

Jan 18, 01 21:54

DRI_LIS.txt

Page 1/30

\$Id: DRI_LIS.txt,v 1.12 2001/01/19 02:54:15 kcain Exp \$

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 November 2000 working meeting.

DATE OF THIS DRAFT: 01/18/2001

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 mostly re-iterated from previous versions of this draft.

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 (01/18/2001), 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 01/17/2001 draft to 01/18/2001 draft

1. Made dates referenced in this document consistent (same versions were associated with two different, but very close, dates)
2. Reversed input parameters to DRI_Init C language binding so that argcp goes before argvp
3. Changed misspelled DRI_UNSEDGED_LONG to DRI_UNSIGNED_LONG
4. Changed DRI_Buffer_get_ptr interface to return a pointer address (not a pointer)

***** Changes from 01/14/2001 draft to 01/17/2001 draft

1. Grammar changes only.

***** Changes from 11/26/2000 draft to 01/14/2001 draft

Jan 18, 01 21:54

DRI_LIS.txt

Page 2/30

1. Removed some text in the DRI_Distribution_create documentation that should have been cut in the previous revision
2. Added a routine, DRI_Buffer_get_ptr that takes a DRI_Buffer_Id input argument, and returns a pointer with which the application can directly manipulate the contents of a data buffer.

***** Changes from 9/19/2000 draft to 11/26/2000 draft

1. Changed names of pre-defined DRI_Overlap_type objects:

```
DRI_OVERLAP_TRUNCATE ==> DRI_OVERLAP_PAD_TRUNCATE
DRI_OVERLAP_TOROIDAL ==> DRI_OVERLAP_PAD_TOROIDAL
```

The idea here was that the presence of "PAD" in the pre-defined object names was inconsistent. It was only used in DRI_OVERLAP_PAD_ZEROS and DRI_OVERLAP_PAD_REPLICATED. Observation is that toroidal overlap and truncate cases are also forms of data "padding".

2. Changed the descriptions of DRI_Distribution_get_nublocks and DRI_Distribution_get_blockinfo to describe the linear (1D) indexing approach that they ultimately support. Wording needs to be clearer; this and many other descriptions could benefit from an up-front "terminology" section including diagrams.
3. DRI_Distribution_get_local_count: change output parameter from local_size to local_count, since this quantity now refers to the number of local data elements, not the number of bytes
4. Removed elem_size field from DRI_Blockinfo structure (data type specification is deferred, and is therefore not available when performing data partitioning calls)
5. Fixed typo from prior modification to DRI_Channel_create_ functions (dataspec parameter not listed in the SYNOPSIS section)
6. Changed DRI_Distribution_create from STANDALONE/ADAPTER classification to CORE.
7. DRI_Layout object specification and general reorganization
 - 7a. Previously there was no official home in the document for DRI_Layout specification.
 - 7b. Changes made in other areas of the document (esp. in DRI_Distribution_create) to reflect decisions made at 11/15/2000 working meeting:
 - 2 forms of distribution create function - with and without layouts
 - DRI_Distribution_create remains (with layouts array parameter)
 - DRI_Distribution_create_simple is a new function (without layouts)
 - removed DRI_LAYOUT_NULL pre-defined object as it now seems unnecessary given the 2-function approach described above
8. Changed function naming convention for DRI_Partition object:

Jan 18, 01 21:54

DRI_LIS.txt

Page 3/30

```

DRI_Partition_block_create      ==> DRI_Partition_create_block
DRI_Partition_blockcyclic_create ==> DRI_Partition_create_blockcyclic
DRI_Partition_whole_create      ==> DRI_Partition_create_whole

```

This follows the broader decision at the 11/15/2000 working meeting to name functions according to the following format:

DRI_Object_verb_adverb

9. Added minsz and mod parameters as valid arguments to the DRI_Partition_create_blockcyclic function, according to agreement made during the August 2000 working meeting.

***** Changes from July 2000 draft to September 2000 (POST SEPTEMBER MEETING) ***** draft:

0. Reorganization of API and associated terminology:

```

CORE
Standalone / pure
Standalone / middleware adapter

```

CORE Data Reorg interfaces must appear in all implementations. They are inherently part of Data Reorg and are not likely to appear in other existing standard APIs.

List of CORE objects/interfaces:

For non-CORE interfaces, there is an implicit overlap with existing middleware services. Implementers can fully implement these services according to the Data Reorg "standalone" specification (akin to re-inventing the wheel), or they can make reasonable choices about how to leverage the existing services in so-called "middleware adapters". In this respect, the Data Reorg specification can be thought of as "META" (i.e., not precisely specified, because it could be instantiated in many ways corresponding to the many middlewares that might "co-layer" with Data Reorg).

The Data Reorg committee will define, along with the first CORE and pure standalone interfaces, an MPI middleware adapter.

List of Standalone objects/interfaces:

1. Removed dataspec parameter from DRI_Global_Data_create

We are deferring data type specification until Channel creation time.

Having a data type input parameter to DRI_Global_Data_create would confuse its distinction as a Data Reorg "CORE" interface. Data types can take many forms, since they are implemented in many other middlewares (e.g., MPI, MPI/RT, VS IPL, ...).

