

Minutes from the June 1999 meeting of the Data Reorganization Forum
Location: Lockheed Martin GES, Moorestown, NJ

June 10 meeting: 1pm – 5pm

June 10 meeting was an executive session, per Tony's request, to organize the discussion on June 11, 1999 and future near-term activities

Attendance:

Tony Skjellum	MPI Software Technology, Inc. / MSU
Ken Cain	MITRE
Karen Lauro	Mercury Computer
James Lebak	MIT Lincoln Laboratory
Jon Greene	Mercury Computer
Tom McClean	Lockheed Martin GES

[EDITORIAL NOTE: There is a lot of detail here from our discussions, but basically the goal was to generate an agenda of discussion topics for the June 11 meeting. That agenda is listed in item #6 in this (June 10 meeting minutes) section of the document. Items 1-5 show how we arrived at that agenda

ANOTHER NOTE: gdo == global data object in the discussion
]

1) Discussion of standalone API versus layered or co-layered DRI with other middleware implementations

- Karen: standalone API to handle every possible parallel communication needed is an attractive idea. Retargeting run-time libraries for parallel software development tools would be vastly easier. No need to choose between alternative parallel communication layers in run-time library implementation.
- Tony: problem with this approach is that you have to re-do all previous message passing efforts again in the DRI effort. DRI will not necessarily ease transition between MPI and MPI/RT based codes. It's more appropriate to first pick the messaging layer appropriate to your problem, and then use the corresponding data reorganization extensions.
- Ken: agreement with Tony – this (MPI or MPI/RT) is the software layer that you “fall back” onto when DRI cannot do the specific operation you want. A goal of this effort, of course, is to provide enough generality to minimize these instances.

2) Discussion of need for high-level data distribution:

- Jon: it's very hard to explain our low-level “part” object to people not involved in this effort
- Jon: current approach is not scalable –low-level part object is too closely tied to a particular global data object (gdo)
- Jon: ease of usability of emerging API's like MPI/RT and DRI needs to be prioritized.
- Ken: it's also an issue of ease-of-implementation issue: Example: how does a DRI implementation decide what data reorganization is being requested by the user based only on these very low level partitioning descriptions?
- Tony: it's possible to describe many useful distribution specifications with formulas
- Formula-based partitioning allows for easy mapping from global to local data points (and vice-versa)

- James/Tony: Block-cyclic data distributions have “closed-form” representations, and encapsulate the vast majority of useful data partitionings
- Example: Block partitioning formula: a “closed-form” based on most-evenly-divided data was hypothesized as a reasonable approach to take (written on whiteboard)
- Jon: Proposed starting point for discussion:
 - Have a data structure that specifies, for each dimension of your distributed data, the distribution “type” (Indivisible, Block, or Block-Cyclic)
 - General discussion and agreement

3) Process set dimensionality discussion:

- Jon: re-emphasized support for providing this in the API
- Jon: this is a tool to facilitate the partitioning process (gives user more control, makes implementation’s job easier)
- Ken: need to have a way to “default” to some reasonable behavior if the user does not want to specify process set dimensionality
- Tony: sufficient high-level “helper” functions can support such things – this is a general theme that can be applied throughout our API design process.

4) Need to address multiple data buffers issue

- Jon: there are 2 ways to think about “multiple buffering”. The forum needs to talk about how to support these things:
 - Overlap communication and computation for a series of equivalent data reorgs (based on a single, complete gdo). Here processing is typically performed on the *entire* local part of the data, and communication takes place (before or) after that. Having multiple user buffers (each with enough space to store the local part of the global data) gives some opportunity to overlap comm and compute
 - Perform data reorg at a “finer grain” (i.e., multiple smaller transfers that in aggregate comprise an entire data reorg for a gdo) – Example here is a transpose in which processing and communication are *interleaved* for a single gdo. Process a row, then ship it out, process the next row, ship the second row, etc.

[EDITORIAL NOTE: This document will refer to the first approach as “multiple buffering”, and the second approach as “flow control”. There is some consensus for this based on the forum’s discussions to date, so let’s adopt this language for clarity in future communications]

- Ken/James: note that the flow control approach has been ruled out in past decisions of the forum, but it’s possible to add this in if somebody submits a concrete proposal to extend the basic API.

