tensorflow / tensorflow

Branch: master ▾ tensorflow / tensorflow / contrib / makefile /

Create new file | Find file | History

hawkinsp committed with tensorflower-gardener Add bitwise LeftShift (aka. tf.bitwise.left_shift) and RightShift (tf... … Latest commit `4c7e02c` 2 days ago

..		
samples	Change soc op type id by input data types and add new hvx ops	5 months ago
sub_makefiles	Merge changes from github.	a month ago
test	Add a makefile for tests in contrib/quantization	a year ago
.gitignore	Fix android build of protobuf with android_ndk r10e	a year ago
Dockerfile	Merge changes from github.	24 days ago
Makefile	Add veneer for SQLite C API	7 days ago
README.md	Merge changes from github.	2 months ago
build_all_android.sh	Make tensorflow::mutex implement a shared (reader/writer) lock, using	2 months ago
build_all_ios.sh	Make tensorflow::mutex implement a shared (reader/writer) lock, using	2 months ago
build_all_linux.sh	Make tensorflow::mutex implement a shared (reader/writer) lock, using	2 months ago
build_helper.subr	Lower number of parallel jobs to prevent OOM.	6 months ago
build_with_docker.sh	Merge changes from github.	24 days ago
compile_android_protobuf.sh	Merge changes from github.	4 months ago
compile_ios_protobuf.sh	Merge changes from github.	3 months ago
compile_ios_tensorflow.sh	Merge changes from github.	3 months ago
compile_linux_protobuf.sh	Merge changes from github.	6 months ago
compile_nsync.sh	Merge changes from github.	24 days ago
compile_pi_protobuf.sh	Merge changes from github.	21 days ago
create_ios_frameworks.sh	Merge changes from github.	3 months ago
download_dependencies.sh	Merge changes from github.	24 days ago
gen_file_lists.sh	Merge changes from github.	a year ago
proto_text_cc_files.txt	Make ResourceHandle not be a proto	4 months ago
proto_text_pb_cc_files.txt	Return more complete device information from the GetDevices() method …	6 months ago
proto_text_pb_h_files.txt	Return more complete device information from the GetDevices() method …	6 months ago
rename_protobuf.sh	Merge changes from github.	a year ago
rename_protoc.sh	Merge changes from github.	a year ago
rename_prototext.sh	Merge changes from github.	a year ago
tf_op_files.txt	Add bitwise LeftShift (aka. tf.bitwise.left_shift) and RightShift (tf...	2 days ago
tf_pb_text_files.txt	Implement ClusterSpec Propagation in TF Master	6 months ago
tf_proto_files.txt	Migrate kernels to boosted_trees.	4 months ago

README.md

TensorFlow Makefile

The recommended way to build TensorFlow from source is using the Bazel open-source build system. Sometimes this isn't possible. For example, if you are building for iOS, you currently need to use the Makefile.

- The build system may not have the RAM or processing power to support Bazel.
- Bazel or its dependencies may not be available.
- You may want to cross-compile for an unsupported target system.

This experimental project supplies a Makefile automatically derived from the dependencies listed in the Bazel project that can be used with GNU's make tool. With it, you can compile the core C++ runtime into a static library.

This static library will not contain:

- Python or other language bindings
- GPU support

You can target:

- iOS
- OS X (macOS)
- Android
- Raspberry-PI

You will compile tensorflow and protobuf libraries that you can link into other applications. You will also compile the benchmark application that will let you check your application.

Before you start (all platforms)

First, clone this TensorFlow repository.

You will need to download all dependencies as well. We have provided a script that does so, to be run (as with all commands) **at the root of the repository**:

```
tensorflow/contrib/makefile/download_dependencies.sh
```

You should only need to do this step once. It downloads the required libraries like Eigen in the `tensorflow/contrib/makefile/downloads/` folder.

You should download the example graph from https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip.

Building on Linux

Note: This has only been tested on Ubuntu.

As a first step, you need to make sure the required packages are installed:

```
sudo apt-get install autoconf automake libtool curl make g++ unzip zlib1g-dev \
git python
```

You should then be able to run the `build_all_linux.sh` script to compile:

```
tensorflow/contrib/makefile/build_all_linux.sh
```

This should compile a static library in `tensorflow/contrib/makefile/gen/lib/libtensorflow-core.a`, and create an example executable at `tensorflow/contrib/makefile/gen/bin/benchmark`.

Get the graph file, if you have not already:

```
mkdir -p ~/graphs
curl -o ~/graphs/inception.zip \
 https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip \
 && unzip ~/graphs/inception.zip -d ~/graphs/inception
```

To run the executable, use:

```
tensorflow/contrib/makefile/gen/bin/benchmark \
 --graph=$HOME/graphs/inception/tensorflow_inception_graph.pb
```

Android

First, you will need to download and unzip the Native Development Kit (NDK). You will not need to install the standalone toolchain, however.