2. Added "name" parameter to DRI_Global_Data_create

This string parameter was added as a way to connect off-line configuration file specifications to run-time calls to the Data Reorg library. Some

Jan 18, 01 21:54

DRI_LIS.txt

Page 4/30

architectures use a configuration file approach to specifying communication resources that will be used at run time.

Another benefit of adding this parameter is to facilitate any future debugging or profiling features either included in DRI itself or provided by a third party as a useful complementary capability.

3. Changed `DRI_Distribution_calc_local_size` to `DRI_Distribution_get_local_count`

In addition to the name change, the return value has changed from number of bytes to number of data elements.

The change from returning bytes to returning elements is necessary because the data type of the data being partitioned has not yet been bound in any prior calls (see related change to `DRI_Global_Data_create`).

4. In `DRI_Distribution_create` documentation, clarified a special case that corresponds to "broadcasting" data.

The case is when the `DRI_Partition` object refers to a "whole" partitioning of a data dimension (i.e., indivisible), but the process group logical topology (`group_dims` parameter) specifies more than 1 process. Here, such distributions will effectively replicate the data across all associated processes.

5. In `DRI_Bufferset_system_create`, renamed `dist` parameter to `dsth` to be consistent with other documentation in this document.

6. `DRI_Distribution_create` - wholesale modifications!!

- 6a) Changed how layouts input array works. There are 3 cases supported:

- layouts is NULL (no layout specified at all - default operation request)
- layouts is a mixture of user-instantiated `DRI_Layout` and pre-defined `DRI_LAYOUT_NULL` objects
- layouts consists entirely of user-instantiated `DRI_Layout` objects

See documentation for `DRI_Distribution_create` for details

- 6b) Removed `group` parameter, in favor of user specifying `group_size` and `myrank`

This was motivated by the need to remove so-called "META" arguments out of "CORE" Data Reorg functions like this

7. Moved `DRI_Dataspec` input parameter into `DRI_Channel_create_send` and `DRI_Channel_create_rcv` functions (from `DRI_Global_Data_create`).

See reasoning above in item 1

8. Changed `DRI_Bufferset_system_create` and `DRI_Bufferset_user_create` to take `bufsize` (in bytes) instead of `DRI_Distribution` object parameter.

This, of course, is because of the change described in item 1 (no `dataspec` is bound early on in the API - it is deferred until `DRI_Channel` creation time)

Jan 18, 01 21:54

DRI_LIS.txt

Page 5/30

9. Added built-in datatypes and DRI_Dataspec_get_size

***** Changes from March 2000 draft to July 2000 (POST JUNE DR MEETING) draft:

1. Changed capitalization conventions as decided in prior meetings, but has not yet been implemented. The new convention is:

DRI_ prefix for everything (objects/types, function names, etc.)
 DRI_Data_Type (capitalized first letter of object/type names)
 DRI_Data_Type_method (lowercase method/function names)
 DRI_Method (for standalone functions without associated objects)

2. Changed DRI_Distspec object to DRI_Partition, per group's decision at the June 2000 meeting

An associated change is that the pre-defined object DRI_DISTSPEC_INDIVISIBLE has been changed to DRI_PARTITION_WHOLE

(2 changes - DISTSPEC to PARTITION, and INDIVISIBLE to WHOLE)

3. Changed DRI_Dist object to DRI_Distribution, per group's decision at the June 2000 meeting

4. Changed DRI_Bufferset_create to DRI_Bufferset_system_create

5. Modified DRI_Bufferset_system_create function to take a DRI_Distribution object parameter (as decided in June 2000 meeting)

6. Created DRI_Bufferset_user_create function to import user-allocated memory into a DRI_Bufferset object (instead of calling DRI_Bufferset_system_create to have the library/run time perform the memory allocation).

7. Added DRI_Init and DRI_Finalize functions to the API specification

8. Specified a number of pre-defined "NULL" objects (one for each major data type in the API specification). DRI_Init creates these NULL objects at runtime.

9. Added a new function DRI_Partition_whole_create that allows the user to get a DRI_Partition object that specifies no partitioning at all (per June meeting).

10. Changed DRI_Group_create parameters to require an "original_group" from which a subset process group is to be created

11. REMOVED the following language from parts of this document, based on decisions made in June 2000 meeting.

<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>

***** Changes from December 1999 draft to March 2000 (PRE DR MEETING) draft:

Jan 18, 01 21:54

DRI_LIS.txt

Page 6/30

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

Jan 18, 01 21:54

DRI_LIS.txt

Page 7/30

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 -----

```

/***** <STANDALONE/ADAPTER> DRI_Init *****/
DRI_Init - Initialize the data reorganization run-time environment

```

SYNOPSIS

```

DRI_Init(argvp, argcp) - C language binding
DRI_Init(???)         - other language bindings

```

PARAMETERS

```

INOUT: argvp (pointer to array of strings) - application command line
        arguments
INOUT: argcp (pointer to integer) - address of integer variable that
        stores the number of command line arguments contained in argvp

```

STANDALONE/ADAPTER NOTES

Co-layer (middleware adapter) implementations based on MPI or MPI/RT may be able to accomplish the necessary Data Reorg init actions within their respective Init functions, depending on how they are implemented.

