Cao Z, Liu XL, Li Q *et al.* An intra-server interconnect fabric for heterogeneous computing. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(6): 976–988 Nov. 2014. DOI 10.1007/s11390-014-1483-0

An Intra-Server Interconnect Fabric for Heterogeneous Computing

Zheng Cao (曹 政), *Member*, *CCF*, *ACM*, Xiao-Li Liu (刘小丽), *Member*, *CCF*, *ACM*
Qiang Li (李 强), *Member*, *CCF*, *ACM*, Xiao-Bing Liu (刘小兵), *Member*, *CCF*, *ACM*
Zhan Wang (王 展), *Student Member*, *CCF*, and Xue-Jun An (安学军), *Member*, *CCF*, *ACM*

State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China

E-mail: {cz, liuxiaoli, liqiang, liuxiaobing, wangzhan, axj}@ncic.ac.cn

Received December 16, 2013; revised July 4, 2014.

Abstract With the increasing diversity of application needs and computing units, the server with heterogeneous processors is more and more widespread. However, conventional SMP/ccNUMA server architecture introduces communication bottleneck between heterogeneous processors and only uses heterogeneous processors as coprocessors, which limits the efficiency and flexibility of using heterogeneous processors. To solve this problem, this paper proposes an intra-server interconnect fabric that supports both intra-server peer-to-peer interconnection and I/O resource sharing among heterogeneous processors. By connecting processors and I/O devices with the proposed fabric, heterogeneous processors can perform direct communication with each other and run in stand-alone mode with shared intra-server resources. We design the proposed fabric by extending the de-facto system I/O bus protocol PCIe (Peripheral Computer Interconnect Express) and implement it with a single chip cZodiac. By making full use of PCIe's original advantages, the interconnection and the I/O sharing mechanism are light weight and efficient. Evaluations that have been carried out on both the FPGA (Field Programmable Gate Array) prototype and the cycle-accurate simulator demonstrate that our design is feasible and scalable. In addition, our design is suitable for not only the heterogeneous server but also the high density server.

Keywords heterogeneous system, interconnection, I/O virtualization, PCI-express

1 Introduction

In order to achieve ultra-high power efficiency, heterogeneous servers mixing many-core processors (such as GPGPU and Xeon Phi) have been widely used in HPC (high performance computing) systems[1-2]. Recently, not only in HPC, but also in datacenter, heterogeneous architectures are becoming more widely used. Research and industry are showing interest in building servers with heterogeneous processors, even using mobile or embedded processors[3]. Guevara *et al.*[4] showed that the proper mixing of Atom and Xeon processors can achieve greater power efficiency, and Zapater *et al.*[5] introduced a mixture of X86 and SPARC processors. Suneja *et al.*[6] used GPGPU to accelerate cloud management tasks. Today, heterogeneity mainly lies at the server level. With the increasing diversity of application needs and hardware, we can expect that the degree of heterogeneity will also keep increasing and heterogeneity in a single server will become common in the future. However, the conventional server architecture, such as SMP/ccNUMA, is not well suited for such heterogeneity.

1) *Limited Communication Efficiency Between Heterogeneous Processors and the Limited Number of Heterogeneous Processors.* Heterogeneous processors are used only as coprocessors, and all of the data exchanging between heterogeneous processors must be forwarded by a master CPU (usually general-purpose CPU). The forwarding process introduces high communication overhead and imposes constraints on the intra-server switching bandwidth.

2) *Heterogeneous Processors Cannot Use Intra-Server I/O Resources Directly.* A single physical I/O device can belong only to one operating system domain (master CPU), thus it cannot be directly operated by heterogeneous processors, neither those working as coprocessors nor those running their own operating systems.

To overcome the above drawbacks, we propose a *network-centric server architecture* shown in Fig.1 to support heterogeneous computing, with features defined as follows.

• Support both the coprocessor and stand-alone heterogeneous processing modes. In the network-centric server, depending on the heterogeneous processor's

Regular Paper
This work was supported by the National Natural Science Foundation of China under Grant No. 61100014.

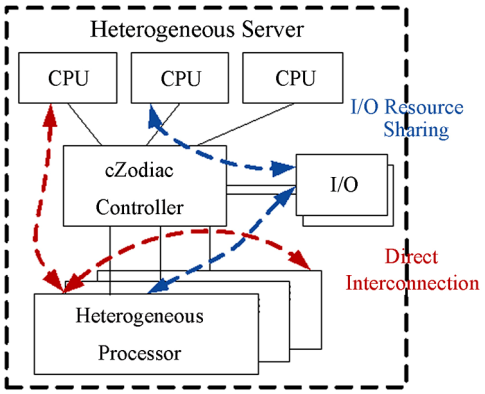


Fig.1. Network-centric server architecture.

type, it can attach to general purpose processors either working as coprocessors or running their own operating system as hosts.

• To improve intra-server communication efficiency, direct interconnections between all of the components (including processors and I/O devices) are implemented.

• To improve the sharing efficiency of intra-server I/O resources, especially network devices for inter-server communication, we provide a hardware-based I/O resource sharing mechanism.

As shown in Fig.1, all of these features are implemented with a central controller cZodiac, which supports intra-server global addressing space and integrates the function of interconnection and intra-server resource pooling into a single fabric. PCIe (PCI-Express), the de-facto intra-server system I/O bus, plays an important role in our network-centric server. Especially in terms of supporting the dual heterogeneous processing mode, every port of cZodiac can be configured as either a PCIe downstream port① in coprocessor mode or a PCIe upstream port/endpoint① in stand-alone mode.

Regarding the intra-server interconnection, we propose an intra-server network design which makes full use of PCIe point-to-point communication features. Regarding the I/O resource sharing, we propose a hardware mechanism of building an I/O pool with PCIe SR-IOV② (Single-Root I/O Virtualization) or multi-function devices, so that resources in the pool can be shared by all of the intra-server processors.

The rest of this paper is organized as follows. Section 2 introduces key issues of designing the interconnect fabric. Section 3 introduces the proposed PCIe compliant interconnection. Section 4 introduces the mechanism of I/O resource pooling. Section 5 introduces the micro-architecture of cZodiac. Section 6 shows the performance evaluations on the prototype of cZodiac. Section 7 introduces related work, and the last section presents our conclusions.

2 Key Issues

2.1 Extending PCIe to Host-to-Host Network

PCIe has been integrated into many processors, such as Xeon, Xeon Phi, GPGPU, and ARM, and is the de-facto I/O bus today. Thus, PCIe is a suitable fabric for unifying the processors' interconnection interface and integrating functions of the interconnection and I/O expansion. What is more, both the communication hierarchy and the server architecture can be simplified.

