#include "tensorflow/core/framework/op.h"

#include "tensorflow/core/framework/shape\_inference.h"

#include "tensorflow/core/framework/op\_kernel.h"

#include "tensorflow/core/util/work\_sharder.h"

#include "tensorflow/core/lib/core/threadpool.h"

#include "tensorflow/core/platform/env.h"

#include "tensorflow/core/platform/mutex.h"

#include "tensorflow/core/platform/logging.h"

#include <iostream>

#include <vector>

using namespace tensorflow;

REGISTER\_OP("ParAdd")

.Input("input\_a: float")

.Input("input\_b: float")

.Output("a\_plus\_b: float")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

c->set\_output(0, c->input(0));

return Status::OK();

});

class ParAddOp : public OpKernel {

public:

explicit ParAddOp(OpKernelConstruction\* context) : OpKernel(context) {

}

void Compute(OpKernelContext\* context) override {

// Grab the input tensor

const Tensor& input\_tensor0 = context->input(0);

auto input\_flat0 = input\_tensor0.flat<float>();

const Tensor& input\_tensor1 = context->input(1);

auto input\_flat1 = input\_tensor1.flat<float>();

OP\_REQUIRES(context, input\_tensor0.shape() == input\_tensor1.shape(),

errors::InvalidArgument("Input tensors must be identical shape."));

// Create an output tensor

Tensor\* output\_tensor = NULL;

OP\_REQUIRES\_OK(context,

context->allocate\_output(0,

input\_tensor0.shape(),

&output\_tensor));

auto output\_flat = output\_tensor->flat<float>();

// PARALLEL ADD

const int N = input\_flat0.size();

// retrieve the thread pool from the op context

auto worker\_threads = \*(context->device()->tensorflow\_cpu\_worker\_threads());

// Definition of the computation thread

auto task = [=, &input\_flat0, &input\_flat1, &output\_flat](int64 start, int64 limit) {

for (; start < limit; ++start) {

output\_flat(start) = input\_flat0(start) + input\_flat1(start);

}

};

// this is a heuristic. high number is likely to be sharded into smaller pieces

int64 cost\_per\_unit = 1;

// let Tensorflow split up the work as it sees fit

Shard(worker\_threads.num\_threads,

worker\_threads.workers,

N,

cost\_per\_unit,

task);

}

};

REGISTER\_KERNEL\_BUILDER(Name("ParAdd").Device(DEVICE\_CPU), ParAddOp);