DESCRIPTION

Parses the application command line for implementation-specific data reorganization library options.

Synchronizes with all other data reorganization processes in the environment, and produces the DRI_GROUP_WORLD object that is used to represent all processes in a data reorganization based application.

If they are not already provided at compile-time, this function creates a pre-defined "null" objects:

- DRI_GROUP_NULL
- DRI_GLOBAL_DATA_NULL
- DRI_DATASPEC_NULL
- DRI_OVERLAP_NULL
- DRI_PARTITION_NULL
- DRI_DISTRIBUTION_NULL
- DRI_CHANNEL_NULL
- DRI_BUFFERSET_NULL
- DRI_BUFFER_ID_NULL

Also, if necessary, creates the following pre-defined objects:

- DRI_PARTITION_WHOLE

COMMUNICATION BEHAVIOR

Jan 18, 01 21:54

DRI_LIS.txt

Page 8/30

Collective. Synchronizes with all other data reorganization processes in the environment, and produces the DRI_GROUP_WORLD object that is used to represent all processes in a data reorganization based application.

RESTRICTIONS / POLICY

DRI_Init must be the first data reorganization library function called.

```

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

```

SYNOPSIS

```
DRI_Global_Data_create(ndims, dimsizes[ndims], name, gdo)
```

PARAMETERS

```

IN:  ndims (integer) - number of dimensions in the global data
IN:  dimsizes (integer array) - size of each dimension of the global data
IN:  name (string) - symbolic name of data object represented by gdo
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` and `dimsizes` parameters. Implementations may place an upper limit on the `ndims` parameter. However, all implementations must minimally support `1 <= ndims <= 3`

```

/***** <STANDALONE/ADAPTER> DRI_Group_create *****/
DRI_Group_create - Create an object to represent a group of processes

```

SYNOPSIS

```
DRI_Group_create(original_group, num_ranks, rank_list, new_group)
```

PARAMETERS

```

IN:  original_group (DRI_Group) - group from which a subset will be taken
      to produce the new_group of processes
IN:  num_ranks (integer) - total number of processes in the group to be
      created
IN:  rank_list (array of integer) - list of logical process ranks from
      the original_group that will form the new_group of processes
OUT: new_group (DRI_Group) - process group object

```

Jan 18, 01 21:54

DRI_LIS.txt

Page 9/30

STANDALONE/ADAPTER NOTES

This is meta at the "object level". That is, some implementations may choose to completely leverage constructs from other middleware APIs (e.g., MPI Communicators) as part of a co-layered implementation with Data Reorg. Implementations that take this approach may elect not to implement DRI_Group objects in the standalone format shown here.

DESCRIPTION

Creates an object to represent a group of unique processes in the parallel processing environment. Groups are one-dimensional logical orderings of processes. Each process is assigned an integer rank, numbered between zero and the total number of processes - 1. The original_group parameter must be a valid data reorganization group. The pre-defined DRI_Group object DRI_GROUP_WORLD must be used to create the first subset group of processes.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

/***** <STANDALONE/ADAPTER> 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

STANDALONE/ADAPTER NOTES

See DRI_Group_create

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

/***** <STANDALONE/ADAPTER> DRI_Group_get_size *****/
DRI_Group_get_size - Return the size of the specified group

SYNOPSIS

DRI_Group_get_size(group, size)

Jan 18, 01 21:54

DRI_LIS.txt

Page 10/30

PARAMETERS

IN: group (DRI_Group) - group object
 OUT: size (integer) - size of the specified group

STANDALONE/ADAPTER NOTES

See notes for DRI_Group_create.

DESCRIPTION

Returns the number of participating processes in the given group

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, overlaph)

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_PAD_TRUNCATE
 DRI_OVERLAP_PAD_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 either DRI_Partition_create_block or DRI_Partition_create_blockcyclic as a left or right overlap argument.

NOTE: Just like the DRI_Partition 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 one of the the DRI_Partition_<block|blockcyclic>create functions.

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, but that is assigned to another processor, based on the data partitioning.

Jan 18, 01 21:54

DRI_LIS.txt

Page 11/30

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_PAD_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_PAD_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_Partition_create_block *****/
/***** DRI_Partition_create_blockcyclic *****/
/***** DRI_Partition_create_whole *****/

```

`DRI_Partition_create_block` - Create a block distribution specification
`DRI_Partition_create_blockcyclic` - Create a block cyclic distribution
`DRI_Partition_create_whole` - Create an indivisible (whole) distribution

SYNOPSIS

```

DRI_Partition_create_block(minsz, mod, lov, rov, part)
DRI_Partition_create_blockcyclic(minsz, mod, lov, rov, blksz, part)
DRI_Partition_create_whole(part)

```

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)

Jan 18, 01 21:54

DRI_LIS.txt

Page 12/30

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: part (DRI_Partition) - high-level data distribution object

DESCRIPTION

These functions create a DRI_Partition object that stores information about either a block, blockcyclic, or indivisible (whole) partitioning of global application data. Users must associate a separate DRI_Partition object with each dimension of partitioned global data. The output object, part, 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_Partition 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 particular global data array is stored in the DRI_Distribution object that can be queried for detailed partitioning information following the DRI_Distribution_create operation.

