GASNet Teams Specification and Design Document

Rajesh Nishtala, Dan Bonachea, Paul Hargrove
rajeshn@cs.berkeley.edu, bonachea@cs.berkeley.edu, PHHargrove@lbl.gov

Computer Science Division, University of California at Berkeley
Future Technologies Group, Lawrence Berkeley National Lab

Revision
Preliminary Work: For Internal Distribution Only

1 Introduction

In a variety of applications the use of collective communication is very important for both expressing the communication pattern of the application as well as achieving the best performance. However, one of the fundamental aspects that makes the collectives useful is the ability to use them on a subset (or team) of the processors rather than all the processors in the program. This document will focus on the design and implementation of these teams within GASNet as well as the API that exposes them to applications or libraries that use GASNet.

Since GASNet allows hierarchical declaration of nodes and images we must be careful about the terminology we use and how we compare our implementation with MPI’s. Henceforth, we will use “node” to refer to a GASNet process. Each of these GASNet nodes can potentially contain many GASNet images (which can be implemented on top of different threads if one chooses). The distinction allows one to perform shared memory communication optimizations across various images within the same node since they will share the same address space. In MPI parlance this hierarchy is not explicit and thus all the threads of control visible to the application are at the same level which we will call “tasks.” Our teams will consist of GASNet images rather than nodes since we want different threads in the same process to be a part of different teams. We will allow images to have their own relative ranks within different teams but the GASNet node identifier will remain fixed throughout the life of the program.

2 Design Goals

- One of the important and fundamental characteristics of GASNet that needs to be addressed in the design of teams is the ability to have multiple “images” of GASNet on the same GASNet process, thus allowing one to perform shared memory/intra-node optimizations for collectives whenever possible. This ability will need to also be effectively exposed in the design of the teams.

- Each team of images will need to be sufficiently isolated from the others so that the communication and synchronization mechanisms will never interfere with those from another team. This will allow libraries to be written on top of GASNet that take teams as arguments and be guaranteed that operations on one team will be isolated from operations performed on another team provided that they do not share members. If they do, then special care must be done at the application level.

- The team creation and destruction must be abstract and expressive enough to easily and quickly build up whatever subset of processors (GASNet images) the user wishes.

- The teams must fit seamlessly into the existing GASNet collectives framework so that collective performance on all the images in a GASNet program is not hindered by the addition of teams.
3 Similarities and Differences with MPI

The problem of communication with a subset of the processors is certainly not a new one and has been around in the MPI specification for many years. In MPI parlance the notion of teams is expressed as communicators. Many of the ideas presented in the API for MPI communicators (and groups) are orthogonal to their use of two-sided communication and their design of the collectives and thus a lot of these ideas are applicable here. However, there are some important differences that will need to be addressed as well.

3.1 Similarities

- **Communicators**: As stated above, the notion of an MPI communicator is very similar to our concept of teams, modulo some of the differences stated below. A team, like a communicator, is a set of GASNet images along with a preallocated set of buffer space and communication meta data (e.g. a distributed buffer space manager) to allow fast collective communication across the various members of the team.

- **Groups**: We will also adopt the idea of an MPI Group into GASNet. The primary function of a GASNet group is to allow easy team construction. The process of creating a team is an expensive process due to the setup of all the required meta-data. Since a group merely an ordered set of images without the additional meta data necessary for communication and synchronization their construction and modification can be done using much simpler, and therefore more efficient, operations. Thus the general model, like MPI, will be to build up a group based on the different operations provided in the API and then construct a team around a group once the group has been finalized.

3.2 Differences

- **GASNet Images**: As stated above the biggest difference (especially in the implementation) that needs to be addressed is the difference between a node and an image and how these relate to the teams. MPI does not have this problem since there is no notion of a hierarchical structure between MPI tasks; they are all at the same level.

- **Scratch Space**: GASNet has an explicit segment which allows one to take advantage of some very important communication optimizations (e.g. RDMA) in one-sided operations. Since we wish to leverage some of these same optimizations in the collectives, the collectives (and thus teams) will need to reclaim some of the space that has been allocated to the GASNet client to provide the best possible performance. The auxiliary scratch space for the initial GASNET_TEAM_ALL will be handled directly within GASNet so that it isn’t visible to the end user, however further team construction will necessitate the GASNet client explicitly managing these buffers.

- **Usage**: Another important and distinct difference is that MPI allows one to use a communicator for isolation of point-to-point messaging operations such as sends and receives. GASNet is a one-sided communication system that lacks such two-sided message passing operations, and the teams are not relevant for one-sided point-to-point communication. GASNet teams are only relevant for use in the collectives library. Thus each image will have one globally unique name for point-to-point communication. The translation routines that are provided so that one can specify the root for rooted collectives relative to the other images in the team.

