YGM

YGM Developers

Mar 13, 2025

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1.1 What is YGM?

YGM is an asynchronous communication library written in C++ and designed for high-performance computing (HPC) use cases featuring irregular communication patterns. YGM includes a collection of distributed-memory storage containers designed to express common algorithmic and data-munging tasks. These containers automatically partition data, allowing insertions and, with most containers, processing of individual elements to be initiated from any runninng YGM process.

Underlying YGM's containers is a communicator abstraction. This communicator asynchronously sends messages spawned by senders with receivers needing no knowledge of incoming messages prior to their arrival. YGM communications take the form of *active messages*; each message contains a function object to execute (often in the form of C++ lambdas), data and/or pointers to data for this function to execute on, and a destination process for the message to be executed at.

YGM also includes a set of I/O primitives for parsing collections of input documents in parallel as independent lines of text and streaming output lines to large numbers of destination files. Current parsing functionality supports reading input as CSV, ndjson, and unstructured lines of data.

1.2 General YGM Operations

YGM is built on its ability to communicate active messages asynchronously between running processes. This does not capture every operation that can be useful, for instance collective operations are still widely needed. YGM uses prefixes on function names to distinguish their behaviors in terms of the processes involved. These prefixes are:

- `async_`: Asynchronous operation initiated on a single process. The execution of the underlying function may occur on a remote process.
- `local_`: Function performs only local operations on data of the current process. In uses within YGM containers with partitioning schemes that determine item ownership, care must be taken to ensure the process a `local_` operation is called from aligns with the item's owner. For instance, calling `ygm::container::map::local_insert` will store an item on the process where the call is made, but the `ygm::container::map` may not be able to look up this location if it is on the wrong process.
- No Prefix: Collective operation that must be called from all processes.

The primary workhorse functions in YGM fall into the two categories of `async_` and `for_all` operations. In an `async_` operation, a lambda is asynchronously sent to a (potentially) remote process for execution. In many cases with YGM containers, the lambda being executed is not provided by the user and is instead part of the function itself, e.g. `async_insert` calls on most containers. A `for_all` operation is a collective operation in which a lambda is executed locally on every process while iterating over all locally held items of some YGM object. The items iterated over can be

items in a YGM container, items coming from a map, filter, or flatten applied to a container, or all lines in a collection of files in a YGM I/O parser.

1.2.1 Lambda Capture Rules

Certain `async_` and `for_all` operations require users to provide lambdas as part of their executions. The lambdas that can be accepted by these two classes of functions follow different rules pertaining to the capturing of variables:

- `async_` calls cannot capture (most) variables in lambdas. Variables necessary for lambda execution must be provided as arguments to the `async_` call. In the event that the data for the lambda resides on the remote process the lambda will execute on, a `ygm::ygm_ptr` should be passed as an argument to the `async_`.
- `for_all` calls assume lambdas take only the arguments inherently provided by the YGM object being iterated over. All other necessary variables *must* be captured. The types of arguments provided to the lambda can be identified by the `for_all_args` type within the YGM object.

These differences in behavior arise from the distinction that `async_` lambdas may execute on a remote process, while `for_all` lambdas are guaranteed to execute locally to a process. In the case of `async_` operations, the lambda and all arguments must be serialized for communication, but C++ does not provide a method for inspection of variables captured in the closure of a lambda. In the case of `for_all` operations, the execution is equivalent to calling std::for_each on entire collection of items held locally.

1.3 Requirements

- C++20 - GCC versions 11 and 12 are tested. Your mileage may vary with other compilers.
- Cereal - C++ serialization library
- MPI
- Optionally, Boost 1.77 to enable Boost.JSON support.

1.4 Using YGM with CMake

YGM is a header-only library that is easy to incorporate into a project through CMake. Adding the following to CMakeLists.txt will install YGM and its dependencies as part of your project:

```
set(DESIRED_YGM_VERSION 0.6)
find_package(ygm ${DESIRED_YGM_VERSION} CONFIG)
if (NOT ygm_FOUND)
    FetchContent_Declare(
        ygm
        GIT_REPOSITORY https://github.com/LLNL/ygm
        GIT_TAG v${DESIRED_YGM_VERSION}
    )
    FetchContent_GetProperties(ygm)
    if (ygm_POPULATED)
        message(STATUS "Found already populated ygm dependency: "
                       ${ygm_SOURCE_DIR}
        )
    else ()
        set(JUST_INSTALL_YGM ON)
```

(continues on next page)

(continued from previous page)

```
        set(YGM_INSTALL ON)
        FetchContent_Populate(ygm)
        add_subdirectory(${ygm_SOURCE_DIR} ${ygm_BINARY_DIR})
        message(STATUS "Cloned ygm dependency " ${ygm_SOURCE_DIR})
    endif ()
else ()
    message(STATUS "Found installed ygm dependency " ${ygm_DIR})
endif ()
```

1.5 License

YGM is distributed under the MIT license.

All new contributions must be made under the MIT license.

See LICENSE-MIT, NOTICE, and COPYRIGHT for details.

SPDX-License-Identifier: MIT

1.6 Release

LLNL-CODE-789122

CHAPTER

TWO

YGM::COMM CLASS REFERENCE.

CHAPTER

THREE

COMMUNICATOR OVERVIEW

The communicator `ygm::comm` is the central object in YGM. The communicator controls an interface to an MPI communicator, and its functionality can be modified by additional optional parameters.

