Front. Comput. Sci., 2014, 8(3): 357–366
DOI 10.1007/s11704-014-3500-9

RESEARCH ARTICLE

The TH Express high performance interconnect networks

Zhengbin PANG[1,2], Min XIE[2], Jun ZHANG[2], Yi ZHENG[2], Guibin WANG (✉)[2], Dezun DONG[1,2], Guang SUO[2]

1 Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China

2 College of Computer, National University of Defense Technology, Changsha 410073, China

Abstract Interconnection network plays an important role in scalable high performance computer (HPC) systems. The TH Express-2 interconnect has been used in MilkyWay-2 system to provide high-bandwidth and low-latency interprocessor communications, and continuous efforts are devoted to the development of our proprietary interconnect. This paper describes the state-of-the-art of our proprietary interconnect, especially emphasizing on the design of network interface. Several key features are introduced, such as user-level communication, remote direct memory access, offload collective operation, and hardware reliable end-to-end communication, etc. The design of a low level message passing infrastructures and an upper message passing services are also proposed. The preliminary performance results demonstrate the efficiency of the TH interconnect interface.

Keywords HPC, network interface chip (NIC), TH Express interconnect, offload collective operation

1 Introduction

As the second generation of massively parallel supercomputers in the TH series designed by National University of Defense Technology (NUDT), the MilkyWay-2 (TH-2) system topped the TOP500 list released in June 2013 [1]. The system consists of 16 000 compute nodes of each one equipped with two Xeon E5-2600 processors and three Xeon Phi accelerators. Such massive parallelism puts high pressure on the design of interconnection network. In order to fulfill the high communication requirement, the MilkyWay-2 system uses proprietary interconnect named TH Express-2 interconnect, which especially emphasizes on high bandwidth and low latency interprocessor communications [2].

While the computing nodes and system become more and more parallel due to architectural improvements and parallel programming paradigms, typically there is only a single network interface. Therefore, it is crucial to efficiently support such available parallelism in network interface [3,4]. We continually improve the design of network interface and message passing services based on the implementation in MilkyWay-2 system. Our goal is to provide highly scalable and efficient message passing services through the design at all levels, namely, hardware, operating system, and communication software. This paper describes the state-of-the-art of our proprietary interconnect, especially emphasizing on the design of network interface.

Received December 16, 2013; accepted March 6, 2014

E-mail: wgbljl@gmail.com

2 Network interface chip (NIC)

NIC provides the software-hardware interface for applications to access the high-performance network. As shown in Fig. 1, it contains a full width 16-Lane PCI-E 3.0 interface connected to the compute node, and uses its network port to communicate with interconnect fabric. NIC contains several advanced mechanisms to support scalable high performance computing, including protected user-level communication, remote direct memory access (RDMA), offloaded collective mechanism, etc.

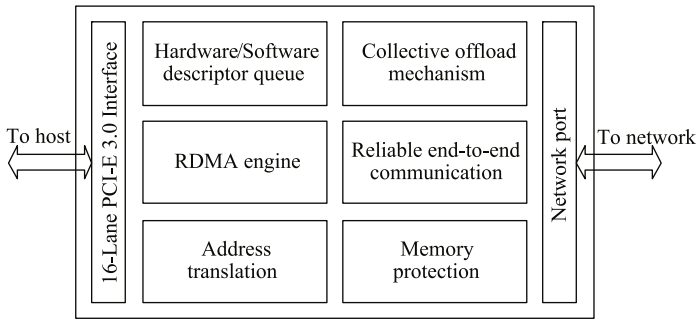

Fig. 1 Architecture of NIC

2.1 Virtual port (VP)

Hardware resources virtualization plays an important role in implementing protected user-level communication [5–7], which provides each process an exclusive programming view for using communication hardware. When several processes run concurrently, communication operations from different processes are isolated without interference.

In order to support protected user-level communications, NIC exploits a mechanism named VP, which is a combination of a small set of memory-mapped registers and a set of in-memory data structures. The address ranges of registers in different VPs are spaced out at least the length of the physical page. All the data structures can be mapped to user space, so that it can be accessed in user space concurrently with protection. The related data structures are organized in several queues, including on-chip hardware descriptor queue (HDQ), and software descriptor queue (SDQ), minipacket queue (MPQ), event queue (EQ), interrupt queue (INTQ), error packet queue (EPQ) in memory. The detailed structure and usage will be described later.

2.2 Hybrid descriptor queue

To support multi-process simultaneously submitting descriptor, NIC supports up to 40 VPs in hardware. Among them, eight VPs receive descriptor through PIO (programmable IO) write and the other 32 VPs fetch descriptor through DMA (direct memory access). The two different types of VP are organized in HDQ and SDQ respectively.