Calling DRI_Partition_create_whole will produce an object equivalent to the pre-defined object DRI_PARTITION_WHOLE. Implementations may in fact return a reference to this pre-defined object as the output of DRI_Partition_create_whole.

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

In block partitionings, the minsz and mod parameters govern the size of the block that is assigned to the calling process. In block cyclic partitionings, the blocks assigned to the calling process may be of different sizes. An example is the case where the blksize parameter does not evenly divide into the number of elements in the data. In such a case, there may be one or more blocks that are smaller than the specified blksize parameter. The minsz and mod parameters constrain the sizes of those exceptional blocks, and must not conflict with the basic blksize parameter specified (i.e., minsz must be \leq blksize).

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 the global data partitioning.

COMMUNICATION BEHAVIOR

Jan 18, 01 21:54

DRI_LIS.txt

Page 13/30

Local

RESTRICTIONS / POLICY

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

COMMUNICATION BEHAVIOR

Local

```

/***** DRI_Layout_create *****/
DRI_Layout_create      - create a local memory layout, general case
DRI_Layout_create_packed - create a local memory layout, packed special case
DRI_Layout_create_uniform - create a local memory layout, uniform special case

```

SYNOPSIS

```

DRI_Layout_create - MORE GENERALIZED MEMORY LAYOUT INTERFACES NOT YET DEFINED
DRI_Layout_create_packed (order)
DRI_Layout_create__uniform (order)

```

PARAMETERS

IN: order (integer) - packing order in linearly-addressed memory

DESCRIPTION

Refer the "BACKGROUND DISCUSSION" section below for background concepts.

DRI_Layout objects associate the following attributes to a single dimension of multi-dimensional data stored in linearly-addressed memory:

- packing order [ranging from 0 .. (number of data dimensions-1)]
- stride (number of data elements between the start of 2 consecutive data elements - stride > 1 implies empty spacing between elements)

Current interfaces (as of the 11/26/2000 draft) do not include support for setting the stride attribute. Ultimately, the stride in the associated data dimension is determined by the specified packing order and the size of the local data determined in the later data distribution call (i.e., DRI_Distribution_create).

If the packing order attribute of a layout is equal to 0, then the associated data dimension will be ordered most contiguously in memory. Higher values of the packing order attribute indicate successively less contiguous ordering of data elements in the associated dimension. The association of a DRI_Layout object with a particular dimension of data is made in a later call to DRI_Distribution_create. See the documentation for that function for the details of how this association is made.

The application programmer can use either manually-created or pre-defined DRI_Layout objects to specify memory layouts. The pre-defined DRI_Layout objects are:

DRI_LAYOUT_PACKED_0 (packing order attribute = 0 most contiguous, stride=1)

Jan 18, 01 21:54

DRI_LIS.txt

Page 14/30

```
DRI_LAYOUT_PACKED_1 (packing order attribute = 1)
DRI_LAYOUT_PACKED_2 (packing order attribute = 2)
DRI_LAYOUT_UNIFORM_0 (packing order attribute = 0, most contiguous)
DRI_LAYOUT_UNIFORM_1 (packing order attribute = 1)
DRI_LAYOUT_UNIFORM_2 (packing order attribute = 2)
```

Implementations must provide pre-defined DRI_Layout objects for at least 3 dimensions of data (i.e, at least up to and including DRI_LAYOUT_PACKED_2 and DRI_LAYOUT_PACKED_UNIFORM_2). However, implementations are encouraged to implement many more pre-defined objects to facilitate programming applications requiring higher dimensional data sets.

The DRI_LAYOUT_PACKED_<N> pre-defined objects are equivalent to those created by manually calling DRI_Layout_create_packed with the order parameter equal to N. The DRI_LAYOUT_UNIFORM_<N> pre-defined objects are the same (to the user) as DRI_LAYOUT_PACKED_<N>. However, when combined with the data distribution (partitioning) step in a later DRI_Distribution_create call, the DRI_LAYOUT_UNIFORM_<N> layout objects request a uniform local memory buffer size in every process dividing the data (even in cases where the number of owned data elements is non-uniform across the process set). This parameter is implementation-motivated, as optimizations in data reorganizations can be achieved on some platforms when uniform memory buffer sizes are involved.

BACKGROUND DISCUSSION

Memory layouts define the order in which multi-dimensional data is arranged in (linearly addressed) memory. Two common examples of memory layout specifications are row-major and column-major orderings of a 2-dimensional matrix. Given a (row,column) multi-dimensional array indexing notation, the row-major memory ordering of an M row by N column matrix follows the pattern shown below:

$$(0,0), (0,1), (0,2), \dots (0,(N-1)), (1,0), (1,1), \dots (1,(N-1)), \dots$$

In the row-major case, consecutive data elements within the same row are stored contiguously. Consecutive elements within the same column are separated by N data elements.

Conversely, the column-major memory ordering of the same matrix follows the pattern shown below:

$$(0,0), (1,0), (2,0), \dots ((M-1),0), (0,1), (1,1), \dots ((M-1),1), \dots$$

In the column-major case, consecutive data elements within the same column are stored contiguously. Consecutive elements within the same row are separated by M data elements.