Communicator Features:

- **Message Buffering** - Increases application throughput at the expense of increased message latency.
- **Message Routing** - Extends benefits of message buffering to extremely large HPC allocations.
- **Fire-and-Forget RPC Semantics** - A sender provides the function and function arguments for execution on a specified destination rank through an *async* call. This function will complete on the destination rank at an unspecified time in the future, but YGM does not explicitly make the sender aware of this completion.

CHAPTER

FOUR

COMMUNICATOR HELLO WORLD

Here we will walk through a basic "hello world" YGM program. The [examples directory](/examples/) contains several other examples, including many using YGM's storage containers.

To begin, headers for a YGM communicator are needed:

```
#include <ygm/comm.hpp>
```

At the beginning of the program, a YGM communicator must be constructed. It will be given `argc` and `argv` like `MPI_Init`.

```
ygm::comm world(&argc, &argv);
```

Next, we need a lambda to send through YGM. We'll do a simple hello_world type of lambda.

```
auto hello_world_lambda = [](const std::string &name) {
        std::cout << "Hello " << name << std::endl;
};
```

Finally, we use this lambda inside of our *async* calls. In this case, we will have rank 0 send a message to rank 1, telling it to greet the world

```
if (world.rank0()) {
        world.async(1, hello_world_lambda, std::string("world"));
}
```

The full, compilable version of this example is found here. Running it prints a single "Hello world".

class **comm**

Public Functions

inline **comm**(int *argc, char ***argv)

inline **comm**(MPI_Comm comm)

inline **~comm**()

inline void **welcome**(std::ostream &os = std::cout)

Prints a welcome message with configuration details.

inline void **stats_reset**()

inline void **stats_print**(const std::string &name = "", std::ostream &os = std::cout)

template<typename **AsyncFunction**, typename ...**SendArgs**>
inline void **async**(int dest, *AsyncFunction* fn, const *SendArgs*&... args)

template<typename **AsyncFunction**, typename ...**SendArgs**>
inline void **async**(int dest, *AsyncFunction* fn, const *SendArgs*&... args) const

template<typename **AsyncFunction**, typename ...**SendArgs**>
inline void **async_bcast**(*AsyncFunction* fn, const *SendArgs*&... args)

template<typename **AsyncFunction**, typename ...**SendArgs**>
inline void **async_mcast**(const std::vector<int> &dests, *AsyncFunction* fn, const *SendArgs*&... args)

inline void **cf_barrier**() const

Control Flow Barrier Only blocks the control flow until all processes in the communicator have called it. See: *MPI_Barrier()*

inline void **barrier**()

Full communicator barrier.

inline void **barrier**() const

inline void **local_progress**()

Checks for incoming unless called from receive queue and flushes one buffer.

inline bool **local_process_incoming**()

template<typename **Function**>
inline void **local_wait_until**(Function fn)

Waits until provided condition function returns true.

Template Parameters
Function –

Parameters
fn – Wait condition function, must match []() -> bool

template<typename **T**>
inline ygm_ptr<*T*> **make_ygm_ptr**(*T* &t)

inline void **register_pre_barrier_callback**(const std::function<void()> &fn)

Registers a callback that will be executed prior to the barrier completion.

Parameters
fn – callback function

template<typename **T**>
inline *T* **all_reduce_sum**(const *T* &t) const

template<typename **T**>
inline *T* **all_reduce_min**(const *T* &t) const

template<typename **T**>
inline *T* **all_reduce_max**(const *T* &t) const

template<typename **T**, typename **MergeFunction**>

inline T **all_reduce**(const T &t, MergeFunction merge) const

Tree based reduction, could be optimized significantly.

Template Parameters

- **T** –
- **MergeFunction** –

Parameters

- **in** –
- **merge** –

Returns

T

inline int **size**() const

inline int **rank**() const

inline MPI_Comm **get_mpi_comm**() const

inline const detail::layout &**layout**() const

inline const detail::comm_router &**router**() const

inline bool **rank0**() const

template<typename **T**>
inline void **mpi_send**(const *T* &data, int dest, int tag, MPI_Comm comm) const

template<typename **T**>
inline void **mpi_send**(const *T* &data, int dest, int tag) const

template<typename **T**>
inline *T* **mpi_recv**(int source, int tag, MPI_Comm comm) const

template<typename **T**>
inline *T* **mpi_recv**(int source, int tag) const

template<typename **T**>
inline *T* **mpi_bcast**(const *T* &to_bcast, int root, MPI_Comm comm) const

template<typename **T**>
inline *T* **mpi_bcast**(const *T* &to_bcast, int root) const

inline std::ostream &**cout0**() const

inline std::ostream &**cerr0**() const

inline std::ostream &**cout**() const

inline std::ostream &**cerr**() const

template<typename ...**Args**>
inline void **cout**(*Args*&&... args) const

template<typename ...**Args**>

inline void **cerr**(*Args*&&... args) const

template<typename ...**Args**>
inline void **cout0**(*Args*&&... args) const

template<typename ...**Args**>
inline void **cerr0**(*Args*&&... args) const

Friends

friend class detail::interrupt_mask

friend class detail::comm_stats

struct **header_t**

Public Members

uint32_t **message_size**

int32_t **dest**

struct **mpi_irecv_request**

Public Members

std::shared_ptr<ygm::detail::byte_vector> **buffer**

MPI_Request **request**

struct **mpi_isend_request**

Public Members

std::shared_ptr<ygm::detail::byte_vector> **buffer**

MPI_Request **request**

CHAPTER

FIVE

YGM::CONTAINER MODULE REFERENCE.

ygm::container is a collection of distributed containers designed specifically to perform well within YGM's asynchronous runtime. Inspired by C++'s Standard Template Library (STL), the containers provide improved programmability by allowing developers to consider an algorithm as the operations that need to be performed on the data stored in a container while abstracting the locality and access details of said data. While insiration is taken from STL, the top priority is to provide expressive and performant tools within the YGM framework.

