FLEXDIRECT EVALUATION GUIDE

WELCOME

This document will help you get started with the evaluation of FlexDirect on a pre-configured computing cluster. 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

The cluster which is provided to you consists of two computing instances:

1. A CPU instance with no physical GPUs
2. A GPU instance with one NVIDIA M60 card containing two Maxwell™ GPUs. Each GPU has 2048 CUDA® cores and 8 GB of GDDR5 memory.

The two instances are connected via commonly available 10 Gb/s Ethernet based networking. FlexDirect is pre-installed on these two instances and no additional configuration is required.

ACCESSING THE CLUSTER

Along with this documentation you received an email with a username, password, and two IP addresses, one IP address for the CPU instance and one IP address for the GPU instance, respectively. You can access each of the two instances using the following command from a Linux terminal:

```
ssh username@ip_address
```

When prompted, please enter the password and you will be forwarded to the home directory on each of the instances.

CLUSTER INSTANCES & SOFTWARE

Each instance is running the exact same operating system and software stack:

- Ubuntu 16.04.02 LTS
- FlexDirect 1.8
- TensorFlow 1.4.1 (git hash: 438604f)
- CUDA 9.0, CuDNN 7, Nvidia Drivers 384.11 (GPU Instance Only)

SECTION 1: BASELINE GPU EXAMPLES

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

```
ssh username@ip_address_gpu_instance
```

The examples you will run were downloaded directly from GitHub and are not modified in any way. The code repositories are summarized in the table below:

URL (Benchmark, Model, Data)	Git Hash
https://github.com/tensorflow/benchmarks	9165a70
https://github.com/tensorflow/models	ec2bb58
http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz	N/A

The home directory on the GPU instance contains several wrapper scripts which execute the above Benchmarks.

The Benchmarks fall into the following categories:

Convolutional Neural Network Benchmarks:

```
gpu_instance_inceptionv3_1gpu.sh
gpu_instance_inceptionv3_2gpu.sh
gpu_instance_resnet50_1gpu.sh
gpu_instance_resnet50_2gpu.sh
gpu_instance_vgg16_1gpu.sh
gpu_instance_vgg16_2gpu.sh
gpu_instance_alexnet_1gpu.sh
gpu_instance_alexnet_2gpu.sh
```

Recurrent Neural Network Benchmarks:

```
gpu_instance_rnn_ptb_1gpu.sh
gpu_instance_rnn_ptb_2gpu.sh
```

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

```
./benchmark_wrapper_name.sh
```

For example:

```
./gpu_instance_inceptionv3_1gpu.sh
```

The GPU instance features an NVIDIA M60 card with two GPUs. You can verify the presence of these GPUs on the instance using the following command:

```
nvidia-smi
```

You will see the following output:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                 Driver Version: 384.111                  |
|-------------------------------+----------------------+----------------------+
| 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 M60           On   | 00000000:00:1D.0 Off |                    0 |
| N/A   23C    P8    13W / 150W |      1MiB /  7613MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 00000000:00:1E.0 Off |                    0 |
| N/A   28C    P8    13W / 150W |      1MiB /  7613MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

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

We have provided wrapper scripts that can execute the benchmarks using 1 or 2 of the provided 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 – taking into account that some of the benchmarks may take some time to run, however, progress is always displayed on the screen. The expected approximate results for the runs on the GPU Instances are shown in the table below:

Convolutional Neural Network Benchmarks:	**Images per Second**
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
gpu_instance_alexnet_1gpu.sh	~ 690
gpu_instance_alexnet_2gpu.sh	~ 1050
Recurrent Neural Network Benchmarks:	**Words per Second (Epoch 3)**
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.

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 as follows:

```
ssh username@ip_address_cpu_instance
```

Next, verify that this instance has no physical GPU by executing the following command:

```
nvidia-smi
```

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

```
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 verify that the FlexDiret has been configured properly for you by issuing the following command:

```
bfboost list_gpus
```

You will see output similar to the following:

```
- server 0 [10.0.1.180:56001]: running 0 tasks
   |- GPU 0: free memory 7613 MiB / 7613 MiB
   |- GPU 1: free memory 7613 MiB / 7613 MiB
```

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

```
- server 0 [10.0.1.180:56001]: running 0 tasks
   |- GPU 0: free memory 7613 MiB / 7613 MiB
   |- GPU 1: free memory 7613 MiB / 7613 MiB
- server 1 [10.0.1.181:56001]: running 0 tasks
   |- GPU 0: free memory 7613 MiB / 7613 MiB
   |- GPU 1: free memory 7613 MiB / 7613 MiB
```

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

```
bfboost run -n 2 nvidia-smi
```

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

```
Requesting GPUs [0 1] with 7613 MiB of memory from server 0...
Locked 2 GPUs with partial memory 1, configuration saved to '/tmp/bfboost740678502'
Running client command 'nvidia-smi' on 2 GPUs, with the following servers:
10.0.1.180 55006 db89e5fb-0573-11e8-8618-0e8b138088fe 56001