Descriptor submission through PIO is easier by writing descriptors to the HDQ queue directly via PCI-E interface. However, the steps required to fetch descriptors through DMA in SDQ are more complicated. First, the software informs NIC to fetch descriptor by writing the number of descriptors prepared in host memory to a specific register within VP. Second, NIC schedules the candidate VPs which have descriptors to submit. Third, NIC requests to read descriptor in host memory through PCI-E interface. Finally, NIC receives the descriptor and stores it in the SDQ buffer.

Once the descriptors arrive at NIC, they are buffered internally, and then read out in round-robin sequence and submitted to corresponding processing units. Both the two descriptor queues have their specific advantages. Compared with SDQ, HDQ implementation has less PCI-E transactions and relatively lower latency from submitting descriptor to processing it. However, SDQ could accommodate much more descriptors. Therefore, the hybrid descriptor queue implementation supports both low latency and high capacity. It is up to system software to choose the proper VPs to submit descriptor.

2.3 Address translation and memory protection

In order to support RDMA in virtual address mode, NIC implements virtual address translation mechanism to translate virtual address into physical address. The whole address translation table (ATT) [8] is allocated in host memory, and NIC provides an on-chip Cache named ATC (address translation Cache) to reduce the long latency of main memory access.

Before address transformation, the legality of each virtual address must be checked. A memory checker is employed to detect the validity of virtual address. The memory checker contains 4 096 items, while each item is composed of several fields, including valid flag, key, read and write bit, base and top page address boundary. The memory checker serves several request sources, and adopts a round-robin arbiter to select one source each time.

The ATC is applied to bridge the gap between current long latency of main memory outside and high requirement for obtaining physical address. The Cache stores one million physical address items recently used. The Cache adopts eight ways set-associative structure and LRU (least recently used) replacement strategy. A deep non-blocking pipeline is implemented to improve performance and bandwidth. When the total number of physical address items used by software is smaller than one mega, the Cache can be configured to a large buffer to achieve substantial improvement on the performance of accessing the physical address. This buffer is initialized by writing all physical address items those will be used later, and then the latency of reading these items can be largely shortened.

It is important that the procedure of refreshing virtual and physical address items must be complete and correct. While the virtual and physical address items are newly added or

deleted on line by software, the memory checker must be updated simultaneously, and the old items in Cache should be cleaned, and the new ones should be reloaded into Cache. The first step of refresh procedure is that a new item is written in the memory checker. The second step is to check whether any tag in Cache is located between the base and top page address sent from memory checker. The final step is to remove this type of tags from Cache, and then set a global register to inform software that the refresh procedure is finished.

2.4 RDMA engine

RDMA is a one-side communication mechanism which is efficient for block data transfer without interference with processor. In TH Express Network, the interconnect fabric achieves reliable link-level packet delivery through link-level CRC and packet retransmission. Besides that, NIC provides a reliable end-to-end connection mechanism. Therefore, block transfer can be started as long as the address and the length of the local and remote buffers are provided. The RDMA engine supports both PUT and GET operations.

To prevent invalid memory access, RDMA in the NIC is protected by a key-based mechanism [9]. Each VP has a unique key that can only be set by software. The keys of local and remote VPs should be provided in RDMA descriptor. Only when keys are matched, can RDMA access memory buffers. This security mechanism provides sufficient protection in our HPC environment.

During RDMA data transfer, NIC uses ATC to translate the virtual address into the PCI address of physical pages. There is an address aligning module in the NIC, which helps to byte-align the local and remote buffer of RDMA. Because the buffers in kernel module are often allocated in contiguous physical memory, RDMA can also be set to bypass ATC, thus PCI address of physical page can be contained directly in RDMA descriptor.

To reduce the latency of short data in RDMA, NIC supports immediate RDMA PUT operation. The source data is contained directly in the immediate RDMA PUT descriptor, therefore NIC can send data directly to the network without accessing local memory.

The status of RDMA operation is checked via event mechanism. When RDMA is completed, NIC can generate a local and a remote event specified by the descriptor, and write it to a registered memory unit specified by software.

2.5 Offloaded collective mechanism

In order to speed up collective communications in MPI, NIC provides offload mechanisms to accelerate collective operations [10,11]. The software is required to construct collective algorithm tree and generate a collective descriptor sequence for NIC to initialize collective communication.

The execution of collective descriptor sequence is triggered upon receiving a special control packet. When the descriptor sequence is executed, NIC may perform a series of swap operations to modify the memory address and VP information of the descriptor in the descriptor sequence using the data from the control packet.

In the broadcast procedure, a non-leaf node in the collective algorithm tree needs to submit a series of descriptors with the same data element to transmit data to its child nodes. In order to reduce the descriptor submission transactions, NIC defines a special descriptor named collective descriptor which can be used for multiple times in hardware. Thus, the software can submit only one collective descriptor to facilitate the broadcast from parent nodes to its child nodes in a broadcast tree.