CHAPTER

SIX

IMPLEMENTED STORAGE CONTAINERS

The currently implemented containers include a mix of distributed versions of familiar containers and distributed-specific containers:

- `ygm::container::bag` - An unordered collection of objects partitioned across processes. Ideally suited for iteration over all items with no capability for identifying or searching for an individual item within the bag.
- `ygm::container::set` - Analogous to `std::set`. An unordered collection of unique objects with the ability to iterate and search for individual items. Insertion and iteration are slower than a `ygm::container::bag`.
- `ygm::container::multiset` - Analogous to `std::multiset`. A set where multiple instances of the same object may appear.
- `ygm::container::map` - Analogous to `std::map`. A collection of keys with assigned values. Keys and values can be inserted and looked up individually or iterated over collectively.
- `ygm::container::multimap` - Analogous to `std::multimap`. A map where keys may appear with multiple values.
- `ygm::container::array` - A collection of items indexed by an integer type. Items can be inserted and looked up by their index values independently or iterated over collectively. Differs from a `std::array` in that sizes do not need to known at compile-time, and a `ygm::container::array` can be dynamically resized through a (potentially expensive) function at runtime.
- `ygm::container::counting_set` - A container for counting occurrences of items. Can be thought of as a `ygm::container::map` that maps items to integer counts but optimized for the case of frequent duplication of keys.
- `ygm::container::disjoint_set` - A distributed disjoint set data structure. Implements asynchronous union operation for maintaining membership of items within mathematical disjoint sets. Eschews the find operation of most disjoint set data structures and instead allows for execution of user-provided lambdas upon successful completion of set merges.

CHAPTER

SEVEN

TYPICAL CONTAINER OPERATIONS

Most interaction with containers occurs in one of two classes of operations: `for_all` and *async_*.

7.1 `for_all` Operations

`for_all`-class operations are barrier-inducing collectives that direct ranks to iteratively apply a user-provided function to all locally-held data. Functions passed to the `for_all` interface do not support additional variadic parameters. However, these functions are stored and executed locally on each rank, and so can capture objects in rank-local scope.

7.2 `async_` Operations

Operations prefixed with `async_` perform operations on containers that can be spawned from any process and execute on the correct process using YGM's asynchronous runtime. The most common *async* operations are:

- `async_insert` - Inserts an item or a key and value, depending on the container being used. The process responsible for storing the inserted object is determined using the container's partitioner. Depending on the container, this partitioner may determine this location using a hash of the item or by heuristics that attempt to evenly spread data across processes (in the case of `ygm::container::bag`).
- `async_visit` - Items within YGM containers will be distributed across the universe of running processes. Instead of providing operations to look up this data directly, which would involve a round-trip communication with the process storing the item of interest, most YGM containers provide `async_visit`. A call to `async_visit` takes a function to execute and arguments to pass to the function and asynchronously executes the provided function with arguments that are the item stored in the container and the additional arguments passed to `async_visit`.

Specific containers may have additional `async_` operations (or may be missing some of the above) based on the capabilities of the container. Consult the documentation of individual containers for more details.

7.2.1 array

template<typename **Value**, typename **Index** = size_t>

class **array** : public ygm::container::detail::base_async_insert_key_value<*array*<*Value*, size_t>, std::tuple<size_t, *Value*>>, public ygm::container::detail::base_misc<*array*<*Value*, size_t>, std::tuple<size_t, *Value*>>, public ygm::container::detail::base_async_visit<*array*<*Value*, size_t>, std::tuple<size_t, *Value*>>, public ygm::container::detail::base_iteration_key_value<*array*<*Value*, size_t>, std::tuple<size_t, *Value*>>, public ygm::container::detail::base_async_reduce<*array*<*Value*, size_t>, std::tuple<size_t, *Value*>>

Public Types

using **self_type** = *array*<*Value*, *Index*>

using **mapped_type** = *Value*

using **key_type** = *Index*

using **size_type** = *Index*

using **for_all_args** = std::tuple<*Index*, *Value*>

using **container_type** = ygm::container::array_tag

using **ptr_type** = typename ygm::ygm_ptr<*self_type*>

Public Functions

array() = delete

inline **array**(ygm::comm &comm, const *size_type* size)

inline **array**(ygm::comm &comm, const *size_type* size, const *mapped_type* &default_value)

inline **array**(ygm::comm &comm, std::initializer_list<*mapped_type*> l)

inline **array**(ygm::comm &comm, std::initializer_list<std::tuple<*key_type*, *mapped_type*>> l)

inline **array**(const *self_type* &rhs)

template<typename **T**>
inline **array**(ygm::comm &comm, const *T* &t)

template<typename **T**>
inline **array**(ygm::comm &comm, const *T* &t)

template<typename **T**>
inline **array**(ygm::comm &comm, const *T* &t)

template<typename **T**>
inline **array**(ygm::comm &comm, const *T* &t)

template<typename **T**>
inline **array**(ygm::comm &comm, const *T* &t)

inline ~**array**()

inline void **local_insert**(const *key_type* &key, const *mapped_type* &value)

