CUDA语义

`torch.cuda`会记录当前选择的GPU，并且分配的所有CUDA张量将在上面创建。可以使用`torch.cuda.device`上下文管理器更改所选设备。

但是，一旦张量被分配，您可以直接对其进行操作，而不考虑所选择的设备，结果将始终放在与张量相同的设备上。

默认情况下，不支持跨GPU操作，唯一的例外是`copy_()`。 除非启用对等存储器访问，否则对分布不同设备上的张量任何启动操作的尝试都将会引发错误。

下面你可以找到一个展示如下的小例子：

```
x = torch.cuda.FloatTensor(1)
# x.get_device() == 0
y = torch.FloatTensor(1).cuda()
# y.get_device() == 0

with torch.cuda.device(1):
    # allocates a tensor on GPU 1
    a = torch.cuda.FloatTensor(1)

    # transfers a tensor from CPU to GPU 1
    b = torch.FloatTensor(1).cuda()
    # a.get_device() == b.get_device() == 1

    c = a + b
    # c.get_device() == 1

    z = x + y
    # z.get_device() == 0

    # even within a context, you can give a GPU id to the .cuda call
    d = torch.randn(2).cuda(2)
    # d.get_device() == 2
```

最佳实践

使用固定的内存缓冲区

当副本来自固定（页锁）内存时，主机到GPU的复制速度要快很多。CPU张量和存储开放了一个`pin_memory()`方法，它返回该对象的副本，而它的数据放在固定区域中。

另外，一旦固定了张量或存储，就可以使用异步的GPU副本。只需传递一个额外的`async=True`参数到`cuda()`的调用。这可以用于将数据传输与计算重叠。

通过将`pin_memory=True`传递给其构造函数，可以使`DataLoader`将batch返回到固定内存中。

使用 nn.DataParallel 替代 multiprocessing

大多数涉及批量输入和多个GPU的情况应默认使用 `DataParallel` 来使用多个GPU。尽管有GIL的存在，单个python进程也可能使多个GPU饱和。

从0.1.9版本开始，大量的GPU(8+)可能未被充分利用。然而，这是一个已知的问题，也正在积极开发。和往常一样，测试你的用例吧。

调用 `multiprocessing` 来利用CUDA模型存在重要的注意事项；使用具有多处理功能的CUDA模型有重要的注意事项; 除非就是需要谨慎地满足数据处理需求，否则您的程序很可能会出现错误或未定义的行为。