5) MPI/RT bindings?

- Jon/Tony: Important to work on this, since the integration of the two APIs was an initial goal
- Need to discuss in tomorrow’s (June 11) meeting with the larger group on how to get started

6) Set preliminary agenda for June 11 meeting

- Discussion of how to get process set (group) dimensionality into the API
- Discuss what types of high-level (scalable) partitioning descriptions we will support
 - API impact
 - How to differentiate software objects for 1) high-level data distribution request with 2) low-level partitioning “reality” achieved when applying the high-level spec to a particular gdo

- Policy impact: now that robust high-level distribution specifications exist, do we still allow programmers to specify their partitionings using the lowest-level interface (based on the April '98 meeting API) ?
- Flow control — REVISITED!
- Refine the "multiple buffer" issue for data transfers
- Review the rest of the API, esp. the post "dist_create" functions (layout, transfer objects)
- Chart the course for MPI/RT binding effort

June 11 meeting: 845am – 4pm

Attendance:

Ken Cain	MITRE
Karen Lauro	Mercury Computer
James Lebak	MIT Lincoln Laboratory
Jon Greene	Mercury Computer
Tom McClean	Lockheed Martin GES
Rick Pancoast	Lockheed Martin GES
Nathan Doss	Lockheed Martin GES
Shane Hebert	MPI Software Technology, Inc. / MSU
Steve Paavola	Sky Computer
Dennis Cattel	SPAWARSYSCEN, S.D.
Randy Judd	SPAWARSYSCEN, S.D.
Arkady Kanevsky	Mercury Computer

James made some opening remarks to start the meeting at 8:45 a.m.

- Mentioned the low-level data partitioning descriptor added to the API (left_overlap, start, end, right_overlap “four-tuple”)
- Mentioned the group’s decision to require separate “transfer_connect” calls – burden is on user to code up the connect’s in the appropriate order to avoid deadlock

Jon recapped the executive session

- Low-level partitioning object too hard to program
- Adding group dimensionality could facilitate the types of partitionings we want to have
- Revisiting multiple-buffering and flow-control issues is needed
- Expressed a desire to accelerate the integration of a DRI and MPI/RT
- Requested additional ideas for the agenda (none heard)

Question: Is there a document available?

- Yes, it consists of the April version of the document plus the C apidraft.c and Language-independent-spec, apidraft.lis (J. Lebak email following the April meeting)
- Group decided to use the standalone apidraft files as the basis of this meeting’s discussions
- Copies of draft document and apidraft files were distributed

Ken: suggested a prioritization of the Agenda: handle items 1 and 2 (group dimensionality and high-level partitioning descriptors) before getting back into multiple buffering and flow-control

James starts off the process set (group) dimensionality discussion:

- “matching” process set dimensions to global data object dimensions makes sense
- Jon: put burden on user to facilitate the partitioning, but user also gets additional control over final partitioning result
- Jon: will need an assortment of helper (high-level) functions to accomplish data partitionings. Should work on the basic high-level descriptions first (i.e., descriptions that require group dimensionality to be specified), and add appropriate convenience functions later
- Dennis/Ken: would like to give the user the ability to not specify dimensionality, and have implementations choose a reasonable default.
- Dennis: Can you change the dimensions of the group dynamically during program execution?
 - James: suggested that providing “views” of a group would handle this issue
 - Alternative: make dimensionality an attribute of a group object
 - Steve: could provide dimensionality parameters to the group_create function

[EDITORIAL NOTE: I don't think we concretely decided the policy on dynamically changing the group dimensionality. Most people did agree that providing multiple "views" of a group is not an attractive option]

High-level partitioning description discussion:

- Jon: user can associate a partitioning "policy" with each dimension of the global data object
 - B = block
 - I = indivisible
 - BC= block-cyclic

James reviews for clarity the current approach to creating a "dist" object in the current API:

- User provides a gdo, a group, and a low-level partitioning descriptor (a "part" object) to the `dri_dist_create_from_parts()` function
- Part object can only be queried after the `dist_create`
- Ken: changing to a high-level data distribution specification requires us to now distinguish between high and low-level partitioning software objects.

