#include "internal/error.h"

#include <twml/Tensor.h>

#include <twml/Type.h>

#include <type\_traits>

#include <algorithm>

#include <numeric>

namespace twml {

using std::vector;

Tensor::Tensor(void \*data, int ndims, const uint64\_t \*dims, const uint64\_t \*strides, twml\_type type) :

m\_type(type), m\_data(data),

m\_dims(dims, dims + ndims),

m\_strides(strides, strides + ndims) {

}

Tensor::Tensor(void \*data,

const vector<uint64\_t> &dims,

const vector<uint64\_t> &strides,

twml\_type type) :

m\_type(type), m\_data(data),

m\_dims(dims.begin(), dims.end()),

m\_strides(strides.begin(), strides.end()) {

if (dims.size() != strides.size()) {

throw twml::Error(TWML\_ERR\_SIZE, "The number size of dims and strides don't match");

}

}

int Tensor::getNumDims() const {

return static\_cast<int>(m\_dims.size());

}

uint64\_t Tensor::getDim(int id) const {

if (id >= this->getNumDims()) {

throw twml::Error(TWML\_ERR\_SIZE, "Requested dimension exceeds tensor dimension");

}

return m\_dims[id];

}

uint64\_t Tensor::getStride(int id) const {

if (id >= this->getNumDims()) {

throw twml::Error(TWML\_ERR\_SIZE, "Requested dimension exceeds tensor dimension");

}

return m\_strides[id];

}

uint64\_t Tensor::getNumElements() const {

return std::accumulate(m\_dims.begin(), m\_dims.end(), 1, std::multiplies<int>());

}

twml\_type Tensor::getType() const {

return m\_type;

}

twml\_tensor Tensor::getHandle() {

return reinterpret\_cast<twml\_tensor>(this);

}

const twml\_tensor Tensor::getHandle() const {

return reinterpret\_cast<const twml\_tensor>(const\_cast<Tensor \*>(this));

}

const Tensor \*getConstTensor(const twml\_tensor t) {

return reinterpret\_cast<const Tensor \*>(t);

}

Tensor \*getTensor(twml\_tensor t) {

return reinterpret\_cast<Tensor \*>(t);

}

#define INSTANTIATE(T) \

template<> TWMLAPI T \*Tensor::getData() { \

if ((twml\_type)Type<T>::type != m\_type) { \

throw twml::Error(TWML\_ERR\_TYPE, \

"Requested invalid type"); \

} \

return reinterpret\_cast<T \*>(m\_data); \

} \

template<> TWMLAPI const T \*Tensor::getData() const { \

if ((twml\_type)Type<T>::type != m\_type) { \

throw twml::Error(TWML\_ERR\_TYPE, \

"Requested invalid type"); \

} \

return (const T \*)m\_data; \

} \

INSTANTIATE(int32\_t)

INSTANTIATE(int64\_t)

INSTANTIATE(int8\_t)

INSTANTIATE(uint8\_t)

INSTANTIATE(float)

INSTANTIATE(double)

INSTANTIATE(bool)

INSTANTIATE(std::string)

// This is used for the C api. No checks needed for void.

template<> TWMLAPI void \*Tensor::getData() {

return m\_data;

}

template<> TWMLAPI const void \*Tensor::getData() const {

return (const void \*)m\_data;

}

std::string getTypeName(twml\_type type) {

switch (type) {

case TWML\_TYPE\_FLOAT32 : return "float32";

case TWML\_TYPE\_FLOAT64 : return "float64";

case TWML\_TYPE\_INT32 : return "int32";

case TWML\_TYPE\_INT64 : return "int64";

case TWML\_TYPE\_INT8 : return "int8";

case TWML\_TYPE\_UINT8 : return "uint8";

case TWML\_TYPE\_BOOL : return "bool";

case TWML\_TYPE\_STRING : return "string";

case TWML\_TYPE\_UNKNOWN : return "Unknown type";

}

throw twml::Error(TWML\_ERR\_TYPE, "Uknown type");

}

uint64\_t getSizeOf(twml\_type dtype) {

switch (dtype) {

case TWML\_TYPE\_FLOAT : return 4;

case TWML\_TYPE\_DOUBLE : return 8;

case TWML\_TYPE\_INT64 : return 8;

case TWML\_TYPE\_INT32 : return 4;

case TWML\_TYPE\_UINT8 : return 1;

case TWML\_TYPE\_BOOL : return 1;

case TWML\_TYPE\_INT8 : return 1;

case TWML\_TYPE\_STRING :

throw twml::Error(TWML\_ERR\_THRIFT, "getSizeOf not supported for strings");

case TWML\_TYPE\_UNKNOWN:

throw twml::Error(TWML\_ERR\_THRIFT, "Can't get size of unknown types");

}

throw twml::Error(TWML\_ERR\_THRIFT, "Invalid twml\_type");

}

} // namespace twml

twml\_err twml\_tensor\_create(twml\_tensor \*t, void \*data, int ndims, uint64\_t \*dims,

uint64\_t \*strides, twml\_type type) {

HANDLE\_EXCEPTIONS(

twml::Tensor \*res = new twml::Tensor(data, ndims, dims, strides, type);

\*t = reinterpret\_cast<twml\_tensor>(res););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_delete(const twml\_tensor t) {

HANDLE\_EXCEPTIONS(

delete twml::getConstTensor(t););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_type(twml\_type \*type, const twml\_tensor t) {

HANDLE\_EXCEPTIONS(

\*type = twml::getConstTensor(t)->getType(););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_data(void \*\*data, const twml\_tensor t) {

HANDLE\_EXCEPTIONS(

\*data = twml::getTensor(t)->getData<void>(););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_dim(uint64\_t \*dim, const twml\_tensor t, int id) {

HANDLE\_EXCEPTIONS(

const twml::Tensor \*tensor = twml::getConstTensor(t);

\*dim = tensor->getDim(id););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_stride(uint64\_t \*stride, const twml\_tensor t, int id) {

HANDLE\_EXCEPTIONS(

const twml::Tensor \*tensor = twml::getConstTensor(t);

\*stride = tensor->getStride(id););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_num\_dims(int \*ndim, const twml\_tensor t) {

HANDLE\_EXCEPTIONS(

const twml::Tensor \*tensor = twml::getConstTensor(t);

\*ndim = tensor->getNumDims(););

return TWML\_ERR\_NONE;

}

twml\_err twml\_tensor\_get\_num\_elements(uint64\_t \*nelements, const twml\_tensor t) {

HANDLE\_EXCEPTIONS(

const twml::Tensor \*tensor = twml::getConstTensor(t);

\*nelements = tensor->getNumElements(););

return TWML\_ERR\_NONE;

}