Assign your NDK location to $NDK_ROOT:

```
export NDK_ROOT=/absolute/path/to/NDK/android-ndk-rxxx/
```

Download the graph if you haven't already:

```
mkdir -p ~/graphs
curl -o ~/graphs/inception.zip \
 https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip \
 && unzip ~/graphs/inception.zip -d ~/graphs/inception
```

Then, execute the following:

```
tensorflow/contrib/makefile/download_dependencies.sh
tensorflow/contrib/makefile/compile_android_protobuf.sh -c
export HOST_NSYNC_LIB=`tensorflow/contrib/makefile/compile_nsync.sh`
export TARGET_NSYNC_LIB=`CC_PREFIX="${CC_PREFIX}" NDK_ROOT="${NDK_ROOT}" \
        tensorflow/contrib/makefile/compile_nsync.sh -t android -a armeabi-v7a`
make -f tensorflow/contrib/makefile/Makefile TARGET=ANDROID
```

At this point, you will have compiled libraries in `gen/lib/*` and the benchmark app compiled for Android.

Run the benchmark by pushing both the benchmark and the graph file to your attached Android device:

```
adb push ~/graphs/inception/tensorflow_inception_graph.pb /data/local/tmp/
adb push tensorflow/contrib/makefile/gen/bin/benchmark /data/local/tmp/
adb shell '/data/local/tmp/benchmark \
 --graph=/data/local/tmp/tensorflow_inception_graph.pb \
 --input_layer="input:0" \
 --input_layer_shape="1,224,224,3" \
 --input_layer_type="float" \
 --output_layer="output:0"
'
```

For more details, see the benchmark documentation.

iOS

Note: To use this library in an iOS application, see related instructions in the iOS examples directory.

Install XCode 7.3 or more recent. If you have not already, you will need to install the command-line tools using `xcode-select`:

```
xcode-select --install
```

If this is a new install, you will need to run XCode once to agree to the license before continuing.

(You will also need to have Homebrew installed.)

Then install automake/libtool:

```
brew install automake
brew install libtool
```

Also, download the graph if you haven't already:

```
mkdir -p ~/graphs
curl -o ~/graphs/inception.zip \
 https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip \
 && unzip ~/graphs/inception.zip -d ~/graphs/inception
```

Building all at once

If you just want to get the libraries compiled in a hurry, you can run this from the root of your TensorFlow source folder:

```
tensorflow/contrib/makefile/build_all_ios.sh
```

This process will take around twenty minutes on a modern MacBook Pro.

When it completes, you will have a library for a single architecture and the benchmark program. Although successfully compiling the benchmark program is a sign of success, the program is not a complete iOS app.

To see TensorFlow running on iOS, the example Xcode project in tensorflow/examples/ios shows how to use the static library in a simple app.

Building by hand

This section covers each step of building. For all the code in one place, see build_all_ios.sh.

If you have not already, you will need to download dependencies:

```
tensorflow/contrib/makefile/download_dependencies.sh
```

Next, you will need to compile protobufs for iOS:

```
tensorflow/contrib/makefile/compile_ios_protobuf.sh
```

Then, you will need to compile the nsync library for iOS:

```
export HOST_NSYNC_LIB=`tensorflow/contrib/makefile/compile_nsync.sh`
export TARGET_NSYNC_LIB=`tensorflow/contrib/makefile/compile_nsync.sh -t ios`
```

Then, you can run the makefile specifying iOS as the target, along with the architecture you want to build for:

```
make -f tensorflow/contrib/makefile/Makefile \
 TARGET=IOS \
 IOS_ARCH=ARM64
```

This creates a library in `tensorflow/contrib/makefile/gen/lib/libtensorflow-core.a` that you can link any xcode project against.

At this point, you will have a library for a single architecture and the benchmark program. Although successfully compiling the benchmark program is a sign of success, the program is not a complete iOS app.

To see TensorFlow running on iOS, the example Xcode project in tensorflow/examples/ios shows how to use the static library in a simple app.

Universal binaries

In some situations, you will need a universal library. In that case, you will still need to run `compile_ios_protobuf.sh` and `compile_nsync.sh`, but this time follow it with:

```
compile_ios_tensorflow.sh
```

In XCode, you will need to use -force_load in the linker flags section of the build settings to pull in the global constructors that are used to register ops and kernels.