However, the standard PCIe was originally designed for host-to-I/O interconnection. For two reasons, the original PCIe is not suitable for multi-root environment, as shown in Fig.2(a).

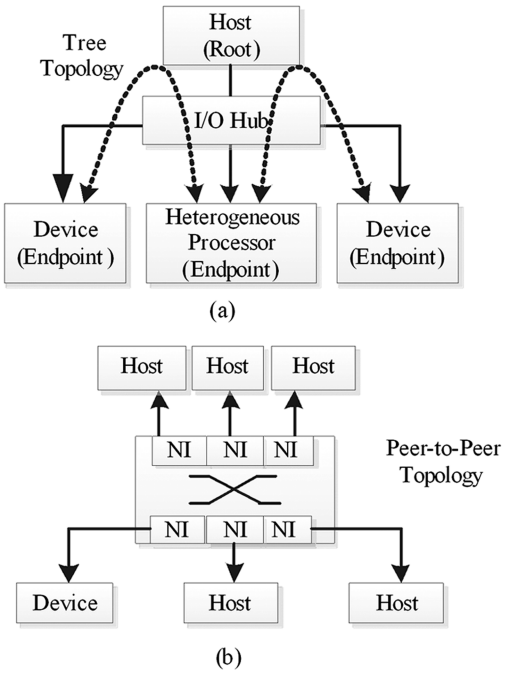


Fig.2. PCIe interconnection system. (a) Legacy PCIe system. (b) Peer-to-peer PCIe system.

1) *Limited Scalability.* PCIe is designed for the single root system, which means only the heterogeneous processors and devices in the same OS domain can communicate with each other. In this case, the heterogeneous processor can only be used as the co-processor, and the

①PCI-SIG. PCI Express base 3.0 specification, Nov. 2010. http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r3.0_10Nov10.pdf, Sept. 2014.

②PCI-SIG. Single root I/O virtualization and sharing 1.1 specification, Jan. 2010. http://www.pcisig.com/memers/downloads/specifications/iov/sr-iov1_1_20Jan10_cb.pdf, Sept. 2014.

number of heterogeneous processors is limited by the physical address space.

2) *Limited Peer-to-Peer (P2P) Communication.* Some work is enabling the peer-to-peer communication between two endpoint devices. For example, GPUDirect[③] can achieve direct communication between GPUs by accessing the same memory block on the master CPU. However, it is still a single node optimization technique that requires sequential accessing to the master's main memory. In addition, not all the heterogeneous processors have the DMA function. Using only load/store operations for peer-to-peer communication is very low efficient.

Facing above limitations, we propose a peer-to-peer PCIe interconnection system that enhances the peer-to-peer communication. As shown in Fig.2(b), we introduce the idea of network into our peer-to-peer architecture and implement a network interface (NI) for each host to perform the peer-to-peer communication. All these NIs are interconnected with a central crossbar.

To analyze the benefit of the proposed peer-to-peer architecture, we build performance models for both the legacy and peer-to-peer architectures. We define the forwarding delay of a PCIe root complex as $T_{\rm rc}$, the transmit delay between two endpoints or an endpoint and the root as $T_{\rm p2p}$, the bandwidth of each link as BW, the number of processors as n, and the average message length in the system as L.

In the legacy PCIe architecture, the switching bandwidth is only BW (the link bandwidth between PCIe switch and root). In this case, the average intra-server message delay is:

$$T = T_{\rm p2p} + T_{\rm rc} + T_{\rm p2p} + L/BW.$$

In a peer-to-peer architecture, the switching bandwidth equals the switching bandwidth provided by the central crossbar (the peak switching bandwidth is $n \times BW$). If the switching throughput of crossbar is ε, then the average intra-server message delay is:

$$T' = T_{\rm p2p} + L/(\varepsilon \times n \times BW), \quad \varepsilon \in \left[\frac{1}{n}, 1\right].$$

Then, the performance improvement G can be defined as:

$$G = T/T'.$$

Because ε is larger than $1/n$ under most communication patterns, the performance improvement is in direct proportion to n and the limit value of G is:

$$\lim G = \begin{cases} (2 + T_{\rm rc}/T_{\rm p2p}), & \text{if } L \to 0, \\ \varepsilon \times n, & \text{if } L \to \infty. \end{cases} \quad (1)$$

As shown in (1), in terms of latency, the peer-to-peer architecture can achieve at least two times performance improvement for short message, while at most n times for long message.

To make the network protocol lightweight, we define the interconnection protocol by extending the transaction layer protocol of PCIe. In addition, this protocol is still compatible with the standard PCIe protocol. The detailed design of PCIe interconnection network is given in Section 3.

2.2 Sharing I/O Devices Between Multiple Hosts

Commercial devices including SR-IOV devices are designed to the single host (root) system, in other words, one device can only accept ID number and physical addresses from one host. As shown in Fig.3(a), once BARs (basic address registers) of the device have been configured by host 0, the configuration from host 1 will conflict with the configuration from host 0. Even each virtual function in an SR-IOV device has its own BAR, such confliction still cannot be avoided, because all BARs of virtual devices belong to a single consecutive space.

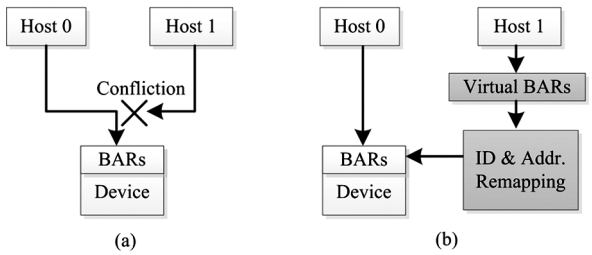


Fig.3. Isolation of I/O spaces among multiple hosts.

To efficiently share the devices, we propose an ID and I/O space remapping mechanism in hardware level. As shown in Fig.3(b), we implement virtual BARs for BARs in the device and implement device ID and physical address remapping between them. In this way, host 1 can own its virtual devices. From the device's view, it is still controlled by a single host (host 0).

In addition, by sharing devices among multiple hosts, the server can achieve several new features, including:

1) *Decoupling of Processors and I/O Devices.* Any host can directly communicate with any device/virtual device.

2) *Elastic I/O Bandwidth Allocation.* On-demand inter-server I/O bandwidth allocation can be achieved

[③]Mellanox Inc., NVIDIA GPUDirect™ technology-accelerating GPU-based systems, Jan. 2010. http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf, Sept. 2014.

by dynamically adding/removing virtual devices to/from hosts.

3) *Parallel Intra-Server Switching.* The network card such as Intel 82599 often contains an embedded L2 switch that performs fast packet switching between virtual devices. By assigning virtual devices belonging to one NIC to different hosts, the switching between hosts can be performed inside NIC. Therefore, in addition to the switching capacity of peer-to-peer PCIe network, all the internal switching capacity inside network cards can be used for intra-server communication.

