tensorflow / **tensorflow**

Branch: r1.3 tensorflow / tensorflow / compiler / aot / **tfcompile.bzl** Find file Copy path

Fetching contributors…

Cannot retrieve contributors at this time

299 lines (274 sloc) 11.3 KB

```
# -*- Python -*-

"""Build macro that compiles a TensorFlow graph into a cc_library.

To use from your BUILD file, add the following line to load the macro:

load("//tensorflow/compiler/aot:tfcompile.bzl", "tf_library")

Then call the macro like this:

tf_library(
    name = "test_graph_tfmatmul",
    config = "test_graph_tfmatmul.config.pbtxt",
    cpp_class = "MatMulComp",
    graph = ":test_graph_tfmatmul.pb",
)
"""

load("//tensorflow:tensorflow.bzl", "if_android", "tf_copts")

def tf_library(name, graph, config,
               freeze_checkpoint=None, freeze_saver=None,
               cpp_class=None, gen_test=True, gen_benchmark=True,
               visibility=None, testonly=None,
               tfcompile_flags=None,
               tfcompile_tool="//tensorflow/compiler/aot:tfcompile",
               deps=None, tags=None):
  """Runs tfcompile to compile a TensorFlow graph into executable code.

  Given an invocation of tf_library(name="foo", ...), generates the following
  build targets:
    foo:           A cc_library containing the generated header and computation.
    foo_test:      A cc_test with simple tests and benchmarks. Only created if
                   gen_test=True.
    foo_benchmark: A cc_binary that runs a minimal-dependency benchmark, useful
                   for mobile devices or other platforms that can't compile the
                   full test libraries. Only created if gen_benchmark=True.

  Args:
    name: The name of the build rule.
    graph: The TensorFlow GraphDef to compile.  If the file ends in '.pbtxt' it
      is expected to be in the human-readable proto text format, otherwise it is
      expected to be in the proto binary format.
    config: File containing tensorflow.tfcompile.Config proto.  If the file ends
      in '.pbtxt' it is expected to be in the human-readable proto text format,
      otherwise it is expected to be in the proto binary format.
    freeze_checkpoint: If provided, run freeze_graph with this checkpoint to
      convert variables into constants.
    freeze_saver: If provided, run freeze_graph with this saver, in SaverDef
      binary form, to convert variables into constants.
    cpp_class: The name of the generated C++ class, wrapping the generated
      function.  The syntax of this flag is
      [[<optional_namespace>::],...]<class_name>.  This mirrors the C++ syntax
      for referring to a class, where multiple namespaces may precede the class
      name, separated by double-colons.  The class will be generated in the
      given namespace(s), or if no namespaces are given, within the global
      namespace.
    gen_test: If True, also generate a cc_test rule that builds a simple
```

```
      test and benchmark.
    gen_benchmark: If True, also generate a binary with a simple benchmark.
      Unlike the output of gen_test, this benchmark can be run on android.
    visibility: Bazel build visibility.
    testonly:   Bazel testonly attribute.
    tfcompile_flags: Extra flags to pass to tfcompile to control compilation.
    tfcompile_tool: The tfcompile binary. A non-default can be passed to
      use a tfcompile built with extra dependencies.
    deps: a list of extra deps to include on the build rules for
      the generated library.
    tags: tags to apply to subsidiary build rules.

  The output header is called <name>.h.
  """
  if not cpp_class:
    fail("cpp_class must be specified")

  tfcompile_graph = graph
  if freeze_checkpoint or freeze_saver:
    if not freeze_checkpoint:
      fail("freeze_checkpoint must be specified when freeze_saver is specified")

    freeze_name = "freeze_" + name
    freeze_file = freeze_name + ".pb"