Optimization

The `compile_ios_tensorflow.sh` script can take optional command-line arguments. The first argument will be passed as a C++ optimization flag and defaults to debug mode. If you are concerned about performance or are working on a release build, you would likely want a higher optimization setting, like so:

```
compile_ios_tensorflow.sh "-Os"
```

For other variations of valid optimization flags, see clang optimization levels.

Raspberry Pi

Building on the Raspberry Pi is similar to a normal Linux system. First download the dependencies, install the required packages and build protobuf:

```
tensorflow/contrib/makefile/download_dependencies.sh
sudo apt-get install -y autoconf automake libtool gcc-4.8 g++-4.8
cd tensorflow/contrib/makefile/downloads/protobuf/
./autogen.sh
./configure
make
sudo make install
sudo ldconfig  # refresh shared library cache
cd ../../../../..
export HOST_NSYNC_LIB=`tensorflow/contrib/makefile/compile_nsync.sh`
export TARGET_NSYNC_LIB="$HOST_NSYNC_LIB"
```

Once that's done, you can use make to build the library and example:

```
make -f tensorflow/contrib/makefile/Makefile HOST_OS=PI TARGET=PI OPTFLAGS="-Os" CXX=g++-4.8
```

If you're only interested in building for Raspberry Pi's 2 and 3, you can supply some extra optimization flags to give you code that will run faster:

```
make -f tensorflow/contrib/makefile/Makefile HOST_OS=PI TARGET=PI \
 OPTFLAGS="-Os -mfpu=neon-vfpv4 -funsafe-math-optimizations -ftree-vectorize" CXX=g++-4.8
```

One thing to be careful of is that the gcc version 4.9 currently installed on Jessie by default will hit an error mentioning `__atomic_compare_exchange`. This is why the examples above specify `CXX=g++-4.8` explicitly, and why we install it using apt-get. If you have partially built using the default gcc 4.9, hit the error and switch to 4.8, you need to do a `make -f tensorflow/contrib/makefile/Makefile clean` before you build. If you don't, the build will appear to succeed but you'll encounter malloc(): memory corruption errors when you try to run any programs using the library.

For more examples, look at the tensorflow/contrib/pi_examples folder in the source tree, which contains code samples aimed at the Raspberry Pi.

Other notes

Supported Systems

The Make script has been tested on Ubuntu and OS X. If you look in the Makefile itself, you'll see it's broken up into host and target sections. If you are cross-compiling, you should look at customizing the target settings to match what you need for your desired system.

Dependency Management

The Makefile loads in a list of dependencies stored in text files. These files are generated from the main Bazel build by running `tensorflow/contrib/makefile/gen_file_lists.sh`. You'll need to re-run this i you make changes to the files that are included in the build.

Header dependencies are not automatically tracked by the Makefile, so if you make header changes you will need to run this command to recompile cleanly:

```
make -f tensorflow/contrib/makefile/Makefile clean
```

Cleaning up

In some situations, you may want to completely clean up. The dependencies, intermediate stages, and generated files are stored in:

```
tensorflow/contrib/makefile/downloads
tensorflow/contrib/makefile/gen
```

Those directories can safely be removed, but you will have to start over with `download_dependencies.sh` once you delete them.

Fixing Makefile Issues

Because the main development of TensorFlow is done using Bazel, changes to the codebase can sometimes break the makefile build process. If you find that tests relying on this makefile are failing with a change you're involved in, here are some trouble-shooting steps:

- Try to reproduce the issue on your platform. If you're on Linux, running `make -f tensorflow/contrib/makefile/Makefile` should be enough to recreate most issues. For other platforms, see the sections earlier in this document.
- The most common cause of breakages are files that have been added to the Bazel build scripts, but that the makefile isn't aware of. Typical symptoms of this include linker errors mentioning missing symbols or protobuf headers that aren't found. To address these problems, take a look at the *.txt files in `tensorflow/contrib/makefile`. If you have a new operator, you may need to add it to `tf_op_files.txt`, or for a new proto to `tf_proto_files.txt`.

- There's also a wildcard system in `Makefile` that defines what core C++ files are included in the library. This is designed to match the equivalent rule in `tensorflow/core/BUILD`, so if you change the wildcards there to include new files you'll need to also update `CORE_CC_ALL_SRCS` and `CORE_CC_EXCLUDE_SRCS` in the makefile.
- Some of the supported platforms use clang instead of gcc as their compiler, so if you're hitting compile errors you may need to tweak your code to be more friendly to different compilers by avoiding gcc extensions or idioms.

These are the most common reasons for makefile breakages, but it's also possible you may hit something unusual, like a platform incompatibility. For those, you'll need to see if you can reproduce the issue on that particular platform and debug it there. You can also reach out to the broader TensorFlow team by filing a Github issue to ask for help.