3 PCIe Compliant Interconnection

Making full use of PCIe as host-to-host interconnection fabric requires carefully designed network interface and internal switching architecture. In addition, to integrate several network interfaces into a single chip, it must be lightweight.

3.1 Network Interface

In terms of the design of network interface, three aspects must be taken into consideration: the user-level interface, the hardware virtualization, and the communication primitive.

The user-level interface, also known as the OS bypass, is the key technology to reduce communication latency in the software layer. In our design, communication processes operate an NIC (network interface controller) with doorbell and their own registered QPs (queue pairs), including send queue, receive queue, and completion queue. Thus each process owns a "virtual network interface."

However, such a "virtual network interface" accepts only the host's physical address, thereby processes on a VM (virtual machine) cannot use the user-level interface. To use cZodiac in the virtualization environment, the network interface supports hardware virtualization and is compliant with PCIe SR-IOV specification.

Finally, the communication primitive determines the communication function offloaded by the network interface. Corresponding to the MPI (both eager and rendezvous models) and PGAS programming model, we define four primitives: DAP (direct access packet), NAP (no address packet), RDMA PUT, and RDMA GET. For intra-server interconnection, the network interface must be lightweight and efficient. Thus, the primitives are connectionless, and memory protection is achieved by performing the magic number matching between the sender and the receiver.

3.1.1 DAP

DAP is designed to support GAS (global addressing space). Maintaining cache coherence between different kinds of processors is difficult. However, the facilitation of intra-server resource sharing requires GAS. Each processor exposes part of its main memory to other processors, and all of the memory regions are addressed into a single space. DAP is initialized with a load/store operation and transmits data of very small size with very low overhead, as shown in Fig.4(a).

3.1.2 NAP

NAP is a type of RDMA operation that can be used to transmit data of less than 2 KB. Because each QP allocates a dedicated receive ring for NAP in the main memory, NAP does not carry any destination information except for the identification of the QP at the receiver side, as shown in Fig.4(b). Each entry of the receive ring occupies 2 KB memory and has a corresponding entry in the completion queue. If the message is very small, then data can be written directly into the DMA descriptor, and the local DMA read is not needed. We call it as NAP immediate, while the normal NAP as NAP indirect.

NAP removes the step of negotiating destination address with the receiver side, but introduces an extra step of copying data from the ring to the application's memory; thus, it is suitable for transmitting only small amounts of data.

3.1.3 RDMA PUT/GET

RDMA PUT can transmit large messages and place them directly into the user space of the application on the receiver side. The destination information, including discrete destination addresses and lengths, is obtained from a handshake between the sender and the receiver through the use of NAP, as shown in Fig.4(c). After the handshake, all memory pages to be written by RDMA PUT are pinned. The improvement of communication efficiency requires that RDMA operations support a huge memory page from 4 KB to 128 MB while not requiring any constraint on address alignment.

Based on RDMA PUT, we implement RDMA GET. The sender directly transmits the RDMA GET descriptor to the receiver. Then, the receiver translates the descriptor into the local RDMA PUT descriptor by exchanging source and destination information, and finally the receiver executes the RDMA PUT procedure, as shown in Fig.4(d).

With the implementation of RDMA GET with the RDMA PUT primitive, all RDMA transactions between the sender and the receiver are remote write operations. Because the implementation of remote write operation is based on the PCIe posted transaction, and the interconnection protocol is simplified by releasing the constraint on the number of PCIe tags used in PCIe

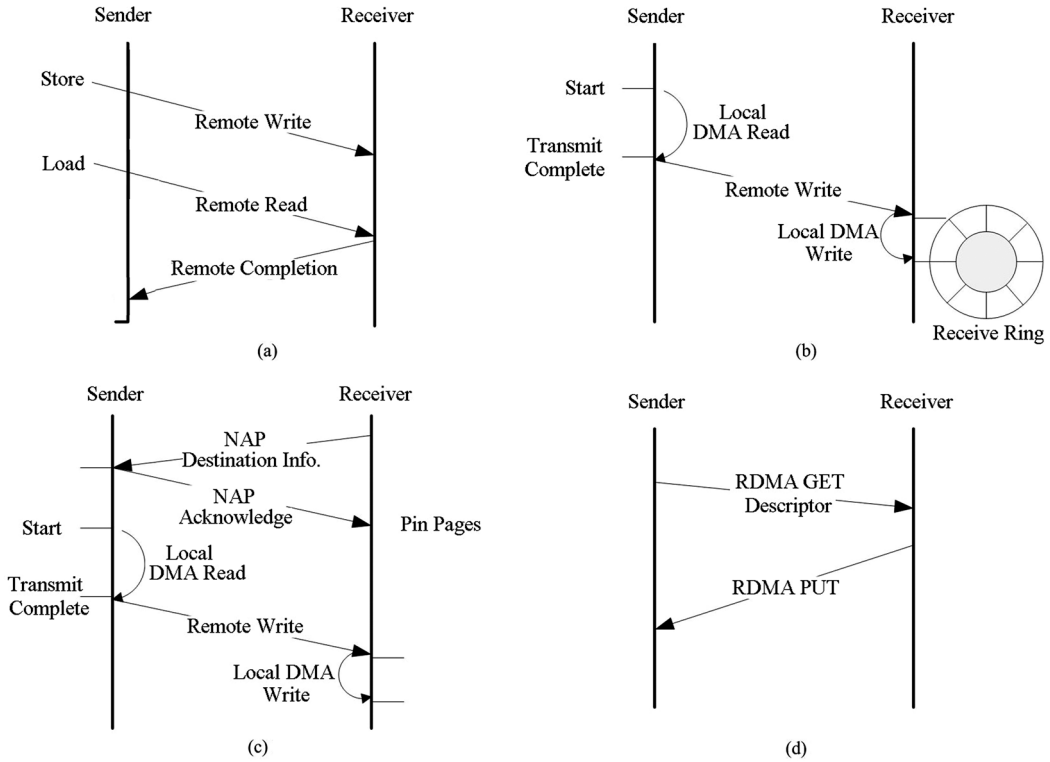


Fig.4. Communication primitives. (a) DAP. (b) NAP. (c) RDMA PUT. (d) RDMA GET.

non-posted transactions. In addition, communication traffic is reduced by the removal of frequent remote request-response interaction.

3.2 Internal Switching

3.2.1 Network Packet Format

To minimize the protocol conversion overhead, we define all the network packet formats with an extension to the PCIe request formats. The formats of the DAP load and DAP completion (not shown in Fig.5) are much the same as the formats of the PCIe read request and PCIe completion. The DAP load redefines some reserved fields in the PCIe read request as identifications of the source processor and the destination processor, while the DAP completion uses only the identification of the destination processor.