2.6 Reliable end-to-end communication

Reliability becomes more and more important to the high performance interconnect. Although link level retransmission can guarantee point-to-point reliability, there still exists the possibility that some errors cannot be found by link level CRC check. Moreover, soft errors happening on router chips may not be corrected by their chip level reliable mechanisms. Reliable end-to-end communication can recover the errors that cannot be solved by reliable mechanisms in link level or in router-chip level, therefore it can improve the reliability of the high performance interconnect further.

A dynamic context based reliable end-to-end communication mechanism is introduced in the network interface. The contexts include sender contexts and receiver contexts. They are all saved on the chips and allocated on demand dynamically. A connection needs to be setup between a sender and a receiver before the sender wants to communicate with the receiver. Accordingly, a sender context and a receiver context are allocated for the connection to save the corresponding information about it. When data transfer is ended, the sender context and the receiver context will be freed to close the connection. User level CRC is used in data packets to ensure the data integrality. When user level CRC errors occur, data packets will be retransferred. A timeout counter is used in the context to detect any packet missing error. Data packets will also be retransferred when packet missing errors occur. The contexts provided by the chip are limited. The sender contexts

constrain the maximum connections it can setup with the receivers simultaneously and the receiver contexts constrain the maximum connections it can accept from the senders simultaneously. The descriptor processing will be paused when the sender runs out of the sender contexts and the connection setup requests will be NACKed and retried when the receiver runs out of the receiver contexts.

In order to reduce the negative effect of connection setup on data transfer delay, the data fetch from the memory operations can overlap with the connection setup process, which means that data fetch can start before the connection has been setup.

2.7 Queue management

In order to support polling hardware status in system software, NIC supports managing several queues in host memory, which includes the MPQ, EQ, INTQ and EPQ. Each memory queue is organized as the FIFO (first-in-first-out) structure, its write port is managed by the NIC, through which the NIC can put various packets into the queue, and the read port is managed by software.

MPQ: This is a queue for receiving a minipacket (MP), a short two-sided communication packet. MPs are sent by descriptors, and queued in the MPQ at the destination VP. To support large queue size, the MPQ supports virtual address mode.

EQ: NIC uses various types of events to report the completion of communication. Each VP has a corresponding EQ. The EQ also supports virtual addressing mode.

INTQ: NIC leverages the INTQ to save the interrupt information, i.e., the interrupt type, source address, VP number, etc. When the INTQ is not empty, NIC triggers hardware interrupt command to system software, then the latter could access the INTQ to check which type of the interrupt is triggered. The NIC supports several types of interrupt, including queue not empty, error, descriptor defined interrupt, etc.

EPQ: EPQ is mainly for hardware debugging purpose. When the NIC receives error network packets, it can be configured to save it in the EPQ, and then system software can fetch the error packets from EPQ to check the specific errors in network.

3 Galaxy Express-2 communication system

Galaxy Express-2 (GLEX2) communication system is the basic message passing infrastructures for other software subsystem to efficiently utilize the new generation of TH Express interconnect. GLEX2 is composed of user-level library libglex, kernel modules gdev, TCP/IP driver gnet, PXE driver gpxenet and some management utilities.

3.1 User-level communication interface

Libglex adopts endpoint to provide data transmission service. Endpoint is an abstract object in user space for a VP in NIC. On initialization, process must create an endpoint to map the hardware sources within a VP, including several registers and related in-memory data structure, into user address space. Based on the virtual memory management, multiple processes could achieve protected, fully user-level communication via endpoint, bypassing the interference of operating system in critical communication paths.

Through the endpoint, process can communicate with all other endpoints via commanding NIC hardware directly. It is up to the upper-level software to choose to use either the MP or RDMA operations to perform a communication. The memory that RDMA accesses should be registered before RDMA operations, through the memory registration interfaces in libglex. The memory registration interface would record the corresponding physical address information into address translation table and memory management table to support direct memory access in NIC.

Besides that, several communication-related data structures, such as SDQ, EQ, MPQ, etc., are allocated in kernel module, and then are mapped into user space, in order to support user-level operations without system calls. In current implementation, the NIC is attached to PCI-E interface. While the compute node is organized as a multi-CPU SMP system, which results in different access latencies via PCI-E among different CPUs. In order to reduce memory access latency, libglex provides an efficient mechanism to permit allocating the above data structures in the memory space of the CPU directly connected with NIC.

Another key feature in libglex is that all the communication operations are non-blocking, in order to better support overlapping communication and computation in upper level software.