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit**(const *key_type* index, *Function* &fn, const *VisitorArgs*&... args)

inline void **async_set**(const *key_type* index, const *mapped_type* &value)

template<typename **BinaryOp**>
inline void **async_binary_op_update_value**(const *key_type* index, const *mapped_type* &value, const *BinaryOp* &b)

inline void **async_bit_and**(const *key_type* index, const *mapped_type* &value)

inline void **async_bit_or**(const *key_type* index, const *mapped_type* &value)

inline void **async_bit_xor**(const *key_type* index, const *mapped_type* &value)

inline void **async_logical_and**(const *key_type* index, const *mapped_type* &value)

inline void **async_logical_or**(const *key_type* index, const *mapped_type* &value)

inline void **async_multiplies**(const *key_type* index, const *mapped_type* &value)

inline void **async_divides**(const *key_type* index, const *mapped_type* &value)

inline void **async_plus**(const *key_type* index, const *mapped_type* &value)

inline void **async_minus**(const *key_type* index, const *mapped_type* &value)

template<typename **UnaryOp**>
inline void **async_unary_op_update_value**(const *key_type* index, const *UnaryOp* &u)

inline void **async_increment**(const *key_type* index)

inline void **async_decrement**(const *key_type* index)

const *mapped_type* &**default_value**() const

inline void **resize**(const *size_type* size, const *mapped_type* &fill_value)

inline void **resize**(const *size_type* size)

inline size_t **local_size**()

inline size_t **size**() const

inline void **local_clear**()

inline void **local_swap**(*self_type* &other)

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **ReductionOp**>
inline void **local_reduce**(const *key_type* index, const *mapped_type* &value, *ReductionOp* reducer)

inline void **sort**()

inline void **async_insert**(const typename std::tuple_element<0, *for_all_args*>::type &key, const typename std::tuple_element<1, *for_all_args*>::type &value)

inline void **async_insert**(const std::pair<const typename std::tuple_element<0, *for_all_args*>::type, typename std::tuple_element<1, *for_all_args*>::type> &kvp)

inline void **clear**()

inline void **swap**(derived_type &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<derived_type> **get_ygm_ptr**()

inline const ygm::ygm_ptr<derived_type> **get_ygm_ptr**() const

template<typename **Visitor**, typename ...**VisitorArgs**>
inline void **async_visit**(const std::tuple_element<0, *for_all_args*>::type &key, *Visitor* visitor, const *VisitorArgs*&... args)

template<typename **Visitor**, typename ...**VisitorArgs**>
inline void **async_visit_if_contains**(const std::tuple_element<0, *for_all_args*>::type &key, *Visitor* visitor, const *VisitorArgs*&... args)

template<typename **Visitor**, typename ...**VisitorArgs**>
inline void **async_visit_if_contains**(const std::tuple_element<0, *for_all_args*>::type &key, *Visitor* visitor, const *VisitorArgs*&... args) const

template<typename **Function**>
inline void **for_all**(*Function* fn)

template<typename **Function**>
inline void **for_all**(*Function* fn) const

template<typename **STLContainer**>
inline void **gather**(*STLContainer* &gto, int rank) const

template<typename **Compare** = std::greater<std::pair<*key_type*, *mapped_type*>>>
inline std::vector<std::pair<*key_type*, *mapped_type*>> **gather_topk**(size_t k, *Compare* comp = *Compare*()) const

template<typename **YGMContainer**>
inline void **collect**(*YGMContainer* &c) const

template<typename **MapType**, typename **ReductionOp**>
inline void **reduce_by_key**(*MapType* &map, *ReductionOp* reducer) const

template<typename **TransformFunction**>
transform_proxy_key_value<derived_type, *TransformFunction*> **transform**(*TransformFunction* ffn)

inline auto **keys**()

inline auto **values**()

inline flatten_proxy_key_value<derived_type> **flatten**()

template<typename **FilterFunction**>
filter_proxy_key_value<derived_type, *FilterFunction*> **filter**(*FilterFunction* ffn)

template<typename **ReductionOp**>
inline void **async_reduce**(const typename std::tuple_element<0, *for_all_args*>::type &key, const typename std::tuple_element<1, *for_all_args*>::type &value, *ReductionOp* reducer)

Public Members

detail::block_partitioner<*key_type*> **partitioner**

Friends

friend class detail::base_misc< array< Value, Index >, std::tuple< Index, Value > >

7.2.2 bag

template<typename **Item**>

class **bag** : public ygm::container::detail::base_async_insert_value<*bag*<*Item*>, std::tuple<*Item*>>, public ygm::container::detail::base_count<*bag*<*Item*>, std::tuple<*Item*>>, public ygm::container::detail::base_misc<*bag*<*Item*>, std::tuple<*Item*>>, public ygm::container::detail::base_iteration_value<*bag*<*Item*>, std::tuple<*Item*>>