For higher dimensional data sets, the number of possible orderings of data in linearly-addressed memory increases (e.g., 3 dimensional data can be ordered in 6 possible ways). Memory layout objects generalize the ordering specification for multi-dimensional data stored in local memory. For a given data dimension, an associated DRI_Layout object specifies the relative packing order of that dimension in linearly-addressed memory.

COMMUNICATION BEHAVIOR

Local

Jan 18, 01 21:54

DRI_LIS.txt

Page 15/30

RESTRICTIONS / POLICY

```

/***** DRI_Distribution_create *****/
DRI_Distribution_create - Create a distribution object for a
                        global data object over a group of processes

```

SYNOPSIS

```

DRI_Distribution_create(gdo, group_size, myrank, group_dims, parts, layouts,
                        disth)

```

PARAMETERS

```

IN:  gdo (DRI_Global_Data) - global data object
IN:  group_size (integer) - total number of processes in group dividing data
IN:  myrank (integer) - my logical process rank within the group
IN:  group_dims (array of integer) - logical dimensions of process group
IN:  parts (array of DRI_Partition) - high-level data distribution specs
     (one array entry per gdo dimension)

IN:  layouts (array of DRI_Layout) - memory layout specifications (one per
     data dimension for local data blocks)

OUT: disth (DRI_Distribution) - data distribution object

```

DESCRIPTION

This function calculates a specific partitioning of the global data referenced by the gdo parameter. The result partitioning is a function of:

- the number of processes in the group dividing the data
- the identity of the calling process within the group dividing the data
- the logical multi-dimensional topology of the processes in the group
- the generic partitioning strategy to be applied to each dimension of the data (e.g., undivided/whole, block, or block-cyclic)

This previously specified information, including local memory layout information, is aggregated into the output DRI_Distribution object. The user will be able to query the resulting detailed partitioning information following the execution of this call. The data partitioning performed here guarantees that each global data element is assigned to a process. It is possible in some cases that some processes will be assigned NO global data elements as a result of this call.

The group_size parameter defines the size of the group (number of processes) that will be dividing the data set.

The myrank parameter uniquely identifies the calling process within the group dividing the data. This is specified by the caller so that a distinct portion of the global data set can be assigned to it by this function.

The group_dims array specifies a logical process set dimensionality for the process group identified by the "group" parameter. The number of elements in the group_dims must be the same as the number of dimensions associated with the gdo parameter (i.e., the same as was specified in an earlier call

Jan 18, 01 21:54

DRI_LIS.txt

Page 16/30

to `DRI_Global_Data_create`). This parameter gives the caller more explicit control over the partitioning process performed by this function.

The `group_dims` array can take one of three forms:

1. NULL (no process set dimensionality specified)
User has no preference and the implementation can choose any values that make the product of `group_dims` equal to `group_size`
2. Mixture of zero and nonzero/positive values

A nonzero/positive value in a `group_dims` array position represents a specific process set dimensionality that should be respected for the associated global data dimension.

Zero valued elements effectively represent a "don't care" in that dimension of the data, and the implementation is free to select an appropriate value that satisfies the overall requirement that the product of the `group_dims` values equal the `group_size` parameter.

3. All nonzero/positive values
In this case the user is providing a complete process set dimensionality that should be used in the partitioning process.

The `layouts` parameter specifies, for each dimension of the global data represented by `gdo`, how the locally stored data is to be arranged in linear memory space. The form of the `layouts` array parameter will control the following 2 characteristics of locally stored data:

- packing order of multi-dimensional data
(e.g., which dimension is ordered "fastest" in linear memory)
- striding of local data

See the `DRI_Layout_create` section of this API for information on the construction of the constituent `DRI_Layout` objects in the `layouts` array.

The `layouts` array can consist of a mixture of user-created and pre-defined `DRI_Layout` objects (e.g., `DRI_LAYOUT_PACKED_0`, `DRI_LAYOUT_UNIFORM_0`, etc.):

- for `layouts` array elements containing user-created `DRI_Layout` objects, the order and striding of the corresponding data dimensions are defined by the properties of the supplied `DRI_Layout` objects

<NOTE: As of 11/26/2000 draft, there is no support for user-created `DRI_Layout` objects>

- for a `layouts` array element equal to one of the other pre-defined `DRI_Layout` objects (e.g., `DRI_LAYOUT_PACKED_<N>` and `DRI_LAYOUT_UNIFORM_<N>`), the meaning is the same as that described in the `DRI_Layout` object documentation (`DRI_Layout_create` and related functions)

The `parts` parameter is an array of `DRI_Partition` objects, one entry per dimension of data being partitioned. The entries in the array are created prior to this call by using one of the `DRI_Partition_<block|blockcyclic>_create` functions.

Jan 18, 01 21:54

DRI_LIS.txt

Page 17/30

Special case:

If one of the data dimensions has a partitioning description (parts parameter) of "whole", but the group_dims parameter specifies more than one process "dividing" the data, then the data distribution approach is to provide copies of all global data in that dimension to each of the processes in that dimension. This can effectively be used to implement a broadcast or replication of the affected data.