3.2 Kernel management module and interface

The gdev kernel module manages NIC hardware resources, including VPs and ATT. Memory registration is implemented in gdev too. The gdev provides libglex and management utilities the system call interface, and also provides kernel-level APIs to other kernel modules such as gnet and Lustre lnet module. The kernel-level API provides nearly the same data

transmission functions as user-level APIs, except the following two key points. Firstly, MP packet adopts active message mode, which permits kernel users to process MP packet via interrupt handler. Secondly, kernel-level RDMA operation supports physical address mode to bypass the address translation table, because most data buffers in kernel module are allocated in continuous physical addresses, and should be regarded as trusted objects.

The MilkyWay compute node exploits GPU or MIC to accelerate computation, so the kernel module provides special mechanism for such accelerators. We implemented GPU-Direct within memory registration codes using a kernel patch provided by NVIDIA to support zero-copy RDMA of CUDA pinned memory. Direct NIC operation from MIC is also assisted from the kernel module.

The gnet module is running above gdev to support TCP/IP protocol, so that traditional network services and MPI above TCP/IP can be running on TH series interconnect network. The gnet is interrupt-driven and utilizes MP and RDMA to transfer network frames contained in Linux sk_buff. Since data in sk_buff is allocated using kmalloc(), RDMA in gnet uses PCI address of data directly to bypass ATT.

3.3 Network booting

The MilkyWay-2 computer system implements PXE specification to support network booting for compute node. The PXE specification adopts DHCP and TFTP protocols based on TCP/IP. The GLEX2 system provides an NIC-based PXE driver named gpxenet, which uses hybrid MP and RDMA communication mechanism to implement the same packet transmission protocol as gnet. With gpxenet, the diskless compute node could be booted via TH interconnect network.

4 GLEX2-based Nemesis Netmod

The MilkyWay-2 MPI, which we named MPICH2-GLEX2, is an optimized port of the Argonne National Laboratory's MPI implementation: MPICH2 [12]. MPICH2 adopts hierarchical structure with the lowest channel level performing data transmission based on specific interconnect interface. One of the most popular channel implementation is Nemesis channel [13].

The design of Nemesis specially emphasizes on a highly optimized on-node messaging system and a multi-method capable framework for implementing Network modules (Netmod). Nemesis provides a simple callback-function approach to interface with Netmods. The MPICH2-GLEX2 mainly extends a Netmod for Nemesis, providing high performance communication with hybrid MP and RDMA data transfer.

4.1 Message passing protocol

There are two data transfer protocols in MPICH2: eager and rendezvous. Eager protocol is used for reducing the message passing latency. Sender assumes that the receiver can handle the message and starts sending message data immediately. If there is no matched receiving request, data will be buffered in receiver. In rendezvous protocol, a scout packet will be sent to receiver first, asking for permission to send message data. Only when the matched receiving request is found can data be transferred, but this may avoid copying in message passing.

In MPICH2-GLEX2, eager protocol is implemented with two kinds of channels: shared RDMA channel and exclusive RDMA channel. While in rendezvous protocol, we've implement zero-copy data transfer.

4.2 Shared RDMA channel

MP has similar semantics with MPI message, but it can only transfer small data, long message must be transferred using RDMA. RDMA GET is one-side communication initiated by data receiver. When several senders send messages to one receiver, receiver can decide where to store these messages to avoid contention by using RDMA GET.

We designed a Shared RDMA (SR) channel which can be used for communication with all other processes. For each SR channel, we create a GLEX2 endpoint, allocate a range of memory and register it using this endpoint. The reserved memory will be divided equally into blocks for sending and receiving data separately, and each sending block corresponds to an event. Sender divides MPI message into blocks according to the length of the sending block. For each message block, sender selects an idle sending block and copies data into the block. Then sender sends an MP request to receiver. When receiver got the MP request, it selects an idle receiving block, and starts data transfer using RDMA GET which will trigger events in both sender and receiver on completion. Receiver will copy data from receiving block to message buffer on detecting the event, while sender will reuse the sending block when the associated event is triggered. Copying and RDMA GET of these message blocks will be performed in pipelining to improve bandwidth.

In SR channel, MPs are queued in the MPQ of the endpoint. However, MPQ is likely to overflow. There must be a flow control protocol to prevent the loss of MP. MPI assumes an all-to-all communication pattern, where each MPI

process can potentially receive incoming messages from every other process. We designed a dynamic credit-based flow control protocol in SR channel [14,15]. Only when sender has credits, can it send MPs to receiver. Credits between processes can be increased, especially for the processes which have intensive communication between them, to reduce the waiting time for credit backfill.

The capacity of MPQ in SR channel is the sum of two parts: the reserved initial credits for every process, and the credits pool for enlarging credits between processes. On frequent and large message passing, sender may be blocked to wait for credit backfill, in this case sender will set "credit increase" flag in the follow up MP requests. Upon receiving this flag, receiver gets some new credits from credits pool, and backfill credits to sender using piggybacking or credit MP, thus credits between two processes are increased.

