使用JIT编译

- 为什么要使用即时（JIT）编译？
- 通过XLA运行TensorFlow图
  - 打开JIT编译
    - 会议
    - 手册
  - 将操作员放置在XLA设备上
- 教程
  - 步骤＃1：准备样本脚本
  - 步骤＃2：不使用XLA运行
  - 步骤＃3：使用XLA运行

原文链接：https://www.tensorflow.org/performance/xla/jit

译文链接：http://www.apache.wiki/pages/viewpage.action?pageId=10029453

贡献者：片刻 ApacheCN Apache中文网

> 注意：TensorFlow必须从源代码编译为包含XLA。

为什么要使用即时（JIT）编译？

TensorFlow / XLA
JIT编译器通过XLA编译并运行部分TensorFlow图形。与标准TensorFlow实现相比，其优点是XLA可以将多个运算符（内核融合）融合到少量的编译内核中。与TensorFlow执行者一样，融合运算符可以减少内存带宽需求并提高性能。

通过XLA运行TensorFlow图

有两种方法通过XLA运行TensorFlow计算，或者通过JIT编译操作员放置在CPU或GPU的设备上，或通过将操作员在XLA_CPU或XLA_GPUTensorFlow设备。将操作员直接放在TensorFlow XLA设备上，强制操作员在该设备上运行，主要用于测试。

> 注意：XLA CPU后端产生快速单线程代码（在大多数情况下），但尚未并行化TensorFlow CPU后端。XLA GPU后端与标准TensorFlow实现相比具有竞争力，有时更快，有时更慢。

打开JIT编译

JIT编译可以在会话级别打开或手动打开选择操作。这两种方法都是零拷贝---在编译的XLA内核和放置在同一设备上的TensorFlow操作符之间传递数据时，不需要复制数据。

会议

在会话级别打开JIT编译将导致所有可能的操作符被贪婪地编译成XLA计算。每个XLA计算将被编译为底层设备的一个或多个内核。

根据一些约束，如果图中有两个相邻的运算符都具有XLA实现，则它们将被编译为单个XLA计算。

在会话初始化期间，通过将配置`global_jit_level`设置为`tf.OptimizerOptions.ON_1`并传递配置，JIT编译在会话级别 打开。

```
# Config to turn on JIT compilation
config = tf.ConfigProto()
config.graph_options.optimizer_options.global_jit_level =
tf.OptimizerOptions.ON_1

sess = tf.Session(config=config)
```

> 注意：在会话级别打开JIT不会导致为CPU编译操作。用于CPU操作的JIT编译必须通过以下所述的手动方法完成。这个决定是由于CPU后端是单线程的。

手册

也可以为一个或多个操作员手动打开JIT编译。这是通过标记运算符来编译该属性来完成的 `_XlaCompile=true`。最简单的方法是通过`tf.contrib.compiler.jit.experimental_jit_scope()`定义的 范围`tensorflow/contrib/compiler/jit.py`。使用示例

```
jit_scope = tf.contrib.compiler.jit.experimental_jit_scope

    x = tf.placeholder(np.float32)
    with jit_scope():
      y = tf.add(x, x)  # The "add" will be compiled with XLA.
```

`_XlaCompileTensorFlow`

将操作员放置在XLA设备上

通过XLA运行计算的另一种方法是将操作员放在特定的XLA设备上。这种方法通常只用于测试。有效的目标是 `XLA_CPU`或`XLA_GPU`。

```
with tf.device("/job:localhost/replica:0/task:0/device:XLA_GPU:0"):
  output = tf.add(input1, input2)
```

`CPUGPUJITXLATensorFlow`

教程

本教程将介绍使用JIT打开的简单版本的MNIST softmax。目前JIT在会话级别，这是用于本教程的，仅支持GPU。

在开始本教程之前，请确认LD_LIBRARY环境变量或ldconfig包含`$CUDA_ROOT/extras/CUPTI/lib64`，其中包含CUDA Profiling Tools Interface（CUPTI）的库。TensorFlow使用CUPTI从GPU中提取跟踪信息。