Public Types

using **self_type** = *bag*<*Item*>

using **value_type** = *Item*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Item*>

using **container_type** = ygm::container::bag_tag

Public Functions

inline **bag**(ygm::comm &comm)

inline **bag**(ygm::comm &comm, std::initializer_list<*Item*> l)

template<typename **STLContainer**>
inline **bag**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **bag**(ygm::comm &comm, const *YGMContainer* &yc)

inline **~bag**()

inline **bag**(const *self_type* &other)

inline **bag**(*self_type* &&other) noexcept

inline *bag* &**operator=**(const *self_type* &other)

inline *bag* &**operator=**(*self_type* &&other) noexcept

inline void **async_insert**(const *Item* &value, int dest)

inline void **async_insert**(const std::vector<*Item*> &values, int dest)

inline void **local_insert**(const *Item* &val)

inline void **local_clear**()

inline size_t **local_size**() const

inline size_t **local_count**(const *value_type* &val) const

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline void **serialize**(const std::string &fname)

inline void **deserialize**(const std::string &fname)

inline void **rebalance**()

template<typename **RandomFunc**>
inline void **local_shuffle**(*RandomFunc* &r)

inline void **local_shuffle**()

template<typename **RandomFunc**>
inline void **global_shuffle**(*RandomFunc* &r)

inline void **global_shuffle**()

inline void **async_insert**(const typename std::tuple_element<0, std::tuple<*Item*>>::type &value)

inline size_t **count**(const std::tuple_element<0, std::tuple<*Item*>>::type &value) const

inline size_t **size**() const

inline void **clear**()

inline void **swap**(*bag*<*Item*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*bag*<*Item*>> **get_ygm_ptr**()

inline const ygm::ygm_ptr<*bag*<*Item*>> **get_ygm_ptr**() const

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<*value_type*> **gather_topk**(size_t k, Compare comp = std::greater<*value_type*>()) const

inline *value_type* **reduce**(MergeFunction merge) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_value<*bag*<*Item*>, TransformFunction> **transform**(TransformFunction ffn)

inline flatten_proxy_value<*bag*<*Item*>> **flatten**()

filter_proxy_value<*bag*<*Item*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

detail::round_robin_partitioner **partitioner**

Friends

friend class detail::base_misc< bag< Item >, std::tuple< Item > >

7.2.3 counting_set

template<typename **Key**>

class **counting_set** : public ygm::container::detail::base_count<*counting_set*<*Key*>, std::tuple<*Key*, size_t>>, public ygm::container::detail::base_misc<*counting_set*<*Key*>, std::tuple<*Key*, size_t>>, public ygm::container::detail::base_iteration_key_value<*counting_set*<*Key*>, std::tuple<*Key*, size_t>>

Public Types

using **self_type** = *counting_set*<*Key*>

using **mapped_type** = size_t

using **key_type** = *Key*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Key*, size_t>

using **container_type** = ygm::container::counting_set_tag

Public Functions

inline **counting_set**(ygm::comm &comm)

counting_set() = delete

inline **counting_set**(ygm::comm &comm, std::initializer_list<*Key*> l)

template<typename **STLContainer**>
inline **counting_set**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **counting_set**(ygm::comm &comm, const *YGMContainer* &yc)

inline void **async_insert**(const *key_type* &key)

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline void **local_clear**()

inline void **clear**()

inline size_t **local_size**() const

inline *mapped_type* **local_count**(const *key_type* &key) const

inline *mapped_type* **count_all**()

template<typename **CompareFunction**>
inline std::vector<std::pair<*key_type*, *mapped_type*>> **topk**(size_t k, *CompareFunction* cfn)

inline std::map<*key_type*, *mapped_type*> **gather_keys**(const std::vector<*key_type*> &keys)

inline ygm::ygm_ptr<*self_type*> **get_ygm_ptr**() const

inline void **serialize**(const std::string &fname)

inline void **deserialize**(const std::string &fname)

inline size_t **count**(const std::tuple_element<0, std::tuple<*Key*, size_t>>::type &value) const

inline size_t **size**() const

inline void **swap**(*counting_set*<*Key*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*counting_set*<*Key*>> **get_ygm_ptr**()

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<std::pair<*key_type*, *mapped_type*>> **gather_topk**(size_t k, Compare comp = Compare()) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_key_value<*counting_set*<*Key*>, TransformFunction> **transform**(TransformFunction ffn)

inline auto **keys**()

inline auto **values**()

inline flatten_proxy_key_value<*counting_set*<*Key*>> **flatten**()

filter_proxy_key_value<*counting_set*<*Key*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

const *size_type* **count_cache_size** = 1024 * 1024

detail::hash_partitioner<std::hash<*key_type*>> **partitioner**

Friends

friend class detail::base_misc< counting_set< Key >, std::tuple< Key, size_t > >

7.2.4 disjoint_set

template<typename **Item**, typename **Partitioner** = detail::old_hash_partitioner<*Item*>>

class **disjoint_set**

Public Types

using **self_type** = *disjoint_set*<*Item*, *Partitioner*>

using **value_type** = *Item*

using **size_type** = size_t

using **ygm_for_all_types** = std::tuple<*Item*, *Item*>

using **container_type** = ygm::container::disjoint_set_tag

using **impl_type** = detail::disjoint_set_impl<*Item*, *Partitioner*>

Public Functions

disjoint_set() = delete

inline **disjoint_set**(ygm::comm &comm, const size_t cache_size = 8192)

template<typename **Visitor**, typename ...**VisitorArgs**>
inline void **async_visit**(const *value_type* &item, *Visitor* visitor, const *VisitorArgs*&... args)

inline void **async_union**(const *value_type* &a, const *value_type* &b)

template<typename **Function**, typename ...**FunctionArgs**>
inline void **async_union_and_execute**(const *value_type* &a, const *value_type* &b, *Function* fn, const *FunctionArgs*&... args)

inline void **all_compress**()

template<typename **Function**>
inline void **for_all**(*Function* fn)

inline std::map<*value_type*, *value_type*> **all_find**(const std::vector<*value_type*> &items)

inline void **clear**()

inline *size_type* **size**()

inline *size_type* **num_sets**()

inline ygm::ygm_ptr<*impl_type*> **get_ygm_ptr**() const

inline ygm::comm &**comm**()