4.3 Exclusive RDMA channel

SR channel has higher message latency because a MP request should be sent before starting RDMA GET. RDMA PUT has lower latency, but RDMA PUT is one-side communication initiated by sender. It may lead to conflict if several senders send to the same receiver without coordination. We use the RDMA based approach proposed in [15]. An Exclusive RDMA (ER) channel will be created between two processes for communication. Ring buffers are allocated and registered in ER channel, while RDMA PUT with event is used for data transfer. For small data, we use immediate RDMA PUT. The number of ring buffers is the initial credit for flow control between sender and receiver. Credit backfill is performed with piggybacking or credit MP as mentioned above.

4.4 Zero-copy data transfer

Nemesis features a long message transfer (LMT) protocol that facilitates implementation of zero-copy transfers for Netmods interfacing to networks that support these types of operations. In LMT protocol, sender and receiver will synchronize first using request to send (RTS) packet, which can be used to carry the address and length of send buffer. In MPICH2-GLEX2, LMT protocol is implemented using the RDMA GET. The completion of RDMA GET triggers events in the sender and receiver. Receiver completes the receiving of a message upon detecting the event, while sender can reuse the message buffer. Because message buffers must be registered before RDMA operations, we use a user space registration cache [16,17] to reduce the overhead of frequent memory registration.

4.5 Processing of out of order messages

The TH Express interconnect adopts a load-balanced routing based on hierarchical look-up table. In this routing protocol, both the MP and RDMA packets maybe transmitted out of order through the network, and could not be expected in the same sequence between sender and receiver. However, MPI specification requires an orderly message passing semantics, so that all the ER, SR channels and zero-copy rendezvous protocols must recover original message sequence for the out-of-order packets.

For out-of-order MP packets, we embed a sequence number for each MP packet, and recover the correct sequence in the receiver side. For out-of-order RDMA packets, we exploit the event mechanism of checking the arrival event to judge the completeness of a RDMA transaction. Besides that, we use a sequential data structure to recover out-of-outer RDMA communications as the submission sequence.

4.6 Hybrid channel approach

ER channel has the lowest message latency at the cost of more resource consumption; on the contrary, SR channel has smaller resource usage. Some MPI parallel applications show a nearest-neighbour communication pattern [18], meaning that each MPI process communicates frequently with several other MPI processes. In MPICH2-GLEX2, ER channels are created between these processes to implement optimized message passing bandwidth and latency. On the contrary, communication with other processes uses the SR channel. In current NIC implementation, only one EQ and one MPQ are polled for detecting the status of all the ER channels and SR channel, which is benefit for improving the scalability. For most parallel applications, the memory consumption of MPI system is also limited.

4.7 Offloaded collective communication

Offloaded optimization of collective communication utilizes the override interfaces of MPICH communicator. Implementation of offloaded collective communication needs three phases: initialization, operation sequence posting, and testing for completion. Initialization is performed on the creation of communicator, a new VP is used for offloaded collective communication, and tree topology is constructed with the information of group members. Each process knows who are its parent and children in the tree, and where to get the addresses of VP and the reserved internal buffers of them. In current implementation, the tree topology can be k-nominal

tree or *k*-ary tree.

Because the "root" parameter in the calling of some collective communication routines may not be the root of the tree constructed in the initialization phase, messages need to be sent to the root of the tree before starting the offloaded collective communication.

In MPI specification, only Barrier operation has the synchronization semantics. When collective routines are called continuously, root should get acknowledgement from children before starting next operation to coordinate the use of internal buffers for collective communication. In real parallel applications, these acknowledgement packets can be transferred in background overlapping with the computation.

5 Evaluation

In this section we present some preliminary performance results of GLEX2 and MPICH2-GLEX2 in the latest network interface for TH Express interconnect. Each compute node in the testbench is equipped with two Intel Xeon E5-2660@2.2 GHz processors with 32 GB memory. Each node also has a NIC card attached to a CPU via PCI-E, therefore, the other CPU has to transfer communication request via QPI interface. The microbenmarks used include benchmark programs we coded using GLEX2 interfaces and OSU MPI benchmarks.

As mentioned above, the VPs in NIO can work in PIO or DMA mode, and the link can switch between connect and connectless mode. We performed several experiments under different mixes.

Table 1 lists the minimum point-to-point latency for immediate RDMA PUT, RDMA PUT and MP descriptors. We use event mechanism to detect the completion of RDMA operation. Among them, conectless immediate RDMA PUT in PIO mode has the lowest latency (0.89 μs). The reason for that is the data of immediate RDMA PUT is contained in the descriptor, so the NIC can start data transfer directly after receiving descriptor. In the connect mode, the latency will be increased to 1.31 μs owing to the cost in setting up connection. For the VPs under DMA mode, the latency for each test will be increased by 300 ns to 400 ns. The extra cost is mainly introduced by reading descriptor across PCI-E.