4 Groups

A GASNet group is defined as an ordered set of GASNet images that will take part in the construction of a GASNet team. A group is designed to be simple and easy to construct based on other groups. All these operations are non-collective. The following is a list of the operations that will operate on and work with a gasnet_group_t.

Every operation that returns a gasnet_group_t value logically creates a new group object that should later be reclaimed by calling gasnet_group_free.

- **Type gasnet_image_t**
  An unsigned integral type capable of representing the largest number of images permitted in the current GASNet implementation.
- **Type gasnet_group_t**
  An abstract type representing ordered set of GASNet images, where each image is assigned a unique, 0-based index of type gasnet_image_t, also referred to as the *rank* of the image within the group. In a gasnet_group_t representing *N* images, the rank indexes shall comprise the set \(\{0, \ldots, (N - 1)\}\).

- **Constant gasnet_image_t GASNET_IMAGE_UNDEFINED**
  A constant defined for an invalid image number.

- **GlobalVar: gasnet_group_t GASNET_GROUP_EMPTY**
  A predefined group representing an empty set of images, to aid in group creation.

- **gasnet_image_t gasnet_group_size(gasnet_group_t g)**
  Returns the total number of images in a gasnet_group_t *g*.

- **gasnet_image_t gasnet_group_size_on_node(gasnet_group_t g, gasnet_node_t n)**
  Returns the number of images residing on node *n* that are a member of group *g*.

- **gasnet_image_t gasnet_group_my_image(gasnet_group_t g)**
  Returns the image rank in *g* of the calling thread, or GASNET_IMAGE_UNDEFINED if the image corresponding to the calling thread is not a member of *g*.

- **void gasnet_group_translate_images(gasnet_group_t A, gasnet_group_t B, gasnet_image_t *input_vec, gasnet_image_t *output_vec, gasnet_image_t vec_len)**
  Given a set of input images in *input_vec* from group *A* the function will translate them to their corresponding ranks in *B* and write the result into *output_vec*. If some images do not exist in node *B*, the corresponding array slot will be filled with GASNET_IMAGE_UNDEFINED.

- **gasnet_node_t gasnet_group_image_to_node(gasnet_group_t g, gasnet_image_t i)**
  Given a relative rank *i* in *g*, the function returns the node *n* that contains the image.

- **void gasnet_group_node_to_images(gasnet_group_t g, gasnet_node_t n, gasnet_image_t *outimages)**
  Fills the output array *outimages* with the ranks of the images in *g* that reside on node *n*. Use gasnet_group_size_on_node to get the count of how many elements this function will return.

- **enum {GASNET_GROUP_IDENT, GASNET_GROUP_SIMILAR, GASNET_GROUP_UNEQUAL} gasnet_group_compare(gasnet_group_t a, gasnet_group_t b)**
  This function returns one of three values:
  1. If *a* and *b* represent the same set of images with the same rank ordering, the function will return GASNET_GROUP_IDENT.
  2. If *a* and *b* represent the same set of images but the ranks of the images are different then the function returns GASNET_GROUP_SIMILAR.
  3. Otherwise the function will return GASNET_GROUP_UNEQUAL.

- **gasnet_group_t gasnet_group_from_team(gasnet_team_t t)**
  Returns a gasnet_group_t that represents the images in *t*.

- **gasnet_group_t gasnet_group_incl(gasnet_group_t g, gasnet_image_t * in_images, gasnet_image_t image_count)**
  Returns a group that consists of the *image_count* images in group *g* with ranks *in_images[0], ..., in_images[n - 1]*; the image with rank *i* in the returned group is the image with rank *in_images[i]* in *g*. Each of the *image_count* elements of *in_images* must be a valid rank in group *g* and all elements must be distinct, or else the program is erroneous. If *image_count* = 0, then newgroup is GASNET_GROUP_EMPTY. This function can, for instance, be used to reorder the elements of a group.
• `gasnet_group_t gasnet_group_excl(gasnet_group_t g, gasnet_image_t * in_images, gasnet_image_t image_count)`
  Returns a `gasnet_group_t` that removes the images with ranks contained in `in_images` from `g`. The relative ordering of the remaining images from `g` will not change. Each of the `image_count` elements of `in_images` must be a valid rank in `g` and all elements must be distinct; otherwise, the program is erroneous. If `image_count = 0`, then the returned group is identical to `g`.

