

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 1/17

This is a language-independent specification of a meta-API for data reorganization. This version of the specification is based on the consensus of the Data Reorganization Forum following its December 1999 meeting.

There are 4 sections to this document:

- 1) Administrative notes
- 2) Change summary between this version and the prior version
- 3) Current version of "critical path" interfaces in the API
- 4) Current version of other interfaces in the API

----- SECTION 1: Administrative Notes -----

These notes are simply re-iterated from the last version of this draft document (dated 02/02/2000).

NOTE #1: A change has been made in the organization of this file so that the functions that are in the "critical path" to creating and initiating a data reorganization function are presented first. Other functions such as object query functions that are not in this "critical path" are presented later.

As of this date (03/07/2000), this draft contains only the critical path calls (i.e., Section 4 is empty). Since there have been many changes to these functions, we should agree on their specification before handling the low-level ("power user") and non-critical functions.

NOTE #2: A new feature is being added for the committee members - a list of changes that have been introduced in this newer version of the API and the corresponding reasons. The goal is to prevent revisiting issues that have been resolved in past working meetings.

----- SECTION 2: Change Summary -----

***** Changes from December 1999 draft to March 1999 (PRE DR MEETING) draft:

1. Marked appropriate functions as "<META>" to indicate areas where Data Reorg will likely use infrastructure from other middleware

Each meta function now has a "META NOTES" section to try to organize the discussion on the meta-nature of the DRI API.
2. Inserted <UNRESOLVED> notation in this document where some remaining decisions are needed. Before final API is settled, we need to go back and remove these notes and replace with the final decisions
3. Inserted destroy functions for the following objects:
 - DRI_global_data
 - DRI_group
 - DRI_overlap
 - DRI_distspec
 - DRI_bufferset
 - DRI_channel.

***** Changes from September 1999 to December 1999 drafts:

1. dri_group_myrank name changed to dri_group_get_rank()
2. dri_dist_create - added a note in RESTRICTIONS/POLICY section reiterating that this call may not involve collective communication, at the implementation's discretion. This of course means that erroneous programs could cause dri_dist_create to calculate an incorrect data partitioning.
3. dri_dist_create - Added useful default layout parameters so the user doesn't have to use dri_layout_create to create commonly-needed objects (e.g., DRI_LAYOUT_PACKED_012). All possible packed layouts for 1, 2, and 3 dimensions are provided. This effectively makes dri_layout_create a non "critical-path" function in the API.
4. dri_dist_create - A NULL pointer argument for the group_dims parameter specifies that the user wants to have the implementation determine an appropriate logical process set topology to use in dividing up the data during the execution of this call.
5. dri_dist_create - Added a note in the DESCRIPTION section that says that a valid entry in the distspecs array parameter is DRI_DISTSPEC_INDIVISIBLE (this capability was accidentally removed from the API in prior edits).
6. dri_dist_create - Noted that the group_dims array parameter corresponds directly to the dimsizes array parameter of the dri_global_data_create function.
7. Added a dri_dist_get_numblocks function
8. Added a dri_dist_get_blockinfo function to return a single structure that gives all needed information about a locally owned block of data following the partitioning process. Returns a new DRI_blockinfo structure type. internally, the DRI_blockinfo structure contains an array of DRI_blockdim structures. This pair of structures replaces the old DRI_part object and bounds_t structure. DRI_part was not adding anything beyond DRI_dist, so we have opted directly query DRI_dist for the low level partitioning information. The bounds_t structure was poorly named, and needed to be more descriptive (we now call it DRI_blockdim). Additional information was needed beyond what the old bounds_t provided, so we just
9. Changed dri_part_calc_local_size to dri_dist_calc_local_size, since the DRI_part object has been removed and we now just query DRI_dist objects.
10. Added the DRI_bufferset object and its dri_bufferset_create function
11. DRI_transfer object has been renamed to DRI_channel

