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

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

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

using namespace tensorflow;

REGISTER\_OP("Add1")

.Attr("T: {float, double, int32}")

.Input("input1: T")

.Output("output: T")

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

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

return Status::OK();

});

template<typename T>

class Add1 : public OpKernel {

public:

explicit Add1(OpKernelConstruction\* context) : OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

// Grab the input tensor

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

auto input = input\_tensor.flat<T>();

// Create an output tensor

Tensor\* output\_tensor = nullptr;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, input\_tensor.shape(),

&output\_tensor));

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

// Add 1 to input and assign to output

const int N = input.size();

for (int i = 0; i < N; i++) {

output\_flat(i) = input(i) + 1;

}

}

};

REGISTER\_OP("Add1Grad")

.Attr("T: {float, double, int32}")

.Input("grad\_output: T")

.Output("grad\_input: T")

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

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

return Status::OK();

});

template<typename T>

class Add1Grad : public OpKernel {

public:

explicit Add1Grad(OpKernelConstruction\* context) : OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

// Grab the input tensor

const Tensor& grad\_output\_tensor = context->input(0);

auto grad\_output = grad\_output\_tensor.flat<T>();

// Create an grad\_input tensor

Tensor\* grad\_input\_tensor = nullptr;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, grad\_output\_tensor.shape(),

&grad\_input\_tensor));

auto grad\_input\_flat = grad\_input\_tensor->flat<T>();

// Copy from grad\_output to grad\_input

const int N = grad\_output.size();

for (int i = 0; i < N; i++) {

grad\_input\_flat(i) = grad\_output(i);

}

}

};

#define REGISTER(Type) \

\

REGISTER\_KERNEL\_BUILDER( \

Name("Add1") \

.Device(DEVICE\_CPU) \

.TypeConstraint<Type>("T"), \

Add1<Type>); \

\

REGISTER\_KERNEL\_BUILDER( \

Name("Add1Grad") \

.Device(DEVICE\_CPU) \

.TypeConstraint<Type>("T"), \

Add1Grad<Type>); \

REGISTER(float);

REGISTER(double);

REGISTER(int32);