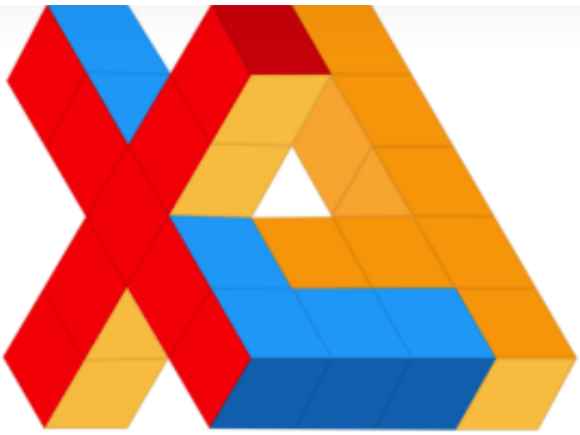


[腾讯机智]TensorFlow XLA工作原理

姜曦楠
外星人

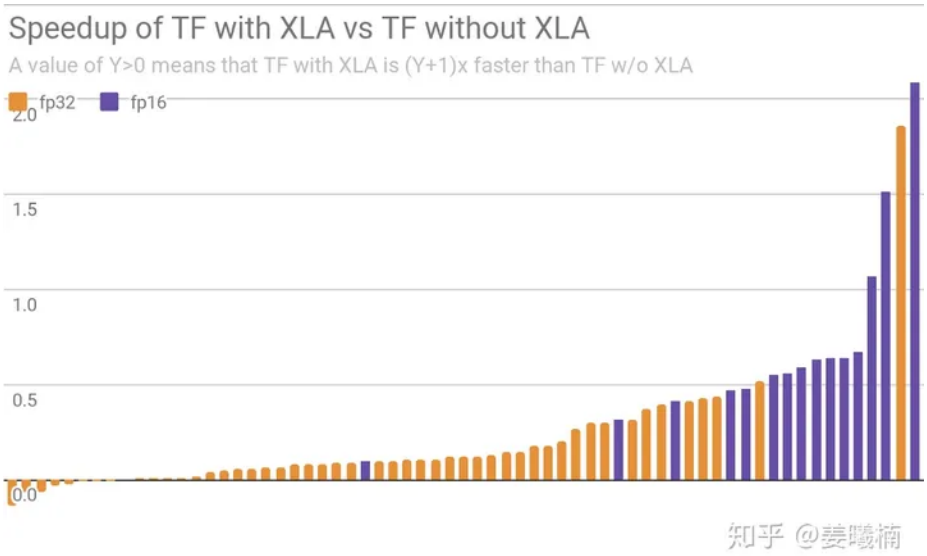
关注

你关注的 吴建明wujianming 赞同

XLA是TensorFlow图表的编译器，只需添加少量代码，即可明显加速TensorFlow ML模型。下图是谷歌官方提供的XLA性能表现。

> 在 Google 基准下的表现
> 下方是所有 XLA 团队基准模型（在 V100 GPU 上运行）的 TensorFlow 在使用和不使用 XLA 时的相对加速/减速图。

利用 XLA 将 GPU 性能推向极限
www.tensorflowers.cn/t/7338

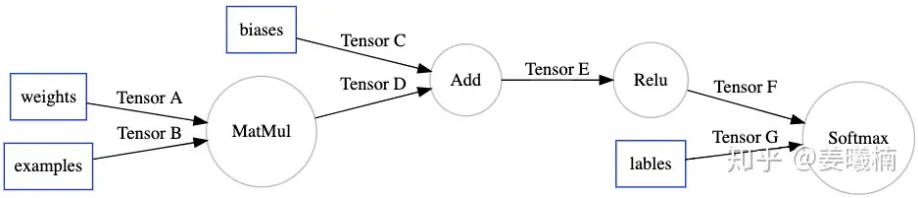


TensorFlow XLA加速对比

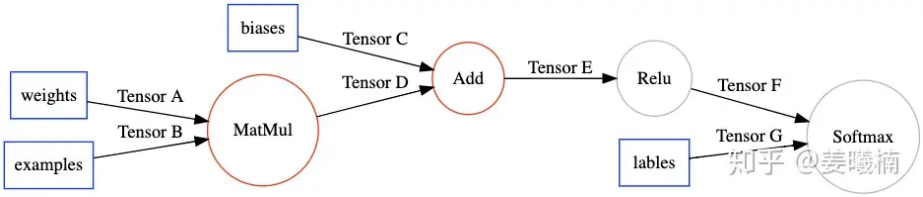
腾讯机智作为国内最专业的AI系统开发团队之一，在TensorFlow内核开发领域积累了丰富经验。本文将介绍XLA的工作原理。

