sketch2sky

What I Cannot Create, I Do Not Understand —Richard Feynman And I

☰ Primary Menu

Tensorflow OpKernel机制详解

1033 Jiang XIAO

2019年8月3日 at pm3:41 (last edited 2020年5月26日 at pm11:47)

OpKernel是Op的具体实现, tf中已经实现的tfop的OpKernel在源码中的tensorflow/core/framework/kernel/中, OpKernel通过注册时使用相同的名字将自己和相应的Op联系到一起.

在tf中, OpKernel进一步可以分为两类, `OpKernel`和`AsyncOpKernel`:

1. `OpKernel`是同步执行的, 即"`Compute()`"返回即认为数据已经被正确处理, 注册OpKernel, 子类需要重写其Compute()方法.
2. `AsyncOpKernel`是对`OpKernel`的封装, 顾名思义, `AsyncOpKernel`执行返回并不意味着数据已经被处理完毕, 数据的真正被处理完毕时通过回调的方式通知Op执行引擎, 注册一个`AsyncOpKernel`, 子类需要实现"`AsyncCompute()`"而不是Compute().

接口形式

无论是哪种OpKernel, 均使用"`REGISTER_KERNEL_BUILDER()`"注册到运行核心.

```
//tensorflow/core/common_runtime/kernels/nccl_ops.cc
#include "third_party/nccl/nccl.h"
#include "tensorflow/core/framework/op_kernel.h"
#include "tensorflow/core/nccl/nccl_manager.h"
namespace tensorflow {
class NcclAllReduceOpKernel : public AsyncOpKernel {
 public:
  explicit NcclAllReduceOpKernel(OpKernelConstruction* c)
      : NcclReduceOpBase(c) {}

  void ComputeAsync(OpKernelContext* c, DoneCallback done) override {
    //...
  }
};
REGISTER_KERNEL_BUILDER(Name("NcclAllReduce").Device(DEVICE_GPU),
                        NcclAllReduceOpKernel);
}
```

注册原理

注册机制的实现代码主要集中在tensorflow/core/framework/op_kernel.h(.cc). 与 Optimization以及Op在注册时直接构造一个static OptimizationPassRegistration(OpRegistrationData)对象的机制略有不同, OpKernel的通过一些trick实现了对OpKernel的延迟构造, 即"REGISTER_OP_KERNEL_BUILDER()"并没有直接构造一个"OpKernel"实例, 而是构造一个" static ::tensorflow::kernel_factory::OpKernelRegistrar "对象, 并借由该构造过程构造并注册一个"KernelRegistration" 对象到 global_regsitry, 该构造过程接受上层传入的, 用于new一个OpKernel的"[](::tensorflow::OpKernelConstruction* context) -> ::tensorflow::OpKernel* { return new __VA_ARGS__(context);}" 函数, 在上层真正需要这个OpKernel的时候, 才会通过一系列调用最终执行该"create_fn()/lambda"来构造一个实实在在的OpKernel对象. 即整体上不再是Registry->Registration(Optimization/Op对象), 而是 Registry->Registrar->Registration->在需要时create_fn()构造OpKernel对象.

构造一个OpKernelRegistrar:

```
//op_kernel.h +1404
#define REGISTER_KERNEL_BUILDER(kernel_builder, ...) \
  REGISTER_KERNEL_BUILDER_UNIQ_HELPER(__COUNTER__, kernel_builder, __VA_ARGS__)

#define REGISTER_KERNEL_BUILDER_UNIQ_HELPER(ctr, kernel_builder, ...) \
  REGISTER_KERNEL_BUILDER_UNIQ(ctr, kernel_builder, __VA_ARGS__)

#define REGISTER_KERNEL_BUILDER_UNIQ(ctr, kernel_builder, ...)         \
  constexpr bool should_register_##ctr##__flag =                       \
      SHOULD_REGISTER_OP_KERNEL(#__VA_ARGS__);                          \
  static ::tensorflow::kernel_factory::OpKernelRegistrar               \
      registrar__body__##ctr##__object(                                 \
          should_register_##ctr##__flag                                 \
              ? ::tensorflow::register_kernel::kernel_builder.Build()  \
              : nullptr,                                                \
          #__VA_ARGS__,                                                 \
          [](::tensorflow::OpKernelConstruction* context)              \
              -> ::tensorflow::OpKernel* {                              \
            return new __VA_ARGS__(context);                            \
          });
```

在OpKernelRegistrar的构造过程中将"KernelRegistration"加入"KernelRegistry":

```
  OpKernelRegistrar(const KernelDef* kernel_def, StringPiece kernel_class_name,OpKernel* (*c
    InitInternal(kernel_def, kernel_class_name, absl::make_unique<PtrOpKernelFactory>(create
      const string key = Key(kernel_def->op(), DeviceType(kernel_def->device_type()), kernel
      auto global_registry = reinterpret_cast<KernelRegistry*>(GlobalKernelRegistry());
        static KernelRegistry* global_kernel_registry = new KernelRegistry;
        return global_kernel_registry;
      global_registry->registry.emplace(key, KernelRegistration(*kernel_def, kernel_class_na
```

