

DARPA Data Re-organization Meeting
December 1, 1998
Bedford, MA

Attendance

Darwin Ammala, MPI Software Technology Inc.	dammala@mpi-softtech.com
Steve Paavola, Sky Computers	paavola@sky.com
Shane Hebert, Mississippi State University	shane@erc.msstate.edu
Richard Games, The Mitre Corp.	rg@mitre.org
Ken Cain, The Mitre Corp.	kcain@mitre.org
Arkady Kanevsky, The Mitre Corp. (Host)	arkady@mitre.org
Dennis Cattel, SPAWAR	dennis@spawar.navy.mil
Nathan Doss, Lockheed-Martin GES	nathan.e.doss@lmco.com
James Lebak, MIT/Lincoln Laboratory	jlebak@ll.mit.edu
Jon Greene, Mercury Computer	greene@mc.com

Review of the Document, minutes and e-mail summary

General Opening Discussion:

Shane from MSU attended for the first time, and asked about the relationship between the VSIP standard and the intention of Data Re-org (DR). VSIP is targeted toward single processors, while DR is concerned with parallel. Parallel VSIP can use DR underneath.

Richard asked why Block Cyclic partitioning was being addressed. Some customers require it. It should not precluded - added for known future growth.

Ken asked if early implementations could ignore Block Cyclic partitioning? Early on this would be possible, but it will never be prohibited from being supported.

Mention was made that the Image Processing community should have more representation in this meeting - Paul Harmon was suggested as one who should be invited.

Partitions

CFAR, Mitre - square partitions – tiles were briefly mentioned, the note taker has little other context for this.

Load balancing handles distribution and sizing?

Process topologies will not be apparent in the API. No automatic mapping will be prescribed. The user will be responsible for partitioning, as each problem is different.

Arkady asked if we could have both - either explicitly by user, or let the standard choose from a default.

Ken is in favor of the choice - codes have built in assumptions, it would be easy to get the implementation up. E.g., PAS can't give you 6,6,2,2,4,4 etc. it only supports one size division for the partition.

The API should allow user flexibility yet provide usual default partitions.

Richard mentioned that rows are atomic. Say the 1st 8 rows go, there is a boundary at row 9. It's in a new group. We should be able to respect 'imaginary' boundaries of any size. E.g., a corner turn on only the 1st N rows.

Steve indicated that there are many ways to partition, the user should be able to have an option to configure the most optimal way for his/her problem, and some standard - simple ones should be there, and an acceptable default.

From this we concluded that we need a partitioning descriptor.

Minutes:

Document: Meta vs. True - Real time is an issue. We have real-time, CFD doesn't. Real-time is addressing performance, but this API can be layered on RT. We should not require real-time. We should consider deterministic behavior, no fluctuations. We should not preclude real-time, We need early binding, and should stay away from quality of service issues, which are best left for real-time API's such as MPI/RT. .

Section 3.2 *Partitioning Descriptor:*

block cyclic not represented.

There are a number of ways to carve up data.

Steve has an example - a piece of data I have is a cube, we need an iterator to determine how we get the next cube. To do this you would need a collective operation; one big distributed step. Iterate through the steps.

Dennis mentioned that you could pipeline a single frame. In SPE we take N pulse rows, treat as a frame, and pass an end of stream mark, we can't afford to wait for all rows, so we go after the 1st one is through., This sounds like DR and Control Flow, much broader issue. May be out of scope. Users with this problem should look for tools to help with flow. Several tools - SPE, Mesa, SMS, Gedae, RT-Express and other shells do this.

Ken pointed out that In section 4.1 we talk about re-org, and overlap.

Shane asked as an example - in data movement we have descrip1, descrip2, transform. Where are descriptions made? Global data, which piece do I have, what will it look like in the next step? e.g., an N x M matrix, N processors each gets 1 row. I have a row, and want a column mapped to process the next. DR specifies maps from row to column.

Memory allocation: we should consider the fastest changing dimension.

Advice to HPC Community

Shared memory NUMA, data is local, but they still must worry about corner turns.

Shared memory is not optimized for the corner turn. Writing algorithms to do corner turns et al. in shard memory is difficult. Vendors will have libraries for doing this. We need input from both shared memory and distributed memory communities.

Shared memory is not a simplification. We must think of how data is organized. High performance shared memory will require many of distributed memory concepts.

Initial Problem Set addressed by API

Provide the tools and capabilities to execute a 2 dimensional corner turn. This is a fundamental operation for many communities in the high performance computing sector.

Rough design of API

Define transforms with descriptor - modify descriptor over time.

Take Ken's STAP example of partitioning object, and generalize it.

We have D, d1-d2, l1-l2.

I have a piece of a virtual object.

I do processing

I now 'put' into another virtual object (data may not necessarily move).

Put into dri, and get from dri are needed.

We need to describe 'big picture'. See 3.2.

We need to describe my piece in physical memory. See 3.3.

We need to describe new context. See 3.2.

Specific calls to propose

Specific contents of descriptors.

Start with 2-d descriptors

Arkady asked about descriptor contents, how much goes into each? What is the functionality? Where are the descriptors used?

