

Using GPUs

Supported devices

On a typical system, there are multiple computing devices. In TensorFlow, the supported device types are CPU and GPU. They are represented as strings. For example:

- `"/cpu:0"`: The CPU of your machine.
- `"/gpu:0"`: The GPU of your machine, if you have one.
- `"/gpu:1"`: The second GPU of your machine, etc.

If a TensorFlow operation has both CPU and GPU implementations, the GPU devices will be given priority when the operation is assigned to a device. For example, `matmul` has both CPU and GPU kernels. On a system with devices `cpu:0` and `gpu:0`, `gpu:0` will be selected to run `matmul`.

Logging Device placement

To find out which devices your operations and tensors are assigned to, create the session with `log_device_placement` configuration option set to `True`.

```
# Creates a graph.
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

You should see the following output:

```
Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: Tesla K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/gpu:0
a: /job:localhost/replica:0/task:0/gpu:0
MatMul: /job:localhost/replica:0/task:0/gpu:0
[[ 22.  28.]
```

```
[ 49.  64.]
```

Manual device placement

If you would like a particular operation to run on a device of your choice instead of what's automatically selected for you, you can use `tf.device` to create a device context such that all the operations within that context will have the same device assignment.

```
# Creates a graph.
with tf.device('/cpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

You will see that now `a` and `b` are assigned to `cpu:0`. Since a device was not explicitly specified for the `MatMul` operation, the TensorFlow runtime will choose one based on the operation and available devices (`gpu:0` in this example) and automatically copy tensors between devices if required.

```
Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: Tesla K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/cpu:0
a: /job:localhost/replica:0/task:0/cpu:0
MatMul: /job:localhost/replica:0/task:0/gpu:0
[[ 22.  28.]
 [ 49.  64.]
```

Allowing GPU memory growth

By default, TensorFlow maps nearly all of the GPU memory of all GPUs (subject to

CUDA_VISIBLE_DEVICES

(<http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#env-vars>) visible to the process. This is done to more efficiently use the relatively precious GPU memory resources on the devices by reducing memory fragmentation

([https://en.wikipedia.org/wiki/Fragmentation_\(computing\)](https://en.wikipedia.org/wiki/Fragmentation_(computing))).

In some cases it is desirable for the process to only allocate a subset of the available memory, or to only grow the memory usage as is needed by the process. TensorFlow provides two Config options on the Session to control this.

The first is the `allow_growth` option, which attempts to allocate only as much GPU memory based on runtime allocations: it starts out allocating very little memory, and as Sessions get run and more GPU memory is needed, we extend the GPU memory region needed by the TensorFlow process. Note that we do not release memory, since that can lead to even worse memory fragmentation. To turn this option on, set the option in the ConfigProto by:

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

The second method is the `per_process_gpu_memory_fraction` option, which determines the fraction of the overall amount of memory that each visible GPU should be allocated. For example, you can tell TensorFlow to only allocate 40% of the total memory of each GPU by:

```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config, ...)
```

This is useful if you want to truly bound the amount of GPU memory available to the TensorFlow process.

Using a single GPU on a multi-GPU system

If you have more than one GPU in your system, the GPU with the lowest ID will be selected by default. If you would like to run on a different GPU, you will need to specify the preference explicitly:

```
# Creates a graph.
with tf.device('/gpu:2'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

If the device you have specified does not exist, you will get `InvalidArgumentError`:

```
InvalidArgumentError: Invalid argument: Cannot assign a device to node 'b':  
Could not satisfy explicit device specification '/gpu:2'  
[[Node: b = Const[dtype=DT_FLOAT, value=Tensor<type: float shape: [3,2]  
values: 1 2 3...>, _device="/gpu:2"]()]]
```

If you would like TensorFlow to automatically choose an existing and supported device to run the operations in case the specified one doesn't exist, you can set `allow_soft_placement` to `True` in the configuration option when creating the session.

```
# Creates a graph.  
with tf.device('/gpu:2'):  
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')  
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')  
    c = tf.matmul(a, b)  
# Creates a session with allow_soft_placement and log_device_placement set  
# to True.  
sess = tf.Session(config=tf.ConfigProto(  
    allow_soft_placement=True, log_device_placement=True))  
# Runs the op.  
print(sess.run(c))
```

Using multiple GPUs

If you would like to run TensorFlow on multiple GPUs, you can construct your model in a multi-tower fashion where each tower is assigned to a different GPU. For example:

```
# Creates a graph.  
c = []  
for d in ['/gpu:2', '/gpu:3']:  
    with tf.device(d):  
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])  
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])  
        c.append(tf.matmul(a, b))  
with tf.device('/cpu:0'):  
    sum = tf.add_n(c)  
# Creates a session with log_device_placement set to True.  
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))  
# Runs the op.  
print(sess.run(sum))
```

You will see the following output.

```
Device mapping:
/job:localhost/replica:0/task:0/gpu:0 -> device: 0, name: Tesla K20m, pci bus
id: 0000:02:00.0
/job:localhost/replica:0/task:0/gpu:1 -> device: 1, name: Tesla K20m, pci bus
id: 0000:03:00.0
/job:localhost/replica:0/task:0/gpu:2 -> device: 2, name: Tesla K20m, pci bus
id: 0000:83:00.0
/job:localhost/replica:0/task:0/gpu:3 -> device: 3, name: Tesla K20m, pci bus
id: 0000:84:00.0
Const_3: /job:localhost/replica:0/task:0/gpu:3
Const_2: /job:localhost/replica:0/task:0/gpu:3
MatMul_1: /job:localhost/replica:0/task:0/gpu:3
Const_1: /job:localhost/replica:0/task:0/gpu:2
Const: /job:localhost/replica:0/task:0/gpu:2
MatMul: /job:localhost/replica:0/task:0/gpu:2
AddN: /job:localhost/replica:0/task:0/cpu:0
[[ 44.  56.]
 [ 98. 128.]]
```

The [cifar10 tutorial](https://www.tensorflow.org/tutorials/deep_cnn) (https://www.tensorflow.org/tutorials/deep_cnn) is a good example demonstrating how to do training with multiple GPUs.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](http://creativecommons.org/licenses/by/3.0/) (<http://creativecommons.org/licenses/by/3.0/>), and code samples are licensed under the [Apache 2.0 License](http://www.apache.org/licenses/LICENSE-2.0) (<http://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

上次更新日期：六月 19, 2017