7.2.5 map

template<typename **Key**, typename **Value**>

class **map** : public ygm::container::detail::base_async_insert_key_value<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_insert_or_assign<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_misc<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_count<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_reduce<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_erase_key<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_erase_key_value<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_batch_erase_key_value<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_visit<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_iteration_key_value<*map*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>

Public Types

using **self_type** = *map*<*Key*, *Value*>

using **mapped_type** = *Value*

using **ptr_type** = typename ygm::ygm_ptr<*self_type*>

using **key_type** = *Key*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Key*, *Value*>

using **container_type** = ygm::container::map_tag

Public Functions

map() = delete

inline **map**(ygm::comm &comm)

inline **map**(ygm::comm &comm, const *mapped_type* &default_value)

inline **map**(ygm::comm &comm, std::initializer_list<std::pair<*Key*, *Value*>> l)

template<typename **STLContainer**>
inline **map**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **map**(ygm::comm &comm, const *YGMContainer* &yc)

inline ~**map**()

inline void **local_insert**(const *key_type* &key)

inline void **local_erase**(const *key_type* &key)

inline void **local_erase**(const *key_type* &key, const *key_type* &value)

inline void **local_insert**(const *key_type* &key, const *mapped_type* &value)

inline void **local_insert_or_assign**(const *key_type* &key, const *mapped_type* &value)

inline void **local_clear**()

template<typename **ReductionOp**>
inline void **local_reduce**(const *key_type* &key, const *mapped_type* &value, *ReductionOp* reducer)

inline size_t **local_size**() const

inline *mapped_type* &**local_at**(const *key_type* &key)

inline const *mapped_type* &**local_at**(const *key_type* &key) const

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args)

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit_if_contains**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args)

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit_if_contains**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args) const

template<typename **STLKeyContainer**>
inline std::map<*key_type*, *mapped_type*> **gather_keys**(const *STLKeyContainer* &keys)

inline std::vector<*mapped_type*> **local_get**(const *key_type* &key) const

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline size_t **local_count**(const *key_type* &key) const

inline void **async_insert**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value)

inline void **async_insert**(const std::pair<const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type, typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type> &kvp)

inline void **async_insert_or_assign**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value)

inline void **async_insert_or_assign**(const std::pair<typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type, typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type> &kvp)

inline size_t **size**() const

inline void **clear**()

inline void **swap**(*map*<*Key*, *Value*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*map*<*Key*, *Value*>> **get_ygm_ptr**()

inline const ygm::ygm_ptr<*map*<*Key*, *Value*>> **get_ygm_ptr**() const

inline size_t **count**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &value) const

inline void **async_reduce**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value, ReductionOp reducer)

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key)

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **async_visit**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args)

inline void **async_visit_if_contains**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args)

inline void **async_visit_if_contains**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args) const

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<std::pair<*key_type*, *mapped_type*>> **gather_topk**(size_t k, Compare comp = Compare()) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_key_value<*map*<*Key*, *Value*>, TransformFunction> **transform**(TransformFunction ffn)

inline auto **keys**()

inline auto **values**()

inline flatten_proxy_key_value<*map*<*Key*, *Value*>> **flatten**()

filter_proxy_key_value<*map*<*Key*, *Value*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

detail::hash_partitioner<std::hash<*key_type*>> **partitioner**

Friends

friend class detail::base_misc< map< Key, Value >, std::tuple< Key, Value > >

7.2.6 multimap

template<typename **Key**, typename **Value**>

class **multimap** : public ygm::container::detail::base_async_insert_key_value<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_misc<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_count<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_erase_key<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_erase_key_value<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_batch_erase_key_value<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_async_visit<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>, public ygm::container::detail::base_iteration_key_value<*multimap*<*Key*, *Value*>, std::tuple<*Key*, *Value*>>

Public Types

using **self_type** = *multimap*<*Key*, *Value*>

using **mapped_type** = *Value*

using **ptr_type** = typename ygm::ygm_ptr<*self_type*>

using **key_type** = *Key*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Key*, *Value*>

using **container_type** = ygm::container::multimap_tag

Public Functions

multimap() = delete

inline **multimap**(ygm::comm &comm)

inline **multimap**(ygm::comm &comm, const *mapped_type* &default_value)

inline **multimap**(ygm::comm &comm, std::initializer_list<std::pair<*Key*, *Value*>> l)

template<typename **STLContainer**>
inline **multimap**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **multimap**(ygm::comm &comm, const *YGMContainer* &yc)

inline **~multimap**()

inline void **local_insert**(const *key_type* &key)

inline void **local_erase**(const *key_type* &key)

inline void **local_erase**(const *key_type* &key, const *key_type* &value)

inline void **local_insert**(const *key_type* &key, const *mapped_type* &value)

inline void **local_clear**()

inline size_t **local_size**() const

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args)

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit_if_contains**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args)

template<typename **Function**, typename ...**VisitorArgs**>
inline void **local_visit_if_contains**(const *key_type* &key, *Function* &fn, const *VisitorArgs*&... args) const

inline std::vector<*mapped_type*> **local_get**(const *key_type* &key)