The header formats of other network packets are given in Fig.5. The fields in blue are defined in the same way as ones in the PCIe specification, while the fields in white are defined as:

1) *SubType*: the type of network packet, which could be RDMA GET, RDMA PUT, or NAP;

2) *Dest CPU*: identification of the destination CPU (regarding the server's scalability, all of the reserved bits shown in Fig.5 can be used as the *Dest CPU*);

3) *Dest VF*: identification of the destination virtual function in the destination network interface;

4) *Dest QP*: identification of the destination queue pair in the destination virtual function;

5) *QP Magic Num*: the verification key used to check whether the sender has been authorized to communicate with the destination queue pair. This key is generated by the receiver.

3.2.2 Switching

We introduce two parallel crossbars for switching. One is called SMALL, which is used to transmit data of very small size, including DAP and RDMA GET; the other is called LARGE, which is used to transmit large amount of data including NAP and RDMA PUT. We also use the SMALL crossbar to avoid the deadlock between RDMA GET and RDMA PUT operations (the RDMA GET primitive is implemented with the RDMA PUT primitive).

The MTU (maximum transmission unit) in the SMALL crossbar is only 128 B (the common payload length in standard PCIe packets). To avoid the request/response deadlock, this crossbar uses two virtual channels: one for DAP load/store and RDMA GET; the other for DAP completion.

The MTU in the LARGE crossbar is 2 KB which is the largest payload size of NAP. This crossbar requires several virtual channels to reduce Head-of-Line blocking. We propose a Dest-Mod strategy for these virtual

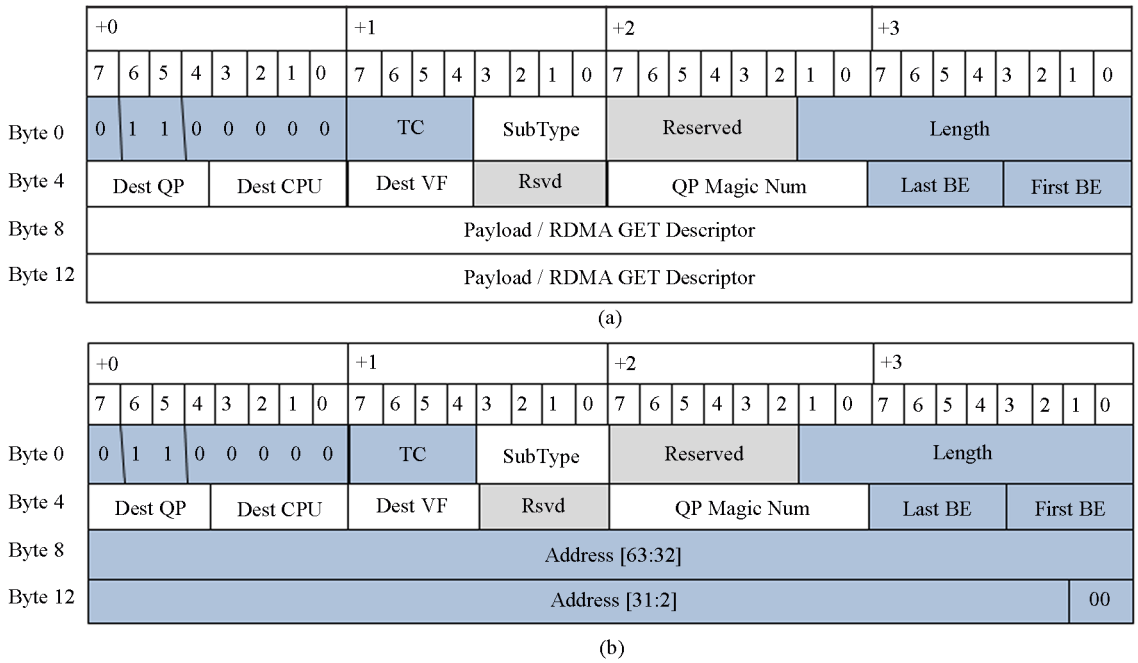


Fig.5. Network packet header. (a) NAP/RDMA GET. (b) RDMA PUT/DAP store.

channels: if the number of virtual channels is n, the number of ports is P, and the message's destination port is d, then the message will be buffered in the virtual channel [d mod n]. The Dest-Mod strategy yields three advantages: 1) the throughput is improved; 2) the packet order is guaranteed; 3) the scale of output arbitrator is still P, instead of $P \times n$:1.

4 I/O Resource Pooling

Sharing I/O devices across multiple operating system domains is just what PCI-SIG Multi-Root I/O Virtualization (MR-IOV)④ aims to achieve. However, MR-IOV makes so many changes to the standard PCIe specification that MR-IOV I/O devices can seldom be found today.

Instead of using PCI-SIG MR-IOV, we propose a mechanism to build an I/O resource pool with the PCIe SR-IOV⑤ or multi-function devices. In addition, the mechanism requires no modification to commercial device drivers. The mechanism involves three key steps:

1) *Unification of the I/O Addressing Space*. Because the commercial I/O device driver runs only in a single operating system domain, multiple processors' I/O addressing spaces should first be mapped into a single one.

2) *Creation of Logic Full-Function Devices*. Because virtual function (VF)⑤ itself can perform only incomplete device functions, it must be virtualized as a full-function device before it is allocated to the processor.

3) *Elastical Allocation of the Logic Devices*. Because processors' I/O requirements change dynamically, logic devices should be dynamically attached to or detached from processors on demand.

4.1 Unification of I/O Addressing Space

Intel VT-d⑥ and AMD IOMMU⑦ are well-known direct I/O assignment technologies. They allocate a device directly to VMs (virtual machines) by creating the DMA remapping between the VM and the physical host. Learning from VT-d, we create remapping between addressing spaces of different processors.

④PCI-SIG. Multi-root I/O virtualization and sharing 1.0 specification, May 2008. http://www.pcisig.com/members/downloads/specifications/iov/mr-iov1.0_12May08.pdf, Sept. 2014.

⑤PCI-SIG. Single root I/O virtualization and sharing 1.1 specification, Jan. 2010. http://www.pcisig.com/memers/downloads/specifications/iov/sr-iov1_1_20Jan10_cb.pdf, Sept. 2014.

⑥Burger T. Intel virtualization technology for directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices, March 2012. http://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices/, Sept. 2014.

⑦AMD Inc. AMD I/O virtualization technology (IOMMU) specification, Nov. 2009. http://support.amd.com/TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf, Sept. 2014.

