jpienaar Update more int64 uses to int64_t ... History

8 contributors

199 lines (179 sloc) | 9.58 KB ...

```
/* Copyright 2016 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
==============================================================================*/

// SparseDenseBinaryOpShared is the shared code for binary coefficient-wise
// (cwise) operations of the following form:
//
//   sparse_t <binary cwise op> dense_t -> new sparse_t
//
// where:
//
//   (1) "binary cwise op" can be, for example, cdiv, cmul, cfloordiv, etc.
//   (2) LIMITATION: we only support broadcasting the dense side to the sparse
//       side.  In other words, NumDims(sparse_t) >= NumDims(dense_t), and if
//       they are equal, each dim size of sparse_t >= that of dense_t.
//   (3) Note that the result is a new sparse tensor, which means the implicitly
//       zero elements of sparse_t do not participate.  (Hence, this should not
//       be used for, say, cadd.)
```

```
//
// The only output is a vector of flat values with shape [nnz], since this op
// does not change neither the indices nor the shape of the sparse operand.
//
// See docs of all registered ops in ../ops/sparse_ops.cc.

#define EIGEN_USE_THREADS

#include "third_party/eigen3/unsupported/Eigen/CXX11/Tensor"
#include "tensorflow/core/framework/op_kernel.h"
#include "tensorflow/core/framework/register_types.h"
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/framework/tensor_util.h"
#include "tensorflow/core/framework/types.h"
#include "tensorflow/core/kernels/cwise_ops.h"
#include "tensorflow/core/kernels/cwise_ops_common.h"
#include "tensorflow/core/util/bcast.h"

using Eigen::TensorRef;
using tensorflow::gtl::ArraySlice;

namespace tensorflow {

typedef Eigen::ThreadPoolDevice CPUDevice;

template <typename Device, typename T, typename Functor>
class SparseDenseBinaryOpShared : public OpKernel {
 public:
  explicit SparseDenseBinaryOpShared(OpKernelConstruction *ctx)
      : OpKernel(ctx) {}

  void Compute(OpKernelContext *ctx) override {
    const Tensor *indices_t, *values_t, *shape_t, *dense_t;
    OP_REQUIRES_OK(ctx, ctx->input("sp_indices", &indices_t));
    OP_REQUIRES_OK(ctx, ctx->input("sp_values", &values_t));
    OP_REQUIRES_OK(ctx, ctx->input("sp_shape", &shape_t));
    OP_REQUIRES_OK(ctx, ctx->input("dense", &dense_t));

    // Validations.
    OP_REQUIRES(ctx, TensorShapeUtils::IsMatrix(indices_t->shape()),
                errors::InvalidArgument(
                    "Input sp_indices should be a matrix but received shape: ",
                    indices_t->shape().DebugString()));
    OP_REQUIRES(ctx,
                TensorShapeUtils::IsVector(values_t->shape()) &&
                    TensorShapeUtils::IsVector(shape_t->shape()),
                errors::InvalidArgument(
                    "Inputs sp_values and sp_shape should be vectors "
                    "but received shapes: ",
```

```
                values_t->shape().DebugString(), " and ",
                shape_t->shape().DebugString()));
    OP_REQUIRES(
        ctx, values_t->dim_size(0) == indices_t->dim_size(0),
        errors::InvalidArgument(
            "The first dimension of values and indices should match. (",
            values_t->dim_size(0), " vs. ", indices_t->dim_size(0), ")"));

    const auto indices_mat = indices_t->matrix<int64_t>();
    const auto shape_vec = shape_t->vec<int64_t>();
    const auto lhs_dims = BCast::FromShape(TensorShape(shape_vec));
    const auto rhs_dims = BCast::FromShape(dense_t->shape());
    BCast b(lhs_dims, rhs_dims, false);  // false for keeping the same num dims.

    // True iff (size(lhs) >= size(rhs)) and all dims in lhs is greater or equal
    // to dims in rhs (from right to left).