步骤＃1：准备样本脚本

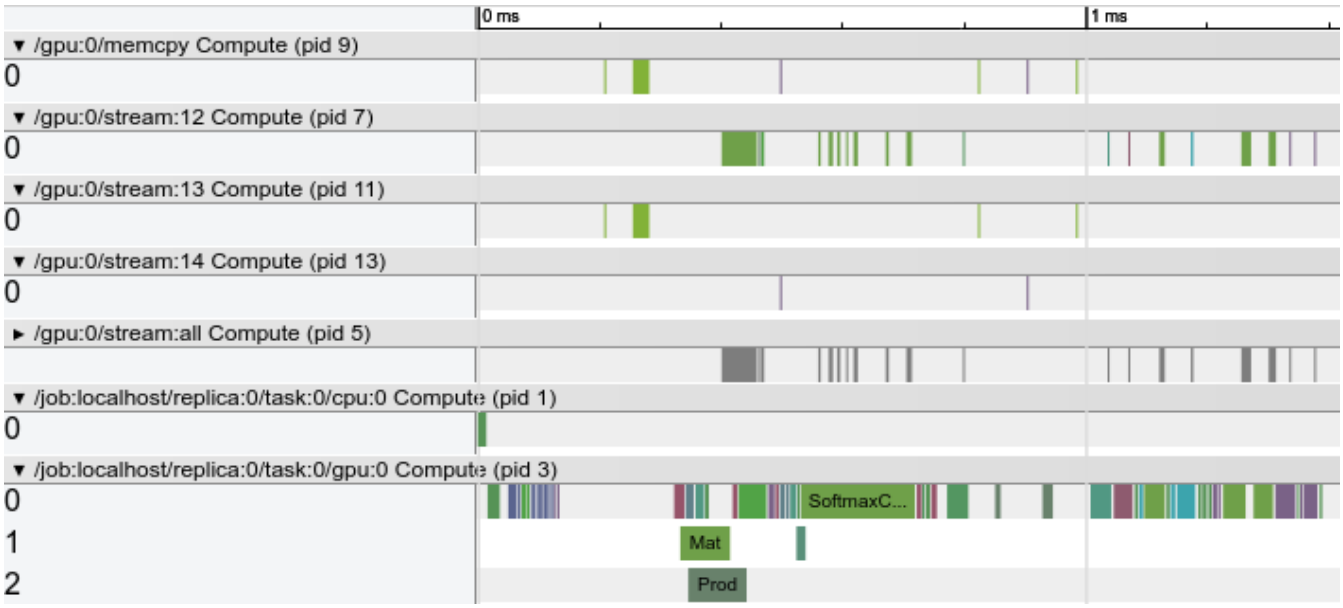
将mnist_softmax_xla.py下载或移动 到TensorFlow源代码树之外的文件夹中。

步骤＃2：不使用XLA运行

执行python脚本来训练没有XLA的模型。

```
python mnist_softmax_xla.py --xla=''
```

使用Chrome跟踪事件分析器（浏览到chrome：//跟踪），打开脚本完成时创建的时间轴文件：`timeline.ctf.json`。渲染的时间线应该类似于下面的图片，标有多个绿色框`MatMul`，可能在多个CPU之间。

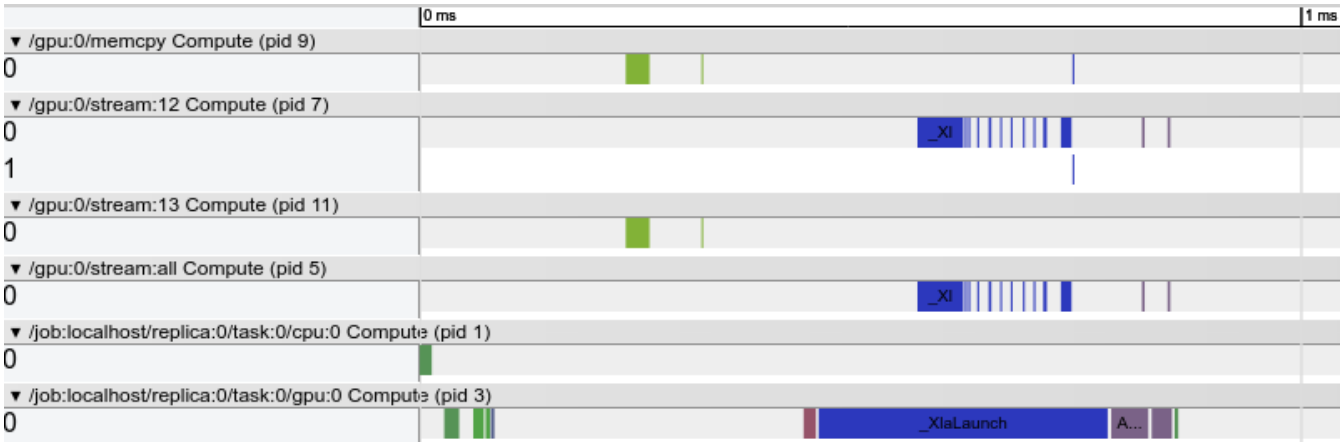


步骤＃3：使用XLA运行

执行python脚本使用XLA训练模型，并通过输出XLA图形的环境变量打开XLA的调试功能。

```
TF_XLA_FLAGS=--xla_generate_hlo_graph=.* python mnist_softmax_xla.py
```