Sane suggested that we describe pieces of virtual object

Describe the physical layout, whether things block or not?

A suggestion is that we should start with blocking calls.

All must call them, but not synchronously.

Collective, but not synchronous.

Nathan advised that we try to make it simple - send/recv. Send/Recv can be viewed many ways: MPI, MPI/RT (set up channel ends, kick off).

Shane pointed out that this is analogous to intra, inter communicators in MPI.

Nathan proposed a sendrecv call which does both.

Checkin,checkout (single sided), do (both).

Steve suggested get/put distinguish. More than one can get it.

Jon mentioned strip mining of buffer queues. May not have room for all - but get a piece at a time. Should we overlay a buffer queue?

Shane pointed out that with big array pieces, allocation must be collective. If one process needs a plane of data all processes must participate. Everyone must call the function, synchronization is not necessary. The transpose operation requires a barrier. We must assume that the call is collective.

The producer can't get out early since they will have to wait until their data is consumed - since consumers may not have spare memory.

Consider:

The way in which the programmer uses the API,

The way that the API is built.

Jon asked: What does 'getting out' mean? You can't program to get out early, it is implementation dependant.

Order is important - avoid deadlock. Even in call sequence.

We have 2 calls begin,end, (put,get).

Nathan: we need good names to symbolize ideas.

3 calls:

do we need to pre-calculate everything up front? Yes

Ken summarized the cases to cover:

Focus on a single problem choose a problem and examine it for the following:

Objects

Partitioning

Layout

Real Memory

Data Descriptors

Transfer Object - opaque

Dimensions of local data change, how? Must dimensions remain fixed.

Semantics of call

Virtual layout:

Issues:

Data Issues

Overall description for a re-org (D)

DR_shape - number of dimensions, size of each dimension, datatype of each element - type,)

Partitioning Objects (P1,P2,

Overlapping, imaging and STAP want it. It's important, doesn't have to be clever.

Can details be hidden? We only need to define our piece. By knowing what others do, implies overlapping. Systematic derivation:

There is public info, dimensions, starting etc.,

This object is a final descriptor of what you HAVE, not what you want.

Proposal, N dimensional pairs.

If a row appears in multiple processes, the destination needs to know which one to get it from.

Arkady: Issues:

1st - do we want to include a stride in the specification - no. - we assumed contiguous data.

2nd - what about holes? Contiguous assumed, this could be an error - as in some data may not be provided.

3rd - James, cyclic calls excluded e.g., 16 columns and 4 processors.

Determine what part you own, beyond that, is there overlap?

Start, end, internal start, internal end

Either system describes, or we define.

DRI_part_block I nuber of dimensions, 4 tuples per dimension,) matching order in DR_SHAPE

Have a mask to represent what is and isn't partitioned. We need to keep the partition type.

0 bit not part, 1 - is partitioned.

Process Sets (M1,M2)

Local Memory layout objects (L1,L2)

Real Memory ; data stored locally

Transfer Objects (T); temporary buffers for use during DR operations

Group to group transfers - or group to self.

Buffer Reuse

Global Knowledge of local partitioning; initially known and computed by DR)

(Sender shouldn't have to know about receiver).

Functionality Issues

"Do: call vs. put/get = we will have all 3. Put(d,

Do is wise to setup, recognizes what is requested, where it's at.)

Can some be do'ing and some getting?

Nathan's API send,recv,sendrecv,do

Groups of Groups ? vs. looping through, chaining etc.

Case of 5 outputs from one group.

We need a proposal to build groups, process sets, and need to subset processes.

Assumption of 1 program, and a mechanism for splitting.

Jon: Postulate the existence of an object - DR layered on top. Leave space in functions, need datatype

DR_group_T

DRI_group_create (group, color, size?, newgroup)

Jon: We need a concept of a "network handle" included in these calls.

Take a list of processes to create a group from it.

DRI_init ()

DRI_creat,query,destroy

DO(transferobject)

Local Partitioning changes dimensions

Pipeline vs. Cliques

Synchronization semantics of DR operations

Local Packing / unpacking around transfer operations

WE NEED A NAMING CONVENTION CHAPTER

Objects:

p.26 DR_shape

End of Regularly scheduled meeting

The regularly scheduled portion of the meeting ended at this point. During the meeting, the group realized that one half-day meetings were inadequate to make good progress. It was agreed that from the next meeting onward, the meetings would be scheduled for a full day. This meeting session was quite productive in that it set forth the framework to begin designing the API. Several members were able to remain and continue the discussion and design issues of the API. The following is a synopsis from Ken Cain, who along with Dennis Cottel, Nathan Doss, Shane Hebert et al. set about to design the API.

***** Preliminary: Recall the list of "issues" that were written on the board.

In the extra session, we covered the following topics:

0) Partitioning objects

1) More on partitioning - API thoughts for TILE partitioning.

2) Synchronization semantics of a data reorg. operation

3) How to handle local pack/unpack around transfer operation in the API

4) Miscellaneous

- group membership rules for data reorganizations.