----- SECTION 3: Current API for "critical path" functions -----

```

/***** dri_global_data_create *****/
dri_global_data_create - Create a global data object

```

SYNOPSIS

```
dri_global_data_create(ndims, dimsizes[ndims], dataspec, gdo)
```

PARAMETERS

IN: ndims (integer) - number of dimensions in the global data

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 3/17

IN: dimsizes (integer array) - size of each dimension of the global data
 IN: dataspec (DRI_dataspec) - data type of each element of the global data
 OUT: gdo (DRI_global_data) - object that describes the global data

DESCRIPTION

Creates a global data object to describe application data. The size information supplied by the user refers to the size of the application data `_without_` considering how the data will eventually be partitioned across a group of processes in the parallel environment

COMMUNICATION BEHAVIOR

Local. All processes that will participate in a future data reorganization involving this data must create this object independently.

RESTRICTIONS / POLICY

All processes that will participate in a data reorganization on the described data must call this function with identical `ndims`, `dimsizes`, and `dataspec` parameters. Implementations may place an upper limit on the `ndims` parameter. However, all implementations must minimally support
`1 <= ndims <= 3`

```

/***** <META> dri_group_create *****/
dri_group_create - Create an object to represent a group of processes
  
```

SYNOPSIS

```
dri_group_create(nprocs, procs[nprocs], group)
```

PARAMETERS

IN: `nprocs` (integer) - total number of processes in the group
 IN: `procs` (array of integers) - array of unique process identifiers
 OUT: `group` (DRI_group) - process group object

META-NOTES

This is meta at the "object level". That is, we may want to entirely leverage process set constructs from other middlewares.

<UNRESOLVED>

What if we are not using standard middleware, but a process set construct exists? Do we want to leverage those (non-portable) approaches to alleviate the need to do process set management in DRI? If we choose to do this, then this object and its methods become unnecessary?

</UNRESOLVED>

DESCRIPTION

Creates an object to represent a group of unique processes in the parallel processing environment. The group is a one-dimensional ordering of processes. The `procs` array must consist of a list of unique identifiers for each process in the group to be formed by this call. Because this is a meta-API, the intent of the group object is to be the same object used by the underlying (or co-) layer, such as MPI or MPI/RT. Also, an implication is that the `procs` array may need to be specified differently depending on the middleware that is used to implement the data reorganization interface. For example, in a general-purpose workstation MPI environment, it may be sufficient for the user to specify a list of MPI ranks in the `procs` array. In some embedded

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 4/17

environments (or others) that involve multiple programs in the run-time environment, more precise information (e.g., system-assigned process id) may be required in the procs array.

COMMUNICATION BEHAVIOR
Local.

RESTRICTIONS / POLICY

A process gets its group rank (logical id within the group) based on where it is referenced in the procs array. The first process listed in procs will be assigned rank 0, for example. Ranks are therefore assigned values in the range 0..(nprocs-1). In order for all processes that call this function to have a consistent understanding of process ranks, the procs array must be specified identically by all callers.

```

/***** <META> dri_group_get_rank *****/
dri_group_get_rank - Return the rank of the calling process in specified group

```

SYNOPSIS

```
dri_group_get_rank(group, rank)
```

PARAMETERS

IN: group (DRI_group) - group object
OUT: rank (integer) - rank of the calling process in the group

META-NOTES

This is meta at the "object level". That is, we may want to entirely leverage process set constructs from other middlewares.

<UNRESOLVED>

What if we are not using standard middleware, but a process set construct exists? Do we want to leverage those (non-portable) approaches to alleviate the need to do process set management in DRI? If we choose to do this, then this object and its methods become unnecessary?

</UNRESOLVED>

DESCRIPTION

Returns the rank (logical process id) in the given group to the caller.

