# Proposal to the GASPI Specification
# Inclusion of `gaspi_alltoallv`

Vanessa End and Christian Simmendinger

January 14, 2016

## Contents



## 1 Introduction and Motivation

Moving towards the exascale era in high performance computing we see the necessity of new communication models for (sparse) collective communication.

Specifically, we see the need to pipeline the execution of alltoallv collective communication with a subsequent evaluation of partial results of this collective communication. We hence suggest the introducton of a notification driven, partially evaluatable alltoallv collective communication into the GASPI standard.

We see a large variety of corresponding use-cases in, e.g., fast fourier transformations (e.g. p3dfft), solvers for sparse matrices (e.g. PETSc), or more generally matrix transposes or any form of data redistribution. We note that especially for large HPC systems with complex network topologies, high-level GASPI implementations (via e.g `gaspi_write_notify`) of such a partially evaluatable collective will be difficult and error-prone. Even more problematic is the fact that these implementations need to be specifically designed for the respective underlying network topology. Last but not least, due to the high numbers of messages involved we expect a correspondingly rather high function call overhead in such a high-level implementation.

We hence propose the addition of 3 new routines to the GASPI standard:

1. `gaspi_get_remote_offsets`
2. `gaspi_alltoallv`
3. `gaspi_alltoallv_reset`

The first of these routines is a convenience function for the user, not necessary but useful for the `gaspi_alltoallv`. Users of an alltoallv communication will require to obtain information about the remote communication entries for a sparse collective communication, including:

- Which data will be transferred to the process from other processes?
- Where will the data be transferred to?
- How much memory has to be reserved for transferred data?

As these preparatory steps are mandatory of an irregular alltoall communication (and identical for all the potential uses cases) we provide the `gaspi_get_remote_offsets` routine, which returns the required information in a convenient format.

The second and third functions, i.e., `gaspi_alltoallv` and `gaspi_alltoallv_reset`, implement the actual alltoall communication.

As GASPI is designed to be scalable, fault tolerant and equipped with a timeout mechanism for non-local communication routines, we have designed an alltoallv which fits these design goals:

Listing 1: `gaspi_alltoallv` Example usage

```
1  //GASPI initialization and group creation has been set up
2  //segment with segment_id_local has been created
3
4  #pragma omp parallel
5  while(gaspi_alltoallv( group
6                       , source_segment_id
7                       , &source_offset[0]
8                       , &remote_ranks[0]
9                       , &remote_segment_ids[0]
10                      , &remote_offset[0]
11                      , &size[0]
12                      , &received_rank
13                      , num_receives
14                      , GASPI_TEST) != GASPI_SUCCESS){
15
16   // work on local offsets
17   // c.f. gaspi_get_remote_offsets()
18   work(received_rank,receive_offset_local);
19
20   gaspi_alltoallv_reset(group, received_rank, 1);
21 }
```

## 2 gaspi_get_remote_offsets

A massively parallel application may require communication in subgroups of processes, where each rank only has knowledge about the data it needs to transfer to remote ranks. Thus the rank will need some information from remote ranks about, e.g., the remote segment to write to or the exact offset to write to. At the same time, it needs information on where the data will be located after a successful transfer from a remote rank. This communication may, e.g., be an alltoallv, a halo exchange or a matrix-vector-multiplication. If the memory of the application is absolutely symmetric, the needed information is directly at hand but in an asymmetric layout, some communication and computation has to be done to have this information. The routine `gaspi_get_remote_offsets` is a convenience routine, communicating and computing relevant information for a future data communication as mentioned above.

The function takes the following input arguments:

- local segment ID
- number of messages to be transferred by the calling rank
- size of the messages to be transferred by the calling rank
- ranks to transfer data to
- the offset, at which the first remote data element should locally be written

and returns

- all remote segment IDs to write to
- the number of messages the calling rank will receive
- the size of the messages it will receive
- the offsets to which the messages are to be written to
- the offsets, where received messages will be stored
- the required size of the buffer where remote ranks will transfer the data to
- the ranks from which it will receive data

such that messages can be written from a local source segment to a remote destination segment and the rank knows where it can find the data written from remote ranks. It is a *non-local*, *asynchronous* and *time-based blocking* routine.

```
GASPI_GET_REMOTE_OFFSETS( group
                        , local_target_segment_id
                        , first_local_offset
                        , target_ranks[num_writes]
                        , offset_remote[num_writes]
                        , local_message_sizes[num_writes]
                        , num_writes
                        , remote_segment_ids[num_writes]
                        , remote_source_ranks[num_receives]
                        , receive_offset_local[num_receives]
                        , remote_message_sizes[num_receives]
                        , num_receives
                        , required_buffer_size
                        , timeout)
```

