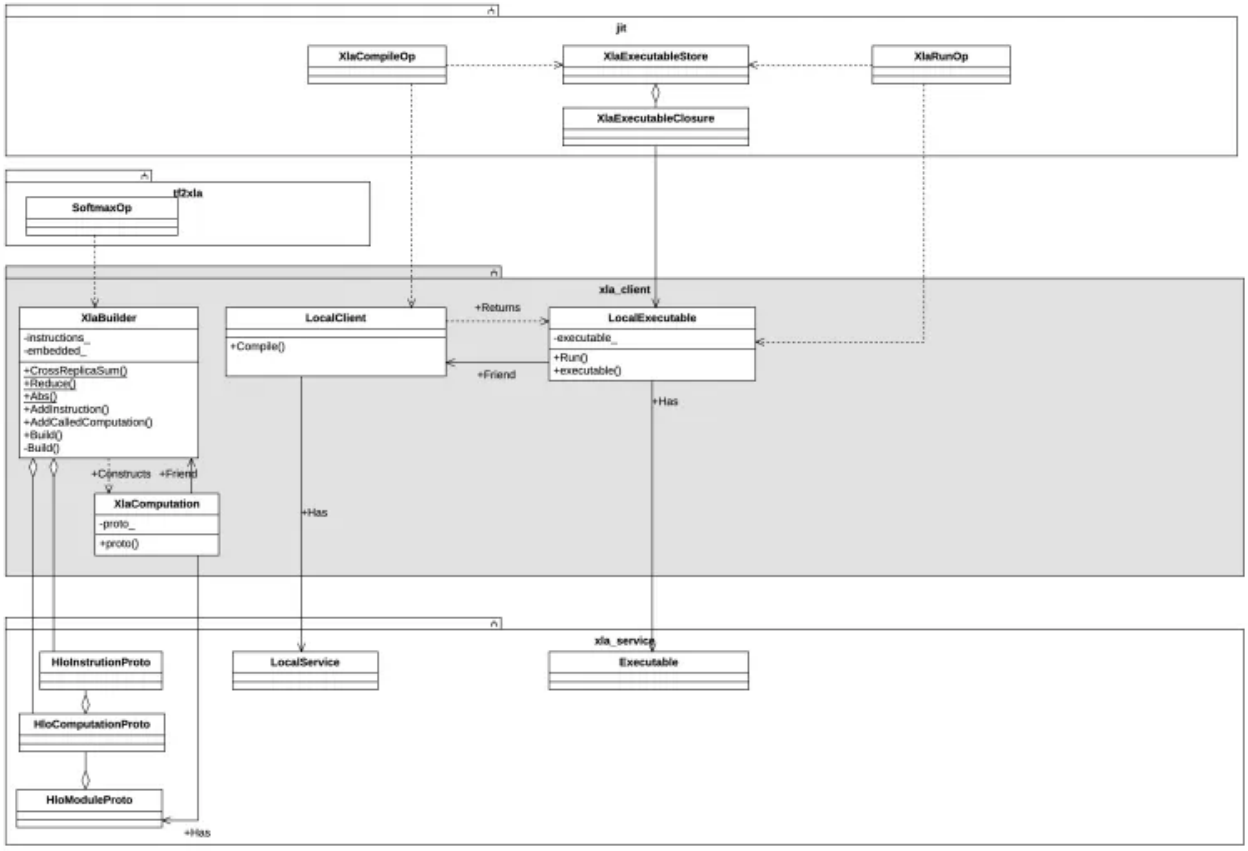
Tensorflow XLA Client | HloModuleProto 详解