James: Introduces a potential problem in combining the ideas of group dimensionality and partitioning

- Example: dividing a 3D gdo (whose partitioning is specified B,I,B over a 2D group (e.g., a 6x5 process set)
- Which gdo dimension gets divided by 6, and which gets divided by 5?
- Jon suggests that group dimensions must correspond 1-to-1 with gdo dimensions
- Shane: force the user to specify a value of "1" in the group dimensionality to correspond to dimensions in the gdo that will not be partitioned (i.e., that are specified as I, "indivisible")
- Shane: new proposal for the block-partition case only:
 - User provides a gdo, and a group with dimensionality, and DOES NOT include a high-level partitioning description of each axis of the gdo. If there is a value of "1" in any of the process set (group) dimensions, then that implies the "indivisible" partitioning policy. A value greater than 1 indicates a block partitioning policy on the corresponding axis of the gdo.
 - Additional proposals are discussed (summary of all approaches discussed follows)

Nathan summarizes approaches to handling high-level partitioning by using group dimensionality

- Option 1: user specifies: gdo + group (with dimensionality) + distribution-spec (B,I,BC symbols) for each corresponding axis of the gdo
- Option 2: User provides partitioning guidance in the distribution-spec:
 - User provides: gdo + group (no dimensionality) + distribution-spec (B,I,BC symbols AND partitioning dimensionality) for each corresponding axis of the gdo
- Option 3: user specifies: gdo + group (with dimensionality)
 - Supports block partitioning and "indivisible" only
 - High-level distribution-spec is derived by examining the gdo and group dimensions (group dimension of 1 indicates an indivisible axis)

[EDITORIAL NOTE: group elects to continue the discussion using Option 2 as the basis]

Discussion on the form of the high-level partitioning information (written very loosely like a C struct or union)

ON A PER-DIMENSION BASIS, USER SPECIFIES THE FOLLOWING INFORMATION:

```
{  
  DRI_dist_type dt; (encodes either block, indivisible, or block-cyclic)  
  Nprocs being applied to the partitioning in this dimension (default = 0, implementation decides)
```

IF dt specifies a block partitioning:

Minimum acceptable size of local partitioning (*default = 0*)
"modulo" requirement (local size should be a multiple of this # of elements) – (*default is 1*)

Left overlap specification (see below for details)
Right overlap specification

IF dt specifies an indivisible axis (no partitioning):

IF dt specifies a block-cyclic partitioning:

Cyclic block size (default size = 1)
Left overlap (yikes!)
Right overlap

}

OVERLAP DETAIL (RECALL, THIS IS SPECIFIED ON A PER-DIMENSION BASIS FOR BLOCK DISTRIBUTIONS)

{

Number of positions (*default = 0*)
Type (either pad, truncate, or toroidal) – (*default = pad*)

Overlap type specifies the policy to implement at the “edges” of the gdo

Pad: pad either with a pad value (e.g., zero), or with replicated data from the last local position

Truncate: do not allocate additional space in the local memory for overlap storage

Toroidal: Store a copy of the “adjacent” processor’s data (adjacency wraps around to the Processor that owns the data on the opposite “edge” of the gdo)

}

Issue: do we still allow the user to specify a partitioning based on a low-level (e.g., four-tuple for the block-partitioning case) description ONLY?

- Group consensus was that we would *disallow* this for now
- Low-level partitioning information object will still need to be maintained, however – processes will need to query this information after the dist_create step

The issue of supporting SPE-like behavior (where a source process doesn't need to specify all the destination processes) came up again: The group decided after some discussion that in our current interface we would need a global commit to implement this. The consensus of the group was that no global commit was desired. We may try to implement this functionality later by requiring a "registry" for desired connections among all processes at system startup.