COMMUNICATION BEHAVIOR
Local

RESTRICTIONS / POLICY

Only members of the specified group may call this function successfully

```

/***** dri_overlap_create *****/
dri_overlap_create - Create an overlap data partitioning object

```

SYNOPSIS

```
dri_overlap_create(ovr_type, num_pos, overlap)
```

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 5/17

PARAMETERS

IN: ovr_type (DRI_overlap_type) - overlap policy to implement at the edges of a global data object. Can be one of:

DRI_OVERLAP_TRUNCATE
 DRI_OVERLAP_TOROIDAL
 DRI_OVERLAP_PAD_ZEROS
 DRI_OVERLAP_PAD_REPLICATED

IN: num_pos (integer) - number of positions to overlap

OUT: overlaph (DRI_overlap) - overlap object

DESCRIPTION

Creates the overlap attribute used in the data distribution high-level specification. The resulting DRI_overlap object is to be passed into the dri_distspec_create function as a left or right overlap argument.

NOTE: Just like the DRI_distspec object, the user is expected to create a DRI_overlap object specification for each dimension of global data (where a nonzero overlap is desired). In the event that no overlap is requested by the user, DRI_NO_OVERLAP can be passed as the left and right overlap arguments to the dri_distspec_create function.

In general, overlap is the storage of extra data in a processor's local data buffer to hold data that is adjacent in the global data context, and that is assigned to another processor, based on the data partitioning. Overlap therefore refers to data that is stored on processor boundaries in the partitioning of the global data.

There are different overlap policies supported:

1) ovr_type == DRI_OVERLAP_TRUNCATE

The local buffer should contain enough space to store copies of num_pos adjacent, non-local elements. At the ends of the global data object, extra storage is not required in the local data buffer, and is truncated accordingly.

2) ovr_type == DRI_OVERLAP_TOROIDAL

The local buffer should contain enough space to store copies of num_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with data from the num_pos elements that start at the opposite end of the global data dimension.

3) ovr_type == DRI_OVERLAP_PAD_ZEROS

The local buffer should contain enough space to store copies of num_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with zeros.

4) ovr_type == DRI_OVERLAP_PAD_REPLICATED

The local buffer should contain enough space to store copies of num_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with a copy of the last num_pos _locally_ held elements.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

/***** dri_distspec_block_create *****/
/***** dri_distspec_blockcyclic_create *****/

```

dri_distspec_block_create - Create a block distribution specification
dri_distspec_blockcyclic_create - Create a block cyclic distribution

SYNOPSIS

```
dri_distspec_block_create(minsz, mod, lov, rov, distspec)
```

```
dri_distspec_blockcyclic_create(lov, rov, blksize, distspec)
```

PARAMETERS

IN: minsz (integer) - minimum number of local elements required
(user specifies 0 to indicate no preference)

IN: mod (integer) - modulo requirement
(user specifies 1 to indicate no preference)

IN: lov (DRI_overlap) - left overlap (DRI_NO_OVERLAP specifies no overlap)

IN: rov (DRI_overlap) - right overlap (DRI_NO_OVERLAP specifies no overlap)

IN: blksize (integer) - block-cyclic partitioning block size
(user specifies 1 for pure cyclic partition)

OUT: distspec (DRI_distspec) - high-level data distribution object

DESCRIPTION

These functions create a DRI_distspec object that stores information about either a block or blockcyclic partitioning of global application data. Users must associate a separate DRI_distspec object with each dimension of partitioned global data. The output object, distspec, is only a high-level specification of the requested data partitioning. It does not store exact partitioning details such as specific global data indices assigned to a particular process. Because a DRI_distspec object is not associated with any single global data array, it can be reused for many different data partitionings. The more exact partitioning information for a global data array is stored in the DRI_dist object that can be queried for detailed partitioning information following the dri_dist_create operation.

Parameter mod specifies that the number of local elements ultimately assigned to the calling process must be some multiple of mod.

Parameters lov and rov specify element overlaps (left and right, respectively). These parameters do not change the mapping of global data indices to processors in the data partitioning. They allow copies of adjacent global data elements at the (left or right) boundaries of the data partitioning to be stored locally. A right overlap refers to overlap in the direction of higher global indices. Consult the section on the DRI_overlap object for additional details about the overlap specification.

Parameter blksize is used in block-cyclic partitionings to define the size (in number of elements) of the blocks that get assigned to processors in

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 7/17

the global data partitioning.

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

This object may NOT be queried until the completion of a subsequent `dri_dist_create` call.

COMMUNICATION BEHAVIOR

Local

