Data Reorganization Interface (DRI): Past Perspective, Future Vision, New Results, and Increased Availability

Kenneth C. Cain Jr.
Mercury Computer Systems, Inc.
kcain@mc.com

Introduction

This presentation will offer – from a standards developer and software/systems vendor perspective – an overall vision and direction for the standard Data Reorganization Interface (DRI)[1] that is focused on increased adoption for DRI and its overall methodology. New results obtained with the Mercury DRI/PAS (Parallel Acceleration System) middleware that directly contributes to this direction will be reported. The results include new free-of-charge DRI/PAS evaluation software for Linux clusters, performance measurements and comparisons obtained with DRI/PAS over InfiniBand, and interfaces facilitating application development with both DRI and the standard Message Passing Interface (MPI).

Figure 1 presents many of the DRI data distribution and reorganization features in the context of radar signal processing. A computational stage (performed in a data-parallel fashion by a set of source processes) is followed by a data reorganization and a second computational stage (performed by a set of destination processes). The source and destination sets may be disjoint (pipeline) or exactly the same (clique). DRI facilitates application development by automating both the data distributions in each stage and the parallel communications for the data transformation between stages.

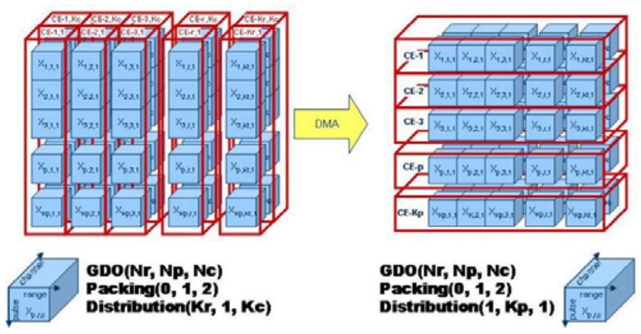

Figure 1. DRI applied to radar signal processing.

The unpartitioned radar data cube (N_r range samples × N_p pulses × N_c channels) is represented in DRI as the Global Data Object (GDO) whose parameters must be the same in both stages. The source processes (consisting of $K_r \times 1 \times K_c$ members) request that the range samples and channels dimensions be partitioned into K_r and K_c blocks respectively. The destination processes (consisting of $1 \times K_p \times 1$ members) request that the pulses dimension be partitioned into K_p blocks. DRI accepts these high-level parameters, then calculates and maintains the mapping of each data element onto a specific processor in the multicomputer. DRI also provides a multi-point communication channel between the source and destination processes. DRI channels calculate the set of point-to-point communications required to complete the global data transformation. The application can perform the entire transformation by issuing a single DRI function call, offering a very high degree of programming productivity. Multi-buffering is also provided to simultaneously enable incoming delivery of the next dataset, processing of the current dataset, and outgoing delivery of the previous dataset.

DRI Direction and Vision

Looking back, DRI was ratified in 2002 by a group consisting of embedded system vendors, software vendors, and government research and acquisition command organizations. It has been the topic of several HPEC workshop presentations[2], including case studies in which DRI was applied to realistic military signal processing applications[3,4]. Its API and extensions have been incorporated into the Mercury Computer Systems' PAS middleware product. Within PAS, its DRI foundation has been extended to support system integration requirements such as communication between embedded compute nodes and run-time hosts, I/O devices, and field programmable gate array (FPGA) compute nodes. Because of these very practical results, the work of the Data Reorganization Forum can be considered a successful development undertaken by the HPEC community during the first 10 years of the workshop.

Currently, additional developments are needed to achieve wider adoption of DRI:

- A freely available reference implementation to foster DRI-based prototyping of applications and library implementations with low or no cost.
- Inter-operation with or inclusion into existing MPI implementations. This will allow applications to incrementally incorporate calls to DRI-based middleware while simultaneously preserving other portions of the application already written in MPI. The recommendation is that HPEC applications can greatly benefit from a programming environment that simultaneously provides MPI and DRI.
- Possibly incorporate DRI like interfaces in MPI using the MPI namespace.

Looking forward, the concepts embodied in DRI will be refined and applied in HPEC systems into the future, continuing to fill its (conventional) role between multiple separate processors, but also now enabling dataflow within individual (multi-core) processors. For example, DRI data distribution and multi-buffered channel concepts are being refined and implemented in Mercury's programming environment for the Cell Broadband Engine™ processor. DRI-based middleware may also serve as a high-productivity, high-performance transport to be used transparently within very high-level programming environments such as parallel VSIPL++[5], and graphical development environments.

Results

Mercury has ported its DRI/PAS middleware to clusters of Linux-based x86, x86_64, and ppc64 blades interconnected with Ethernet and InfiniBand switched fabrics. DRI/PAS already targets Mercury's embedded multicomputers of Linux- and MCexec-based PowerPC processors interconnected with RACE++, and RapidIO switched fabrics. The port from embedded to blade systems requires a new transport layer implementation inside DRI/PAS that provides these services:

- Remote Direct Memory Access (RDMA) interface
- Multicomputer services (named memory endpoints to use with RDMA, semaphores for multi-process level mutual exclusion and interrupt-driven synchronization)

For DRI/PAS clusters, there are two transport implementations: one using only TCP/IP BSD sockets for interprocess communication, and one that uses both sockets and RDMA over InfiniBand (RDMA via the user-level Direct Access Programming Library standard, uDAPL[6,7]).

Performance measurements obtained from running DRI/PAS over uDAPL over InfiniBand will be presented, including relative comparisons to MPI performance over InfiniBand verbs (MVAPICH-gen2 package[8,9]). DRI/PAS will demonstrate the ability to achieve comparable point-to-point, single-buffered bandwidth to MVAPICH-gen2 over IB verbs, validating its transport design and implementation. Additional multi-point dataflow patterns will be measured and reported, such as scatter, gather, and many-to-many. A performance comparison to MVAPICH-gen2 over uDAPL will also be made, providing a more direct comparison due to the common use of uDAPL.

This presentation will announce the availability of the TCP/IP-based DRI/PAS software for the HPEC community to use, free of charge, to evaluate DRI and related software interfaces. A report of user experiences with this software will be provided based on an advance deployment of this software on the HPEC-SI program's blade cluster at Georgia Tech Research Institute.

New interfaces are available in DRI/PAS that facilitate a dual use of MPI and DRI/PAS in the same application. A set of basic MPI integration definitions will be suggested. The specific interfaces available in DRI/PAS to meet these requirements will be detailed. Test applications that exercise these interfaces will be described, including successful results achieved running those tests on multiple systems (Mercury PowerStream 7000, Linux ppc64 cluster with TCP/IP transport, Linux ppc64 cluster with RDMA transport) and using multiple MPI implementations (MVAPICH-gen2 over IB verbs, Verari MPI/Pro for RapidIO, and MPICH).

References

[1] Data Reorganization Forum, Document for the *Data Reorganization Interface (DRI-1.0) Standard*, http://www.data-re.org/.

[2] K. Cain and A. Skjellum, *Data Reorganization Interface*, High Performance Embedded Computing Workshop, 2002.

[3] J. Rudin and K. Cain and L. Cico and E. Luce and T. O'Connor and M. Prelle, *The Scalable Software Interconnect for Distributed Radar Signal Processing*, High Performance Embedded Computing Workshop, 2005.

[4] B. Sroka, *HPEC-SI Demonstration: Common Imagery Processor – APG-73 Image Formation*, High Performance Embedded Computing Workshop, 2002.

[5] M. Mitchell and J. Oldham, *VSIPL++: Intuitive Programming using C++ Templates*, High Performance Embedded Computing Workshop, 2002.

[6] Direct Access Transport (DAT) Collaborative, *uDAPL: User Direct Access Programming Library*, http://www.datcollaborative.org/udapl.html#spec

[7] Open Fabrics Alliance, http://www.openib.org/

[8] Ohio State University Network-Based Computing Laboratory. *MPI for InfiniBand Project* http://nowlab.cse.ohio-state.edu/projects/mpi-iba/

[9] J. Liu and J. Wu and D. Panda, *High Performance RDMA-Based MPI Implementation over InfiniBand*, Proceedings of 17th Annual ACM International Conference on Supercomputing. San Francisco Bay Area. June, 2003.