The remapping is implemented with two look-up tables: an RID table, which performs RID (PCIe request ID[8]) remapping and an MMIO table, which performs MMIO address remapping (MSI-X interrupt is treated as PCIe memory write). RID is the identification of a PCIe device, and the MMIO address is the addressing space allocated to a PCIe BAR[8]. Each RID owns six BARs at maximum.

To facilitate the explanation of the remapping mechanism, we take the scenario of two processors sharing one PCIe SRIOV device as an example. As shown in Fig.6, the processor that owns the device is called OWNER processor, and the one that shares the device is called USER processor.

First, the OWNER processor discovers and configures the SR-IOV device that has one physical function (PF) and the N virtual functions (VF). Both the physical function and N virtual functions are mapped to the OWNER's address space with OID_i ($i \in [0, N]$) and $OBAR_{i\_0\sim5}$ ($i \in [0, N]$), where OID_i represents the RID of the i-th function, and $OBAR_{i\_0\sim5}$ represents the MMIO addresses assigned to $BAR_0 \sim BAR_5$ of the i-th function.

Second, USER starts its device discovery procedure. VF_j, the virtual function allocated to USER is assigned with UID_j and $UBAR_{j\_0\sim5}$ ($j \in [0, N]$) by USER, where UID_j and $UBAR_{j\_0\sim5}$ represent the RID and MMIO addresses in USER's address space respectively. After this procedure, the RID table (mapping between OID and UID, N entries) and the MMIO table (mapping between OBAR and UBAR, $6\times N$ entries) are completely configured.

Finally, UIDs and UBARs in the PCIe requests (from the USER processor to the device) are translated to the OIDs and OBARs with the two tables, as the blue lines indicate in Fig.6. In the opposite direction (from the device to the USER processor), only the OIDs in the PCIe requests are translated to UIDs, because the addresses that the PCIe requests carried are in the USER processor's space (e.g., addresses in the DMA descriptor), as the green lines in Fig.6.

As a result, all USER processors' I/O addressing spaces are unified to the space of the OWNER processor. cZodiac sets up one RID table and one MMIO table for each endpoint device, regardless of the number of USER processors.

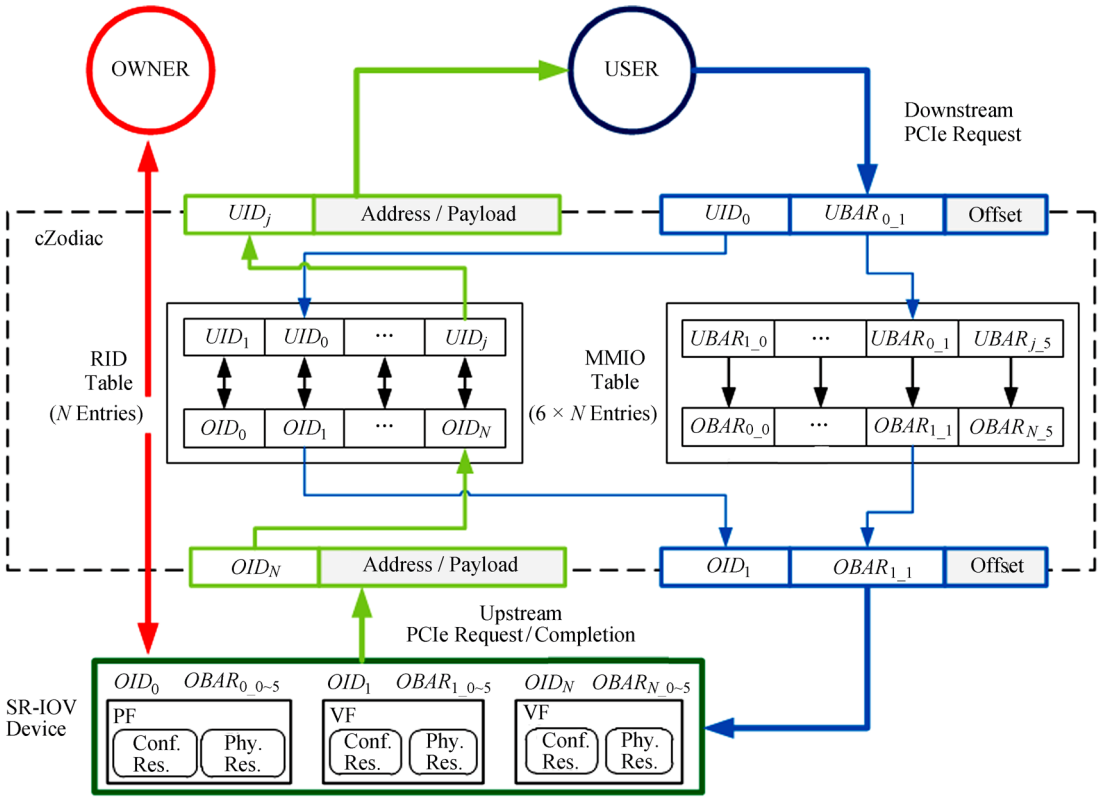


Fig.6. I/O remapping mechanism.

[8]PCI-SIG. PCI Express base 3.0 specification, Nov. 2010. http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r3.0_10Nov10.pdf, Sept. 2014.

4.2 Creation of Logic Full-Function Devices

We must create logic full-function devices for VFs or PFs that are allocated to USER processors, because of:

1) *Incorrect Responses.* VF does not possess complete configuration space, and many registers are read-only copies of PF (physical function). When a processor reads these read-only registers with VF's RID, only meaningless value (for example, all 1s) is returned.

2) *Forbidden Write Operations.* Even though PF possesses complete configuration space, control registers and BAR registers can only be written by the OWNER processor.

Thus, the logic full-function device must implement all of the registers that the USER processor cannot read or write. In addition, it intercepts all the transactions between the I/O device and the USER processor and decides whether the request is forwarded to the I/O device or responded by itself. Fig.7 shows the desired configuration space emulated by the logic device. All PCIe requests to colored fields are completely intercepted by the logic device, while requests to other fields are directly passed through to the physical device. As shown in Fig.7, green fields are emulated because of *incorrect responses*; blue fields are emulated because of *forbidden write operations*; orange fields are emulated for both reasons. For example, not only PF's but VF's device ID and vendor ID must be returned to the USER processor. In addition, if multiple VFs are allocated to a single USER processor, the head type of "multi-function device" will be returned to the USER processor.

00	Device ID		Vendor ID	
04	Status		Command	
08	Class Code		Revision ID	
0C	BIST	Head Type	Latency Timer	Cache Line Size
10	BAR 0			
14	BAR 1			
18	BAR 2			
1C	BAR 3			
20	BAR 4			
24	BAR 5			
28	Cardbus CIS Pointer			
2C	Subsystem ID		Subsystem Vendor ID	
30	Expansion ROM Base Address			
34	Reserved			Capability Pointer
38	Reserved			
3C	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Fig.7. Configuration space header of the logic device.