JIT-Just In Time Compilation

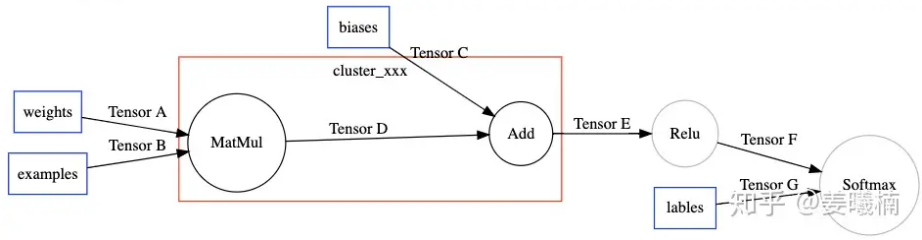
我们先来看XLA如何作用于TensorFlow的计算|



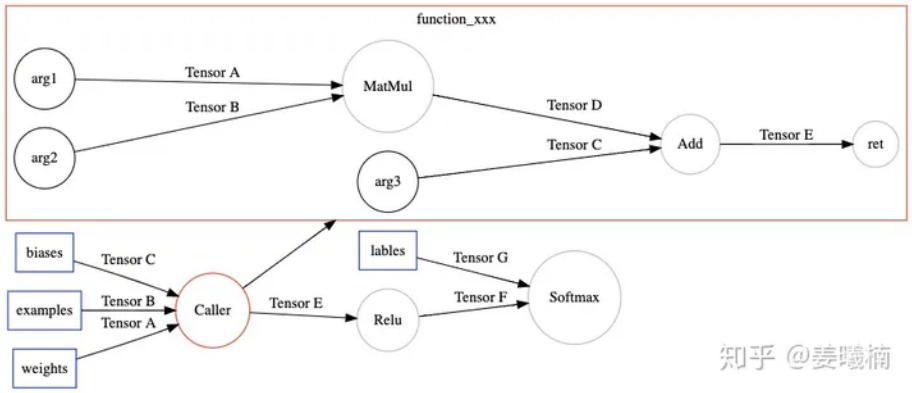
XLA通过一个TensorFlow的图优化Pass(MarkForCompilation)，在TensorFlow计算图中找到适合被JIT编译的区域。这里我们假设XLA仅支持MatMul和Add。



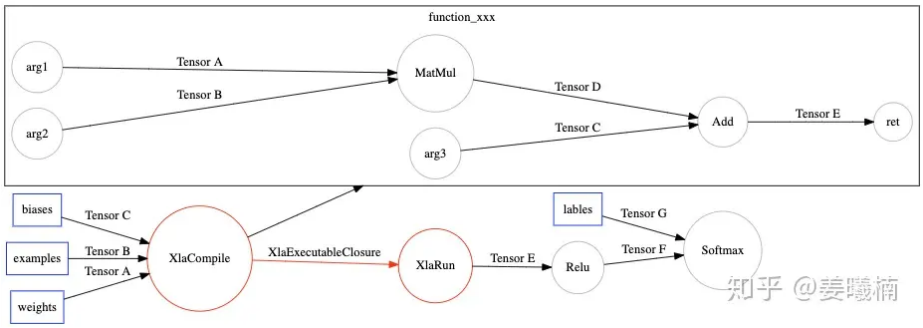
TensorFlow XLA把这个区域定义为一个Cluster，作为一个独立的JIT编译单元，在TensorFlow计算图中通过Node Attribute标示。



然后另一个TensorFlow的图优化Pass(EncapsulateSubgraphs)，把cluster转化成TensorFlow的一个Function子图。在原图上用一个Caller节点表示这个Function在原图的位置。



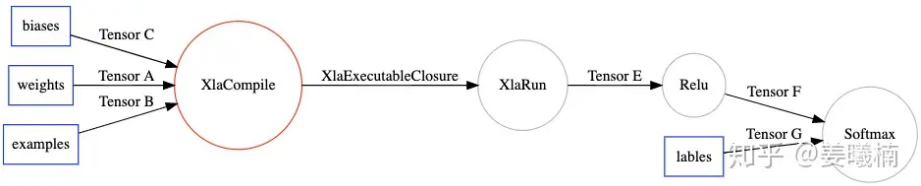
最后调用TensorFlow的图优化Pass(BuildXlaOps)，把Function节点转化成特殊的Xla节点。



