Program Listing for File holoinfer_buffer.hpp

Return to documentation for file (modules/holoinfer/src/include/holoinfer_buffer.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2022-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * 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. */ #ifndef HOLOINFER_SRC_INCLUDE_HOLOINFER_BUFFER_HPP #define HOLOINFER_SRC_INCLUDE_HOLOINFER_BUFFER_HPP #include <cuda_runtime_api.h> #include <sys/stat.h> #include <algorithm> #include <fstream> #include <iostream> #include <iterator> #include <map> #include <memory> #include <mutex> #include <numeric> #include <string> #include <utility> #include <vector> #include "holoinfer_constants.hpp" #define _HOLOSCAN_EXTERNAL_API_ __attribute__((visibility("default"))) namespace holoscan { namespace inference { uint32_t get_element_size(holoinfer_datatype t) noexcept; class DeviceAllocator { public: bool operator()(void** ptr, size_t size) const; }; class DeviceFree { public: void operator()(void* ptr) const; }; class DeviceBuffer { public: explicit DeviceBuffer(holoinfer_datatype type = holoinfer_datatype::h_Float32); DeviceBuffer(size_t size, holoinfer_datatype type); void* data(); size_t size() const; size_t get_bytes() const; void resize(size_t number_of_elements); ~DeviceBuffer(); private: size_t size_{0}, capacity_{0}; holoinfer_datatype type_ = holoinfer_datatype::h_Float32; void* buffer_ = nullptr; DeviceAllocator allocator_; DeviceFree free_; }; class HostBuffer { public: explicit HostBuffer(holoinfer_datatype data_type = holoinfer_datatype::h_Float32) : type_(data_type) {} void* data() { return static_cast<void*>(buffer_.data()); } size_t size() const { return number_of_elements_; } void set_type(holoinfer_datatype in_type) { type_ = in_type; resize(size()); } void resize(size_t number_of_elements) { buffer_.clear(); number_of_elements_ = number_of_elements; buffer_.resize(number_of_elements * get_element_size(type_)); } private: std::vector<byte> buffer_; size_t number_of_elements_{0}; holoinfer_datatype type_; }; class DataBuffer { public: explicit DataBuffer(holoinfer_datatype data_type = holoinfer_datatype::h_Float32,

```
int device_id = 0); std::shared_ptr<DeviceBuffer> device_buffer; HostBuffer
host_buffer; holoinfer_datatype get_datatype() const { return type_; } int get_device()
const { return device_id_; } private: holoinfer_datatype type_; int device_id_; }; using
DataMap = std::map<std::string, std::shared_ptr<DataBuffer>>; using Mappings =
std::map<std::string, std::string>; using DimType = std::map<std::string,
std::vector<std::vector<int64_t>>>; using MultiMappings = std::map<std::string,
std::vector<std::string>>; struct InferenceSpecs { InferenceSpecs() = default;
InferenceSpecs(const std::string& backend, const Mappings& backend_map, const
Mappings& model_path_map, const MultiMappings& pre_processor_map, const
MultiMappings& inference_map, const Mappings& device_map, const Mappings&
temporal_map, bool is_engine_path, bool oncpu, bool parallel_proc, bool use_fp16,
bool cuda_buffer_in, bool cuda_buffer_out) : backend_type_(backend),
backend_map_(backend_map), model_path_map_(model_path_map),
pre_processor_map_(pre_processor_map), inference_map_(inference_map),
device_map_(device_map), temporal_map_(temporal_map),
is_engine_path_(is_engine_path), oncuda_(!oncpu),
parallel_processing_(parallel_proc), use_fp16_(use_fp16),
cuda_buffer_in_(cuda_buffer_in), cuda_buffer_out_(cuda_buffer_out) {} Mappings
get_path_map() const { return model_path_map_; } Mappings get_backend_map()
const { return backend_map_; } Mappings get_device_map() const { return
device_map_; } Mappings get_temporal_map() const { return temporal_map_; }
std::string backend_type_{""}; Mappings backend_map_; Mappings
model_path_map_; MultiMappings pre_processor_map_; MultiMappings
inference_map_; Mappings device_map_; Mappings temporal_map_; bool
is_engine_path_ = false; bool oncuda_ = true; bool parallel_processing_ = false; bool
use_fp16_ = false; bool cuda_buffer_in_ = true; bool cuda_buffer_out_ = true;
DataMap data_per_tensor_; DataMap output_per_model_; }; InferStatus
allocate_buffers(DataMap& buffers, std::vector<int64_t>& dims, holoinfer_datatype
datatype, const std::string& keyname, bool allocate_cuda, int device_id); } // 
namespace inference } // namespace holoscan #endif/*
HOLOINFER_SRC_INCLUDE_HOLOINFER_BUFFER_HPP */
```