`createdtimeline.ctf.json_XlaLaunch`



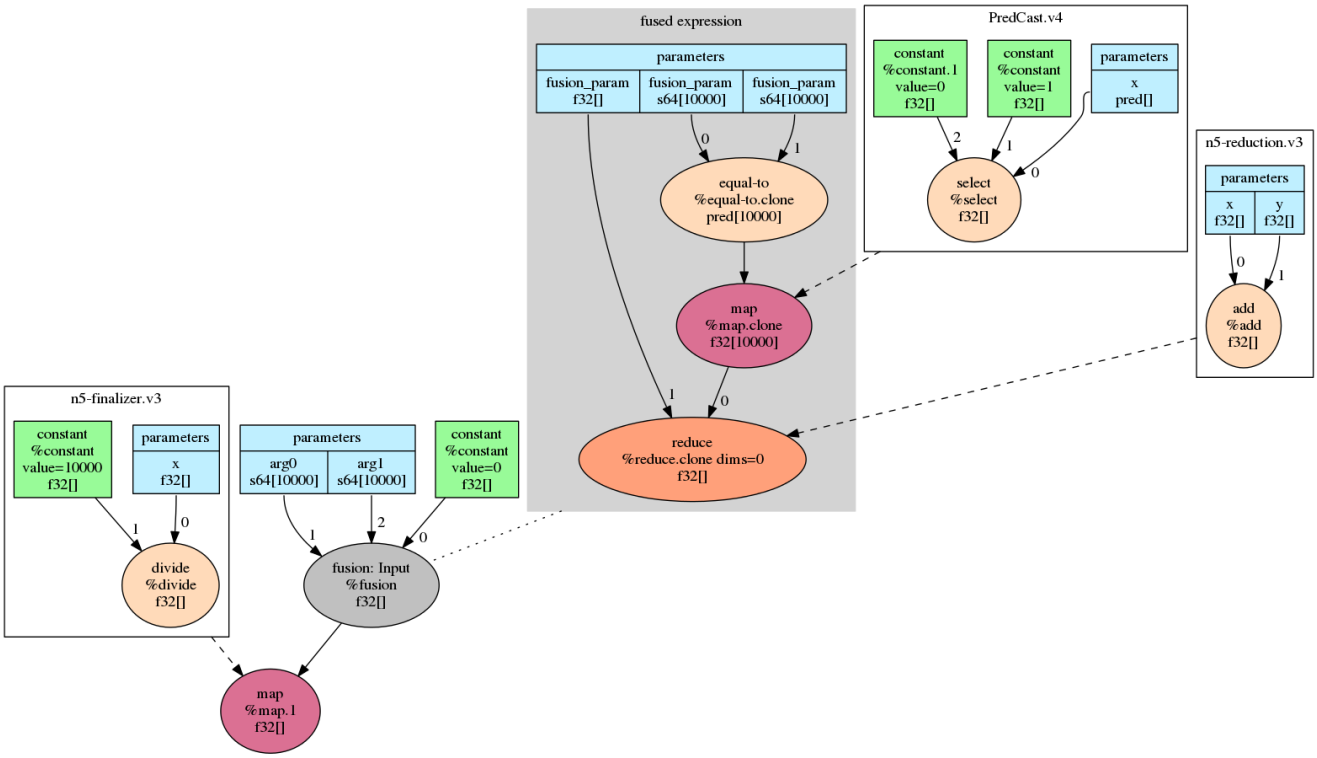
要了解发生了什么`_XlaLaunch`，请查看控制台输出，以获得类似于以下内容的语句：

```
computation cluster_0[_XlaCompiledKernel=true,_XlaNumConstantArgs=1].v82
[CPU:
pipeline start, before inline]: /tmp/hlo_graph_0.dot
```

`hlo_graph_xx.dotXLAhlo_graph_0.dotXLA`

要将.dot文件渲染为png，请安装 GraphViz并运行：

```
dot -Tpng hlo_graph_80.dot -o hlo_graph_80.png
```



fusion: after fusion, before fusion
cluster_1[_XlaCompiledKernel=true,_XlaNumConstantArgs=0].v10