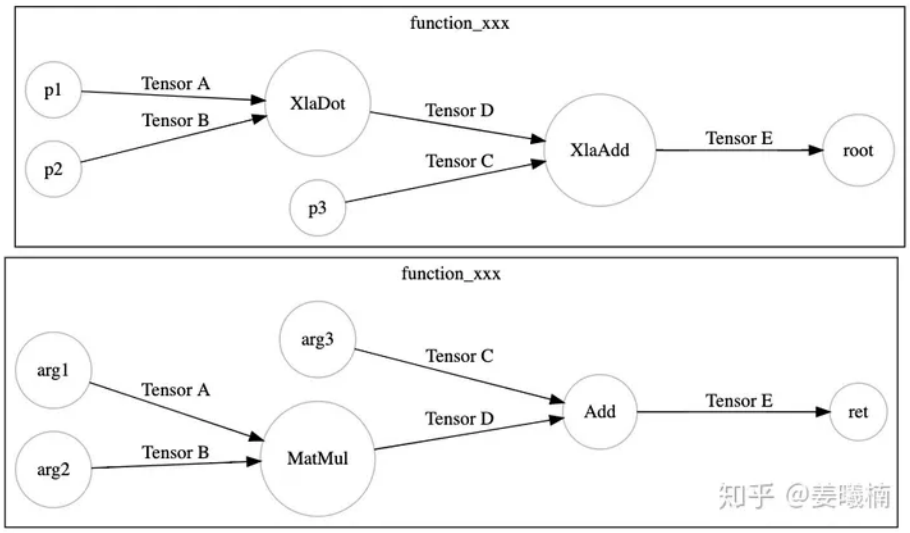
在TensorFlow运行时，运行到XlaCompile时，编译Xla cluster子图，然后把编译完的Executable可执行文件通过XlaExecutableClosure传给XlaRun运行。

TF2XLA

TensorFlow运行到XlaCompile节点时。



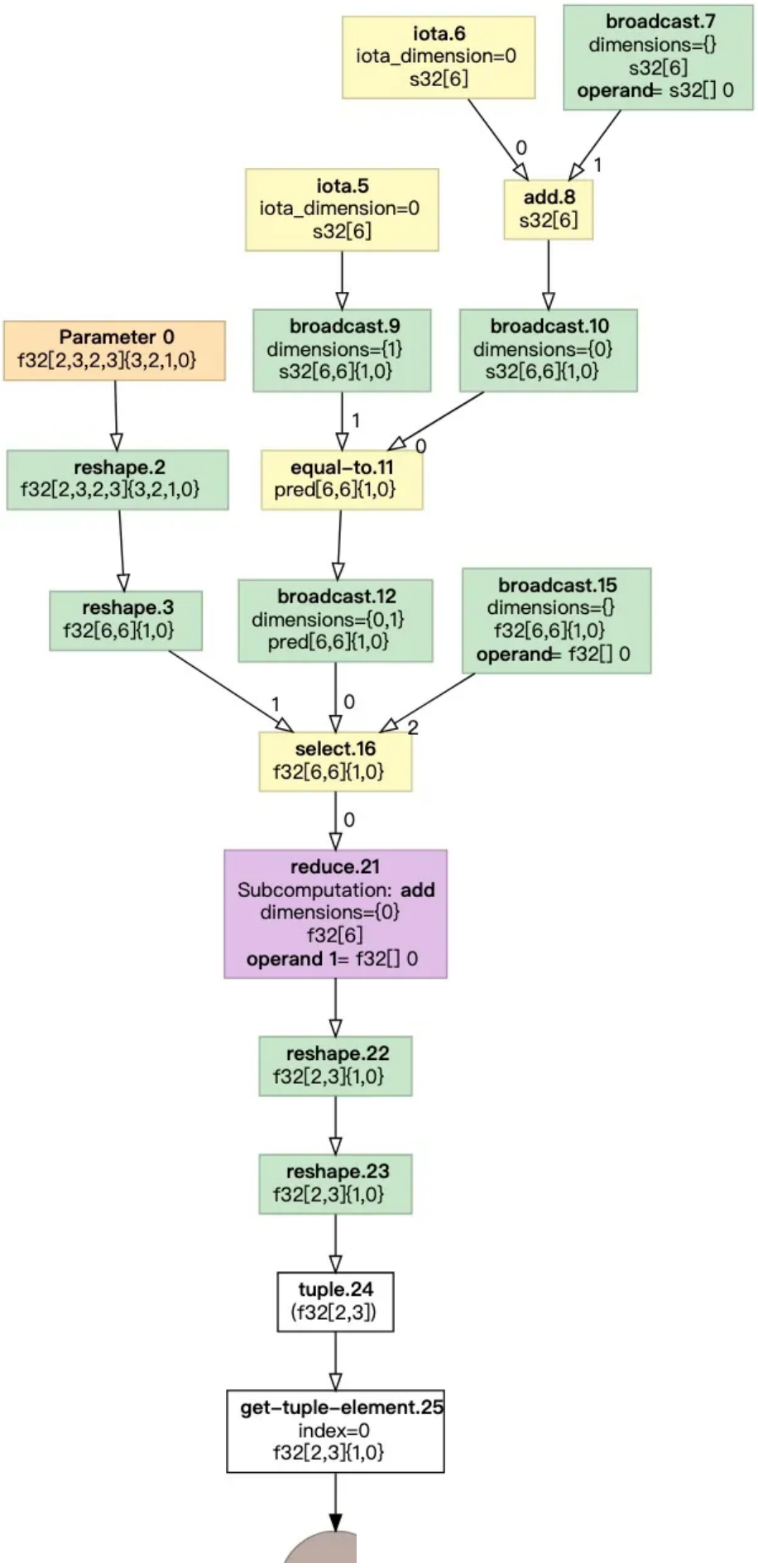
为了编译这个Function，通过把TensorFlow子图所有的节点翻译成XLA HLO Instruction虚拟指令的形式表达，整个子图也由此转化成XLA HLO Computation。