Table 1 Minimum point-to-point latency in GLEX2 /μs

		IMM_PUT (EVT)	PUT (EVT)	MP
PIO	Connectless	0.89	1.30	1.30
	Connect	1.31	1.35	1.64
DMA	Connectless	1.29	1.59	1.40
	Connect	1.64	1.67	1.71

For MP descriptor, sender has to submit a fixed-length descriptor, and receiver has to receive a fully-length packet into MPQ no matter how long the valid data payload. Therefore, the minimum latency for MP is a little larger than that of immediate RDMA PUT.

Figure 2 illustrates the unidirectional bandwidth and bidirectional bandwidth for RDMA PUT. In the unidirectional bandwidth tests, RDMA PUT reaches peak bandwidth of 11.7 GB/s after the data size extends 4 KB. In the bidirectional bandwidth tests, both the VPs under PIO and DMA mode achieve nearly the same peak bandwidth. After the data size extends 8 KB, the peak bandwidth is 20.6 GB/s, which is almost twice of the results under unidirectional test.

Table 2 lists the minimum point-to-point latency in MPI. The ER channel uses immediate RDMA PUT for small-size messages, whose minimum latency is 1.18 μs. Compared with latency of 0.89 μs in tests of GLEX2, MPI introduces about 300 ns delay. The main cause is that both sender and receiver have a copy cost in the ER channel implementation. Besides that, the progress engine in MPICH may also introduce some cost. We plan to optimize it in the future. SR channel uses MP for small-size message and achieves a minimum latency of 1.26 μs, which is neatly the same as the result in GLEX2 tests. The result demonstrates that MPI almost does

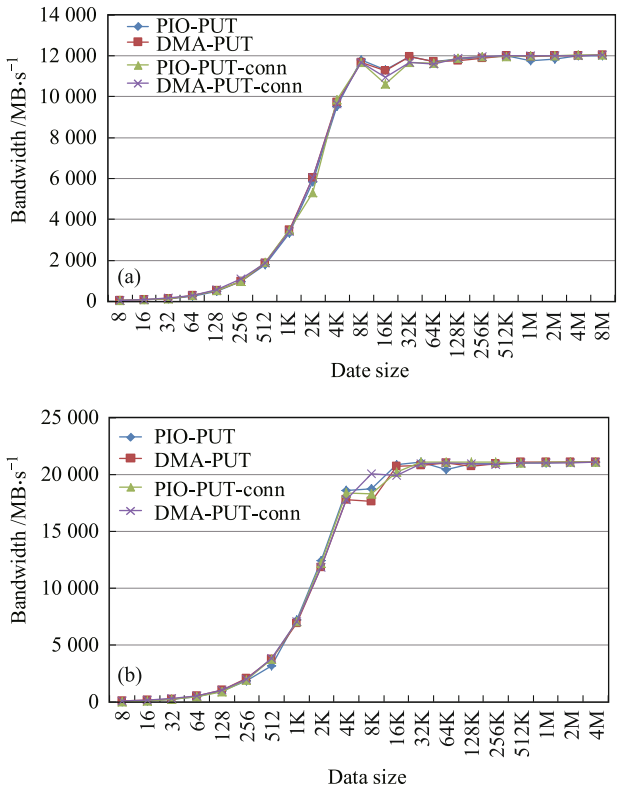

Fig. 2 Unidirectional bandwidth and bidirectional bandwidth for RDMA PUT in GLEX2. (a) Unidirectional bandwidth; (b) Bidirectional bandwidth

Table 2 Minimum point-to-point latency in MPI /μs

		ER Channel	SR Channel
PIO	Connectless	1.18	1.26
	Connect	1.53	1.61
DMA	Connectless	1.40	1.38
	Connect	1.76	1.69

not introduce any cost for small-size message in SR channel.

Figure 3 illustrates the achievable bidirectional bandwidth in MPI under PIO and DMA VPs. From the results, we can see that the ER channel is always superior to the SR channel no matter which type of VP is used. The main reason is that the ER channel uses RDMA PUT directly, while SR channel uses RDMA GET, which introduces one more immediate MP message. However, as mentioned above, SR channel consumes relatively smaller resource, so we introduce hybrid channel implementation to balance the efficiency and overhead.