    # First run tfcompile to generate the list of out_nodes.
    out_nodes_file = "out_nodes_" + freeze_name
    native.genrule(
        name=("gen_" + out_nodes_file),
        srcs=[config],
        outs=[out_nodes_file],
        cmd=("$(location " + tfcompile_tool + ")" +
             " --config=$(location " + config + ")" +
             " --dump_fetch_nodes > $@"),
        tools=[tfcompile_tool],
        # Run tfcompile on the build host, rather than forge, since it's
        # typically way faster on the local machine.
        local=1,
        tags=tags,
    )

    # Now run freeze_graph to convert variables into constants.
    freeze_args = (" --input_graph=$(location " + graph + ")" +
                   " --input_binary=" + str(not graph.endswith(".pbtxt")) +
                   " --input_checkpoint=$(location " + freeze_checkpoint + ")" +
                   " --output_graph=$(location " + freeze_file + ")" +
                   " --output_node_names=$$(<$(location " + out_nodes_file +
                   "))")
    freeze_saver_srcs = []
    if freeze_saver:
      freeze_args += " --input_saver=$(location " + freeze_saver + ")"
      freeze_saver_srcs += [freeze_saver]
    native.genrule(
        name=freeze_name,
        srcs=[
            graph,
            freeze_checkpoint,
            out_nodes_file,
        ] + freeze_saver_srcs,
        outs=[freeze_file],
        cmd=("$(location //tensorflow/python/tools:freeze_graph)" +
             freeze_args),
        tools=["//tensorflow/python/tools:freeze_graph"],
        tags=tags,
    )
    tfcompile_graph = freeze_file

  # Rule that runs tfcompile to produce the header and object file.
  header_file = name + ".h"
  object_file = name + ".o"
  ep = ("__" + PACKAGE_NAME + "__" + name).replace("/", "_")
  native.genrule(
      name=("gen_" + name),
      srcs=[
          tfcompile_graph,
          config,
      ],
      outs=[
          header_file,
          object_file,
      ],
      cmd=("$(location " + tfcompile_tool + ")" +
           " --graph=$(location " + tfcompile_graph + ")" +
           " --config=$(location " + config + ")" +
           " --entry_point=" + ep +
           " --cpp_class=" + cpp_class +
           " --target_triple=" + target_llvm_triple() +
           " --out_header=$(@D)/" + header_file +
           " --out_object=$(@D)/" + object_file +
           " " + (tfcompile_flags or "")),
      tools=[tfcompile_tool],
      visibility=visibility,
      testonly=testonly,
      # Run tfcompile on the build host since it's typically faster on the local
      # machine.
      #
      # Note that setting the local=1 attribute on a *test target* causes the
      # test infrastructure to skip that test.  However this is a genrule, not a
      # test target, and runs with --genrule_strategy=forced_forge, meaning the
      # local=1 attribute is ignored, and the genrule is still run.
      #
      # https://www.bazel.io/versions/master/docs/be/general.html#genrule
      local=1,
      tags=tags,
  )

  # The cc_library rule packaging up the header and object file, and needed
  # kernel implementations.
  native.cc_library(
      name=name,
      srcs=[object_file],
      hdrs=[header_file],
      visibility=visibility,
      testonly=testonly,
      deps = [
          # TODO(cwhipkey): only depend on kernel code that the model actually needed.
          "//tensorflow/compiler/tf2xla/kernels:gather_op_kernel_float_int32",
          "//tensorflow/compiler/tf2xla/kernels:gather_op_kernel_float_int64",
          "//tensorflow/compiler/tf2xla/kernels:index_ops_kernel_argmax_float_1d",
          "//tensorflow/compiler/tf2xla/kernels:index_ops_kernel_argmax_float_2d",
          "//tensorflow/compiler/aot:runtime",
          "//tensorflow/compiler/tf2xla:xla_local_runtime_context",
          "//tensorflow/compiler/xla/service/cpu:runtime_conv2d",
          "//tensorflow/compiler/xla/service/cpu:runtime_matmul",
          "//tensorflow/compiler/xla/service/cpu:runtime_single_threaded_conv2d",
          "//tensorflow/compiler/xla/service/cpu:runtime_single_threaded_matmul",
          "//tensorflow/compiler/xla:executable_run_options",
          "//third_party/eigen3",
          "//tensorflow/core:framework_lite",
          ] + (deps or []),
      tags=tags,
  )