```

/***** <META> dri_dist_create *****/
dri_dist_create - Create a distribution object for a specific global data
                  object over a specific process group

```

SYNOPSIS

```
dri_dist_create(gdo, group, group_dims, distspecs, layout, disth)
```

PARAMETERS

```

IN:  gdo (DRI_global_data) - global data object
IN:  group (DRI_group) - process group
IN:  group_dims (array of integer) - logical dimensions of process group
IN:  distspecs (array of DRI_distspec) - high-level data distribution specs
      (one array entry per gdo dimension)

IN:  layout (DRI_layout) - memory layout of local data buffers
OUT: disth (DRI_dist) - data distribution object

```

META-NOTES

This function is meta because it takes parameters of type `DRI_distspec` and `DRI_group`, which are meta at the "object level". The `DRI_dist` object itself is not meta - it is unique to Data Reorganization. It is meta at an "interface level"

DESCRIPTION

This function aggregates all of the input objects into a single container, a `DRI_dist` object. It also calculates explicitly the data block(s) of the global data that will be assigned to processes (and stores that detailed information in the resulting `DRI_dist` object). The user will be able to query this low-level information following the execution of this call. Note that the data partitioning performed here guarantees that each global data element is assigned to a process. It is unlikely, but possible that some processes could be assigned NO global data elements as a result of this call.

The layout parameter may have been created by a prior call to the `dri_layout_create` function, or it can be specified with one of the following pre-defined layouts (if the global data described by the `gdo` parameter is either two or three-dimensional):

- `DRI_LAYOUT_PACKED_012`
- `DRI_LAYOUT_PACKED_021`
- `DRI_LAYOUT_PACKED_102`
- `DRI_LAYOUT_PACKED_120`
- `DRI_LAYOUT_PACKED_201`
- `DRI_LAYOUT_PACKED_202`

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 8/17

- DRI_LAYOUT_PACKED_01
- DRI_LAYOUT_PACKED_10

These pre-defined layout objects specify the order in which a multidimensional local buffer is organized in linear memory space. The numeric codes at the end of each predefined symbol serve this purpose. The most contiguously arranged dimension is indicated by the LAST digit of the predefined object name. The least contiguously arranged dimension is the FIRST digit shown in the object name. The term PACKED refers to data that is compactly stored in memory, meaning that the user is requesting no strides between consecutive elements of the local data.

The `group_dims` array specifies a logical process set dimensionality for the process group identified by the "group" parameter. The number of elements in `group_dims` must be equal to the number of dimensions specified for the `gdo` parameter in a prior call to `dri_global_data_create` (i.e., `group_dims` corresponds directly to the `dimsizes` parameter of `dri_global_data_create`). The product of all values in `group_dims` must equal the total number of processes in the process group "group". This parameter gives the caller more explicit control over the global data partitioning process performed by `dri_dist_create`.

Just like the layout parameter, the `group_dims` parameter also allows for a Default setting. A NULL pointer can be passed in place of an integer array to specify that the user has no preference for how the process group is viewed logically.

The logical view of processes specified in `group_dims` is only effective during the execution of the `dri_dist_create()` function. Other calls to `dri_dist_create` involving the same DRI_group object parameter, but a different DRI_global_data may use alternative "views" of the process group to yield a different type of partitioning to be applied to the (also different) data set.

The `distspecs` parameter is an array of DRI_distspec objects, one entry per dimension of data being partitioned. The entries in the array are created prior to this call by using the `dri_distspec_create` function. An exception is when a data dimension will not be partitioned at all - in that case, the corresponding array entry in the `distspecs` parameter here can contain the pre-defined object DRI_DISTSPEC_INDIVISIBLE.

COMMUNICATION BEHAVIOR

At the implementation's discretion, this can be performed either as a collective operation, or as a local operation.

RESTRICTIONS / POLICY

It is possible that an implementation will choose to not communicate collectively among the members of the process group during the execution of this function. It is therefore possible that the constituent processes that make up the group could (erroneously) supply different specifications for the following important parameters: `gdo`, `group_dims`, `distspecs`. In that case the resulting data distribution that is computed and stored in the `DRI_dist` output parameter may be incorrect.

The user must be able to query the low-level partitioning details that result from this call immediately following completion of this call. This is true even if the implementation does not perform any communication between processes in the specified group during the execution of this

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 9/17

function.