0) Partitioning objects

Here, we were trying to identify what type of information would need to be stored in a partitioning object. In the course of this discussion, we realized that our current partitioning descriptions were not adequate to indicate the range of possible partitionings that one might want to describe. For example, what does it really mean to partition a matrix using PART_XY? Here's what it could mean:

a) Parallel processing can take place at the "point" granularity (i.e., we could deal out individual matrix elements in any fashion to the processors)

OR b) We want to split the matrix so that elements that are "contiguous" in the global matrix are grouped together when the matrix is partitioned among the processors. Imagine the following scenario that fits into this case: We want to split a 4x8 matrix (which is declared to be row-major) among 3 processors, enforcing this "contiguous" partitioning constraint. A straight-forward partitioning might do the following calculation:

$$32/3 = 10 \text{ remainder } 2$$

each processor will get either 10 or 11 elements.

P0 might get the *first* 11 elements from the global matrix
P1 might get the *next* 11 elements
and finally... P2 might get the *last* 10 elements

Note that this policy will end up "cutting" rows 1 & 2 and will leave rows 0 and 3 intact (i.e., stored on one processor only)

OR c) We want to split the matrix so that we end up with "tiles" being assigned to processors. That is, we want to enforce the constraint that each processor gets a rectangular region of the matrix.

So... we decided that we should have descriptions for each type of partitioning:

- PART_XY should describe the most general partitioning (this allows you to say that dimensions X and Y are completely independent with respect to parallel processing)

- CONTIG_XY for the second case

- TILE_XY for the third case

We also decided that TILE_XY is a reasonable thing to support in an initial API, with follow-on efforts (or ambitious implementations?) identifying a way to handle the two more general cases.

So, in order to support TILE_XY type partitionings, what would a partitioning object (minimally) need to store (and allow the user to see and possibly specify)?

- global x start position
- global x length
- global y start
- global y length

We called this a FOURTUPLE during our discussion, but this applies only to 2D data. One might internally represent this structure for the general case like this:

```
int x_start[MAX_DIMS]
int x_len[MAX_DIMS]
int y_start[MAX_DIMS]
int y_len[MAX_DIMS]
```

where MAX_DIMS is a hard limit on the number of supported dimensions (see topic #1 just below for a discussion on that)

Let's just call it a TUPLE for now.

1) API issues for TILE partitionings

You could do it in 2 ways:

- specify an explicit partitioning yourself (each processor could compute and specify its own TUPLE indicating the region of the global object that it will "own")
- Call the API with an indication of which dimensions should be TILE partitioned, and accept the default partitioning

Preliminary API idea for strategy a:

```
DRI_PART_BLOCK_CREATE (int num_dim, TUPLE *tuples, DRI_PART *part);
```

(DRI_PART is the datatype of the partitioning object)

Preliminary API idea for strategy b:

- A bitmask is used to indicate which dimensions of the global data are to be TILE partitioned (supported # of dimensions is therefore limited to the number of bits in the mask). Each bit position corresponds to a dimension of the global data - a 1 in a bit position indicates that the corresponding dimension should be partitioned.

```
DRI_PART_TILE_CREATE (DD global_picture, process_group grp,  
int bitmask, int overlap  
DRI_PART *part);
```

where:

"global_picture" is the global data descriptor

"grp" is the process group that will divide the data

"overlap" specifies the amount of overlap applied to every dimension (is this bogus to require the same overlap in every dimension?)

"part" is the returned partitioning object

2) Synchronization semantics of a data reorg operation

The group basically decided that a DR operation should return as soon as all of its "local work" is done. So, if a process is only a "sender" in a reorg, then it can return as soon as it is again "safe" for the application to reuse the send buffer. There is no need in this situation for the senders to wait for the entire data reorg operation to complete before proceeding.

This topic centers around whether operations should "block" or not. So far, we have discussed blocking send and receive semantics.

3) How to handle local pack/unpack around transfer operation in the API

A reorg operation can be thought of as 3 separate steps:

- Local data packing (may require some reordering on the source side)
- Data transfer
- Local data unpacking (reordering of received data from the transfer)

The DR draft document talks about separating these three operations as a way to possibly overlap communication and computation (where DMA engines are present)

Basically, on this topic we agreed that a multiple buffering scheme could be used internal to the library in order to overlap some of the pack/unpack operations with the data transfer. Needs to be thought out further, though, because we clearly want to take advantage of this in an implementation.

4) Miscellaneous

- group membership rules for data reorgs

Question: what is legal?

Example:

P0 is the sole member of the source group for the reorg

P0, P1, P2, and P3 are members of the destination group

(here, P0 is both a sender and receiver of data, but the remaining processes are just receivers)

Cases:

src pset exactly equals dest pset (clique data reorg)

src pset disjoint from dest pset (bipartite, or "pipeline")

src pset and dest pset partially intersect (shown in the example above)

The consensus was that each of these cases should be allowed.

Next Meeting

Early February 2nd prior to the MPI/RT Forum meeting at the Island Palms Resort Hotel, San Diego.
Dennis Cottel will host.