  # Variables used for gen_test and gen_benchmark.
  no_ns_name = ""
  cpp_class_split = cpp_class.rsplit("::", maxsplit=2)
  if len(cpp_class_split) == 1:
    no_ns_name = cpp_class_split[0]
  else:
    no_ns_name = cpp_class_split[1]
  sed_replace = (
      "-e \"s|{{TFCOMPILE_HEADER}}|$(location " + header_file + ")|g\" " +
      "-e \"s|{{TFCOMPILE_CPP_CLASS}}|" + cpp_class + "|g\" " +
      "-e \"s|{{TFCOMPILE_NAME}}|" + no_ns_name + "|g\" ")

  if gen_test:
    test_name = name + "_test"
    test_file = test_name + ".cc"
    # Rule to rewrite test.cc to produce the test_file.
    native.genrule(
        name=("gen_" + test_name),
        testonly=1,
        srcs=[
            "//tensorflow/compiler/aot:test.cc",
            header_file,
        ],
        outs=[test_file],
        cmd=("sed " + sed_replace +
             " $(location //tensorflow/compiler/aot:test.cc) " +
             "> $(OUTS)"),
        tags=tags,
    )

    # The cc_test rule for the generated code.
    native.cc_test(
        name=test_name,
        srcs=[test_file],
        deps=[
            ":" + name,
            "//tensorflow/compiler/tf2xla:xla_local_runtime_context",
            "//tensorflow/compiler/aot:runtime",
            "//tensorflow/compiler/aot:tf_library_test_main",
            "//tensorflow/compiler/xla:executable_run_options",
            "//third_party/eigen3",
            "//tensorflow/core:lib",
            "//tensorflow/core:test",
            ],
        tags=tags,
    )

  if gen_benchmark:
    benchmark_name = name + "_benchmark"
    benchmark_file = benchmark_name + ".cc"
    benchmark_main = ("//tensorflow/compiler/aot:" +
        "benchmark_main.template")

    # Rule to rewrite benchmark.cc to produce the benchmark_file.
    native.genrule(
        name=("gen_" + benchmark_name),
        srcs=[
            benchmark_main,
            header_file,
        ],
        testonly = testonly,
        outs=[benchmark_file],
        cmd=("sed " + sed_replace +
             " $(location " + benchmark_main + ") " +
             "> $(OUTS)"),
        tags=tags,
    )

    # The cc_benchmark rule for the generated code.
    #
    # Note: to get smaller size on android for comparison, compile with:
    #    --copt=-fvisibility=hidden
    #    --copt=-D_LIBCPP_TYPE_VIS=_LIBCPP_HIDDEN
    #    --copt=-D_LIBCPP_EXCEPTION_ABI=_LIBCPP_HIDDEN
    native.cc_binary(
        name=benchmark_name,
        srcs=[benchmark_file],
        testonly = testonly,
        copts = tf_copts(),
        linkopts = if_android(["-pie", "-s"]),
        deps=[
            ":" + name,
            "//tensorflow/compiler/tf2xla:xla_local_runtime_context",
            "//tensorflow/compiler/aot:benchmark",
            "//tensorflow/compiler/aot:runtime",
            "//tensorflow/compiler/xla:executable_run_options",
            "//third_party/eigen3",
        ] + if_android([
            "//tensorflow/compiler/aot:benchmark_extra_android",
        ]),
        tags=tags,
    )
```

```python


def target_llvm_triple():
  """Returns the target LLVM triple to be used for compiling the target."""
  # TODO(toddw): Add target_triple for other targets.  For details see:
  # http://llvm.org/docs/doxygen/html/Triple_8h_source.html
  return select({
      "//tensorflow:android_armeabi": "armv5-none-android",
      "//tensorflow:android_arm": "armv7-none-android",
      "//tensorflow:android_arm64": "aarch64-none-android",
      "//tensorflow:android_x86": "i686-none-android",
      "//tensorflow:linux_ppc64le": "ppc64le-ibm-linux-gnu",
      "//tensorflow:darwin": "x86_64-none-darwin",
      "//conditions:default": "x86_64-pc-linux",
  })
```