*Parameter:*
*(in) group:* the group of ranks, which participate in the routine
*(in) local_target_segment_id:* the segment id of the local target segment
*(in) first_local_offset:* offset of first data element to be written to local segment
*(in) target_ranks[num_writes]:* the ranks actually written to from the calling process
*(out) offset_remote[num_writes]:* the calculated remote offsets in bytes to write to
*(in) local_message_sizes[num_writes]:* the sizes in bytes of the data to be written
*(in) num_writes:* the number of messages to be written
*(out) remote_segment_ids:* the segment ids of the remote target segments
*(out) remote_source_ranks[num_receives]:* the ranks that will transfer data to the calling process
*(out) receive_offset_local[num_receives]:* offsets of received data
*(out) remote_message_sizes[num_receives]:* the sizes in bytes of the data received
*(out) num_receives:* number of messages received
*(out) required_buffer_size:* the size in bytes necessary to hold all received messages
*(in) timeout:* the timeout

```
gaspi_return_t
gaspi_get_remote_offsets( gaspi_group_id_t group
                        , gaspi_segment_id_t
                          local_target_segment_id
                        , gaspi_offset_t first_local_offset
                        , gaspi_rank_t *target_ranks
                        , gaspi_offset_t *offset_remote
                        , gaspi_size_t *local_message_sizes
                        , gaspi_number_t num_writes
                        , gaspi_segment_id_t *remote_segment_id
                        , gaspi_rank_t *remote_source_ranks
                        , gaspi_offset_t *receive_offset_local
                        , gaspi_size_t *remote_message_sizes
                        , gaspi_number_t *num_receives
                        , gaspi_size_t *required_buffer_size
                        , gaspi_timeout_t timeout)
```

```
function gaspi_get_remote_offsets(group, &
&           local_target_segment_id, first_local_offset, &
&           target_ranks, offset_remote, &
&           local_message_sizes, num_writes, remote_segment_id, &
&           remote_source_ranks, receive_offset_local, &
&           remote_message_sizes, num_receives, &
&           required_buffer_size, timeout ) &
&          result(res) bind(C, name="gaspi_get_remote_offsets")
  integer(gaspi_group_t), value :: group
  integer(gaspi_segment_id_t) :: local_target_segment_id
  integer(gaspi_offset_t), value :: first_local_offset
  type(c_ptr) :: target_ranks
  type(c_ptr) :: offset_remote
  type(c_ptr), value :: local_message_sizes
  integer(gaspi_number_t), value :: num_writes
  integer(gaspi_segment_id_t) :: remote_segment_id
  type(c_ptr) :: remote_source_ranks
  type(c_ptr) :: receive_offset_local
  type(c_ptr) :: remote_message_sizes
  integer(gaspi_number_t) :: num_receives
  integer(gaspi_size_t) :: required_buffer_size
  integer(gaspi_timeout_t, value) :: timeout
  integer(gaspi_return_t) :: res
end function gaspi_get_remote_offsets
```

*Execution phase:*
Working

*Return values:*

GASPI_SUCCESS: operation has returned successfully
GASPI_TIMEOUT: operation has run into a timeout
GASPI_ERROR: operation has finished with an error ⌟

After successful procedure completion, i. e., return value GASPI_SUCCESS, the necessary size *required_buffer_size* of the local target segment or buffer will be computed. In addition, all communication relevant information will be computed, such that the message of size *local_message_size[i]* can be transferred to *target_ranks[i]* on *remote_segment_id[i]* at offset *offset_remote[i]*. Accordingly the message received from rank *remote_source_ranks[i]* in the following communication can be found on *local_segment_id* at offset *receive_offset_local[i]*.

The remote offsets, where the data shall be written will have been computed in dependance of *first_local_offset* and returned in *offset_remote*. The segment IDs of the remote target buffers will be returned in *remote_segment_id*.

