Developing a new backend for XLA

This preliminary guide is for early adopters that want to easily retarget TensorFlow to their hardware in an efficient manner. The guide is not step-by-step and assumes knowledge of LLVM (http://llvm.org), Bazel (https://bazel.build/), and TensorFlow.

XLA provides an abstract interface that a new architecture or accelerator can implement to create a backend to run TensorFlow graphs. Retargeting XLA should be significantly simpler and scalable than implementing every existing TensorFlow Op for new hardware.

Most implementations will fall into one of the following scenarios:

1. Existing CPU architecture not yet officially supported by XLA, with or without an existing LLVM (http://llvm.org) backend.
2. Non-CPU-like hardware with an existing LLVM backend.
3. Non-CPU-like hardware without an existing LLVM backend.

Note: An LLVM backend can mean either one of the officially released LLVM backends or a custom LLVM backend developed in-house.

Scenario 1: Existing CPU architecture not yet officially supported by XLA

In this scenario, start by looking at the existing XLA CPU backend (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/cpu/) . XLA makes it easy to retarget TensorFlow to different CPUs by using LLVM, since the main difference between XLA backends for CPUs is the code generated by LLVM. Google tests XLA for x64 and ARM64 architectures.

If the hardware vendor has an LLVM backend for their hardware, it is simple to link the backend with the LLVM built with XLA. In JIT mode, the XLA CPU backend emits code for the host CPU. For ahead-of-time compilation, `xla::AotCompilationOptions` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/compiler.h) can provide an LLVM triple to configure the target architecture.

If there is no existing LLVM backend but another kind of code generator exists, it should be possible to reuse most of the existing CPU backend.

Scenario 2: Non-CPU-like hardware with an existing LLVM backend

It is possible to model a new `xla::Compiler` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/compiler.h) implementation on the existing `xla::CPUCompiler` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/cpu/cpu_compiler.cc) and `xla::GPUCompiler` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/gpu/gpu_compiler.cc) classes, since these already emit LLVM IR. Depending on the nature of the hardware, it is possible that many of the LLVM IR generation aspects will have to be changed, but a lot of code can be shared with the existing backends.

A good example to follow is the GPU backend (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/gpu/) of XLA. The GPU backend targets a non-CPU-like ISA, and therefore some aspects of its code generation are unique to the GPU domain. Other kinds of hardware, e.g. DSPs like Hexagon (which has an upstream LLVM backend), can reuse parts of the LLVM IR emission logic, but other parts will be unique.

Scenario 3: Non-CPU-like hardware without an existing LLVM backend

If it is not possible to utilize LLVM, then the best option is to implement a new backend for XLA for the desired hardware. This option requires the most effort. The classes that need to be implemented are as follows:

- StreamExecutor (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/stream_executor/stream_executor.h): For many devices not all methods of `StreamExecutor` are needed. See existing `StreamExecutor` implementations for details.
- xla::Compiler (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/compiler.h): This class encapsulates the compilation of a HLO computation into an `xla::Executable`.

- `xla::Executable` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/executable.h): This class is used to launch a compiled computation on the platform.
- `xla::TransferManager` (https://www.github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/compiler/xla/service/transfer_manager.h): This class enables backends to provide platform-specific mechanisms for constructing XLA literal data from given device memory handles. In other words, it helps encapsulate the transfer of data from the host to the device and back.

Last updated August 17, 2017.