(see earlier DRI_Partition_create_whole, and DRI_PARTITION_WHOLE descriptions)

COMMUNICATION BEHAVIOR

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

RESTRICTIONS / POLICY

This function may or may not be implemented in a collective fashion. 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, parts. In that case the resulting data distribution that is computed and stored in the DRI_Distribution 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 this call collectively among the affected processes.

```

/***** DRI_Distribution_create_simple *****/
DRI_Distribution_create - Create a distribution object for a
                        global data object over a group of processes
                        (NO LAYOUT SPECIFICATION)

```

SYNOPSIS

```

DRI_Distribution_create_simple (gdo, group_size, myrank, group_dims, parts,
                               disth)

```

PARAMETERS

```

IN:  gdo (DRI_Global_Data) - global data object
IN:  group_size (integer) - total number of processes in group dividing data
IN:  myrank (integer) - my logical process rank within the group
IN:  group_dims (array of integer) - logical dimensions of process group
IN:  parts (array of DRI_Partition) - high-level data distribution specs
     (one array entry per gdo dimension)
OUT: disth (DRI_Distribution) - data distribution object

```

DESCRIPTION

This function is identical to the DRI_Distribution_create function, but eliminates the layouts array input parameter.

The local memory layout will follow the natural ordering of multidimensional data, based on the associated dimension specifications of the gdo parameter. The first specified dimension of the global data (i.e., dimsizes[0] in the earlier call to DRI_Global_Data_create) will be the dimension that is most

Jan 18, 01 21:54

DRI_LIS.txt

Page 18/30

contiguously stored in memory. The last specified dimension will be the least contiguously stored dimension in local memory.

COMMUNICATION BEHAVIOR

Same behavior as DRI_Distribution_create

RESTRICTIONS / POLICY

Same as DRI_Distribution_create

```

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

```

SYNOPSIS

```
DRI_Distribution_get_numblocks (dsth, nblocks)
```

PARAMETERS

IN: dsth (DRI_Distribution) - 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, created in an earlier DRI_Distribution_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.

A typical use of this function is to set up a subsequent loop to process the data in all local data blocks. The processing of a data block will often require a call to DRI_Distribution_get_blockinfo, using an index in the range 0..(nblocks-1) to refer to the specific data block.

The output parameter nblocks does not reflect the multi-dimensional data block indexing that can occur when a block-cyclic partitioning is performed on data with 2 or more dimensions. This interface, along with the complementary DRI_Distribution_get_blockinfo function, is provided for applications that want to use a simpler linear indexing approach. The library implementation is ultimately responsible for determining the data block index assignments for multi-dimensional, block-cyclic partitionings.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS/POLICY

```

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

```

Jan 18, 01 21:54

DRI_LIS.txt

Page 19/30

SYNOPSIS

```
DRI_Distribution_get_blockinfo (disth, block_num, blockinfo)
```

PARAMETERS

```
IN: disth (DRI_Distribution) - 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

Given a low-level data partitioning object (disth) and local, linear block index (block_num), return a structure describing that data block.

This function uses a linear (1 dimensional) index to refer to a local data block. However, in block-cyclic partitionings on multi-dimensional data, a multi-dimensional indexing scheme is possible. This function provides a simpler linear index alternative for those applications that do not require a multi-dimensional indexing approach. The library implementation is responsible for assigning linear index numbers to each of the local data blocks.

The DRI_Blockinfo structure is defined as follows:

```
struct DRI_Blockinfo
{
    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.
    dims[ndims] (array of DRI_Blockdim structures) - detailed information
        (on a per-dimension basis) about the range of global indices covered
        by the local block of data referred to by this DRI_Blockinfo structure
}
```

The DRI_Blockdim structure referred to above is defined as follows:

```
struct DRI_Blockdim
{
    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.

Jan 18, 01 21:54

DRI_LIS.txt

Page 20/30

RESTRICTIONS/POLICY

```

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

```

SYNOPSIS

```
DRI_Distribution_get_local_count(disth, local_count)
```

PARAMETERS

```

IN:  disth (DRI_Distribution) - low-level data partitioning object
OUT: local_count (integer) - number of elements in the data
    buffers associated with this data distribution

```

DESCRIPTION

This function tells the caller the size of data buffers (in elements) associated with the disth data distribution. The returned local_count parameter is calculated based on a combination of

- user-specified partitioning parameters
- user-specified memory layout parameters
- and implementation-imposed local memory layout policies

The number of elements returned in the local_count parameter specifies the size of a memory buffer large enough to hold all local blocks from a data partitioning. This is 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

```

/***** <STANDALONE/ADAPTER> DRI_Bufferset_system_create *****/
DRI_Bufferset_system_create - create shared application/library buffers
                                for processing and data reorganization

```

SYNOPSIS

```
DRI_Bufferset_system_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) - number of bytes needed for each buffer in bufferset
    buffer sizes that will be created in the set
OUT: bufset (DRI_Bufferset) - buffer set object created

```

STANDALONE/ADAPTER NOTES:

Because of similar constructs defined in other APIs (most notably MPI/RT), buffersets can be implemented in a middleware adapter, or in a completely standalone fashion, at the implementation's discretion.