In case the offsets and segment size could not be computed and completely communicated in the time given through *timeout*, the return value is GASPI_TIMEOUT. A subsequent call of gaspi_get_remote_offsets has to be invoked in order to complete the alltoallv.

In case of error, the return value is GASPI_ERROR.

Invoking gaspi_get_remote_offsets in any other execution phase than the working phase yields undefined behavior.

# 3 GASPI Alltoallv

The GASPI Alltoallv routine is a split-phase collective communication routine, distributing data among the participating processes in a user-defined mapping. The gaspi_alltoallv and the gaspi_alltoallv_reset are tightly coupled and both need to be called every time an alltoallv is performed. A source and a target segment will have to be created by the user in order to successfully use an gaspi_alltoallv. The size of the target segments need to be large enough to hold all remotely written data. To get the necessary size, a previous call to gaspi_get_remote_offsets can be made.

## 3.1 gaspi_alltoallv

The gaspi_alltoallv is a collective operation, distributing a maximum of *group_size* data elements to *group_size* processes in a user-defined mapping. It is a *non-local, asynchronous* and *time-based blocking* routine.

The application needs to have knowledge of the number of messages written from the local segment (*num_writes*) and of the number of received messages (*num_receives*). Latter can be retrieved from gaspi_get_remote_offsets. To process the data after it is visible to the process, the application needs knowledge of the offsets the remote ranks have written to. These offsets can also be retrieved from gaspi_get_remote_offsets. The source and target segments needed for the alltoallv must previously be created by the user.

```
GASPI_ALLTOALLV( group
               , source_segment_id
               , source_offset[num_writes]
               , remote_ranks[num_writes]
               , remote_segment_ids[num_writes]
               , remote_offset[num_writes]
               , size[num_writes]
               , received_rank
               , num_receives
               , timeout)
```

*Parameter:*
*(in) group:* the group of ranks participating in the alltoallv
*(in) source_segment_id:* the ID of the local segment to write data from
*(in) source_offset[num_writes]:* the local offsets in bytes to write data from
*(in) remote_ranks[num_writes]:* the ranks that are written to
*(in) remote_segment_ids:* the segment ids to write to
*(in) remote_offset[num_writes]:* the remote offsets in bytes to write to
*(in) size[num_writes]:* the sizes of the data blocks to be transferred
*(out) received_rank:* the source rank of a successfully written message
*(in) num_receives:* the number of messages to be received
*(in) timeout:* the timeout

```
gaspi_return_t
gapsi_alltoallv( gaspi_group_t group
               , gaspi_segment_id_t source_segment_id
               , gaspi_offset_t *source_offset
               , gaspi_rank_t *remote_ranks
               , gaspi_segment_id_t *remote_segment_ids
               , gaspi_offset_t *remote_offset
               , gaspi_size_t *size
               , gaspi_rank_t *received_rank
               , gasi_number_t num_receives
               , gaspi_timeout_t timeout)
```

```
function gaspi_alltoallv( group, source_segment_id, &
&          source_offset, remote_ranks, target_segment_id, &
&          offset_remote,size, received_rank, num_receives, &
&          timeout) &
&          result(res) bind(C, name="gaspi_alltoallv")
  integer(gaspi_group_t), value :: group
  integer(gaspi_segment_id_t), value :: source_segment_id
  type(c_ptr), value :: source_offset
  type(c_ptr), value :: remote_ranks
  integer(gaspi_segment_id_t), value :: remote_segment_ids
  type(c_ptr), value :: remote_offset
  type(c_ptr), value :: size
  integer(gaspi_rankt_t) :: received_rank
  integer(gaspi_number_t), value :: num_receives
  integer(gaspi_timeout_t), value :: timeout
  integer(gaspi_return_t) :: res
end function gaspi_alltoallv
```

*Execution phase:*
Working

*Return values:*
`GASPI_SUCCESS`: operation has returned successfully
`GASPI_TIMEOUT`: operation has run into a timeout
`GASPI_ERROR`: operation has finished with an error ⌟

For each process, `gaspi_alltoallv` posts transfers *num_writes* contiguous data blocks per process to the peers given in *remote_ranks*. The source of the data written to *remote_ranks[i]* is given through *source_segment_id, source_offset[i]* and the target through *remote_segment_id, remote_offset[i]*.