在需要的时候, 调用之前注册的factory接口构造OpKernel实例:

```
PyEval_EvalCodeEx()
  PyEval_EvalFrameEx()
    _wrap_TF_FinishOperation()
      tensorflow::ShapeRefiner::AddNode()
        tensorflow::ShapeRefiner::RunShapeFn()
          tensorflow::ShapeRefiner::EvaluateConstantTensorForEdge()
            tensorflow::EvaluateConstantTensor()
              tensorflow::GraphRunner::Run()
```

```
 9.                   tensorflow::NewLocalExecutor()
10.                     tensorflow::(anonymous namespace)::ExecutorImpl::Initialize()
11.                       tensorflow::CreateNonCachedKernel()
12.                         tensorflow::CreateOpKernel()
13.                           const KernelRegistration* registration;
14.                           FindKernelRegistration()
15.                             FindKernelRegistration()
16.                               const string key = Key(node_op, device_type, label);
17.                               auto typed_registry = GlobalKernelRegistryTyped();
18.                               tf_shared_lock lock(typed_registry->mu);
19.                               auto regs = typed_registry->registry.equal_range(key);
20.                               for (auto iter = regs.first; iter != regs.second; ++iter)
21.                                 *reg = &iter->second;
22.                           OpKernelConstruction context();
23.                           *kernel = registration->factory->Create(&context);
24.                             (*create_func_)(context);
```

-12-构造OpKernel入口 op_kernel.cc
-14-获取KernelRegistration的入口
-16-用于检索KenelRegitration的key。由于一个Op可以有多个OpKernel实现版本， 所以检索用于构造OpKernel的KernelRegistration时，不能只根据Op，还要结合其他信息， 典型的比如device_type, 这里的key就是将node_op, device_type和label组合一起的string。
-23-PtrOpKernelFactory是OpKernelFactory的子类，这里的Create()也是来自OpKernelFactory，其实质就是调用注册时传入的create_func_()。
-24-真正的构造OpKernel实例

调试方法

同Op一样, 也有一些可以用于调试的常用接口, 只不过封装思路不同, 相关的方法并不在Registry或任何类中:

```
//op_kernel.h
// Checks whether a given kernel is registered on device_type.
bool KernelDefAvailable(const DeviceType& device_type, const NodeDef& node_def);

// If node of node_name, experimental_debug_info, node_op, node_device and
// node_attrs has a corresponding kernel registered on device_type, returns OK
// and fill in the kernel def and kernel_class_name. <def> and
// <kernel_class_name> may be null.
Status FindKernelDef(
    const DeviceType& device_type, StringPiece node_name,
    bool has_experimental_debug_info,
    const NodeDef_ExperimentalDebugInfo& experimental_debug_info,
    StringPiece node_op, StringPiece node_device, AttrSlice node_attrs,
    const KernelDef** def, string* kernel_class_name);

// If node_def has a corresponding kernel registered on device_type,
// returns OK and fill in the kernel def and kernel_class_name. <def> and
// <kernel_class_name> may be null.
Status FindKernelDef(const DeviceType& device_type, const NodeDef& node_def,
                     const KernelDef** def, string* kernel_class_name);

// Writes a list of all registered kernels to LOG(INFO), to help users debug
// missing kernel errors.
void LogAllRegisteredKernels();
```

```
// Gets a list of all registered kernels.
KernelList GetAllRegisteredKernels();

// Gets a list of all registered kernels for which predicate returns true
KernelList GetFilteredRegisteredKernels(
    const std::function<bool(const KernelDef&)>& predicate);

// Gets a list of all registered kernels for a given op
KernelList GetRegisteredKernelsForOp(StringPiece op_name);
```

Related:
Tensorflow XLA HLO I — BufferLiveness
Tensorflow XLA Service 详解 II
Tensorflow XLA Service 详解 I
Tensorflow XLA Client | HloModuleProto 详解
Tensorflow XlaOpKernel | tf2xla 机制详解
Tensorflow JIT 技术详解
Tensorflow JIT/XLA UML
Tensorflow OpKernel机制详解
Tensorflow Op机制详解
Tensorflow Optimization机制详解
Tensorflow 图计算引擎概述

📂 技术 🏷 Tensorflow, 技术

‹ Tensorflow Op机制详解

Tensorflow 编译方法 ›

One comment on “Tensorflow OpKernel机制详解”

Pingback: Tensorflow 图计算引擎概述 - sketch2sky

Leave a Reply

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Quick search... Go