A dynamically loadable XLA plugin proposal

Avijit Chakraborty, Intel

Motivation

- XLA backends must currently be built with TensorFlow
  - Steep learning curve
  - Upstreaming plugins developed by various contributors may take up significant resources from core TensorFlow team
  - Changes in the “plugin” code (though unrelated to TensorFlow code) need to up upstreamed
- We have made minimal modifications to XLA to
  - Load XLA backends at runtime
  - Co-exists with traditional statically linked plugin backends

A dynamically loaded XLA Backend

- Just a normal dynamic shared object library, loaded at run time (using dlopen)
- The plugin is written using c++ and located outside of TensorFlow source tree
  - Depends on include files from TF installation (i.e., from Python site packages/tensorflow/include)
  - Links with libtensorflow_framework.so
- The plugin is placed in a well known location
  - For example TF installation/plugins directory
  - May have an optional configuration file (co-located)
- This idea is very similar to TensorFlow “Adding a New Op”

TensorFlow changes

- TensorFlow will have a plugin “adapter” that will connect with one or more plugin DSO libraries
  - The adapter will be initialized statically and will discover the plugin(s)
  - Load the plugin (.so), configure it, and query necessary data
  - If successful, register the plugin device with various TF registries
    - Platform, Compiler, Transfer Manager, Device
    - Connect the corresponding implementation components from the plugin DSO
- At run time, user scripts requests computation placement on this plugin device
  - Will follow the usual TensorFlow computation placement and execution

TensorFlow code changes

- Need to modify the BUILD scripts
  - Add additional header files (xla) to be included with the usual “include” target
  - Add additional library objects to the libtensorflow_framework.so
  - Add the additional directories to be packaged with Python wheel
- Need to implement the plugin “adapter” code
  - Compiler/plugin/<adapter directory>
  - Code is modeled from the existing XLA plugin code
    - Some refactoring to separate out the static registration parts and functional parts
- Timeframe
  - We already have written the code to test out this concept. Can submit the code for PR very soon – in the next few days.
  - Have included an example plugin that uses the HloEvaluator to run simple computations