```

/***** dri_dist_get_numblocks *****/
dri_dist_get_numblocks - Return the number of locally stored blocks from a
data partitioning

```

SYNOPSIS

```
dri_dist_get_numblocks (dsth, nblocks)
```

PARAMETERS

```

IN:  dsth (DRI_dist) - data distribution object
OUT: nblocks (integer) - number of blocks associated with the low-level
      partitioning referred to by the dsth parameter

```

DESCRIPTION

This function returns the number of blocks assigned as part of a low-level data partitioning (described by the dsth parameter that was created in an earlier dri_dist_create call). For block data partitionings, this function will return a value of 1 in the nblocks output parameter. For block-cyclic partitionings, a value greater than 1 may be returned in the nblocks parameter.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS/POLICY

```

/***** dri_dist_get_blockinfo *****/
dri_dist_get_blockinfo - Get detailed information about a local block of data

```

SYNOPSIS

```
dri_dist_get_blockinfo (dsth, block_num, blockinfo)
```

PARAMETERS

```

IN:  dsth (DRI_dist) - data distribution object
IN:  block_num (integer) - local block number for which information is needed
OUT: blockinfo (DRI_blockinfo) - returned structure containing detailed
      information about the block

```

DESCRIPTION

For a specified low-level data partitioning object (dsth) and local block index (block_num), allocates and returns a structure to the user containing the following:

```

{
  ndims (integer) - number of dimensions in the local data block described
  first_offset (integer) - offset (in elements) from the beginning of the local
      application's memory buffer to the first "owned"
      element of this data block. It therefore in some
      cases does not identify the first data element in
      the block, since the first element in storage could
      be the result of an overlapped data partitioning.
  elem_size (integer) - number of bytes per data element in the local block.
      This can be obtained by querying other objects
      (DRI_global_data and DRI_distspec), but is provided in
      this structure for user convenience.
  dims[ndims] (array of DRI_blockdim structures) - detailed information
      (on a per-dimension basis) about the range of global indices covered

```

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 10/17

```

    by the local block of data referred to by this DRI_blockinfo structure
}

```

The DRI_blockdim structure referred to above is defined as follows:

```

{
  lov (DRI_overlap) - left overlap in this dimension
  rov (DRI_overlap) - right overlap in this dimension
  global_begin_ix (integer) - global index of the first "owned" data element in
    the block in this dimension
  length (integer) - number of "owned" data elements in this dimension
  stride (integer) - number of elements between consecutive data elements
    in the local data buffer in this dimension.  If this value is 1, then
    the data is densely packed, with no spacing between consecutive elements.
}

```

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS/POLICY

```

/***** dri_dist_calc_local_size *****/
dri_dist_calc_local_size - Calculate size of local buffers associated with one
                          side of a data reorganization

```

SYNOPSIS