• `gasnet_group_t gasnet_group_diff(gasnet_group_t A, gasnet_group_t B)`
  Returns a `gasnet_group_t` that is the set difference of `A` and `B`. The resultant set will be all the images of `A` that are not in `B`, ordered by their relative rankings in `A`.

• `gasnet_group_t gasnet_group_intersect(gasnet_group_t A, gasnet_group_t B)`
  Returns a `gasnet_group_t` that is the set intersection of `A` and `B`. The images are ordered according to their ordering in `A`.

• `gasnet_group_t gasnet_group_union(gasnet_group_t A, gasnet_group_t B)`
  Returns a `gasnet_group_t` that is the set union of `A` and `B`. The images of `A` are placed before the images of `B` in the ordering.

• `size_t gasnet_group_to_node_scratch_size(gasnet_group_t g)`
  Given a group `g` the function returns the amount of node-local scratch space the calling thread would need to provide in order to construct a team from `g` using `gasnet_team_from_group`. Guaranteed to return zero if the image associated with the calling thread is not a member of `g`.

• `size_t gasnet_split_to_node_scratch_size(gasnet_team_t t, gasnet_image_t color, gasnet_image_t relrank)`
  This is the only group-like function that is a collective call - it must be called collectively by all images comprising team `t`. The call returns the amount of node-local scratch space the calling thread would need to provide in order to perform a split of the team `t` with the corresponding `color` and `relrank` arguments that one would use for the `gasnet_team_split()` function. If there are multiple teams that will be created on the same node, the function will return a valid size for the scratch space for one representative image from each of the teams that will reside on the node. A nonzero return value indicates that this image must provide the indicated amount of scratch space for the corresponding team split. Guaranteed to return zero to any thread which passes `GASNET_IMAGE_UNDEFINED` as `relrank`.

• `void gasnet_group_free(gasnet_group_t g)`
  This function will perform the required housekeeping to reclaim any resources associated with group `g`.

5 Team Operators

All team constructor operations are collectively invoked over a parent team. All team constructor operations require local scratch space to be provided, with a size indicated by the corresponding group operations. The scratch space provided must reference valid memory residing within the bounds of the local GASNet segment (for `GASNET_SEGMENT_EVERYTHING`, any valid local memory location is suitable). The provided scratch space is consumed by the team construction routine, and must not be accessed by the client throughout the lifetime of the created team.

• GlobalVar: `GASNET_TEAM_ALL`
  Initial team that all images are a part of. Comparable to `MPI_COMM_WORLD`.

• `gasnet_image_t gasnet_team_my_image(gasnet_team_t t)`
  Given a team the function returns a relative rank of the calling image within `t`. If the calling image is not contained in `t` then `GASNET_IMAGE_UNDEFINED` is returned.

• `gasnet_image_t gasnet_team_size(gasnet_team_t t)`
  Returns the number of images in `t`. 
gasnet_team_t gasnet_team_split(gasnet_team_t t, gasnet_image_t color, gasnet_image_t relrank, void *local_scratch_space)
This function must be called collectively by all images comprising team \( t \). This function splits the team \( t \) based on the \( \text{color} \) and \( \text{relrank} \) arguments. All images that call the function with the same \( \text{color} \) argument will be separated into the same team. The \( \text{relrank} \) argument specifies the relative position of the image within the new team. If multiple images call the split function with the same color and relrank argument the results are undefined. Note that an image is responsible for allocating the local_scratch_space if the corresponding call to gasnet_split_to_node_scratch_size() returned a nonzero value. One can pass GASNET_IMAGE_UNDEFINED to the \( \text{relrank} \) to indicate that the image is not to be part of any team.

Note this function can be used to duplicate an existing team \( t \), by having all images pass \( \text{color} = 0 \) and \( \text{relrank} = \) gasnet_team_my_image(t).

gasnet_team_t gasnet_team_from_group(gasnet_team_t t, gasnet_group_t g, void *local_scratch_space)
This function must be called collectively by all images comprising team \( t \). This function will create a child team of \( t \) from the group \( g \) with the given scratch space. The results are undefined if all members of \( g \) are not part of \( t \).

void* gasnet_team_free(gasnet_team_t t)
This function must be called collectively by all images comprising team \( t \). Free a team by cleaning up all associated resources with the team. Once the free routine returns it is safe to free the scratch space that was passed in to create the team. As a convenience, the original argument scratch space argument is returned to each image to facilitate reclaiming the associated storage.

6 Notes to Self About Implementation.

1. Keep Team Hierarchy
2. Reference Count Groups