Program Listing for File messagelabel.hpp

Return to documentation for file (include/holoscan/core/messagelabel.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2023-2024 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 HOLOSCAN_CORE_MESSAGELABEL_HPP #define HOLOSCAN_CORE_MESSAGELABEL_HPP #include <chrono> #include <iterator> #include <string> #include <unordered_set> #include <vector> #include "./forward_def.hpp" namespace holoscan { *// The initially reserved length of each path in message_paths* #define DEFAULT_PATH_LENGTH 10 *// The initially reserved number of paths in message_paths* #define DEFAULT_NUM_PATHS 5 static inline int64_t get_current_time_us() { return static_cast<int64_t> (std::chrono::duration_cast<std::chrono::microseconds>(std::chrono::system_clock::now().time_since_epoch()) .count()); } struct OperatorTimestampLabel { public: OperatorTimestampLabel() = default; explicit OperatorTimestampLabel(Operator* op) : operator_ptr(op), rec_timestamp(get_current_time_us()), pub_timestamp(-1) {} OperatorTimestampLabel(Operator* op, int64_t rec_t, int64_t pub_t) : operator_ptr(op), rec_timestamp(rec_t), pub_timestamp(pub_t) {} OperatorTimestampLabel(const OperatorTimestampLabel& o) : operator_ptr(o.operator_ptr), rec_timestamp(o.rec_timestamp), pub_timestamp(o.pub_timestamp) {} OperatorTimestampLabel& operator=(const OperatorTimestampLabel& o); void set_pub_timestamp_to_current() { pub_timestamp = get_current_time_us(); } Operator* operator_ptr = nullptr; *// The timestamp when an Operator receives from an input // For a root Operator, it is the start of the compute call* int64_t rec_timestamp = 0; *// The timestamp when an Operator publishes an output // For a leaf Operator, it is the end of the compute call* int64_t pub_timestamp = 0; }; class MessageLabel { public: using TimestampedPath = std::vector<OperatorTimestampLabel>; using PathOperators =

```
std::unordered_set<std::string>; MessageLabel() { // By default, allocate DEFAULT_NUM_PATHS paths in the message_paths message_paths.reserve(DEFAULT_NUM_PATHS); } MessageLabel(const MessageLabel& m) : message_paths(m.message_paths), message_path_operators(m.message_path_operators) {} MessageLabel& operator=(const MessageLabel& m) { if (this != &m) { this->message_paths = m.message_paths; this->message_path_operators = m.message_path_operators; } return *this; } int num_paths() { return message_paths.size(); } std::vector<std::string> get_all_path_names(); std::vector<TimestampedPath> paths() { return message_paths; } int64_t get_e2e_latency(int index); double get_e2e_latency_ms(int index) { return ((double)get_e2e_latency(index) / 1000); } static double get_path_e2e_latency_ms(TimestampedPath path) { int64_t latency = path.back().pub_timestamp - path.front().rec_timestamp; return (static_cast<double>(latency) / 1000); } TimestampedPath get_path(int index); std::string get_path_name(int index); OperatorTimestampLabel& get_operator(int path_index, int op_index); void set_operator_pub_timestamp(int path_index, int op_index, int64_t pub_timestamp); void set_operator_rec_timestamp(int path_index, int op_index, int64_t rec_timestamp); std::vector<int> has_operator(std::string op_name); void add_new_op_timestamp(holoscan::OperatorTimestampLabel o_timestamp); void update_last_op_publish(); void add_new_path(TimestampedPath path); std::string to_string() const; static std::string to_string(TimestampedPath path); void print_all(); private: std::vector<TimestampedPath> message_paths; std::vector<PathOperators> message_path_operators; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_MESSAGELABEL_HPP */
```