1173 Jiang XIAO

2019年9月26日 at pm1:12 (last edited 2020年6月20日 at am9:30)

compiler/aot/	以AOT的方式将tf2xla/接入TF引擎
compiler/jit/	以JIT的方式将tf2xla/接入TF引擎， 核心是9个优化器和3个tfop，其中XlaCompileOp调用tf2xla的“编译”入口完成功能封装，XlaRunOp调用xla/client完成“运行”功能。
compiler/tf2xla/	对上提供xla_compiler.cc:XlaCompiler::CompileFunction()供jit:compile_fn()使用将cluster转化为XlaComputation。核心是利用xla/client提供的接口，实现对XlaOpKernel的“Symbolic Execution”功能。每个XlaOpKernel子类均做的以下工作: 从XlaOpKernelContext中取出XlaExpression或XlaOp, 调用xla/client/xla_buidler.h提供的方法完成计算, 将计算结果的XlaOp存入XlaKernelContext.**
compiler/xla/client/	对上提供**xla_builder.cc:Builder**等供CompileFunction()使用，将Graph由Op表达转化为HloModuleProto:HloComputationProto:HloInstructionProto表达并保存在XlaComputation中。 对上提供**local_client.cc:LocalClient::Compile()**，作为编译入口供jit:BuildExecutable()使用，将已经得到的XlaComputation交给service并进一步编译为二进制。 对上提供**local_client.cc:LocalExecutable::Run()**，作为运行入口供jit/kernels/xla_ops.cc:XlaRunOp使用，通过Key找到相应的二进制交给service层处理
compiler/xla/service/	对上提供local_service.cc:LocalService::BuildExecutable()供LocalClient::Compile()使用实现真正的编译，承接XlaComputation封装的HloProto, 将其转化为HloModule:HloComputation:HloInstruction表达, 对其进行优化之后, 使用LLVM后端将其编译为相应Executable后端的二进制代码 对上提供executable.cc:Executable::ExecuteOnStream()供LocalExecutable::Run()使用实现真正的执行二进制。

compiler/xla/client 向上为tf2xla/下的XlaOpKernel的实现提供支撑, 将上层请求转换为HloModule交给下层xla/service优化并编译.



接口上, client做上表中的三件事 , 实际上, 只有XlaOpKernel->HloProto在Client完成, 而对于另外两个编译和执行，都是在service中完成的，或者说，**Client本质上是Service的Facade + Proxy**，对繁琐的构造HloModuleProto的过程进行了封装，降低了XLA Service与上层模块的耦合

client.h:Client	Client基类, 用于多态实现
client_library.h:ClientLibarary	使用单例构造client_library对象，用于检索/构造所需的Client实例
lib/	同builder一起实现”Symbolic Execution”
local_client.h:LocalClient, LocalExecutable	JIT 使用的LocalClient定义, 是Service相关方法的Proxy
xla_builder.h:XlaBuilder	提供接口用tf2xla使用实现”Symbolic Execution”, 是其中XlaBuilder::Build()是构造client构造HloModuleProto的核心方法。对于需要多个步骤完成初始化的类， 我们会使用**Builder模式**， 这就是个例子
xla_computation.h:XlaComputation	XlaComputation对象是对HloModuleProto的封装, 用于进一步二进制编译

职责1: 构造HloModuleProto

编译二进制之前首先要完成Graph表达方式的映射: Client之前的tf2xla的Graph由使用Op表达, Client之后的Service的Graph使用HloInstruction表达, Client负责完成这种转化, 具体地, 就是将Op转化为HloProto格式, 再交给

Service解析为Hlo格式, 其中的HloProto就是封装在XlaComputation中. 所以, 这个过程可以看做是"编译"的准备工作. 在这个过程中, Graph, Cluster, XlaComputation, HloModuleProto, HloModule是一一对应的，都是"Program"的概念， 这一点从源码的ProgramShape等变量名中都可以体现

按照OOP的方式理解OOP的代码, 在说明Client是如果构造HloModuleProto之前, 来介绍几个概念, HloModuleProto-HloComputationProto- HloInstructionProto, 这3个概念是XLA Service提供给上层调用者(Here, XLA Client)的用于构造HloModule-HloComputation-HloInstruction的protobuf形式的"原材料", Module-Computation-Instruction, 在概念上可以对应**程序-函数-语句**