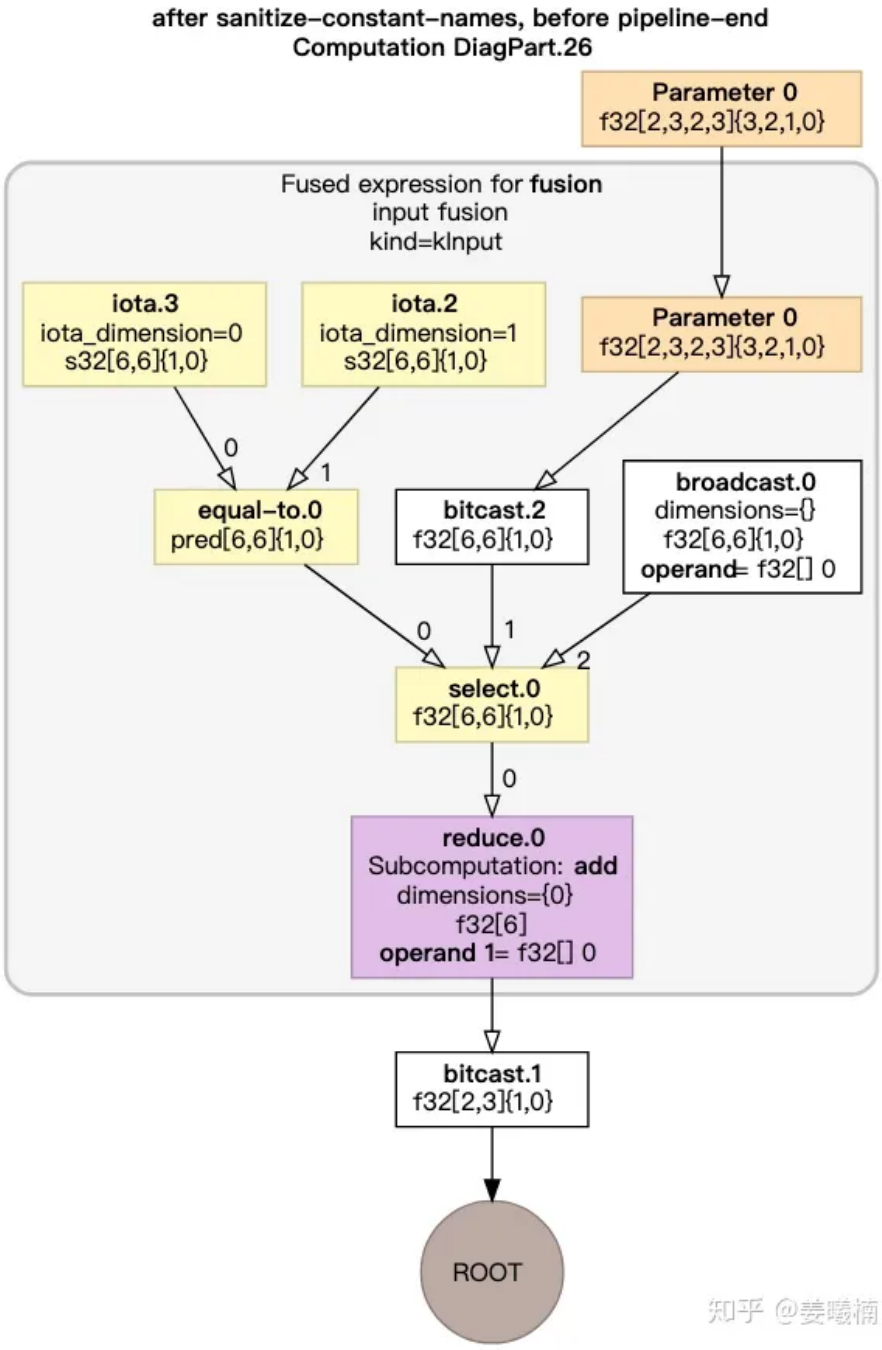
XLA-HLO

XLA在HLO的图表达上进行图优化。聚合可在同一个GPU Kernel中执行的HLO指令。

HLO图优化前


after pipeline-start, before dynamic-index-splitter
Computation DiagPart.26
iota.6
iota_dimension=0
s32[6]
broadcast.7
dimensions={}
s32[6]
operand= s32[] 0
0
1
iota.5
iota_dimension=0
s32[6]
add.8
s32[6]
Parameter 0
f32[2,3,2,3]{3,2,1,0}
broadcast.9
dimensions={1}
s32[6,6]{1,0}
broadcast.10
dimensions={0}
s32[6,6]{1,0}
1
0
reshape.2
f32[2,3,2,3]{3,2,1,0}
equal-to.11
pred[6,6]{1,0}
reshape.3
f32[6,6]{1,0}
broadcast.12
dimensions={0,1}
pred[6,6]{1,0}
broadcast.15
dimensions={}
f32[6,6]{1,0}
operand= f32[] 0
1
0
2
select.16
f32[6,6]{1,0}
0
reduce.21
Subcomputation: add
dimensions={0}
f32[6]
operand 1= f32[] 0
reshape.22
f32[2,3]{1,0}
reshape.23
f32[2,3]{1,0}
tuple.24
(f32[2,3])
get-tuple-element.25
index=0
f32[2,3]{1,0}

HLO图优化后



代码生成

首先根据虚拟指令分配GPU Stream和显存。

然后IrEmitter把HLO Graph转化成由编译器的中间表达LLVM IR表示的GPU Kernel。LLVM IR如下所示：

```
; ModuleID = 'cluster_36__XlaCompiledKernel_true__XlaNumConstantArgs_1__XlaNumResource
source_filename = "cluster_36__XlaCompil
target datalayout = "e-i64:64-i128:128-v
```

```
target triple = "nvptx64-nvidia-cuda"

@0 = private unnamed_addr constant [4 x i8] zeroinitializer
@1 = private unnamed_addr constant [4 x i8] zeroinitializer
@2 = private unnamed_addr constant [4 x i8] zeroinitializer
@3 = private unnamed_addr constant [4 x i8] zeroinitializer