Through the implementation of logic devices, different functions in a physical I/O device can be directly accessed by different processors simultaneously. The original device driver can be used on the USER processor without any modification.

4.3 Elastical Allocation of logic Devices

The I/O resource pool is built with logic devices through the use of two remapping tables. However, such an I/O resource pool can only statically allocate resources to processors during the power-on phase. Since the processor's I/O requirement may vary drastically over time, we must dynamically assign or withdraw logic devices on demand. To achieve this goal, we propose a virtual hot plug & play mechanism based on PCIe hot plug & play.

For each endpoint device, we set a virtual plug module. All hot plug & play registers (slot capabilities, slot status, and slot control) in the downstream ports' configuration spaces are implemented in this module. The module generates corresponding MSI interrupts according to the I/O reallocation and responds to all PCIe requests (from USER processors to downstream ports) without performing any real actions to slot. In addition, write and read requests to the I/O device during the plug and unplug procedure are directly passed to a logic full-function device, and the logic device guarantees that these operations are harmless.

5 Micro-Architecture

The micro-architecture of cZodiac is shown in Fig.8. It contains several CommPorts and three parallel crossbars. The LARGE crossbar and the SMALL crossbar are used for host-to-host interconnection, while the IOV crossbar is used for I/O resource pooling. The IOV crossbar has the same architecture as the SMALL crossbar, which has been introduced in Section 3.

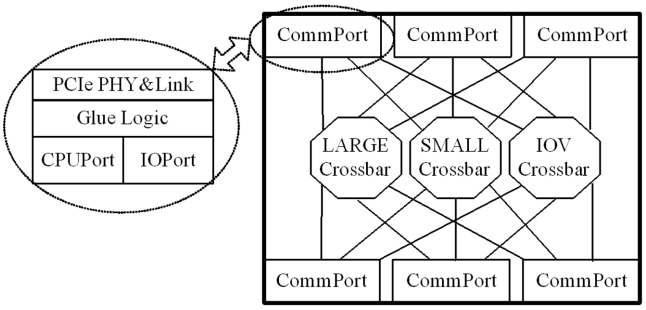


Fig.8. Micro-architecture of cZodiac.

CommPorts are designed to support both the coprocessor and stand-alone heterogeneous processing modes. When a CommPort interconnects with a coprocessor or an I/O device, IOPort is enabled; otherwise, CPUPort is enabled. The processing mode selection, as well as the control of I/O reallocation, is configured by the system administrator from OWNER processor.

5.1 CPUPort

CPUPort is designed to connect with the host processor and can be treated as a PCIe device with two functions. As shown in Fig.9, its first function, called communication engine, is a PCIe endpoint device with one physical function and seven virtual functions (each VF is statically allocated with four QPs), while the second function is a P2P (PCIe-to-PCIe) bridge, which is used for the I/O resource pooling.

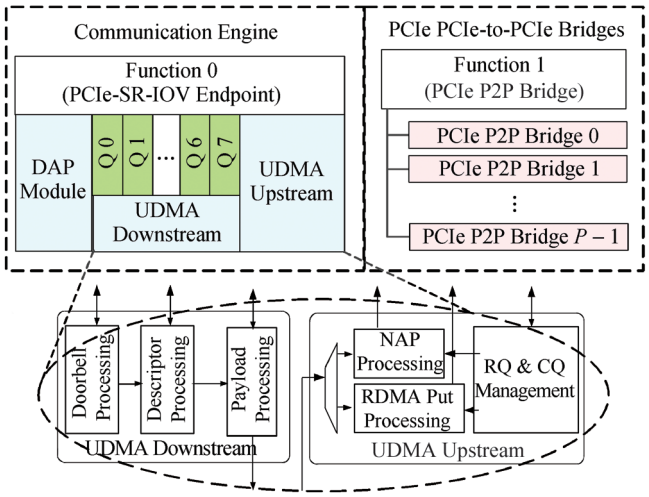


Fig.9. Micro-architecture of the CPUPort.

The communication engine implements the network interface described in Subsection 3.1. The DAP module deals only with DAP primitives and can be used only by the physical function. Except for packet format conversion, the DAP module forwards requests based on a GAS lookup table in the downstream direction, while it manages local PCIe source tags for DAP load and DAP completion in the upstream direction.

NAP and RDMA doorbells from PF and VFs are buffered into separate doorbell queues (Q 0∼Q 7) and fairly scheduled to UDMA downstream for execution with a matrix arbiter[7]. UDMA downstream executes doorbells in a 3-stage pipeline: "doorbell", "descriptor", and "payload". UDMA downstream first retrieves NAP/RDMA descriptors from main memory via the information in the doorbell, then retrieves payload data according to descriptors, and finally packages the payload into network packets. Network packets are designed based on PCIe request formats, thus the upload procedure in UDMA upstream is simple and efficient.

Regarding I/O resource sharing, cZodiac implements a standard PCIe switch first. The PCIe switch is made up of several PCIe-to-PCIe bridges; thus, for a P-port cZodiac, each CPUPort should contain one PCIe-to-PCIe bridge, and each IOPort should contain P PCIe-to-PCIe bridges. These bridges have many registers in common, thereby we implement all bridges in CPUPorts to save hardware resources and facilitate the comparisons of PCIe RIDs and BAR addresses. As shown in Fig.9, function 1 is the bridge that should be implemented in CPUPort (PCIe upstream port), while the other P bridges are related with P IOPorts (PCIe downstream ports).

5.2 IOPort

IOPort is the main module to implement I/O resource sharing. As shown in Fig.10, it contains three main sub-modules: virtual hot plug, remapping table, and logic full-function devices. The virtual hot plug and logic full-function devices use RAMs (random access memories) to store corresponding PCIe configuration spaces, while the remapping table uses both CAMs (content-addressable memories) and RAMs to build the RID table and MMIO table.