```
dri_dist_calc_local_size(disth, local_size)
```

PARAMETERS

```

IN:  disth (DRI_dist) - low-level data partitioning object
OUT: local_size (integer) - number of bytes specifying the size of data
    buffers that will be used in future data reorganizations

```

DESCRIPTION

This function tells the caller what size (in bytes) is required of application data buffers that participate in data reorganizations associated with the data partitioning object disth. The returned local_size parameter is calculated based on a combination of user-specified partitioning parameters (at dri_dist_create-time) and implementation-imposed decisions regarding local memory layouts.

The number of bytes returned in the local_size parameter specifies the size of a memory buffer that is large enough to hold all local blocks from a data partitioning. This is particularly relevant for block-cyclic partitionings, in which it is possible and likely that multiple blocks of data are assigned to a single process.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

/***** <META> dri_bufferset_create *****/
dri_bufferset_create - create shared application/library buffers
                      for processing and data reorganization

```

SYNOPSIS

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 11/17

`dri_bufferset_create (nbufs, bufsize bufset)`**PARAMETERS**

IN: nbufs (integer) - number of buffers of size bufsize that will make up the buffer set to be created by this function

IN: bufsize (integer) - size (in bytes) of an individual buffer in the buffer set to be created by this function

OUT: bufset (DRI_bufferset) - buffer set object created

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Creates a buffer set object that will be associated with a later data reorganization (represented by a DRI_channel object). After this call, the user will never directly query or manipulate the DRI_bufferset object created. Once the association of the buffer set is made with a channel object, all access to the buffer set's constituent buffers will be made through the associated channel object. In that interaction, the user will work with individual DRI_buffer_id objects that are given to the application with a call to dri_channel_get and returned to the library (and hence the DRI_bufferset) with a call to dri_channel_put. See the documentation for those functions for additional details on buffer set management.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

/***** <META> dri_channel_create *****/
dri_channel_create - create data reorganization communication channel

```

SYNOPSIS

There are two forms of this call:

```

dri_channel_create_send(name, srcDist, srcBufs, channel);
dri_channel_create_recv(name, destDist, destBufs, channel);

```

PARAMETERS

IN name: (string/integer?) Identifier for the channel

IN srcDist: (DRI_dist) distribution object on the send side

IN destDist: (DRI_dist) distribution object on the receive side

IN srcBufs: (DRI_bufferset) send side data buffers

IN destBufs: (DRI_bufferset) receive side data buffers

OUT channel: (DRI_channel) Data reorganization (channel) object created

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

The send channel object allows the calling process to participate in a data reorganization as a sender. The receive channel object has a similar (obvious) function. To properly set up data reorganizations in which the caller is both a sender and receiver of data, both forms must be called, resulting in two DRI_channel objects.

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 12/17

COMMUNICATION BEHAVIOR

Local. Processes create channel objects independently and in any order.

RESTRICTIONS / POLICY

Buffers are assumed to be big enough to contain all the data transferred. To find the size of the buffer, use the function `dri_dist_calc_local_size`.

Currently, we assume that data reorganizations are either bi-partite (pipeline) or clique-based (Single Program Multiple Data). Intermediate cases, that is, partially overlapping process groups, are disallowed. If any process is both a sender and a receiver, all processes must be both senders and receivers, or an error will result at the time of the subsequent `dri_channel_connect` call.

```
/***** <META> dri_channel_connect *****/
```

SYNOPSIS

`dri_channel_connect(chan)` - Pipeline channel connect

PARAMETERS

INOUT `chan`: (DRI_channel) channel object to be connected

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Enables a given pipeline data reorganization: calculates which processors are Sending to and receiving from which other processors.

COMMUNICATION BEHAVIOR

The connect call is a synchronization point between all processors in the send and receive sides of the data reorganization identified by the `chan` parameter: it is collective and blocking.

RESTRICTIONS / POLICY

Multiple channel objects must be connected in the correct order by the involved parties or deadlock may (will probably) result.