@4 = private unnamed_addr constant [4 x i8] zeroinitializer
@5 = private unnamed_addr constant [4 x i8] zeroinitializer
@6 = private unnamed_addr constant [4 x i8] zeroinitializer

define void @fusion_1(i8* align 16 dereferenceable(3564544) %alloc2, i8* align 64 dere
entry:
  %output.invar_address = alloca i64
  %output_y.invar_address = alloca i64
  %arg0.1.raw = getelementptr inbounds i8, i8* %alloc2, i64 0
  %arg0.1.typed = bitcast i8* %arg0.1.raw to [944 x [944 x float]]*
  %fusion.1.raw = getelementptr inbounds i8, i8* %temp_buf, i64 0
  %fusion.1.typed = bitcast i8* %fusion.1.raw to [944 x float]*
  %0 = call i32 @llvm.nvvm.read.ptx.sreg.tid.x(), !range !4
  %thread.id.x = sext i32 %0 to i64
  %thread.x = urem i64 %thread.id.x, 944
  %thread.y = udiv i64 %thread.id.x, 944
  %1 = alloca float
  %partial_reduction_result.0 = alloca float
  %2 = load float, float* bitcast ([4 x i8]* @0 to float*)
  %3 = getelementptr inbounds float, float* %partial_reduction_result.0, i32 0
  store float %2, float* %3
  %current_output_linear_index_address = alloca i64
  %4 = alloca i1
  store i1 false, i1* %4
  %5 = call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x(), !range !5
  %block.id.x = sext i32 %5 to i64
  %6 = udiv i64 %block.id.x, 1
  %7 = urem i64 %6, 1
  %8 = udiv i64 %block.id.x, 1
  %9 = urem i64 %8, 8
  %10 = udiv i64 %block.id.x, 8
  %block_origin.0 = mul i64 %10, 1
  %block_origin.1 = mul i64 %9, 1
 ...
```