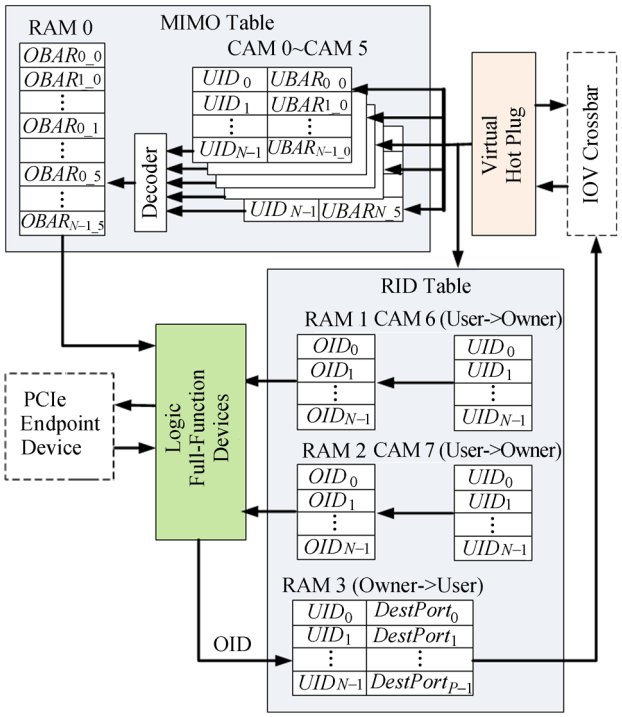


Fig.10. Micro-architecture of the IOPort.

As designed in Subsection 4.1, the MMIO table is used to translate UBAR to OBAR. Thus, it contains six CAMs (CAM 0∼CAM 5) for BAR 0 ∼ BAR 5, respectively. If the PCIe endpoint device has N functions (including PFs and VFs), then each CAM will have at maximum N entries (one entry per function). OBARs are stored in an RAM indexed by the combination of the VF/PF's function number and the BAR's sequence number. Thus, by using the decoded matching result from six CAMs as the read address, the value of OBAR can be gotten from the RAM.

The RID table performs bi-direction remapping. Regarding the USER to OWNER remapping, two N-entry CAMs (CAM 6 and CAM 7) and two N-entry RAMs (RAM 1 and RAM 2) are used (N is the number of virtual and physical functions). RAM 1 and CAM 6 perform the remapping for PCIe requests, while RAM 2 and CAM 7 perform the remapping for PCIe completions. In fact, RAM 1 and RAM 2 store exactly the same content, so do CAM 6 and CAM 7. We use two pairs of RAM and CAM to improve the remapping performance. In the opposite direction, because one device has only N functions and OIDs are sequential numbered (we choose the last nine bits), an N-entry RAM (RAM 3 in Fig.10) indexed by OID can perform the OWNER to USER remapping. $DestPort_i$ ($i \in [0, P-1]$, P is the number of cZodiac's ports) stored in RAM 3 is the destination port number used for internal switching.

6 Evaluation

6.1 Prototype

As shown in Fig.11, the prototype is a 3-port cZodiac implemented with Xilinx Virtex6 XC6VLX365T. All of the 24 high speed serial links (GTH transceivers) in the FPGA are used to implement the three ports (each port is 8× PCIe Gen2). FPGA runs at 250 MHz and uses a 128 bit internal data bus. According to the PAR (place and route) report generated by Xilinx PlanAhead, each CommPort occupies 21% FPGA's logic resources (CPUPort uses 10%, IOPort uses 3%, and PCIe controller uses 8%). Based on such resource consumption, an 8-port cZodiac can be implemented with the latest Xilinx Virtex7 XC7VX690T (contains 80 GTH transceivers and almost double logic resources).

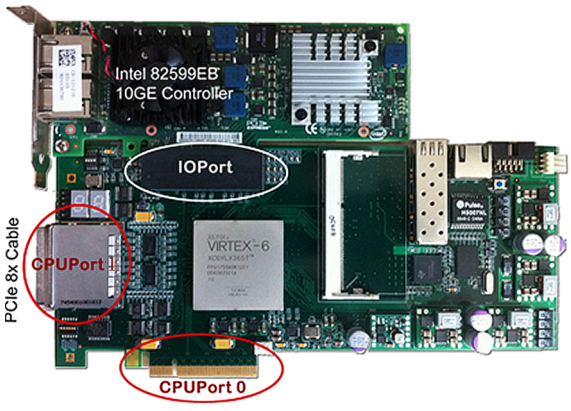


Fig.11. Prototype: 3-port cZodiac.

We configure the three ports as two CPUPorts and one IOPort to verify both the interconnection and I/O sharing mechanisms. Both LARGE and SMALL crossbars have two virtual channels. CPUPort 0 shown in Fig.11 connects with the OWNER server by plugging into the server's PCIe slot, CPUPort 1 connects with a USER server by using a PCIe cable, and the IOPort connects with an Intel 82599EB 10GE controller by using the PCIe slot on the FPGA board. In order to fully evaluate cZodiac's performance, we use two Intel Xeon nodes (Xeon i5-3470, 3.2 GHZ, 16X PCIe Gen2 slots) with high I/O performance as an OWNER server and a USER server respectively.

6.2 PCIe Compliant Interconnection

The evaluation of the point-to-point communication is carried out in the user-level communication library layer. DAP store gets the lowest latency of 0.53 μs. As shown in Fig.12, the minimum latencies of NAP and RDMA are 1.12 μs and 1.73 μs respectively. When transmitting short messages, NAP can achieve lower latency than RDMA. However, as the message length becomes larger, its latency increases quickly because of the time spent in memory copy. Because the RDMA GET involves an additional process of transmitting its descriptor in the SMALL crossbar, its latency is slight higher than that of the RDMA PUT.

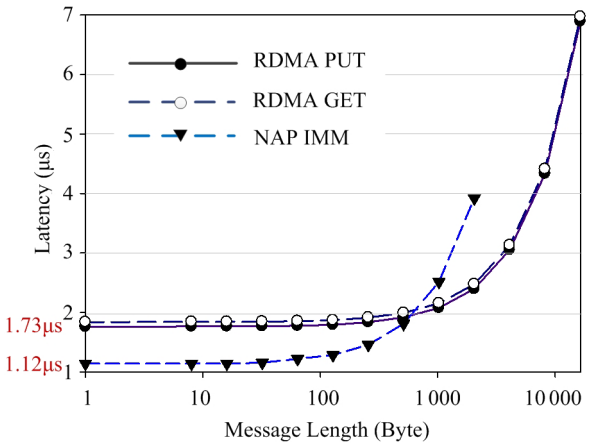


Fig.12. Point-to-point communication latency.

Bandwidth of the point-to-point communication is shown in Fig.13. There are few differences in the bandwidth between RDMA PUT and RDMA GET. The maximum bandwidth that can be achieved is 3.19 GBps, which is 79.8% of PCIe peak bandwidth (8× PCIe Gen2). Because the maximum payload size of PCIe TLP packets in our prototype is 128 B and the TLP packet header is 16 B, the theoretical maximum throughput is 88.9%. That means 89.6% of PCIe valid bandwidth has been achieved by using RDMA PUT.