Jan 18, 01 21:54

DRI_LIS.txt

Page 21/30

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 (in a later call do DRI_Channel_create), all access to the buffer set's constituent buffers will be made through that associated channel object. In that interaction, the user will work with individual DRI_Buffer_Id objects that are obtained with a call to DRI_Channel_get and returned to the channel with a call to DRI_Channel_put. See the documentation for the get/put functions for additional details on buffer set management.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

/***** <STANDALONE/ADAPTER> DRI_Bufferset_user_create *****/
DRI_Bufferset_user_create - create shared application/library buffers
                           from user-allocated memory
                           for processing and data reorganization

```

SYNOPSIS

```
DRI_Bufferset_user_create (nbufs, buffer_ptrs[], bufsize, bufset)
```

PARAMETERS

```

IN:  nbufs (integer) - number of buffers that will make up
      the buffer set to be created by this function
IN:  buffer_ptrs (array of pointer) - addresses of user-allocated buffers
IN:  bufsize (integer) - number of bytes needed for each buffer in bufferset
OUT: bufset (DRI_Bufferset) - buffer set object created

```

STANDALONE/ADAPTER NOTES

See notes for DRI_Bufferset_system_create

DESCRIPTION

Creates a buffer set object (to be used in conjunction with an associated DRI_Channel object) from user-allocated memory. Although the application programmer will have access to the addresses of each buffer using this approach, "safe" use of these memory areas must be negotiated by calling DRI_Channel_get and DRI_Channel_put for the associated channel object.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The buffers that are supplied as input parameters here must provide a sufficient amount of memory to insure correct function of

Jan 18, 01 21:54

DRI_LIS.txt

Page 22/30

the associated channel operation (data reorg) to take place. See DRI_Channel_create_send and DRI_Channel_create_recv for guidance on how to appropriately size the buffers.

```

/***** <STANDALONE/ADAPTER> DRI_Buffer_get_ptr *****/
DRI_Buffer_get_ptr - get a pointer for direct application access to
                    a buffer represented by a DRI_Buffer_Id

```

SYNOPSIS

```
DRI_Buffer_get_ptr (buf, buf_ptr)
```

PARAMETERS

```
IN:  buf (DRI_Buffer_Id) - buffer handle
```

```
OUT: buf_ptr (pointer address) - application-accessible pointer to the buffer
```

STANDALONE/ADAPTER NOTES

DESCRIPTION

This routine is typically called following a Channel_get operation to allow the application to directly manipulate the contents of a buffer. A pointer is returned.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The DRI_Buffer_Id parameter passed as input to this routine must be currently under the control of the application, meaning that it must have been returned to the application by a prior Channel_get operation. Control of the buffer must not have been relinquished to the library with a Channel_put operation.

```

/***** <STANDALONE/ADAPTER> DRI Built-In Datatypes *****/

```

Equivalence table between DRI built in datatypes, and other middleware:

DRI	MPI(C/FORTRAN)	VSIPL Scalar Datatype	ANSI C
DRI_Dataspec	MPI_Datatype		
-----	-----	-----	-----
DRI_FLOAT	MPI_FLOAT	vsipl_scalar_f	float
DRI_DOUBLE	MPI_DOUBLE	vsipl_scalar_d	double
DRI_COMPLEX	NA / MPI_COMPLEX	vsipl_cscalar_f	NA
DRI_DOUBLE_COMPLEX	NA / MPI_DOUBLE_COMPLEX	vsipl_cscalar_d	NA
DRI_COMPLEX_SPLIT	NA / NA	vsipl_cscalar_f	NA
DRI_DOUBLE_COMPLEX_SPLIT	NA / NA	vsipl_cscalar_d	NA
DRI_INTEGER	MPI_INTEGER	vsipl_scalar_i	int
DRI_SHORT	MPI_SHORT	vsipl_scalar_si	signed short int
DRI_UNSIGNED_SHORT	MPI_UNSIGNED_SHORT	vsipl_scalar_us	unsigned short int
DRI_LONG	MPI_LONG	vsipl_scalar_li	signed long int
DRI_UNSIGNED_LONG	MPI_UNSIGNED_LONG	vsipl_scalar_ul	unsigned long int

```

/***** <STANDALONE/ADAPTER> DRI_Dataspec_get_size *****/
DRI_Dataspec_get_size - Get the number of bytes needed to store a data type

```

Jan 18, 01 21:54

DRI_LIS.txt

Page 23/30

SYNOPSIS

```
DRI_Dataspec_get_size (dataspec, nbytes)
```

PARAMETERS

```
IN dataspec (DRI_Dataspec) data type descriptor
OUT nbytes: (integer) number of bytes needed to store data of type described
                by the dataspec parameter
```

STANDALONE/ADAPTER NOTES

Datatypes are commonly implemented in other APIs (see table above). This is an area where DRI implementations can take advantage of this existing functionality, which will almost always offer a broader feature set compared to the standalone DRI interfaces for data types.

DESCRIPTION

Returns amount of memory (in bytes) needed to store a single data element of the type specified in the dataspec parameter

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/****** <STANDALONE/ADAPTER> DRI_Channel_create_send *****/
/****** <STANDALONE/ADAPTER> DRI_Channel_create_rcv  *****/
DRI_Channel_create - create data reorganization communication channel
```

SYNOPSIS

There are two forms of this call:

```
DRI_Channel_create_send(name, dataspec, srcDist, srcBufs, channel);
DRI_Channel_create_rcv(name, dataspec, destDist, destBufs, channel);
```

PARAMETERS

```
IN name: (string/integer?) Identifier for the channel
IN dataspec (DRI_Dataspec) type of data stored in associated buffers
IN srcDist: (DRI_Distribution) distribution object on the send side
IN destDist: (DRI_Distribution) 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
```

STANDALONE/ADAPTER NOTES

Channel constructs appear in other APIs (e.g., MPI/RT) and so DRI_Channel can be implemented in a middleware adapter or in a completely standalone fashion, at the discretion of the implementation.