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline size_t **local_count**(const *key_type* &key) const

inline void **async_insert**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value)

inline void **async_insert**(const std::pair<const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type, typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type> &kvp)

inline size_t **size**() const

inline void **clear**()

inline void **swap**(*multimap*<*Key*, *Value*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*multimap*<*Key*, *Value*>> **get_ygm_ptr**()

inline const ygm::ygm_ptr<*multimap*<*Key*, *Value*>> **get_ygm_ptr**() const

inline size_t **count**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &value) const

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key)

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, const typename std::tuple_element<1, std::tuple<*Key*, *Value*>>::type &value)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **async_visit**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args)

inline void **async_visit_if_contains**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args)

inline void **async_visit_if_contains**(const std::tuple_element<0, std::tuple<*Key*, *Value*>>::type &key, Visitor visitor, const VisitorArgs&... args) const

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<std::pair<*key_type*, *mapped_type*>> **gather_topk**(size_t k, Compare comp = Compare()) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_key_value<*multimap*<*Key*, *Value*>, TransformFunction> **transform**(TransformFunction ffn)

inline auto **keys**()

inline auto **values**()

inline flatten_proxy_key_value<*multimap*<*Key*, *Value*>> **flatten**()

filter_proxy_key_value<*multimap*<*Key*, *Value*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

detail::hash_partitioner<std::hash<*key_type*>> **partitioner**

Friends

friend class detail::base_misc< multimap< Key, Value >, std::tuple< Key, Value > >

7.2.7 multiset

template<typename **Value**>

class **multiset** : public ygm::container::detail::base_async_insert_value<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_erase_key<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_batch_erase_key<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_contains<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_insert_contains<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_count<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_misc<*multiset*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_iteration_value<*multiset*<*Value*>, std::tuple<*Value*>>

Public Types

using **self_type** = *multiset*<*Value*>

using **value_type** = *Value*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Value*>

using **container_type** = ygm::container::set_tag

using **key_type** = std::tuple_element_t<0, std::tuple<*Value*>>

Public Functions

inline **multiset**(ygm::comm &comm)

inline **multiset**(const *self_type* &other)

inline **multiset**(*self_type* &&other) noexcept

inline **multiset**(ygm::comm &comm, std::initializer_list<*Value*> l)

template<typename **STLContainer**>
inline **multiset**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **multiset**(ygm::comm &comm, const *YGMContainer* &yc)

inline ~**multiset**()

multiset() = delete

inline *multiset* &**operator=**(const *self_type* &other)

inline *multiset* &**operator=**(*self_type* &&other) noexcept

inline void **local_insert**(const *value_type* &val)

inline void **local_erase**(const *value_type* &val)

inline void **local_clear**()

inline size_t **local_count**(const *value_type* &val) const

inline size_t **local_size**() const

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline void **serialize**(const std::string &fname)

inline void **deserialize**(const std::string &fname)

inline void **async_insert**(const typename std::tuple_element<0, std::tuple<*Value*>>::type &value)

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Value*>>::type &key)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **async_contains**(const std::tuple_element<0, std::tuple<*Value*>>::type &value, Function fn, const FuncArgs&... args)

inline void **async_insert_contains**(const std::tuple_element<0, std::tuple<*Value*>>::type &value, Function fn, const FuncArgs&... args)

inline size_t **count**(const std::tuple_element<0, std::tuple<*Value*>>::type &value) const

inline size_t **size**() const

inline void **clear**()

inline void **swap**(*multiset*<*Value*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*multiset*<*Value*>> **get_ygm_ptr**()

inline const ygm::ygm_ptr<*multiset*<*Value*>> **get_ygm_ptr**() const

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<*value_type*> **gather_topk**(size_t k, Compare comp = std::greater<*value_type*>()) const

inline *value_type* **reduce**(MergeFunction merge) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_value<*multiset*<*Value*>, TransformFunction> **transform**(TransformFunction ffn)

inline flatten_proxy_value<*multiset*<*Value*>> **flatten**()

filter_proxy_value<*multiset*<*Value*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

detail::hash_partitioner<std::hash<*value_type*>> **partitioner**

Friends

friend class detail::base_misc< multiset< Value >, std::tuple< Value > >

7.2.8 set

template<typename **Value**>

class **set** : public ygm::container::detail::base_async_insert_value<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_erase_key<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_batch_erase_key<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_contains<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_async_insert_contains<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_count<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_misc<*set*<*Value*>, std::tuple<*Value*>>, public ygm::container::detail::base_iteration_value<*set*<*Value*>, std::tuple<*Value*>>

Public Types

using **self_type** = *set*<*Value*>

using **value_type** = *Value*

using **size_type** = size_t

using **for_all_args** = std::tuple<*Value*>

using **container_type** = ygm::container::set_tag

using **key_type** = std::tuple_element_t<0, std::tuple<*Value*>>

Public Functions

inline **set**(ygm::comm &comm)

inline **set**(const *self_type* &other)

inline **set**(*self_type* &&other) noexcept

inline **set**(ygm::comm &comm, std::initializer_list<*Value*> l)

template<typename **STLContainer**>
inline **set**(ygm::comm &comm, const *STLContainer* &cont)

template<typename **YGMContainer**>
inline **set**(ygm::comm &comm, const *YGMContainer* &yc)

inline **~set**()

set() = delete

inline *set* &**operator=**(const *self_type* &other)

inline *set* &**operator=**(*self_type* &&other) noexcept

inline void **local_insert**(const *value_type* &val)