Tue Jan 30 04:12:59 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                 Driver Version: 384.111                  |
|-------------------------------+----------------------+----------------------+
| 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 M60           On   | 00000000:00:1D.0 Off |                    0 |
| N/A   23C    P8    14W / 150W |      1MiB /  7613MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 00000000:00:1E.0 Off |                    0 |
| N/A   28C    P8    13W / 150W |      1MiB /  7613MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

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

Releasing GPUs from config file '/tmp/bfboost740678502'...
Released GPUs on 1 servers and removed generated config file '/tmp/bfboost740678502'
```

Let's dissect the above command first:

```
bfboost                 # 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:

```
Requesting GPUs [0 1] with 7613 MiB of memory from server 0...
Locked 2 GPUs with partial memory 1, configuration saved to '/tmp/bfboost740678502'
Running client command 'nvidia-smi' on 2 GPUs, with the following servers:
10.0.1.180 55006 db89e5fb-0573-11e8-8618-0e8b138088fe 56001
```

These lines inform you that 2 GPUs, each with 7613 MiB of memory, were allocated for the application request on server 0. 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). Finally, the IP address of the GPU server is shown, as this is where the CPU instance allocates the 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. The home directory on the CPU Instance contains several wrapper scripts which execute the same Benchmarks as before, but this time using remote vGPUs:

Convolutional Neural Network Benchmarks:

```
cpu_instance_inceptionv3_1rgpu.sh
cpu_instance_inceptionv3_2rgpu.sh
cpu_instance_resnet50_1rgpu.sh
cpu_instance_resnet50_2rgpu.sh
cpu_instance_vgg16_1rgpu.sh
cpu_instance_vgg16_2rgpu.sh
cpu_instance_alexnet_1rgpu.sh
cpu_instance_alexnet_2rgpu.sh
```

Recurrent Neural Network Benchmarks:

```
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:

```
./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 table below:

Convolutional Neural Network Benchmarks:	**Images per Second**
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
cpu_instance_alexnet_1rgpu.sh	~ 625
cpu_instance_alexnet_2rgpu.sh	~ 975
Recurrent Neural Network Benchmarks:	**Words per Second (Epoch 3)**
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 and using other runtime optimizations that an application may utilize. GPU memory is a scarce resource, however, more often than not applications over-allocate the amount of memory they need. Consequently, the amount of parallel applications or processes which can take advantage of a complete GPU memory space may become limited. To experience this limitation, please access the GPU instance via:

```
ssh username@ip_address_cpu_instance
```

In the home directory you will find the following script which you can execute as follows:

```
./gpu_instance_rnn_ptb_1gpu_small.sh
```

The above script executed an RNN on a language model, but it executes in on 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 to progress to see which one may converge the fastest, and you may try something along the following lines:

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

Similarly, two developers, may be accessing the same systems and trying to experiment with these small models while accessing this GPU in a similar manner. Unfortunately, in both cases, the first process or user would monopolize the resource, while the second process or user would see an error message equivalent to something as follows:

```
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 contents are as follows:

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

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

```
bfboost run -n 1 -p 0.5 "python models/tutorials/rnn/ptb/ptb_word_lm.py
--data data/simple-examples/data/ --model=small --num_gpus 1"
```

This command is just about the same one that we used in the "hello world" example in the previous section. The only new flag that is introduced is -p, or --partial and corresponds to the fraction of the GPU memory that you want to allocate for the application – where 1 represent 100% of the GPU memory, 0.5 represent 50%, and so on.

The exact same command will work on the GPU instances to allocate a partial vGPU as well as on the CPU instances. However, to make it easier and avoid copy/paste mistake we have wrapped the above command into scripts on both of the instances already:

On GPU Instance:

```
gpu_instance_rnn_ptb_0.5gpu_small.sh
gpu_instance_rnn_ptb_0.25gpu_small.sh
```

On CPU Instance:

```
cpu_instance_rnn_ptb_0.5rgpu_small.sh
cpu_instance_rnn_ptb_0.25rgpu_small.sh
```

For example, please execute 4 instantiations on the GPU instance as follows:

```
./gpu_instance_rnn_ptb_0.5gpu_small.sh &
./gpu_instance_rnn_ptb_0.5gpu_small.sh &
./gpu_instance_rnn_ptb_0.5gpu_small.sh &
./gpu_instance_rnn_ptb_0.5gpu_small.sh &
```

Each one of these applications 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 by running nvidia-smi in a separate terminal window which is connected to the GPU instance.

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, and this brief guide serves only as an introduction. For some of the more advanced features please take a look at the help documentation available either as part of the command:

```
bfboost help
```

or online at: https://docs.bitfusion.io

For any question, comments, and feedback please contact Bitfusion at: http://support.bitfusion.io