After successful procedure completion, i. e., return value `GASPI_SUCCESS`, all communication requests will have been posted, the remote data will have been written to the local data segment and is visible to the process.

If the routine exits with `GASPI_TIMEOUT`, not all data elements have been written to the local segment from the remote ranks or not all other ranks are ready for communication and subsequent calls to the routine are necessary for successful completion.

*received_rank* will hold the rank of the source process of one message successfully written to the local data segment.

Alltoallv is exclusive per group, i. e., only one alltoallv operation on a given group can run at a time.

The data to be transferred needs to reside in the global address space. The source and target segments have to be previously allocated and are identified through *source_segment_id* and *remote_segment_id* respectively. The segments need to have appropriate sizes to hold all data written within the `gaspi_alltoallv`.

In case of error, the return value is GASPI_ERROR. The error vector should be examined.

> *User advice:* When using gaspi_alltoallv in a *non-blocking* manner, the routine will only return with GASPI_SUCCESS if the (possibly hierarchical) communication schedule for sending data has been completed and every *rank*'s message has been acknowledged as received, i.e., *num_receives* calls to gaspi_alltoallv will have to be made. When using gaspi_alltoallv in a blocking manner, the argument *rank* will hold the number of ranks participating in the routine. ⌟

## 3.2 gaspi_alltoallv_reset

gaspi_alltoallv_reset resets a remote notification thus allowing the remote process to overwrite the local memory associated with the *rank*. It is a *non-local, blocking* procedure.

```
GASPI_ALLTOALLV_RESET( group
                     , ranks[num]
                     , num)
```

*Parameter:*
*(in) group:* the group participating in the alltoallv
*(in) ranks:* the ranks on which to reset the notifications
*(in) num:* the number of message IDs to be reset

```
gaspi_return_t
gaspi_alltoallv_reset( gaspi_group_t group
                     , gaspi_rank_t *ranks
                     , gaspi_number_t num)
```

```
function gaspi_alltoallv_reset(group, ranks, num) &
&         result( res ) bind(C, "gaspi_alltoallv_reset")
  integer(gaspi_group_t), value :: group
  type(c_ptr), value :: ranks
  integer(gaspi_number_t), value :: num
  integer(gaspi_return_t) :: res
end function gaspi_alltoallv_reset
```

*Execution phase:*
Working

*Return values:*
GASPI_SUCCESS: operation has returned successfully
GASPI_ERROR: operation has finished with an error ⌟

`gaspi_alltoallv_reset` resets the *num* notifications associated with the messages received in the `gaspi_alltoallv` thus allowing remote *ranks* to overwrite the locally reserved and associated buffer in the next `gaspi_alltoallv`. The notifications on all *rank*s have to be reset before a successive call to `gaspi_alltoallv` can be successful.

After successful procedure completion, i. e. return value `GASPI_SUCCESS`, the notifications has been posted to the collective queue.

In case of return value `GASPI_ERROR`, the request could not be posted to the collective queue.

Invoking `gaspi_alltoallv_reset` in any other execution phase than the working phase yields undefined behavior.

> *User advice:* In a multi-threaded environment every thread can post a request to reset the notification associated to the data it just worked on immediately. In a single-threaded environment, one call to `gaspi_alltoallv_reset` can be used to reset all notifications at once. ⌟

# 4 Needed Resources

- A maximum of `max_num_alltoallv` notification segments, as defined through the configuration. We need the notification buffers plus a segment piece in the PGAS of size `group_size * sizeof(gaspi_offset_t)` or `group_size * sizeof(gaspi_size_t)` - whichever is larger - to communicate the offsets and message sizes to be used in the `gaspi_alltoallv`
- some internal structure, mapping the given group to the internal segment
- for each group using the alltoallv, we need a fully connected infrastucture for one-sided communication in that group

# 5 Additional (necessary) Changes to the Standard

- section 3.10 GASPI collective communication will have to be adapted to include alltoallv
- setion 5.2 config will have to be changed - add num alltoallv and num get_remote_offsets
- section 5.5.2 list of collective communication routines/other routines changing state vector
- section 2.3 defines `GASPI_TEST` to wait for data, sections 3.9 and 11.1 define `GASPI_TEST` to perform an atomic portion of work.