1. HloProto中的函数调用都是以"**半inline**"方式实现, inline的部分是指类似"a = add(x, y);"这样的语句, 会将add函数体整个copy到该语句的上下文, 在执行的时候调用之, 在HloProto中, 被调用的XlaComputation也会被copy到调用语句的上下文, "上下文"就是调用语句HloInstructionProto所处的XlaComputation(对应一个HloModuleProto)的XlaBuilder实例. "半"的部分是指被copy的XlaComputation并不是直接在调用语句处展开, 而是在调用语句的HloModuleProto中存储被调用的XlaComputation的Id, 执行的时候直接跳转执行.
2. 类似多文件编程, 构造的过程可以有多个HloModuleProto, 但由于上一条所述, 最终所有的被调用函数均会被copy到根HloModuleProto的上下文, 最终只有也只需要根HloModuleProto就足以以描述整个程序的逻辑
3. 一个C/C++程序有main作为入口, 对应到HloModuleProto就是entry_computation, 一个函数的第一条语句, 对应到HloComputationProto就是root_id

下面我们简要分析下代码, 既然Hlo是描述一个程序, 那么就需要解决一个基本的问题: 描述出函数->语句->函数->语句…这样的递归结构, OOP的Composite模式, 或者内核的kset-kobj结构, 都是解决递归的好方式, Hlo模块的实现也比较简单, 可以认为是kset-kobj的结构的C++版本：在HloComputationProto里记录HloInstructionProto，在HloInstructionProto里记录被调用的另一个HloCProto实例Id

```
tensorflow::XlaCompilationCache::Compile(kernel/out_compilation_result, executable/ou
  return tensorflow::XlaCompilationCache::CompileImpl(out_compilation_result, out_exe
    if (!entry->compiled):
      e->std::function<Status(XlaCompiler* compiler,...(entry->compilation_result)
        tensorflow::XlaCompiler::CompileFunction(out_compilation_result/result, fn_na
          tensorflow::XlaCompiler::CompileGraph()
            ExecuteGraph(context/xla_context)
              tensorflow::GraphCompiler::Compile()
                for node in topo_sorted_nodes:
                  tensorflow::XlaCompilationDevice::Compute(op_context)
                    tensorflow::XlaOpKernel::Compute()
                      XlaOpKernelContext xla_context(context)
                      Compile(&xla_context);
                          //xla/client/xla_builder.cc
                          //CrossReplicaSum
                          b = CreateSubBuilder("sum");
                            return sub_builder = absl::make_unique<XlaBuilder>(comput
                          Add(b->Parameter(),b->Parameter();
                          auto computation = b->Build();
                          AddCalledComputation(computation, &instr)
                            for e : computation.computations():
                              HloComputationProto new_computation(e);
                              computation_id = GetNextId();
                              remapped_ids[new_computation.id()] = computation_id
                              for instruction : *new_computation.mutable_instructions
                                int64 instruction_id = GetNextId();
                                remapped_ids[instruction.id()] = instruction_id;
                                new_computation.set_root_id(remapped_ids.at(new_compu
                                imported_computations.push_back(new_computation);
                            instr->add_called_computation_ids(remapped_ids.at(computa
```

```
                                for (auto& imported_computation : imported_computations):
                                  for (auto& instruction : *imported_computation.mutable_
                                    for operand_id : *instruction.mutable_operand_ids():
                                      operand_id = remapped_ids.at(operand_id);
                                    for control_predecessor_id : *instruction.mutable_con
                                      control_predecessor_id = remapped_ids.at(control_pr
                                    for alled_computation_id : *instruction.mutable_calle
                                      called_computation_id = remapped_ids.at(called_comp
                                  computation_id = imported_computation.id();
                                  embedded_.insert({computation_id, std::move(imported_co
                              AddInstruction(HloInstructionProto instr)
                                instr.set_id(handle);
                                instr.set_opcode(HloOpcodeString(opcode));
                                instr.set_name()
                                handle_to_index_[handle] = instructions_.size();
                                instructions_.push_back(std::move(instr));
                                XlaOp op(handle, this);
                                return op
                    BuildComputation()
                      computation_status = builder->Build();
                        Build(instructions_.back().id()...)
                            for instruction in instructions_:
                              entry.add_instructions()->Swap(&instruction);
                          XlaComputation computation(entry.id());
                          HloModuleProto* module = computation.mutable_proto();
                          module->set_name(entry.name());
                          module->set_id(entry.id());
                          module->set_entry_computation_name(entry.name());
                          module->set_entry_computation_id(entry.id());
                          *module->mutable_host_program_shape() = entry.program_shape();
                          for e in embedded_:
                            module->add_computations()->Swap(e.second)
                          module->add_computations()->Swap(&entry);
                      computation = computation_status.ConsumeValueOrDie();
              e->tensorflow::XlaCompilationCache::BuildExecutable()
```