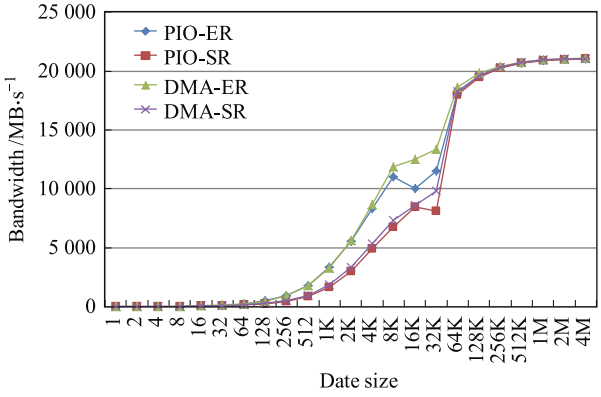

Fig. 3 Bidirectional bandwidth in MPI

6 Related works

User-level operations and zero-copy data transfer are two key state-of-the-art techniques for high performance communication. Most existing interconnect systems adopt these techniques, such as IBM BlueGene/Q [19,20], Fujitsu Tofu [21], Cray Gemini [22], and InfiniBand [9]. Through the user-level operation and improved logic design, we achieve rather small point-to-point latency in TH Express interconnect.

Reliability becomes more and more crucial for future large-scale computer systems, especially for highly complicated interconnect [23]. Gemini interconnect does not provide end-to-end reliability in hardware, while resorts to software protocol to maintain reliability. In InfiniBand, a one-to-one connection is required between two processes to deliver end-to-end reliable data transfer, which requires more resources at large scale. In the new generation of TH Express interconnect, we introduce a reliable end-to-end communication in NIC, while simplify the implementation of scalable message passing services. The efficiency of hardware reliability will be further evaluated in the future.

Hardware assisted collective operations have been investigated extensively in literature. Some of the recent approaches in InfiniBand are [10,24,25], which optimize collective communication at different layers in the interconnect fabric. In PERCS [26], a special Collective Acceleration Unit is used to speed up collective operation. Our previous work [4] demonstrated that we can achieve good performance results using NIC-assisted collective operations. We believe it is necessary to explore hardware-software co-design to achieve the best result, especially exploiting the hardware offload collective to overlap computation and communication.

7 Conclusions

In this paper, we introduced the state-of-the-art of the TH Express interconnect networks. With the support from user-level communication and zero-copy RDMA, our implementation of network interface and message passing services can deliver scalable high performance communication. We also described the design of hardware assisted collective mechanism which can be used to accelerate barrier, broadcast, and other collective operations.

There are several directions for our future work. The study and evaluation of improved interconnect topology and routing protocol is one of the most important directions for future larger-scale MilkyWay systems. Besides that, an improved network interface and software stack is also crucial to support more and more parallelism within compute node.

Acknowledgements This work was partially supported by the National High-tech R&D Program of China (863 Program) (2012AA01A301, 2013AA014301, 2013AA01A208), and by the National Basic Research Program of China (973 Program) (2011CB309705), and by the National Natural Science Foundation of China (Grant Nos. 61120106005, 61303063 and 61272482).

References

1. Top500, http://www.top500.org, 2013
2. Liao K X, Xiao Q L, Yang Q C, Lu T Y. MilkyWay-2 supercomputer system and application. Submitted to Frontiers of Computer Science, 2013
3. Pritchard H, Gorodetsky I, Buntinas D. A ugni-based mpich2 nemesis network module for the cray xe. In: Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface. 2011, 110–119

