripts up one directory...
$ # You only need to do this once on the CPU server, of course.
$ mv ./cpu/* .
$
$ # Now, you can run any of them
$ ./cpu_<benchmark_wrapper_name>.sh
```

We encourage you to examine these scripts to see how easy it is to use FlexDirect to run all these benchmarks. The expected results for the above benchmarks are shown in the tables below:

CONVOLUTIONAL NEURAL NETWORK BENCHMARKS	IMAGES PER SECOND (P100 GPU)
cpu_instance_alexnet_1rgpu.sh	~ 625
cpu_instance_alexnet_2rgpu.sh	~ 975
cpu_instance_inceptionv3_1rgpu.sh	~ 45
cpu_instance_inceptionv3_2rgpu.sh	~ 90
cpu_instance_resnet50_1rgpu.sh	~ 65
cpu_instance_resnet50_2rgpu.sh	~ 125
cpu_instance_vgg16_1rgpu.sh	~ 45
cpu_instance_vgg16_2rgpu.sh	~ 80

RECURRENT NEURAL NETWORK BENCHMARKS	WORDS PER SECOND (P100 GPU)
cpu_instance_rnn_ptb_1rgpu.sh	~ 2525
cpu_instance_rnn_ptb_2rgpu.sh	~ 4450

Additionally, please note that the performance results on the CPU instance are expected to be within a few percentage points of the results you obtained on the GPU instance—you can compare the table above to the table in Section 1, or you can compare your actual results. Minor performance variations can be explained by occasional network variability in bandwidth and latency.

Section 3: FlexDirect—Partial GPU Examples

A novel feature of FlexDirect is the ability allocate partial vGPUs for application requests by limiting the amount of GPU memory an application may use at runtime. GPU memory is a scarce resource. And more often than not, applications over-allocate the amount of memory they need. Consequently, the number of parallel applications or processes which could take advantage of the complete GPU memory space may be unnecessarily limited. To observe this limitation, please access the GPU instance via:

Shell

```
$ ssh username@ip_address_gpu_server
```

Then, in your working directory execute the following script:

Shell

```
$ ./gpu_instance_rnn_ptb_1gpu_small.sh
```

The above script executes an RNN on a language model, but processes a small data set which in reality requires only a fraction of the GPU memory. Naturally, you would assume that for experimental purposes it would make sense to run several of these models in parallel and monitor the progress to see which one converges fastest, trying something similar to this:

Shell

```
$ ./gpu_instance_rnn_ptb_1gpu_small.sh &
$ ./gpu_instance_rnn_ptb_1gpu_small.sh &
```

Similarly, two developers may be accessing the same system and trying to experiment with these small models while accessing the GPU. Unfortunately, in both cases, the first process or user would monopolize the resource, while the second process or user would see an error message similar to the following:

Shell

```
ResourceExhaustedError (see above for traceback): OOM when allocating tensor
```

As previously discussed, this is a serious problem, because the above model only requires a fraction of the GPU. As such, without FlexDirect, GPU resources are severely underutilized. FlexDirect however, makes it very easy to remedy the above situation. If you examine the script above, you will see the command it runs is as follows:

Shell

```
$ python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path data/simple-examples/data/
--model=small --num_gpus 1
```

To allocate 0.5 of a vGPU for our application, the above command simply needs to be changed to the following:

Shell

```
$ flexdirect run -n 1 -p 0.5 “python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path
data/simple-examples/data/ --model=small --num_gpus 1”
```

This command is similar to the one we used in the “hello world” example in the previous section. The only new flag introduced is -p (or --partial) and specifies the fraction of the GPU memory to allocate for the application–where 1 represents 100% of the GPU memory, 0.5 represents 50%, and so on.

The exact same command to allocate a partial vGPU will work whether invoked from the GPU instances or from the CPU instances. To help avoid copy/paste errors as you work through this guide, however, we have provided both cpu and gpu scripts which launch multiple, concurrent instances of the above command.

On the GPU Instance:

Shell

```
$ gpu_instance_alexnet_0.25gpu.sh
$ gpu_instance_alexnet_0.5gpu.sh
$ gpu_instance_inceptionv3_0.25gpu.sh
$ gpu_instance_inceptionv3_0.5gpu.sh
$ gpu_instance_resnet50_0.25gpu.sh
$ gpu_instance_resnet50_0.5gpu.sh
$ gpu_instance_rnn_ptb_0.25gpu_small.sh
$ gpu_instance_rnn_ptb_0.5gpu_small.sh
$ gpu_instance_vgg16_0.5gpu.sh
```

On the CPU Instance:

Shell

```
$ cpu_instance_alexnet_0.25rgpu.sh
$ cpu_instance_alexnet_0.5rgpu.sh
$ cpu_instance_inceptionv3_0.25rgpu.sh
$ cpu_instance_inceptionv3_0.5rgpu.sh
$ cpu_instance_resnet50_0.25rgpu.sh
$ cpu_instance_resnet50_0.5rgpu.sh
$ cpu_instance_rnn_ptb_0.25rgpu_small.sh
$ cpu_instance_rnn_ptb_0.5rgpu_small.sh
$ cpu_instance_vgg16_0.5rgpu.sh
```

For example, you can execute four simultaneous 50%-sized partial instantiations on the GPU instances as follows:

Shell

```
$ cat cpu_instance_rnn_ptb_0.5rgpu_small.sh
flexdirect run -n 1 -p 0.5 "python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path
data/simple-examples/data/ --model=small --num_gpus 1" &
flexdirect run -n 1 -p 0.5 "python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path
data/simple-examples/data/ --model=small --num_gpus 1" &
flexdirect run -n 1 -p 0.5 "python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path
data/simple-examples/data/ --model=small --num_gpus 1" &
flexdirect run -n 1 -p 0.5 "python models/tutorials/rnn/ptb/ptb_word_lm.py --data_path
data/simple-examples/data/ --model=small --num_gpus 1" &
wait # block until preceeding background processes complete
$
$
$ ./cpu_instance_rnn_ptb_0.5rgpu_small.sh
```

Each one of the applications launched by this script will be permitted to uses exactly one-half of the GPU memory, essentially splitting the two GPU cores into four vGPUs, and you will not run out of memory. You can verify the proper memory allocation while the applications are running by executing `nvidia-smi` in a separate terminal window which is connected to the GPU instance.

Note: If the amount of memory required by a test exceeds the amount of memory available in its partial GPU, the test will fail. If this occurs, you can edit the script to user smaller batch sizes until it succeeds.

Conclusion

FlexDirect makes it possible to disaggregate GPUs from CPU instances and to only request them when needed, leading to much more flexible cluster designs, better utilization, and an overall lower cost of ownership. Essentially it is an elastic, virtual GPU architecture.

FlexDirect has many more advanced features for managing and allocating vGPUs; this brief guide serves only as an introduction. For some of the more advanced features, please take a look at the online documentation at: *https://www-review.vmware.com/solutions/business-critical-apps/hardwareaccelerators-virtualization.html* or from the FlexDirect `help` command:

Shell

```
$ flexdirect help [run | list_gpus | ...]
```

For any question, comments, or feedback, please contact Bitfusion at: *askbitfusion@vmware.com*

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com Copyright © 2019 VMware, Inc.
All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at vmware.com/go/patents. VMware is a registered trademark or trademark of VMware, Inc. and its subsidiaries in the United States and other jurisdictions.
All other marks and names mentioned herein may be trademarks of their respective companies. Item No: VMW-0518-1843_VMW_CPBU Technical White Papers_Bitfusion Docs_09 Flex Direct On-Premise Evaluation Guide_1.3_YQ 8/19