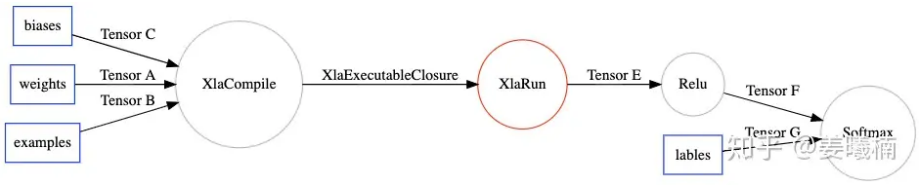
由LLVM生成nvPTX（Nvidia定义的虚拟底层指令表达形式）表达，进而由NVCC生成CuBin可执行代码。PTX如下所示:

```
//
// Generated by LLVM NVPTX Back-End
//

.version 6.0
.target sm_70
.address_size 64

    // .globl   fusion_1

.visible .entry fusion_1(
    .param .u64 fusion_1_param_0,
    .param .u64 fusion_1_param_1
)
.reqntid 944, 1, 1
{
    .reg .pred  %p<9>;
```

```
    .reg .f32   %f<25>;
    .reg .b32   %r<31>;
    .reg .b64   %rd<61>;

    ld.param.u64    %rd27, [fusion_1_param_0];
    ld.param.u64    %rd28, [fusion_1_param_1];
    cvta.to.global.u64  %rd29, %rd28;
    cvta.to.global.u64  %rd1, %rd27;
    cvta.global.u64     %rd2, %rd29;
    mov.u32     %r3, %tid.x;
    cvt.u64.u32     %rd3, %r3;
    mov.u32     %r1, %ctaid.x;
    setp.eq.s32     %p1, %r1, 7;
    @%p1 bra    LBB0_4;
    bra.uni     LBB0_1;
LBB0_4:
    selp.b64    %rd4, 48, 128, %p1;
    cvt.u32.u64     %r26, %rd3;
    shl.b64     %rd47, %rd3, 2;
    add.s64     %rd48, %rd47, %rd1;
    add.s64     %rd59, %rd48, 3383296;
    or.b32      %r27, %r26, 845824;
    mul.wide.u32    %rd49, %r27, 582368447;
    shr.u64     %rd50, %rd49, 39;
    cvt.u32.u64     %r28, %rd50;
    mul.lo.s32  %r29, %r28, 945;
    sub.s32     %r2, %r27, %r29;
    mov.f32     %f23, 0f00000000;
...
```

代码执行

当TensorFlow运行到XlaRun时运行由XlaCompile编译得到的GPU可执行代码（Cubin或PTX）。



后记

We're Hiring!

腾讯机器学习系统团队正在快速发展，诚邀各系统软件领域专业人士加盟。招聘方向包括但不限于机器学习框架（TensorFlow，PyTorch，MxNet等），编译器（LLVM，MLIR，XLA，Glow，TVM，PlaidML，Pluto等），GPU算子（CUDA，OpenCL等），分布式系统（Kubernetes，Spark等），AutoML（Ray，AutoKeras）。

简历投递xinanjiang@tencent.com

编辑于 2019-12-20 07:06

TensorFlow 学习