```
/***** <META> dri_channel_connect_sendrecv *****/
```

SYNOPSIS

`dri_channel_connect_sendrecv(c_send, c_rcv)` - Clique channel connect

PARAMETERS

INOUT `c_send`: (DRI_channel) object managing the "send side" of a data reorg
INOUT `c_rcv`: (DRI_channel) object managing the "receive side" of a data reorg

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Enables a given clique data reorganization: calculates which processors are Sending to and receiving from which other processors.

COMMUNICATION BEHAVIOR

The connect call is a synchronization point between all processors in the

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 13/17

send and receive sides of the given data reorganization: it is collective and blocking.

RESTRICTIONS / POLICY

Multiple channel objects must be connected in the correct order by the involved parties or deadlock may (will probably) result.

```
/***** <META> dri_channel_get *****/
/***** <META> dri_channel_put *****/
```

SYNOPSIS

```
dri_channel_get (chan, buf) - Receive data reorg buffer / Get free buffer
dri_channel_put (chan, buf) - Send data reorg buffer / Return used buffer
```

PARAMETERS

```
INOUT chan: (DRI_channel) channel object managing a data reorganization
OUT buf: (DRI_buffer_id) handle to memory buffer
```

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Discussion of dri_channel_get:

If the channel object argument refers to the "receive side" of a data reorganization, this function returns a buffer that represents the received data from a set of sending processes. If the channel object argument refers to the "send side" of a data reorganization, this function returns an available buffer to the application so that it can produce the data that will be sent in a subsequent data reorganization operation.

Discussion of dri_channel_put:

If the channel object argument refers to the "send side" of a data reorganization, this function initiates the communication using the data provided in the input buffer argument. If the channel object refers to the "receive side" of a data reorganization, then this call simply returns the buffer to the DRI library so that it can be filled up with received data in a subsequent data reorganization operation.

General discussion of put and get in context:

In pipeline data reorganizations, incoming buffers are obtained by calling `dri_channel_get` with a "receive side" channel object input argument. If, after processing the received buffer, the program needs to send the data "downstream"

in the pipeline, the same buffer can be used as input to a `dri_channel_put` call, but with a separate channel object (representing the "send side" of a different data reorganization). In cases where the calling program is at the beginning or end of an application pipeline, the buffer may be returned to the buffer set by calling `dri_channel_put` with the same channel object parameter that was used in the earlier `dri_channel_get`.

For clique data parallel applications, there are two channel objects associated

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 14/17

with the same data reorganization (one for the send side, one for the receive side). To execute clique data reorganizations, the program calls `dri_channel_get` with the send-side channel object as input. The returned buffer is filled and a data reorganization is initiated with a call to `dri_channel_put` (passing again as input the send-side channel object and the buffer id). The program then calls `dri_channel_get`, using the second channel object (associated with the receive side of the data reorganization).

COMMUNICATION BEHAVIOR

`dri_channel_get` is a blocking call and does not return until a full buffer of received data is available

`dri_channel_put` is a non-blocking call and returns immediately to the calling application, regardless of whether the associated communication has completed. The channel object will manage the availability of the buffers associated with the data reorganization, protecting the buffer from future application use (via `dri_channel_get`) until the communication has completed and it is safe to reuse the buffer.

RESTRICTIONS / POLICY

It is possible to use the same `DRI_channel` object for two different data reorganizations when using a clique data-parallel design. The receive-side channel object from the first data reorganization executed can also act as the send-side channel object for a second, distinct data reorganization. This is permissible when the data buffer sizes do not change as a result of application processing between the two data reorganizations.

----- SECTION 4: Current API for remaining functions -----

```

/***** dri_global_data_destroy *****/
dri_global_data_create - destroy a global data object

```

SYNOPSIS

```
dri_global_data_destroy(global_data)
```

PARAMETERS

INOUT: `global_data` (`DRI_global_data`) - object that describes the global data

DESCRIPTION

Destroys the global data object referred to by the `global_data` input parameter.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function should only free resources associated with the `global_data` object when necessary. That is, all references to the global data object must be "destroyed" via this call before the actual internal resources used by the global data object are freed and returned to the system.