Tom brings up a deficiency in the current API:

- Early-binding emphasis limits usefulness of this API in applications where the size of the data changes dynamically. An “enumerated list” of all possible sizes is also not available (setting up all possible data reorgs based on known sizes is one approach of combating this problem)
- Ken: this is an important issue to resolve, but should be done via proposals to the email discussion list. It's a huge problem to tackle in one meeting

Discussion on what parts of the API involve “collective” communication among groups of processes

- Is dri_dist_create_... a collective operation?
 - Benefit of implementing this collectively is to check for appropriate “partitioning coverage” of all positions in the gdo
 - Consensus is that the user *must* be able to query for low-level partitioning information (i.e., what the process has) after the dist_create step is completed
 - Ken: propose to make this optionally a collective step (at the implementation’s discretion). Implementations may choose to defer any error detection until the transfer_connect phase

- **Optional collective implementation + hard requirement to allow the user to query low-level partitioning information after dist_create is adopted as the policy of the API**

Discussion of local memory “layout” object

- Possible layout high-level descriptions (policies)
 - Dense/packed
 - Strided layout
 - User specifies explicit strides to specify the distance between successive elements in each dimension of the local data buffer
- Steve: Alignment parameter
 - Example: make the start of each row in your local memory be aligned on some specified boundary
 - Should we only allow this requirement for 1 dimension of the local data, or all?
 - This allows the user to perform local processing optimizations, and to be able to explain the memory layout in those terms to the data reorg implementation. If the user has no special alignment requirements, then that case can also be accommodated
 - Can also support alignment of the beginning of the local data buffer only (as opposed to a series of alignments within the local buffer, as described above)
- Jon: “Equal stride”
 - Example: 34 column matrix gdo block-partitioned along the columns dimension, with 12 cols to process p0, 12 to p1, and 10 to p2
 - Equal stride layout policy would make stride between successive row elements = 12 on all processors (assuming a row-major layout of the local data)
- Forms of the layout structures
 - Dense/packed (order permutation array: e.g., [0,1] for row-major matrix, and [1, 0] for column major matrix)
 - Strided layout (array of explicit strides for each dimension specified by user)
 - Group Agreement: details of layout structure can be hammered out later

Dennis: could use stride specification in memory layout to facilitate flow-control

- Example: receiving data that constitutes columns of a larger (row-major) matrix
- Group decided that we should revisit this particular use of memory layouts for the flow-control discussion

Revisit where the layout object should be specified in the chain of calls to establish a data transfer:

- Approach to date has been to specify the layout as part of the transfer_create (after the data distribution has been specified in the dist_create)
- Group agrees to make the user supply the memory layout as part of the dist_create
 - BENEFIT: implementation can come back after the dist_create and tell the user a single value that indicates the number of bytes in the local data buffer. The user then takes this value and allocates buffers of the appropriate size before creating the transfer object.

[EDITORIAL NOTE: Now that we have required the memory layout to be described before dist_create time, the user cannot specify explicit strides (because the user doesn't know what the exact size of the local data buffer will be)]

Multiple buffering discussion

- Send and recv sides of a transfer need to be able to acquire and release a buffer at a time (from the multiple buffer pool created by the user and supplied at transfer_create-time)
- There is an issue of what to call the operations:
 - Do (our original “verb” to describe the invocation of a transfer)

- Acquire/release?
- Insert/remove?
- Put/get?
- Jon: the API needs to support a non-blocking transfer operation. The current “do” syntax does not provide this
 - The “buffer get” (appropriate verb TBD) should return an available buffer, or a status to indicate that all buffers are currently in use
- Steve: consider the idea of acquiring a buffer from a recv transfer handle, processing it in-place, and then inserting it into a send transfer handle
 - Starting to look like MPI/RT buffer iterators
-

At the end of the meeting, we talked about overlapping buffers in memory when two sides of a transfer are the same process. The most prominent subset of this case is the in-place reuse of buffers in clique data reorganizations. The group decided to disallow this, with the idea that we could relax this restriction later.

Nathan comment on collective nature of transfer_connect call:

- There is a potential problem if you are building data reorg using MPI/RT.
- Where can collective communication take place? It seems logical that transfer_connect calls would occur before the call to MPIRT_Commit. But no communication can take place until MPIRT_Commit is called!
- Ken suggestion: relax the rules in the data reorganization API that allows MPI/RT based implementations to defer execution of the transfer_connect calls until the MPIRT_Commit call executes. The MPIRT implementation would presumably be a co-layer to the data reorganization layer, so the connect calls could be accumulated in a queue before commit-time.