-1- jit/xla_compilation_cache.cc
-2- jit的下边界, XlaCompiler::CompileFunction(tf2xla/xla_compiler.cc), 进入tf2xla
-3- 对于cache中没有的entry, 真正的去编译
-4- compile_fn()
-5- tf2xla/xla_compiler.cc
-9- 每一个真正执行编译的node
-14- tf2xla/kernel的边界, 用于构造XlaComputation, 个人认为是Client的核心文件, 里面实现的XlaOp是client对上(Here, tfxla)提供的接口
-15- CrossReplicaSum恰好是10个需要调用其他XlaComputation的XlaOp(Call, Conditional, CrossReplicaSum, Map, Reducem ReduceWindow, Scatter, SelectAndScatterWithGeneralPadding, Sort, While)之一, 简单又全面, 这里作为例子
-16- XlaBuilder里构造XlaBuilder, 类似编译a.c的时候遇到了对b.c的依赖, 将b.c 构造好在作为依赖项copy到a.c
-18- XlaComputation的函数体
-19- XlaBuilder::Build(), 参考
-51-, Build()的作用就是将该XlaBuilder下存储在instructions_的HloInstructionProto和embedded_的HloComputation构造HloModuleProto, 并封装到XlaComputation中并返回
-20- CrossReplicaSum通过AddCalledComputation添加所调用的XlaComputation, 这个函数整体上做2件事: **1. 将被调用的XlaComputation deep copy到当前的XlaBuilder的instruction_和embedded_, 2. 在调用点的**

instruction处记录该XlaComputation的entry_computation_id
-41- 将准备好的Instruction添加到当前的XlaBuilder, 注意, CrossReplicaSum通过CreateSubBuilder创造的XlaBuilder实例是局部变量, 调用方需要的信息已经完全copy到了调用方的XlaBuilder实例, 所以这个局部变量随函数结束一起销毁没有任何问题, 其他9个类似的XlaOp同理, 这也就是前文例子中使用"inline"的原因, 如果理解了inline, 也就不难理解, 这里同样有inline的问题: 如果一个XlaComputation被多次调用, 那么就会copy多次, 产生同inline一样的代码膨胀
-47- XlaOp本质就是一个在当前XlaBuilder下instructions_的handle
-49- 前面已经将所有的XlaOpKernel都转换为HloInstructionProto存储在instructions_中, 所有依赖的"函数"XlaComputations也都存储在了embedded_中, 是时候构造最终的"main"函数的XlaComputation了
-64-构造完成

职责2: 编译cubin

```
e->tensorflow::XlaCompilationCache::BuildExecutable(entry->compilation_result, &entry->execu
  xla::ExecutableBuildOptions build_options;
  compile_result = xla::LocalClient::Compile()
    executable = xla::LocalService::CompileExecutable()
```

职责3: 执行cubin

```
tensorflow::XlaRunOp::Compute()
  run_result = xla::LocalExecutable::Run() //friend class LocalClient
    return executable_->ExecuteOnStreamWrapper()
```

Related:
Tensorflow XLA Service Buffer优化详解
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, XLA, 技术

Tensorflow XlaOpKernel | tf2xla 机制详解
Tensorflow XLA Service 详解 I