4. Xie M, Lu Y, Liu L, Cao H, Yang X. Implementation and evaluation of network interface and message passing services for Tianhe-1a supercomputer. In: Proceedings of the 19th IEEE Annual Symposium on High Performance Interconnects. 2011, 78–86
5. Chun B N, Mainwaring A, Culler D E. Virtual network transport protocols for myrinet. IEEE Micro, 1998, 18(1): 53–63
6. Araki S, Bilas A, Dubnicki C, Edler J, Konishi K, Philbin J. User-space communication: a quantitative study. In: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (CDROM). 1998, 1–16
7. Bhoedjang R A, Ruhl T, Bal H E. User-level network interface protocols. Computer, 1998, 31(11): 53–60
8. Schoinas I, Hill M D. Address translation mechanisms in network interfaces. In: Proceedings of the 4th International Symposium on High-Performance Computer Architecture. 1998, 219–230
9. InfiniBand Architecture Specification: Release 1.0. InfiniBand Trade Association, 2000
10. Graham R L, Poole S, Shamis P, Bloch G, Bloch N, Chapman H, Kagan M, Shahar A, Rabinovitz I, Shainer G. Overlapping computation and communication: Barrier algorithms and connectx-2 core-direct capabilities. In: Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. 2010, 1–8
11. Kandalla K, Subramoni H, Vienne J, Raikar S P, Tomko K, Sur S, Panda D K. Designing non-blocking broadcast with collective offload on infiniband clusters: A case study with hpl. In: Proceedings of the 19th IEEE Annual Symposium on High Performance Interconnects. 2011, 27–34
12. MPICH2: High-performance and Widely Portable MPI. http://www.mcs.anl.gov/research/projects/mpich2/
13. Buntinas D, Goglin B, Goodell D, Mercier G, Moreaud S. Cache-efficient, intranode, large-message mpi communication with mpich2-nemesis. In: Proceedings of the 2009 International Conference on Parallel Processing. 2009, 462–469
14. Lauria M, Pakin S, Chien A. Efficient layering for high speed communication: Fast messages 2. x. In: Proceedings of the 7th International Symposium on High Performance Distributed Computing. 1998, 10–20
15. Liu J, Panda D K. Implementing efficient and scalable flow control schemes in MPI over infiniband. In: Proceedings of the 2004 International Parallel and Distributed Processing Symposium. 2004, 183b
16. Tezuka H, O'Carroll F, Hori A, Ishikawa Y. Pin-down cache: a virtual memory management technique for zero-copy communication. In: Proceedings of the 1998 Symposium on Parallel and Distributed Processing. 1998, 308–314
17. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE, 2013
18. Vetter J S, Mueller F. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. Journal of Parallel and Distributed Computing, 2003, 63(9): 853–865
19. Chiu G. The IBM blue gene project. IBM Journal of Research and Development, 2013, 57(1): 1–6
20. Chen D, Eisley N A, Heidelberger P, Senger R M, Sugawara Y, Kumar S, Salapura V, Satterfield D L, Steinmacher-Burow B, Parker J J. The IBM blue gene/q interconnection fabric. IEEE Micro, 2012, 32(1): 32–43
21. Ajima Y, Takagi Y, Inoue T, Hiramoto S, Shimizu T. The tofu interconnect. In: Proceedings of the 19th IEEE Annual Symposium on High Performance Interconnects. 2011, 87–94
22. Alverson R, Roweth D, Kaplan L. The gemini system interconnect. In: Proceedings of the 18th IEEE Annual Symposium on High Performance Interconnects. 2010, 83–87
23. Schroeder B, Gibson G A. Understanding failures in petascale computers. In: Journal of Physics: Conference Series. 2007, Article 012022
24. Graham R L, Poole S, Shamis P, Bloch G, Bloch N, Chapman H, Kagan M, Shahar A, Rabinovitz I, Shainer G. Connectx-2 infiniband management queues: first investigation of the new support for network offloaded collective operations. In: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. 2010, 53–62
25. Subramoni H, Kandalla K, Sur S, Panda D K. Design and evaluation of generalized collective communication primitives with overlap using connectx-2 offload engine. In: Proceedings of the 18th IEEE Annual Symposium on High Performance Interconnects. 2010, 40–49
26. Arimilli B, Arimilli R, Chung V, Clark S, Denzel W, Drerup B, Hoefler T, Joyner J, Lewis J, Li J. The percs high-performance interconnect. In: Proceedings of the 18th IEEE Annual Symposium on High Performance Interconnects. 2010, 75–82

Zhengbin Pang received the BS, MS, and PhD degrees in computer science from National University of Defense Technology (NUDT), China. He is a professor in College of Computer, NUDT. His research interests include parallel and distributed computing, and high performance computer systems.

Min Xie is a professor in College of Computer at National University of Defense Technology (NUDT), China. His research interests include high-speed interconnects, system software and parallel and distributed computing. He has a PhD in computer science from NUDT.

Jun Zhang received the MS degree in computer science from National University of Defense Technology (NUDT), China. Currently he is an assistant professor at the university. His research interests include high speed communication and ASIC design.

Yi Zheng received the PhD degrees in computer science from National University of Defense Technology (NUDT), China. Currently he is an associate professor at the university. His research interests including high performance computer architecture and high performance networks.

Guibin Wang received the BS, MS, and PhD degrees from National University of Defense Technology (NUDT), China in 2004, 2007, and 2011, respectively. Currently, he is an assistant professor in

College of Computer, NUDT. His research interests include high-performance computer systems, heterogeneous parallel systems.

Dezun Dong received the BS, MS, and PhD degrees from the National University of Defense Technology (NUDT), China in 2002, 2004, and 2010, respectively. Currently, he is an associate professor in College of Computer, NUDT, China. His research interests include high-performance computer systems, distributed computing, and wireless networks. He is a member of ACM and IEEE.

Guang Suo received his BS in computer science from National University of Defense Technology (NUDT), China in 2003, and received his MS and PhD in computer science from NUDT in 2005 and 2009, respectively. He is an assistant professor in Institute of Computers, NUDT. He has played an important role in the implementation and optimization of MPI library of MilkWay supercomputers. His research interests are in parallel copmuting, operating system, and HPC runtime systems.