    auto VecGreaterEq = [](ArraySlice<int64_t> lhs, ArraySlice<int64_t> rhs) {
      if (lhs.size() < rhs.size()) return false;
      for (size_t i = 0; i < rhs.size(); ++i) {
        if (lhs[lhs.size() - 1 - i] < rhs[rhs.size() - 1 - i]) return false;
      }
      return true;
    };
    OP_REQUIRES(ctx, VecGreaterEq(lhs_dims, rhs_dims) && b.IsValid(),
                errors::InvalidArgument(
                    "SparseDenseBinaryOpShared broadcasts dense to sparse "
                    "only; got incompatible shapes: [",
                    absl::StrJoin(lhs_dims, ","), "] vs. [",
                    absl::StrJoin(rhs_dims, ","), "]"));

    Tensor *output_values = nullptr;
    Tensor dense_gathered;
    const int64_t nnz = indices_t->dim_size(0);
    OP_REQUIRES_OK(ctx,
                   ctx->allocate_output(0, TensorShape({nnz}), &output_values));
    OP_REQUIRES_OK(
        ctx, ctx->allocate_temp(DataTypeToEnum<T>::value, TensorShape({nnz}),
                                &dense_gathered));
    bool op_is_div = false;
    if (absl::StrContains(ctx->op_kernel().type_string_view(), "Div")) {
      op_is_div = true;
    }
    // Pulls relevant entries from the dense side, with reshape and broadcasting
    // *of the dense side* taken into account.  Use a TensorRef to avoid blowing
    // up memory.
    //
    // We can directly use the sparse indices to look up dense side, because
    // "b.y_reshape()" and "b.y_bcast()" are guaranteed to have rank "ndims".
    auto dense_gathered_flat = dense_gathered.flat<T>();
```

```
    const int ndims = lhs_dims.size();
    switch (ndims) {
#define CASE(NDIM)                                                          \
  case NDIM: {                                                              \
    TensorRef<Eigen::Tensor<const T, NDIM, Eigen::RowMajor>> rhs_ref =      \
        dense_t->shaped<T, NDIM>(b.y_reshape())                             \
            .broadcast(BCast::ToIndexArray<NDIM>(b.y_bcast()));             \
    Eigen::array<Eigen::DenseIndex, NDIM> idx;                              \
    bool indices_valid = true;                                              \
    for (int i = 0; i < nnz; ++i) {                                         \
      for (int d = 0; d < NDIM; ++d) {                                      \
        idx[d] = internal::SubtleMustCopy(indices_mat(i, d));               \
        if (!FastBoundsCheck(idx[d], rhs_ref.dimension(d))) {               \
          indices_valid = false;                                            \
        }                                                                   \
      }                                                                     \
      OP_REQUIRES(                                                          \
          ctx, indices_valid,                                               \
          errors::InvalidArgument("Provided indices are out-of-bounds w.r.t. " \
                                  "dense side with broadcasted shape"));    \
      dense_gathered_flat(i) = rhs_ref.coeff(idx);                          \
      if (op_is_div) {                                                      \
        OP_REQUIRES(ctx, dense_gathered_flat(i) != 0,                       \
                    errors::InvalidArgument(                                \
                        "SparseDenseCwiseDiv cannot divide by zero,"        \
                        "but input dense tensor contains zero "));          \
      }                                                                     \
    }                                                                       \
    break;                                                                  \
  }

      CASE(1);
      CASE(2);
      CASE(3);
      CASE(4);
      CASE(5);
      default:
        OP_REQUIRES(
            ctx, false,
            errors::InvalidArgument("Only tensors with ranks between 1 and 5 "
                                    "are currently supported.  Tensor rank: ",
                                    ndims));
#undef CASE
    }

    output_values->flat<T>().device(ctx->eigen_device<Device>()) =
        values_t->flat<T>().binaryExpr(dense_gathered_flat,
                                       typename Functor::func());
  }
```

```
};

// NOTE(aselle): If Div is extended to non-reals, make sure to use the same
// separation of operator semantics as done for dense cwise ops. I.e. you
// should make SparseDenseCwiseRealDiv, SparseDenseCwiseTruncateDiv,
// SparseDenseCwiseFloorDiv, and then deprecate, SparseDenseCwiseDiv.
// TODO(zongheng): extend to other eligible cwise operations as requested.
#define REGISTER_KERNELS(T)                                                 \
  REGISTER_KERNEL_BUILDER(                                                  \
      Name("SparseDenseCwiseMul").Device(DEVICE_CPU).TypeConstraint<T>("T"), \
      SparseDenseBinaryOpShared<CPUDevice, T, functor::mul<T>>)             \
                                                                            \
  REGISTER_KERNEL_BUILDER(                                                  \
      Name("SparseDenseCwiseDiv").Device(DEVICE_CPU).TypeConstraint<T>("T"), \
      SparseDenseBinaryOpShared<CPUDevice, T, functor::div<T>>)             \
  REGISTER_KERNEL_BUILDER(                                                  \
      Name("SparseDenseCwiseAdd").Device(DEVICE_CPU).TypeConstraint<T>("T"), \
      SparseDenseBinaryOpShared<CPUDevice, T, functor::add<T>>)

TF_CALL_REAL_NUMBER_TYPES(REGISTER_KERNELS);
#undef REGISTER_KERNELS

}  // namespace tensorflow
```