inline void **local_erase**(const *value_type* &val)

inline void **local_clear**()

inline size_t **local_count**(const *value_type* &val) const

inline size_t **local_size**() const

template<typename **Function**>
inline void **local_for_all**(*Function* fn)

template<typename **Function**>
inline void **local_for_all**(*Function* fn) const

inline void **serialize**(const std::string &fname)

inline void **deserialize**(const std::string &fname)

inline void **async_insert**(const typename std::tuple_element<0, std::tuple<*Value*>>::type &value)

inline void **async_erase**(const typename std::tuple_element<0, std::tuple<*Value*>>::type &key)

inline void **erase**(const Container &cont)

inline void **erase**(const Container &cont)

inline void **async_contains**(const std::tuple_element<0, std::tuple<*Value*>>::type &value, Function fn, const FuncArgs&... args)

inline void **async_insert_contains**(const std::tuple_element<0, std::tuple<*Value*>>::type &value, Function fn, const FuncArgs&... args)

inline size_t **count**(const std::tuple_element<0, std::tuple<*Value*>>::type &value) const

inline size_t **size**() const

inline void **clear**()

inline void **swap**(*set*<*Value*> &other)

inline ygm::comm &**comm**()

inline const ygm::comm &**comm**() const

inline ygm::ygm_ptr<*set*<*Value*>> **get_ygm_ptr**()

inline const ygm::ygm_ptr<*set*<*Value*>> **get_ygm_ptr**() const

inline void **for_all**(Function fn)

inline void **for_all**(Function fn) const

inline void **gather**(STLContainer &gto, int rank) const

inline std::vector<*value_type*> **gather_topk**(size_t k, Compare comp = std::greater<*value_type*>()) const

inline *value_type* **reduce**(MergeFunction merge) const

inline void **collect**(YGMContainer &c) const

inline void **reduce_by_key**(MapType &map, ReductionOp reducer) const

transform_proxy_value<*set*<*Value*>, TransformFunction> **transform**(TransformFunction ffn)

inline flatten_proxy_value<*set*<*Value*>> **flatten**()

filter_proxy_value<*set*<*Value*>, FilterFunction> **filter**(FilterFunction ffn)

Public Members

detail::hash_partitioner<std::hash<*value_type*>> **partitioner**

Friends

friend class detail::base_misc< set< Value >, std::tuple< Value > >

CHAPTER

EIGHT

YGM CONTAINER EXAMPLE

```cpp
// Copyright 2019-2021 Lawrence Livermore National Security, LLC and other YGM
// Project Developers. See the top-level COPYRIGHT file for details.
//
// SPDX-License-Identifier: MIT

#include <ygm/comm.hpp>
#include <ygm/container/map.hpp>

int main(int argc, char **argv) {
  ygm::comm world(&argc, &argv);

  ygm::container::map<std::string, std::string> my_map(world);

  if (world.rank0()) {
    my_map.async_insert("dog", "bark");
    my_map.async_insert("cat", "meow");
  }

  world.barrier();

  auto favorites_lambda = [](auto key, auto &value, const int favorite_num) {
    std::cout << "My favorite animal is a " << key << ". It says '" << value
              << "!' My favorite number is " << favorite_num << std::endl;
  };

  // Send visitors to map
  if (world.rank() % 2) {
    my_map.async_visit("dog", favorites_lambda, world.rank());
  } else {
    my_map.async_visit("cat", favorites_lambda, world.rank() + 1000);
  }

  return 0;
}
```

CHAPTER

NINE

CONTAINER TRANSFORMATION OBJECTS

`ygm::container` provides a number of transformation objects that can be applied to containers to alter the appearance of items passed to `for_all` operations without modifying the items within the container itself. The currently supported transformation objects are:

- `filter` - Filters items in a container to only execute on the portion of the container satisfying a provided boolean function.
- `flatten` - Extract the elements from tuple-like objects before passing to the user's `for_all` function.
- `map` - Apply a generic function to the container's items before passing to the user's `for_all` function.

CHAPTER

TEN

DOCUMENTS FOR YGM DEVELOPERS

10.1 Developing YGM

This page contains information for YGM developers.

10.1.1 Build Read the Docs (RTD)

Here is how to build RTD document using Sphinx on your machine.

Listing 1: How to build RTD docs locally

```
# Install required software
brew install doxygen graphviz sphinx-doc
pip install breathe sphinx_rtd_theme

# Set PATH and PYTHONPATH, if needed
# For example:
# export PATH="/opt/homebrew/opt/sphinx-doc/bin:${PATH}"
# export PYTHONPATH="/path/to/python/site-packages:${PYTHONPATH}"

git clone https://github.com/LLNL/ygm.git
cd ygm
mkdir build && cd build

# Run CMake
cmake ../ -DYGM_RTD_ONLY=ON

# Generate Read the Docs documents using Sphinx
# This command runs Doxygen to generate XML files
# before Sphinx automatically
make sphinx
# Open the following file using a web browser
open docs/rtd/sphinx/index.html

# For running doxygen only
make doxygen
# open the following file using a web browser
open docs/html/index.html
```

Rerunning Build Command

Depending on what files are modified, one may need to rerun the CMake command and/or `make sphinx`. For instance:

- Require running the CMake command and `make sphinx`:
  - Adding new RTD-related files, including configuration and .rst files
  - Modifying CMake files
- Require running only `make sphinx`
  - **Existing** files (except CMake) are modified

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Y