```

/***** <META> dri_group_destroy *****/
dri_group_destroy - Destroy an object representing a group of processes

```

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 15/17

SYNOPSIS

dri_group_destroy(grp)

PARAMETERS

INOUT: grp (DRI_group) - process group object

META-NOTES:

This is meta at the "object level". That is, we may want to entirely leverage process set constructs from other middlewares.

<UNRESOLVED>

What if we are not using standard middleware, but a process set construct exists? Do we want to leverage those (non-portable) approaches to alleviate the need to do process set management in DRI? If we choose to do this, then this object and its methods become unnecessary?

</UNRESOLVED>

DESCRIPTION

Destroys the process set group object referred to by the grp input parameter.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function should only free resources associated with the group object when necessary. That is, all references to the group object must be "destroyed" via this call before the actual internal resources used by the group object are freed and returned to the system.

```

/***** dri_overlap_destroy *****/
dri_overlap_destroy - Destroy an overlap data partitioning object

```

SYNOPSIS

dri_overlap_destroy(ov)

PARAMETERS

INOUT: ov (DRI_overlap) - overlap object

DESCRIPTION

Destroys the object referred to by the ov parameter

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function should only free resources associated with the overlap object when necessary. That is, all references to the overlap object must be "destroyed" via this call before the actual internal resources used by the overlap object are freed and returned to the system.

```

/***** dri_distspec_destroy *****/
dri_distspec_destroy - Destroy a data distribution specification object

```

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 16/17

SYNOPSIS

```
dri_distspec_destroy(distspec)
```

PARAMETERS

INOUT: distspec (DRI_distspec) - high-level data distribution object

DESCRIPTION

Destroys the object referred to by the distspec parameter. This parameter can refer to either a block or block-cyclic distribution object (created by dri_distspec_block_create or dri_distspec_blockcyclic_create, respectively).

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

This function should only free resources associated with the distspec object when necessary. That is, all references to the distspec object must be "destroyed" via this call before the actual internal resources used by the distspec object are freed and returned to the system.

```

/***** <META> dri_bufferset_destroy *****/
dri_bufferset_destroy - destroy shared application/library buffers
                        for processing and data reorganization

```

SYNOPSIS

```
dri_bufferset_destroy (nbufs, bufsize bufset)
```

PARAMETERS

INOUT: bufset (DRI_bufferset) - buffer set object destroyed

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Destroys the object referred to by the bufset parameter.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function should only free resources associated with the bufferset object when necessary. That is, all references to the bufferset object must be "destroyed" via this call before the actual internal resources used by the bufferset object are freed and returned to the system.

```

/***** <META> dri_channel_destroy *****/
dri_channel_destroy - destroy data reorganization communication channel

```

SYNOPSIS

```
dri_channel_destroy(chan);
```

PARAMETERS

INOUT chan: (DRI_channel) Data reorganization (channel) object destroyed

Mar 07, 00 22:36

pre_March2000_DRI_LIS.txt

Page 17/17

META-NOTES

This is an area where MPI/RT can be leveraged. Needs to be thought out further by the committees (MPI/RT and Data Reorganization)

DESCRIPTION

Destroys the channel referred to by the chan parameter. Frees all internal resources used by the channel, including temporary buffers that may have been created during the earlier dri_channel_connect() call.

COMMUNICATION BEHAVIOR

<UNRESOLVED>

It would be nice to be able to "shut down" a channel gracefully (i.e., in a "collective" fashion). This could be difficult with respect to process synchronization in pipeline application architectures, where many processes participate in two data reorganization channels. This scenario forces a specific order in which channels must be destroyed (or else deadlock could occur). Since this will apparently be pushed to the application level, a proposal would be to make dri_channel_destroy have local communication behavior, and to have applications use other middlewares for the necessary "graceful synchronization".

</UNRESOLVED

RESTRICTIONS / POLICY

This destroy call is different than most others because all internal resources associated with the channel can be freed without checking for other "references". Channels in the Data Reorganization API cannot be referenced more than once.