DESCRIPTION

The send channel object allows the calling process to participate in a

Jan 18, 01 21:54

DRI_LIS.txt

Page 24/30

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.

COMMUNICATION BEHAVIOR

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

RESTRICTIONS / POLICY

Buffers supplied here are assumed to be large enough to contain all the data transferred. To find the appropriate size for these buffers, use the functions DRI_Distribution_get_local_count (number of elements) and DRI_Dataspec_get_size (number of bytes/element). Alternatively, the user can have the DRI implementation allocate the associated bufferset using the DRI_Bufferset_system_create call.

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.

On a given "side" of a channel (send or receive), all of the participating processes must provide buffersets that contain the same number of local buffers as every other process. The number of buffers on the send side of a channel `_can_` be different than the number of buffers in the bufferset associated with the receive side of the same channel. The reason for the restriction is to enable high-performance implementations. The middleware will be able to compute in advance:

- the explicit pairings of send/rcv buffers in the data reorganizations to be performed with this channel
- the precise order in which the pairings will occur (if there are multiple buffers on the send and receive sides of the channel)

```

/***** <STANDALONE/ADAPTER> DRI_Channel_connect *****/
DRI_Channel_connect(chan) - Pipeline channel connect

```

SYNOPSIS

```
DRI_Channel_connect(chan)
```

PARAMETERS

INOUT chan: (DRI_Channel) channel object to be connected

STANDALONE/ADAPTER NOTES

See notes for DRI_Channel_create.

DESCRIPTION

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

Jan 18, 01 21:54

DRI_LIS.txt

Page 25/30

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.

/***** <STANDALONE/ADAPTER> DRI_Channel_connect_sendrecv *****/

SYNOPSIS

DRI_Channel_connect_sendrecv(c_send, c_recv) - Clique channel connect

PARAMETERS

INOUT c_send: (DRI_Channel) object managing the "send side" of a data reorg

INOUT c_recv: (DRI_Channel) object managing the "receive side" of a data reorg

STANDALONE/ADAPTER NOTES

See notes for DRI_Channel_create.

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 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.

/***** <STANDALONE/ADAPTER> DRI_Channel_get *****/

/***** <STANDALONE/ADAPTER> 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

STANDALONE/ADAPTER NOTES

Jan 18, 01 21:54

DRI_LIS.txt

Page 26/30

See notes for DRI_Channel_create.

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 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

Jan 18, 01 21:54

DRI_LIS.txt

Page 27/30

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_destroy - 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.

/***** <STANDALONE/ADAPTER> DRI_Group_destroy *****/
DRI_Group_destroy - Destroy an object representing a group of processes

SYNOPSIS

DRI_Group_destroy(grp)

PARAMETERS

INOUT: grp (DRI_Group) - process group object

STANDALONE/ADAPTER NOTES

See notes for DRI_Group_create.

DESCRIPTION

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

COMMUNICATION BEHAVIOR

Jan 18, 01 21:54

DRI_LIS.txt

Page 28/30

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_Partition_destroy *****/
 DRI_Partition_destroy - Destroy a data distribution specification object

SYNOPSIS

DRI_Partition_destroy(part)

PARAMETERS

INOUT: part (DRI_Partition) - high-level data distribution object

DESCRIPTION

Destroys the object referred to by the part parameter. This parameter can refer to either a block or block-cyclic distribution object (created by DRI_Partition_create_block or DRI_Partition_create_blockcyclic, respectively).

COMMUNICATION BEHAVIOR

Local

Jan 18, 01 21:54

DRI_LIS.txt

Page 29/30

RESTRICTIONS / POLICY

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

```

/***** <STANDALONE/ADAPTER> 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

STANDALONE/ADAPTER NOTES

See DRI_Bufferset_create notes.

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.

```

/***** <STANDALONE/ADAPTER> 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

STANDALONE/ADAPTER NOTES

See DRI_Channel_create notes.

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.

Jan 18, 01 21:54

DRI_LIS.txt

Page 30/30

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.

```
/***** <STANDALONE/ADAPTER> DRI_Finalize *****/
DRI_Finalize - Free resources used by the data reorganization
                run-time environment
```

SYNOPSIS

DRI_Finalize()

PARAMETERS

STANDALONE/ADAPTER NOTES

Co-layer (middleware adapter) implementations based on MPI or MPI/RT may be able to accomplish the necessary Data Reorg finalize actions within their respective Finalize functions, depending on how they are implemented.

DESCRIPTION

Frees any internal resources used by the data reorganization implementation.

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY