Designing Efficient Small Message Transfer Mechanism for Inter-node MPI Communication on InfiniBand GPU Clusters

Rong Shi[1], Sreeram Potluri[1], Khaled Hamidouche[1], Mingzhe Li[1], Davide Rossetti[2], and Dhabaleswar K. (DK) Panda[1]

[1] *Department of Computer Science and Engineering, The Ohio State University*
{shir, potluri, hamidouc, limin, panda}@cse.ohio-state.edu

[2] *NVIDIA Corporation*
drossetti@nvidia.com

***Abstract*—Increasing number of MPI applications are being ported to take advantage of the compute power offered by GPUs. Data movement on GPU clusters continues to be the major bottleneck that keeps scientific applications from fully harnessing the potential of GPUs. Earlier, GPU-GPU inter-node communication has to move data from GPU memory to host memory before sending it over the network. MPI libraries like MVAPICH2 have provided solutions to alleviate this bottleneck using host-based pipelining techniques. Besides that, the newly introduced GPUDirect RDMA (GDR) is a promising solution to further solve this data movement bottleneck. However, existing design in MPI libraries applies the rendezvous protocol for all message sizes, which incurs considerable overhead for small message communications due to extra synchronization message exchange.**

In this paper, we propose new techniques to optimize inter-node GPU-to-GPU communications for small message sizes. Our designs to support the eager protocol include efficient support at both sender and receiver sides. Furthermore, we propose a new data path to provide fast copies between host and GPUs memories. To the best of our knowledge, this is the first study to propose efficient designs for GPU communication for small message sizes, using eager protocol. Our experimental results demonstrate up to 45% and 63% reduction in latency for GPU-to-GPU and CPU-to-GPU point-to-piont communications, respectively. These designs boost the uni-directional bandwidth by 7.3x and 1.7x, respectively. We also evaluate our proposed design with two end-applications: GPULBM and HOOMD-blue. Performance numbers on Kepler GPUs shows that, compared to the best existing GDR design, our proposed designs achieve up to 23.4% latency reduction for GPULBM and 20.1% increase in average TPS for HOOMD-blue, respectively.

***Keywords*-MPI, CUDA, InfiniBand, GPU Direct RDMA**

I. Introduction

It is becoming increasingly common for High Performance Computing clusters to use accelerators, such as GPUs, to push their peak compute capabilities. This trend is evident in the TOP500 list released in November 2013, where 53 systems make use of accelerator technology [1]. NVIDIA has been the leader in GPU technology and its latest generation of GPU architecture, Kepler, provides innovative features (e.g. GPUDirect RDMA) that make GPUs applicable to a wider range of scientific applications [2]. An increasing number of scientific applications are also being ported to take advantage of such clusters [3, 4]. Lots of scientific applications use CUDA, the primary parallel programming model for NVIDIA GPUs, in conjunction with high-level programming models like MPI. Usually, CUDA is responsible for the kernel computation and data movement between local CPU host and GPU device. And high-level programming models like MPI are responsible for inter-process communications. Several designs have been proposed to reduce the overhead of data movement by enabling MPI for communication directly from GPU device memory [5]. In addition, MPI libraries have taken advantage of advanced CUDA features like CUDA IPC and optimizations like pipelining to improve GPU-to-GPU communication, which are transparent to the application users [6].

GPUDirect is a set of features offered by CUDA to enable efficient data movement among GPUs and between GPUs and other PCI Express (PCIe) devices [7]. Since CUDA 5.0, the initial GPUDirect in CUDA 4.1 was extended with GPUDirect RDMA (GDR) feature, which allows network adapters to directly read from or write to GPU device memory while completely bypassing the host. GDR avoids the additional hop in original design where the data has to be copied to the host before sending it out to other nodes.

*1) **Motivation**:* MPI libraries like MVAPICH2 [8] and Open MPI [9] have incorporated GDR techniques into CUDA-Aware MPI communications. By taking advantage of the performance potential offered by GDR while tackling the limitations observed on the current generation systems, MVAPICH2 achieves faster path for moving data between GPU memory to a remote GPU or a remote host [10].

Nevertheless, there are some limitations in the existing GDR-based communication design. Current GPU-to-GPU communication relies on rendezvous protocol for all message sizes. However, the CTS/RTS header exchange incurs additional overhead for small message transfers in handshake stage and an explicit synchronization between the sender and the receiver will limit the overlap between communication and computation. Furthermore, the bandwidth limitation brings inefficient use of the GDR feature when the network adapter and the GPU are on different sockets.

Existing eager Protocol does not take advantage of the GDR and uses host-based copies. Furthermore, it needs additional copies using the temporary buffers. To avoid these limitations, rendezvous protocol is proposed and used by current MVAPICH2 for all message sizes. Obviously, all above limitation have to be considered while designing MPI libraries to effectively take advantage of GDR for GPU-to-GPU communication.

In this paper, we discuss the existing GDR design in MVAPICH2 library. Also, we propose novel techniques that efficiently take advantage of the performance potential offered by GDR and InfiniBand Verbs. We make the following key contributions through this paper:

1) Propose an sglist-based (scatter-gather list) method to optimize the eager protocol that combines the transfer of meta-data (header) and message data in one IB Verb operation.
2) Propose a loopback design to optimize the eager protocol that takes advantage of the GDR features.
3) Propose a new Fastcopy technique to replace the existing CUDA Memcpy APIs for Host to Device transfers.
4) Present a quantitative evaluation of the proposed designs across clusters with diverse GPU configurations using micro benchmarks and end-applications

The rest of the paper is organized as follows. In Section II, we describe the background related to our work. In Section III, we discuss the current design and present the proposed design in details. We present experimental results in Section V. Finally, we summarize the related work in Section VI and conclude the paper in Section VII.

II. Background

In this section, we describe the background for this paper. We introduce the GPU architecture and CUDA programming models. Then we describe the GPUDirect RDMA technology and InfiniBand architecture. In addition, we summarize the existing support for GPU-to-GPU communication in MVAPICH2 library.

A. GPU and CUDA Programming Model

GPU is a peripheral device that is connected to the host through the high speed IO slot. (e.g. PCIe) In this paper, we focus on the NVIDIA GPU architecture. The latest architectural revision of NVIDIA GPUs for High Performance Computation (HPC) is named Kepler. A Kepler GK110 packs around 7.1 billion transistors, delivering over 1 TFLOP of double precision throughput and up to 3x the performance per watt of a Fermi. It includes up to 15 Streaming Multiprocessor (SMX) units and 6 64-bit memory controllers. Each of the SMX units features 192 single precision cores, 64 double precision units, 32 special function units (SFU), and 32 load/store units. Each SMX has 64KB of configurable shared memory/L1 cache and a 48KB of read-only data cache. Tesla K20C features 1536KB of dedicated L2 cache and 5GB of DRAM. The SMX allows four warps to be issued and executed concurrently where a warp is a group of 32 threads. Keplers quad warp scheduler can dispatch two independent instructions per warp in each cycle. Unlike Fermi, Kepler also allows double precision instructions to be paired with other instructions. Kepler also boasts features like Dynamic Parallelism and HyperQ that are aimed at increasing the utilization of GPUs. NVIDIA provides a software framework called the Compute Unified Device Architecture (CUDA) [1] for programming its GPUs. Code that runs on the GPU is often called a CUDA kernel. GPUs are connected as peripheral devices on the I/O bus (PCI express). Communication between GPU and host as well as among GPUs used to be a performance bottleneck as it involved multiple costly transfers over the PCIe bus. Since CUDA 4.0, NVIDIA introduced two new technologies: Unified Virtual Addressing (UVA) and GPUDirect Peer-to-Peer (P2P) communication among GPUs. UVA enables a process to have a unified address space across main (host) memory and device memories of GPUs connected on a single node. Thanks to P2P, data can be moved directly from one GPU to another, thereby bypassing temporary staging in host memory. However, communication between GPU buffers used by different processes is still needed to go through main memory. NVIDIA addressed this shortcoming in CUDA 4.1 by supporting Peer-to-Peer communication among processes through CUDA Inter-Process Communication (IPC) technology. IPC makes it possible for a process to directly access device memories belonging to other processes, either on the same or different GPUs, sitting on the same host, without involving main memory.

B. GPUDirect RDMA

NVIDIA's GPUDirect technology provides a set of features that enable efficient communication among GPUs and between GPUs and other devices. The initial version of GPUDirect, released in CUDA 4.0, enabled the same host memory regions to be registered by both a network adapter and a GPU device. Registration is necessary for HW DMA engines to directly access memory regions and GPUDirect technology essentially allowed the DMA engines of these two PCIe devices, the GPU and the IB network host adapter, to transfer data into and out of the same host region. Host memory is registered with the GPUs DMA to enable faster asynchronous copies between GPU device and CPU host. Communication over IB also requires registration of the host buffers involved in the transfer. The above feature of GPUDirect avoided an additional copy in host memory when data copied from the GPU has to be transferred over the InfiniBand network. MPI libraries like MVAPICH2 already take advantage of this feature.

In the most recent release of CUDA, GPUDirect has been extended to allow third party PCIe devices to directly read/write data from/to GPU device memory. This is called GPUDirect RDMA (GDR). The existing work utilizes this upgraded version of GPUDirect to enhance inter-node MPI communication from GPU memory.

C. *InfiniBand*

InfiniBand [11] is a very popular interconnect that is used by 41% of the Top500 supercomputers [1]. One important feature of InfiniBand is the support for remote direct memory access (RDMA). RDMA enables directly reading from and writing to the remote node. Besides, InfiniBand provides atomtic operations on remote memory regions. InfiniBand needs to pin and register the memory before it accesses these memory spaces. Communication through InfiniBand is handled between queue-pairs (QPs). A QP consists of two queues to handle send and receive requests, respectively. The RDMA requests are submitted to the send queue. Every queue is associated with a completion queue, where the InfiniBand interface or host channel adapter (HCA) confirms the completion of either send or receive operation.

D. *CUDA-AWare MPI Library*

Message Passing Interface (MPI) is the de-facto standard for parallel application development in the HPC domain. MVAPICH2 is a popular open-source implementation of MPI for InfiniBand, 10Gigabit Ethernet/iWARP and the emerging RDMA over Converged Enhanced Ethernet (RoCE). MVAPICH2 unifies data movement from/to GPU and host memories through the standard MPI semantics. This design was first proposed in [5]. MVAPICH2 achieves this through the Unified Virtual Addressing (UVA) feature that is provided starting from CUDA 4.0. MVAPICH2 further optimizes the performance of inter-node GPU-to-GPU communication by pipelining transfers from GPU to host memory, host memory to remote host memory over InfiniBand and finally from remote host to destination GPU memory. The three stage pipelining results in a significant boost in communication performance between GPUs on different nodes. MVAPICH2 also provides multiple solutions for intra-node GPU-to-GPU communication between processes, using shared memory and CUDA IPC.

III. Overview of the Existing Design

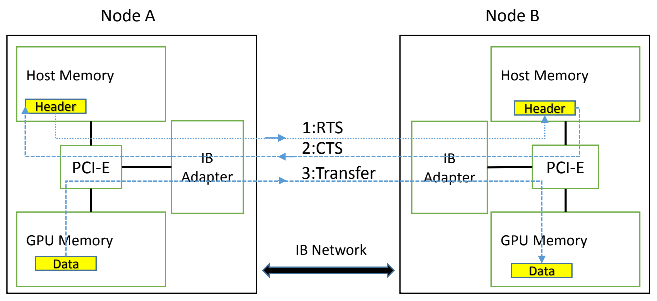


Figure 1: Overview of GDR Rendezvous Protocol Design

For GPU-to-GPU communication, meta-data (header), which includes the necessary information for communication (e.g. data start address, data size, tag information), usually resides on CPU host buffers, while message data that needs to be exchanged is located on GPU device buffers.

The eager protocol is traditionally the most performing design for small message sizes. One straightforward implementation for GPU-to-GPU communications relies on the host-to-host IB transfers. Sender first copies data from device buffer to temporary pre-registered host memory buffer, then it directly write data to pre-registered target buffer sitting in receiver host memory. When the receiver detects the arrival of data, it copies data from its host temporary buffer to the final destination in device memory. Obviously, the two cudaMemcpy calls, one at the sender and the other at the receiver side, make this design inefficient. A detailed presentation of an efficient alternative implementation is shown in the next section.

Some previous work [10] has shown that, when GDR is available for GPU-to-GPU communications, it is more efficient to disable the eager protocol and to employ the rendezvous protocol for all messages sizes. Figure 1 illustrates the three steps to transfer data from a GPU device on one node to a GPU device on a remote node using rendezvous protocol. In the beginning, the source process starts a handshake with the target process to get the target address information (1:RTS and 2:CTS). After that, the source process writes data to the target process using RDMA (3:Transfer). With this design, an initial synchronization phase is required for all message sizes, which adds a relatively large overhead for small messages. The upside is the ability to write the data directly to the final GPU memory destination through RDMA, thereby avoiding additional memory copies.

IV. Proposed Design for GPU-to-GPU Communication Eager Protocol

In this section, we propose an improved implementation of the eager protocol for the GPU-to-GPU communication. Figure 2 shows the overview of our proposed designs, which include two parts: sender side and receiver side. Number “1” and “2” represent the first and second communication steps in the GPU-to-GPU communication. In the following subsections, we discuss these two parts, respectively.

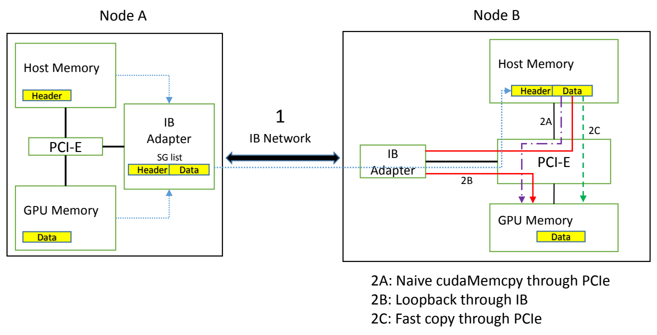


Figure 2: Overview of Proposed GDR Eager Designs

A. *Sender Side Design*

On sender side, we introduce the sglist-based design. The sglist (scatter-gather list) scheme leverages the gather/scatter feature provided by InfiniBand Memory semantics. It takes advantage of the HW capability of processing non-contiguous buffers on both send and receive sides, the so-called Send-Gather and Receive-Scatter operations.

For GPU-to-GPU communication, the meta-data (header) is usually stored on CPU host buffer, while message data is located on GPU device memory. There are two solutions to transfer these two parts. One solution is to explicitly combine meta-data and data, by copying the data from device to host memory, and then sending the combined data. The other method is to use two IB send operations back-to-back, respectively for the header and the data. For the latter method, we need to ensure the delivery ordering between header and data parts: the header should arrive at the destination before the data. Clearly, neither method is ideal, due to additional operations involved.

To overcome the limitation, we introduce the sglist-based design. By using sglist, sender can combine the transfer of header and data parts by launching one IB operation. Compared to the first solution, sglist utilizes low-level IB communication verbs and thus demonstrates much better performance.

B. *Receiver Side Design*

1) Naive Design: On the receive side, the naive design, shown as "2A" path in Figure 2, uses CUDA memory copy API (cudaMemcpy) to move the data between host buffer and device buffer. After the receiver side gets the data from the sender, it checks whether the destination is on device or not through header information. If the destination is on device, it will use the cudaMemcpy API to copy the message data from the temporary receive buffer, located on host memory, to the final destination in device memory, through the PCIe.

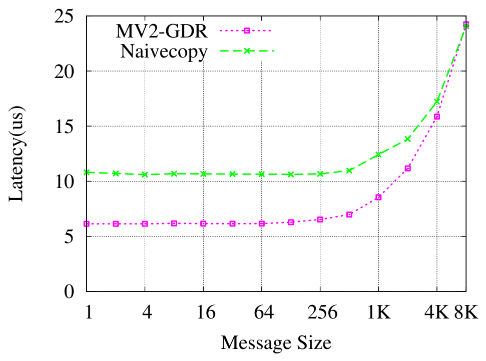
For the naive design, full PCIe bandwidth is available but it incurs a large initial overhead, due to the cudaMemcpy HW/SW implementation. Figures 3(a) and 3(b) compare the GPU-to-GPU latency and bandwidth between current rendezvous GDR protocol ("MV2-GDR") and eager protocol using naive cudaMemcpy for host-to-device data transfer ("NaiveCopy"), respectively. "NaiveCopy" shows around 10 μs latency for small message sizes while "MV2-GDR" achieves lower latency, which is around 6 μs. This is largely due to the fact that cudaMemcpy is a very expensive operation, which contributes around 8 μs to "NaiveCopy" latency. Also, "NaiveCopy" only achieves an average of 30% of the peak bandwidth achieved by "MV2-GDR", on the small message sizes (ranging from 0 to 8KB). Consequently, in the evaluation part, we take the existing rendezvous GDR protocol as the baseline and evaluate and compare its performance with new proposed designs.

2) Loopback Design: To avoid the expensive cudaMemcpy operation used in naive design, we propose the Loopback design. This design relies on the IB verbs to implement the transfer from host to GPU memory, using GPUDirect RDMA. After the matching with header data, the IB adapter is programmed to read the data from the temporary receive buffer on host to the destination memory on device. The "2B" path in Figure 2 illustrates this procedure. This design fully utilizes the GDR capability of IB and thus avoids the calling of expensive cudaMemcpy API used in naive design. Basically, Loopback design has full PCIe bandwidth on GDR and delivers lower latency. However, its performance is affected by PCIe peer-to-peer HW limitations (e.g. on Intel Sandy Bridge) and it consumes additional bandwidth available to the network adapter, as data has to travel three times across its PCIe link.

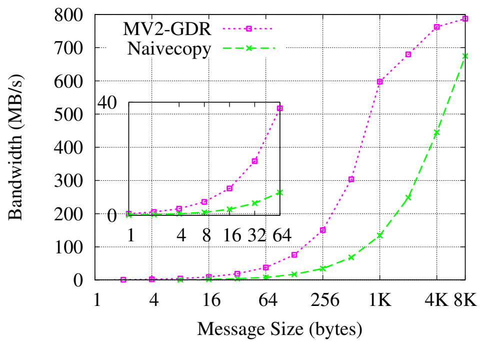
3) Fastcopy Design: Fastcopy is a new low-latency host-GPU copy data-path experimentally provided by NVIDIA. Basically it offers three sets of primitives:

- Pin/Unpin, implemented through the kernel-mode GPUDirect RDMA APIs, used to setup/tear down HW mappings of GPU memory buffers; those mappings are backed by one PCIe BAR of the GPU and come in 64KB-aligned chunks.
- Map/Unmap, which pages are memory-mapped into a contiguous user-space CPU address range. Once mapped in user-space, the CPU can use standard load/-store instructions (MMIO) to access the GPU memory.
- Copy to/from PCIe BAR, a couple of highly tuned functions implemented in terms of Intel SSE instructions.

The use of these APIs is labeled as "2C" path in Figure 2. To avoid the unnecessary buffer pinning/unpinning and mapping operations, we extended the registration cache design to keep track of the GPU buffers. When the application calls CUDA APIs to allocate the device buffer in the first time, the MVAPICH2 library registers this buffer. If the application reuses the same buffer, the MVAPICH2 library will fetch out the registered buffer directly instead of re-registering it. By taking advantage of the registration cache, MVAPICH2 can average the cost of pinning/unpinning and mapping operations. Also, for unpinning operations, we intercept the CUDA APIs related to device memory release and unregister the device buffer from registered buffer list. After getting all necessary information for the data transfer, we call the low-level copy API to move the data between host and device. As for the cudaMemcpy API, there are two corresponding low-level APIs: Host-to-Device and Device-to-Host copies. From the perspective of the access to the GPU BAR, we name Host-to-Device as WRITE and Device-to-Host as READ. Probably thanks to HW level architectural optimizations like write-posting and write-combining which allow for efficient bus utilization, WRITE bandwidth is around 4 GB/s, while READ bandwidth only reaches around 20 MB/s. By taking



(a) D-D Latency



(b) D-D Bandwidth

Figure 3: Comparison of Rendezvous GDR and Eager Naive Designs

this limitation into consideration in our design, we utilize the Fastcopy techniques for Host-to-Device transfer only. Compared to Loopback design, Fastcopy only burns CPU cycles because of its CPU driven nature. However, it is still limited by PCIe bandwidth and affected by NUMA effect.

V. Performance Evaluation

In this section, we describe our experimental testbed and evaluate the proposed design with micro benchmarks and end-applications.

A. Experiment Setup

Cluster Specification	A	B	Wilkes
CPU Processor	Intel E5-2670	Intel E5-2690 v2	Intel E5-2630
CPU Clock	2.60 GHz	3.00 GHz	2.60 GHz
Socket Type	Dual-Socket	Dual-Socket	Dual-Socket
Cores per Socket	8-Core	10-Core	8-Core
CPU Memory	32 GB	128 GB	64 GB
NVIDA GPUs	Tesla K40c	Tesla K40m	Tesla K20
GPUs per Node	2	1	2
GPU Memory	12 GB	12 GB	5 GB
Compilers	gcc 4.4.7	gcc 4.4.7	gcc 4.4.7
MPI Library	MVAPICH2-GDR	MVAPICH2-GDR	MVAPICH2-GDR
Mellanox Interconnect	IB FDR	Connect-IB FDR	IB FDR

Table I: Experimental Environment

We used three clusters in our experiments and their specifications are listed in Table I. Point-to-point experiments and the evaluation of GPULBM were conducted on Cluster A, We evaluated the collective operations and HOOMD-blue application on Wilkes Cluster with up to 64 GPU nodes where each node has two NVIDIA Tesla K20 GPU accelerators. In addition, we evaluated the point-to-point latency and bandwidth on Cluster B, which is equipped with Intel IvyBridge architecture and NVIDIA K40m GPUs. The Wilkes is deployed in November 2013 and is the UK's fastest academic cluster. It achieves a 240TF performance for High Performance Linpack, which ranks it the 166 in latest Top500 list. Also, the Wilkes is ranked second in the worldwide Green500 ranking.

The Fastcopy design requires inserting additional linux driver into each GPU device. As we do not have permission to insert driver module on remote clusters, we only evaluated the MV2-GDR and proposed Loopback design on Wilkes. Based on the results and analysis on cluster A with the Fastcopy design, we are confident that the overall performance of the Fastcopy design on other clusters will follow similar trend as that on cluster A.

B. Evaluation of Point-to-Point Micro Benchmark

In this section we evaluate our proposed design by comparing it with the existing GDR solution using rendezvous protocol (MV2-GDR). MV2-GDR utilizes the potential of GDR to facilitate the data transfer between CPU memory and GPU memory for both inter-node and intra-node communications.

We chose three representative micro benchmarks from OMB-GPU [12]. OMB-GPU is an extension to OMB (OSU Micro-benchmark) that allows users to compare the performance of MPI libraries on GPU clusters. OMB-GPU allocates the data on device memory and use them directly in MPI communication operations. Since point-to-point communication pattern is widely used in GPGPU applications, our evaluation of micro benchmarks provides good perspective for real GPGPU applications as well.

To verify the benefit of proposed design, we evaluate the point-to-point osu_latency, osu_bw (bandwidth) and osu_mbw_mr (Multiple Bandwidth / Message Rate) benchmarks.

Basically, we measure the micro benchmarks in two communication patterns. In the graphs, "H-D" represents that sender side generates the data on CPU host buffer and sends it to the device buffer on receiver side. "D-D" represents that sender side generates the data on GPU device buffer and sends it to the device buffer on receiver side over the network. These two patterns can measure and compare the host to device transfer on receiver side across different designs.

1) Evaluation of Latency: Figures 4(a) and 4(b) compare the performance across different designs for "H-D" and "D-D" latencies, respectively. In general, both Loopback and

Fastcopy designs achieve better performance than existing GDR design. In the case of "H-D", compared to GDR, Loopback and Fastcopy achieve up to 50% and 63% latency reduction, respectively. And Fastcopy achieves an additional 27% lower latency compared to Loopback.

In the case of "D-D", compared to GDR, Loopback and Fastcopy achieve up to 12% and 45% latency reduction, respectively. And Fastcopy achieves an additional 44% lower latency compared to Loopback for 4KB message size.

2) Evaluation of Bandwidth: Figures 5(a) and 5(b) compare the performance across different designs for "H-D" and "D-D" bandwidths, respectively. Similar to latency, both Fast-Copy and Loopback designs achieve better performance than existing GDR design. In the case of "H-D", compared to GDR, Loopback and Fastcopy achieve up to 5.5x and 7.3x bandwidth improvement, respectively. And Fastcopy gains an average of 1.8x improvement compared to Loopback.

In the case of "D-D", compared to GDR, Loopback and Fastcopy achieves up to 1.1x and 1.7x bandwidth improvement, respectively. The reason for this small "D-D" bandwidth improvement is due to SandyBridge internal limitations.

Figures 6(a) and 6(b) show the latency and bandwidth performance of "D-D" on Cluster B. Compared to MV2-GDR, Loopback achieves an average of 4% lower latency, while Fastcopy gains 35% lower latency on average. On Cluster B, the presence of the PCIe switch connecting the IB card and the GPU improves radically the performance for "D-D". The Sandy Bridge bandwidth cap at roughly 800 MB/s disappears, so it reaches 2.2GB/s. Loopback and Fastcopy bandwidth are respectively up to 1.1x and 2x better than MV2-GDR, while Fastcopy is 1.7x better than Loopback.

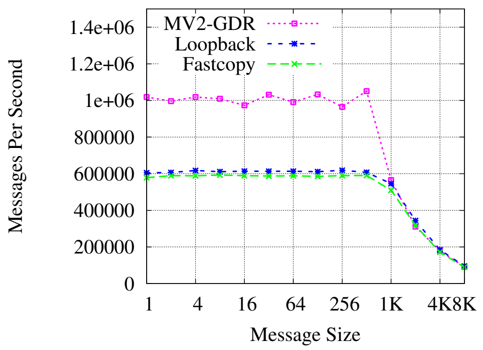


Figure 7: Evaluation of D-D Message Rate with 2 GPU Nodes on Cluster A

3) Evaluation of MBW_MR: Figure 7 compares the performance across different designs for "D-D" osu_mbw_mr. In this case, compared to GDR, Loopback design looses 5% bandwidth and message rate on average; However, Fastcopy gains an average of 1.6x improvement on both bandwidth and message rate. For messages larger than 512 Bytes, the message rates of all three designs drop. This is true considering the fact that we increase the message size while the maximum bandwidth is fixed for certain node configuration.

C. Evaluation of Collective Micro Benchmark

Besides the point-to-point communications, we evaluated the proposed designs on WILKES with collective communications. In general, MPI collective communications are implemented over point-to-point operations. We have evaluated Gather, Allgather and Alltoall, which are widely used collective operations in large-scale applications.

The existing MVAPICH2 library provides an optimization for all collective communication from GPU memory. For small messages, MVAPICH2 copies data from device memory onto the host memory, performs the collective operation on CPU hosts and finally copies the accumulated data back on the respective device memory. This design reduces the number of CUDA memory copies between hosts and devices. While for large messages, MVAPICH2 relies on the host-based pipelining design to hide the CUDA memory copy overheads. Beyond that, MVAPICH2 offers an advanced design for Alltoall [13], which uses a dynamic scheme to pipeline the communication at the level of collective algorithm.

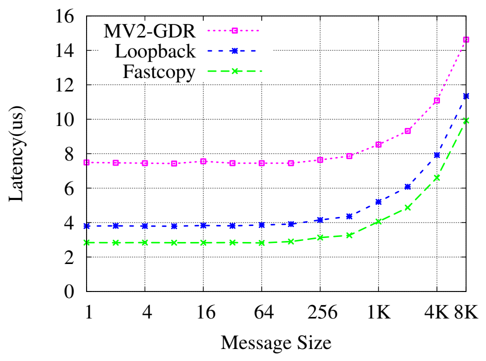
In the current MVAPICH2, Gather operation is implemented with the root posting non-blocking receives from all processes and every other process posting a blocking send to the root. The non-blocking nature of GDR allows the transfers to progress in parallel and thus shows lower latency compared to CUDA memory copies that happen at the start and end of the previous implementation in MVAPICH2.

Figure 8 shows the latency comparison between the existing MV2-GDR design and proposed Loopback design for three collective operations. Compared to MV2-GDR, Loopback achieves up to 49%, 47% and 67% latency reduction in Gather, Allgather and Alltoall latency for 256, 32 and 256 Bytes message sizes, respectively.

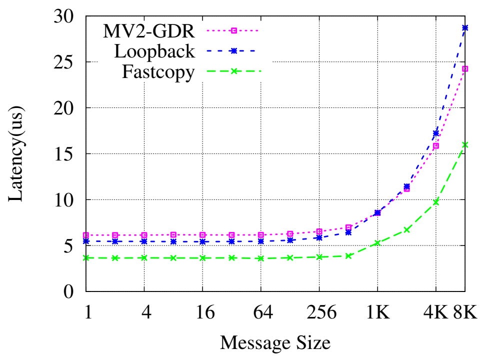
D. Evaluation of End-Applications

In this section, we evaluate the proposed design across two representative end-applications: GPULBM and HOOMD-blue.

1) Evaluation of GPULBM: Lattice Boltzmann Method (LBM) is a popular computational fluid dynamics (CFD) method that is used when a high level of detail is required in a relatively small computational domain. Since the multiphase LBM can simulate the interaction and dynamics of multiple fluids, it has been used widely across the world. GPULBM is a parallel distributed CUDA implementation of Lattice Boltzmann Method (LBM) for multiphase flows with large density ratios [14]. It is an iterative application that operates on 3D data grids. The decomposition of data is along the Z-axis. And the number of elements involved in communication can be calculated as the product of X and

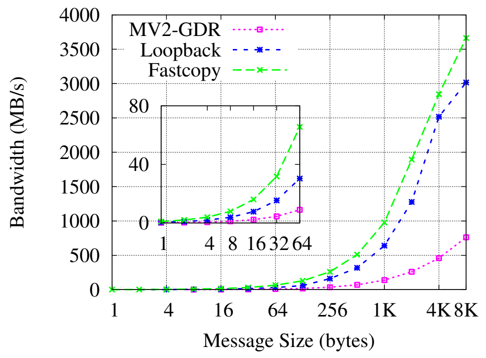


(a) H-D Latency

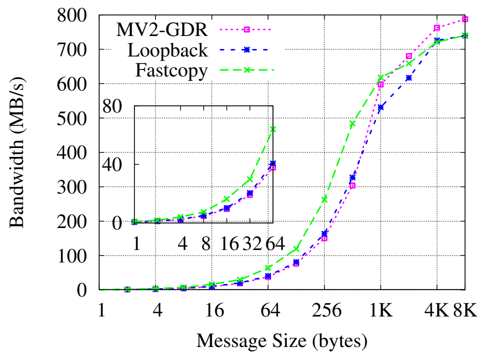


(b) D-D Latency

Figure 4: Evaluation of Latency with 2 GPU Nodes on Cluster A

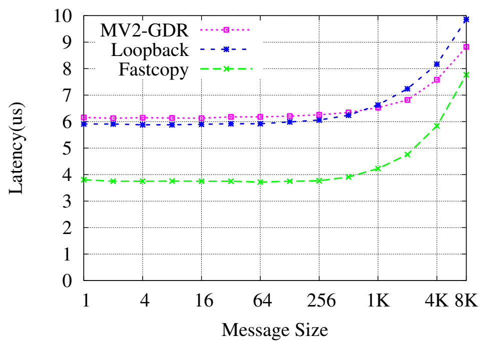


(a) H-D Bandwidth

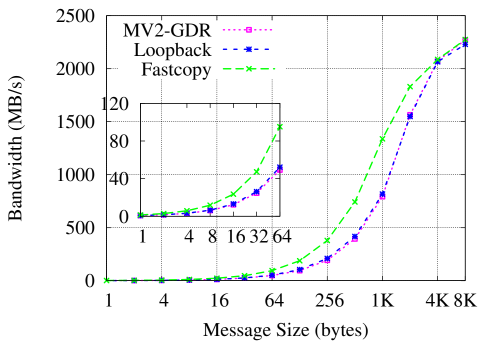


(b) D-D Bandwidth

Figure 5: Evaluation of Bandwidth with 2 GPU Nodes on Cluster A



(a) D-D Latency

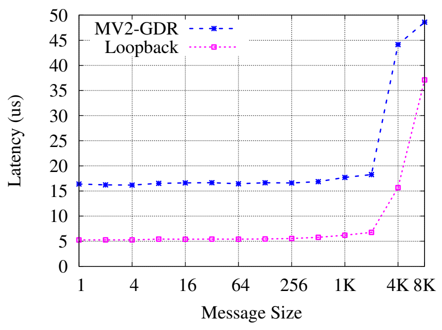


(b) D-D Bandwidth

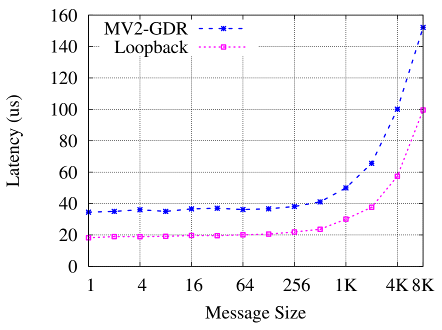
Figure 6: Evaluation of D-D Communication with 2 GPU Nodes on Cluster B

Y dimensions of the grid times the required 6 degrees of freedom, with each element being of float type. To compare the performance across three designs, we conducted the experiment on cluster A using two GPU nodes equipped with Tesla K40c accelerators. Since we target the evaluation of Loopback and Fastcopy designs, we restrict the sizes of 3D data grids within the eager threshold (e.g. 16KB). Also, LBM requires the dimension size of X and Y to be the multiple of 32. As a result, we only conducted the experiment with various sizes in Z dimension.

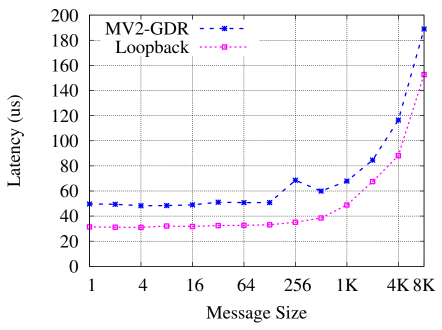
Figure 9(a) measures the overall execution time of LBM with various grid sizes. Loopback and Fastcopy designs achieve up to 18.8% and 23.4% reduction in total execution time for 32x32x4 3D grid size. Figure 9(b) demonstrates the decomposition time of LBM application in terms of computation time and total MPI communication time for 32x32x4 grid case. Compared to MV2-GDR, 5.3% and 15.4% reductions in MPI communication time are achieved by Loopback



(a) Gather



(b) Allgather



(c) Alltoall

Figure 8: Evaluation of Collective Operations with 16 GPU Nodes on Wilkes

and Fastcopy, respectively. Besides the reduction in MPI communication time, the new design also enables better overlap between computation and communication, which contributes much more reduction in overall execution time.

2) Evaluation of HOOMD-blue: HOOMD-blue (Highly Optimized Object-oriented Many-particle Dynamics) is a general-purpose particle simulation toolkit [15] that is widely used in molecular dynamics areas. Users can define particle initial conditions and interactions in a high-level python script. The HOOMD-blue benefits from the advanced features of NVIDIA GPUs (e.g. GDR) and scale well to thousands of GPUs.

GDR provides a direct point-to-point data path between GPU and IB and thus delivers a significant decrease in GPU-GPU communication latency for HOOMD-blue. FDR InfiniBand interconnect on Wilkes cluster allows HOOMD-blue to achieve good scalability. HOOMD-blue utilizes non-blocking MPI_Isend/MPI_Irecv and collectives for most data transfers. For HOOMD-blue, we use cluster A to measure the benefit gained from the Loopback and Fastcopy designs. And for the scalability of HOOMD-blue, we further measure the weak and strong scaling on Wilkes. Similar to evaluating the GPULBM, we choose comparatively small particle sizes in order to enable the eager protocol used in MVAPICH2. Even though using more GPU nodes with small particle sizes makes the application communication-bound, the comparison still highlight the benefit gained from proposed design in terms of MPI communication time and overall performance. In our experiment, bundled lj-liquid-bmark script is used as the input file to evaluate the HOOMD-blue.

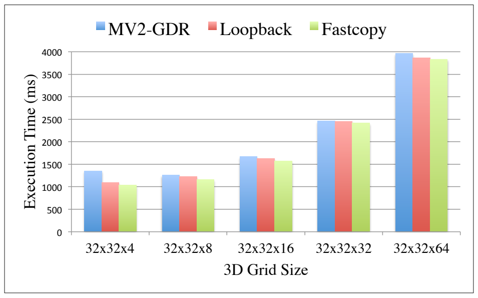
Figure 10(a) evaluates both Loopback and Fastcopy designs. Compared to MV2-GDR design, Loopback and Fastcopy achieves up to 12% and 12.5% improvement in TPS (Transaction per Second) for 4K particles. And Figure 10(b) compares the performance between MV2-GDR and Loopback designs with 64 GPU nodes on Wilkes. Proposed Loopback design outperforms the existing MV2-GDR design by up to 20.1% improvement in TPS for 64K particles. Furthermore, we measure the strong and weak scaling of HOOMD-blue on Wilkes. For strong scaling, we fix the total number of particles to 64K, while varying the number of GPU nodes used. In the experiment for weak scaling, we keep the number of particles per GPU node to 2K. Figure 11(a) and Figure 11(b) show the overall performance in average TPS for weak and strong scaling. As is shown, using 16 GPU nodes in strong scaling achieves the highest average TPS for both designs. However, the overall performance drops in weak scaling with increasing number of GPU nodes due to the small-scale number of particles involved in the computation. As a result, the application becomes communication-bound with larger number of GPUs. Compared to MV2-GDR, Loopback design demonstrates up to 17% and 20.1% performance improvement in average TPS for cases of 64 GPUs on strong and weak scaling, respectively.

VI. Related Work

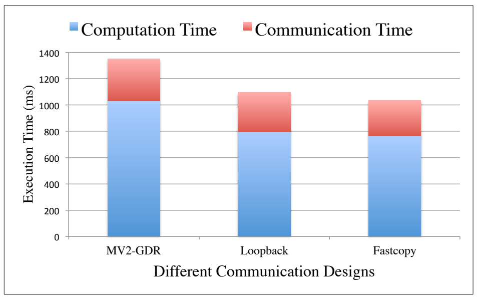
In HPC area, enabling the use of accelerators such as GPUs and co-processing with programming models such as MPI and PGAS has been explored by many researchers [16–18].

Making MPI libraries CUDA-Aware and extending MPI to support communication on GPU clusters have also been the focus for several years. Wang et.al proposed optimized CUDA-based GPU-to-GPU communication for InfiniBand clusters which uses CPU hosts as intermediate buffering units for pipelined internode GPU communication across MPI processes [5]. Some researchers have also explored optimization of GPU-to-GPU communication that involves non-contiguous MPI datatypes [19–22]. Potluri et.al proposed the first solution to harness NVIDIA's GPU Direct RDMA (GDR) feature for MPI libraries and introduced hybrid solutions that benefit from the best of both GDR and host-assisted GPU communication [10].

In addition, the idea of sourcing and sinking the network traffic from GPUs has been explored as well. Stuart et.al introduces DCGN (Distributed Computing on GPU Networks) a framework that allows GPU threads to send and

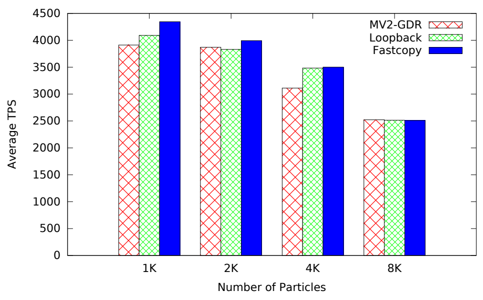


(a) 3D Grid with Various Z Dimension Sizes

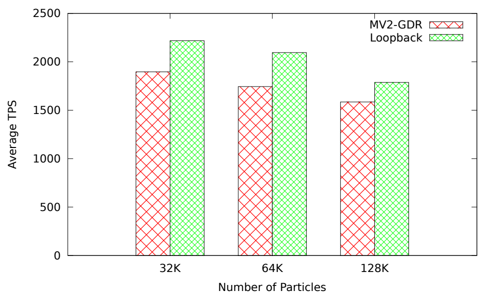


(b) Decomposition Time of 32x32x4 3D Grid

Figure 9: Evaluation of GPULBM with 2 GPU Nodes on Cluster A

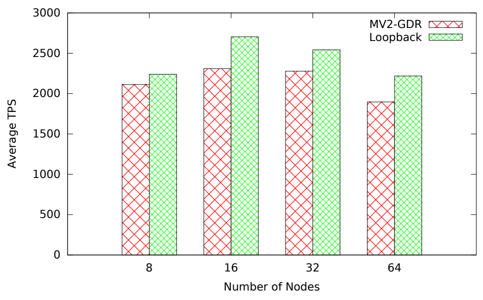


(a) 2 GPUs on Cluster A

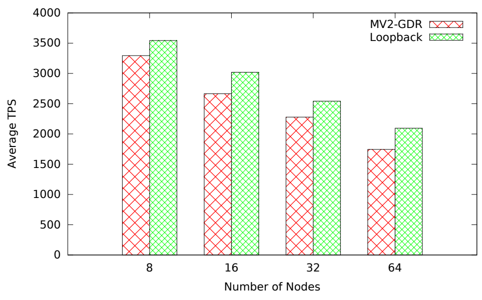


(b) 64 GPUs on Wilkes

Figure 10: Evaluation of HOOMD-blue with Increasing Number of Particles



(a) Strong Scaling with 64KB Particles



(b) Weak Scaling with 2KB particles per GPU

Figure 11: Evaluation of HOOMD-blue with Strong and Weak Scaling on Wilkes

receive data with commands similar to MPI [23]. Oden et.al proposes the framework that allows the GPU to control the network device and independently source and sink network traffic, by modifying the device drivers and user space libraries of InfiniBand network cards and GPUs [24]. However, this approach deteriorates the performance because of the overhead of work request generation on GPUs that is suitable for CPUs, which are highly optimized for single-threaded work. Aji et al. introduces the MPI-ACC, a framework for GPU support that aims to provide portability for both CUDA and OpenCL [25].

VII. Conclusion and Future Work

In this paper, we have proposed several techniques to improve the GPU-to-GPU communication using eager protocol. With the new proposed design, the MVAPICH2 library can boost more efficient CUDA-Aware MPI communications achieving better latency and bandwidth. Also, the proposed designs seamlessly integrate existing designs so as to exhibit equal performance for larger messages using rendezvous protocol. The experimental results exhibit sustained improvement of proposed designs. Our experimental results across three platforms with different GPU configurations validate the portability and stability of our designs.

In the future, we plan to explore proposed design with other communications involving GPUs. For instance, Fastcopy technique can be investigated in GPU-to-GPU communication involving non-contiguous MPI derived Datatypes. Support for the new proposed designs will be available in future MVAPICH2-GDR releases.

VIII. Acknowledgement

This research is supported in part by National Science Foundation grants #OCI-0926691, #OCI-1148371 and #CCF-1213084. We would like to thank Davide Rossetti from NVIDIA for providing the driver patch and library for Fastcopy functions. We also would like to thank Mark Arnold, our system administrator, for helping setup the testbed. This work would not have been possible without their support.

References

[1] Top500, http://www.top500.org.

[2] NVIDIA Kepler Architecture, http://www.nvidia.com content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf/.

[3] A. Nukada, Y. Maruyama, and S. Matsuoka, "High performance 3-d fft using multiple cuda gpus," in *Proceeding of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units (GPGPU-5)*, London, UK, 2012.

[4] R. Shi, S. Potluri, K. Hamidouche, X. Lu, K. Tomko, and D. Panda, "A scalable and portable approach to accelerate hybrid hpl on heterogeneous cpu-gpu clusters," in *Proceeding of CLUSTER (CLUSTER'13)*, Indianapolis, Indiana, USA, 2013, pp. 1–8.

[5] H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, and D. K. Panda, "MVAPICH2-GPU: Optimized GPU to GPU Communication for InfiniBand Clusters," *Comput. Sci.*, no. 3-4, Jun. 2011.

[6] S. Potluri, H. Wang, D. Bureddy, A. K. Singh, C. Rosales, and D. K. Panda, "Optimizaing MPI Communication on Multi-GPU Systems using CUDA Inter-Process Communication," in *Proceedings of the International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), in conjunction with International Parallel and Distributed Processing Symposim (IPDPS'12)*, 2012.

[7] NVIDIA, "GPUDirect," https://developer.nvidia.com/gpudirect.

[8] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, http://mvapich.cse.ohio-state.edu/.

[9] OpenMPI: Open Source High Performance Computing, http://www.open-mpi.org/.

[10] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, " Efficient Inter-node MPI Communication using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs," in *Proceeding of International Conference on Parallel Processing (ICPP'12)*, Pittsburgh, PA, USA, 2012.

[11] InfiniBand, http://en.wikipedia.org/wiki/Infiniband.

[12] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda, "Omb-gpu: A micro-benchmark suite for evaluating mpi libraries on gpu clusters," in *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'12, Berlin, Heidelberg, 2012.

[13] A. K. Singh, S. Potluri, H. Wang, K. Kandalla, S. Sur, and D. K. Panda, "Mpi alltoall personalized exchange on gpgpu clusters: Design alternatives and benefit," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 420–427.

[14] C. Rosales, "Multiphase lbm distributed over multiple gpus," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–7.

[15] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," *J. Comput. Phys.*, vol. 227, no. 10, pp. 5342–5359, May 2008.

[16] T. Miyoshi, H. Irie, K. Shima, H. Honda, M. Kondo, and T. Yoshinaga, "Flat: a gpu programming framework to provide embedded mpi," in *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*. ACM, 2012, pp. 20–29.

[17] D. Cunningham, R. Bordawekar, and V. Saraswat, "Gpu programming in a high level language," in *Proceedings of the ACM SIGPLAN X10'11 Workshop*, 2011.

[18] M. Luo, H. Wang, and D. K. Panda, "Multi-Threaded UPC Runtime for GPU to GPU communication over InfiniBand," in *The 6th Conference on Partitioned Global Address Space Programming Models 2012*, Dec. 2012.

[19] O. Lawlor, "Message Passing for GPGPU Clusters: CudaMPI," in *Proceeding of Cluster Computing and Workshops (CLUSTER '09)*, New Orleans, Louisiana, USA, 2009.

[20] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda, "GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation," *IEEE Transactions on Parallel and Distributed Systems*, 2013.

[21] J. Jenkins, J. Dinan, P. Balaji, T. Peterka, N. F. Samatova, and R. Thakur, "Processing MPI Derived Datatypes on Noncontiguous GPU-Resident Data," *IEEE Transactions on Parallel and Distributed Systems*, 2013.

[22] R. Shi, X. Lu, S. Potluri, K. Hamidouche, J. Zhang, and D. K. Panda, " HAND: A Hybrid Approach to Accelerate Non-contiguous Data Movement using MPI Datatypes on GPU Clusters ," in *Proceeding of International Conference on Parallel Processing (ICPP'14) (to be appear)*, Minneapolis, USA, 2014.

[23] J. A. Stuart and J. D. Owens, "Message passing on data-parallel architectures," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, ser. IPDPS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12.

[24] L. Oden, H. Frning, and F.-J. Pfreundt, "InfiniBand-Verbs on GPU: A case study of controlling an InfiniBand network device from the GPU," in *Proceedings of the International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), in conjunction with International Parallel and Distributed Processing Symposim (IPDPS'14)*, 2014.

[25] A. M. Aji, J. Dinan, D. Buntinas, P. Balaji, W.-c. Feng, K. R. Bisset, and R. Thakur, "Mpi-acc: An integrated and extensible approach to data movement in accelerator-based systems," in *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, ser. HPCC '12, Washington, DC, USA, 2012.