We use a cycle-accurate simulator written in SystemVerilog to evaluate cZodiac's switching performance. We use the RDMA PUT primitive, 2 KB message length, Dest-Mod virtual channel allocation strategy, and random distributed traffic pattern. Conside-

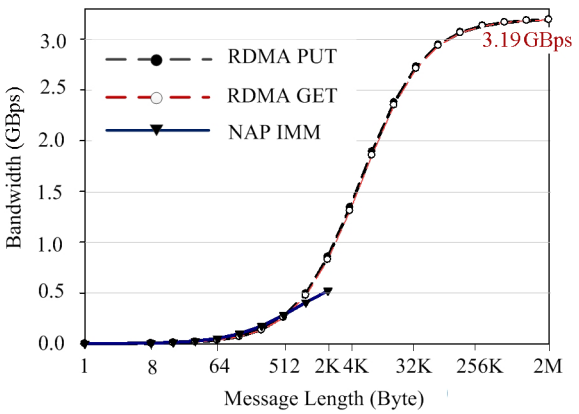


Fig.13. Point-to-point communication bandwidth.

ring the limited space and power budgets, even using high power efficient processors, the radix of 16 is quite large for the interconnection in a 1 U∼2 U server. Therefore, the maximum number of port in this evaluation is set to 16.

As shown in Fig.14, the highest throughput of 71% is achieved when the LARGE crossbar has four virtual channels in the 4-port cZodiac. That is just the VoQ (virtual output queuing) architecture without HOL (head of line blocking). Compared with the highest throughput (71%), the 4-port cZodiac with two VCs only decreases by 4%. For the 8-port cZodiac, the maximum throughput of using two VCs is 65.5%, while the one of using four VCs is 68.75%. Therefore, using two virtual channels for the 8-port cZodiac can achieve the best cost performance. However, for the 16-port cZodiac, it is necessary to use four VCs.

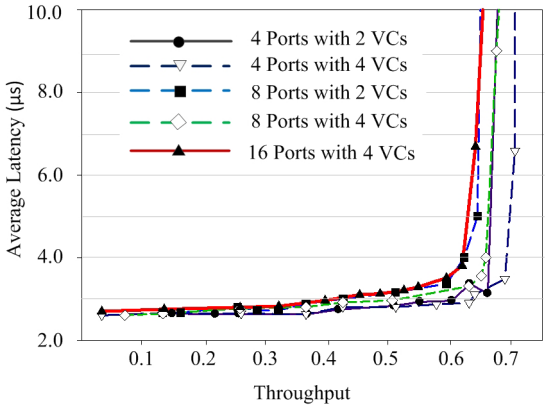


Fig.14. Switching performance.

6.3 Multi-Root Sharing of Intel 82599EB

As shown in Fig.15, except for the OWNER and USER servers, we introduce another Intel server marked as TESTER to communicate with the Intel 82599 adapter attached to the IOPort. In the software layer, the Intel 82599 PF driver (ixgbe-3.14.5) runs on the OWNER server, while the Intel 82599 VF driver (ixgbevf-2.7.12) runs on the USER server. Also, we run four virtual machines (VMM is KVM) on the USER server to test the performance of multi-client I/O sharing. We use iperf-2.0.5[⑨] for our tests. The MTU is set only to 1 500 B, because the 82599EB does not allow MTU larger than 1 500 B when the feature of SR-IOV is enabled.

On five paths, we carry out the evaluations.

• *Path* 1. Between the OWNER and the TESTER, this path includes cZodiac's forwarding overhead. If the OWNER acts as the iperf client, we name the path as *o*2*t*. If the OWNER acts as the iperf server, we name the path as *t*2*o*.

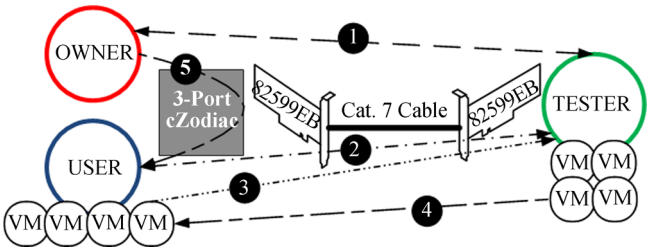


Fig.15. I/O sharing evaluation environment.

• *Path* 2. Between the USER and the TESTER, this path includes cZodiac's forwarding and remapping overhead. If the USER acts as the iperf client, we name the path as *u*2*t*. If the USER acts as the iperf server, we name the path as *t*2*u*.

• *Path* 3 (*v*2*t*). Between VMs on the USER and VMs on the TESTER, we allocate one Intel 82599 virtual function to each VM in pass-through mode. Each VM on the USER acts as one iperf client. The name *v*2*t*-*i* means that *i* VMs act as iperf clients.

• *Path* 4 (*t*2*v*). Between VMs on the USER and VMs on the TESTER, on both the USER and the TESTER, we allocate one Intel 82599 virtual function to each VM in pass-through mode. Each VM on the TESTER acts as one iperf client, while each VM on the USER acts as one iperf server. The name *t*2*v*-*i* means that *i* pairs of VMs are under testing.

• *Path* 5 (*o*2*u*). Between the OWNER and the USER, the OWNER communicates with the USER by using the internal L2 switching inside the Intel 82599EB.

As shown in Fig.16, without using cZodiac, the raw bandwidth between the two hosts is 9.41 Gbps. As the function of I/O resource sharing is completely implemented in hardware, it introduces very low overhead.

⑨http://sourceforge.net/projects/iperf, June 2014.

Almost all of our tests can approach the maximum value 9.41 Gbps. For both paths $v2t$ and $t2v$, the bandwidth is shared fairly among VMs. For example, in the path $v2t\_4$, bandwidths allocated to VM 0∼VM 3 are 2.42 Gbps, 2.24 Gbps, 2.35 Gbps, and 2.37 Gbps, respectively. Such fair sharing is guaranteed by the QoS supported by Intel 82599EB. Therefore, if the processor owns more virtual devices, it owns more bandwidth.

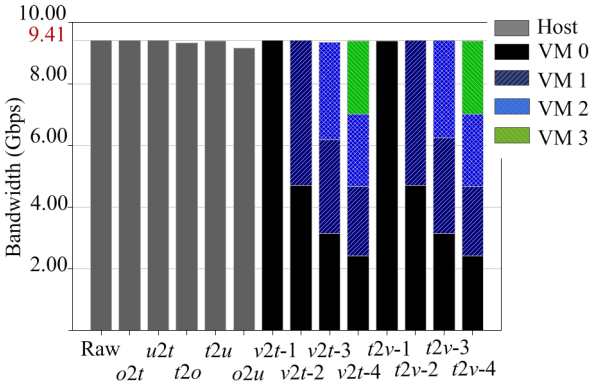


Fig.16. Network sharing performance.

The path $o2u$ gets the lowest bandwidth of 9.17 Gbps. That is because the IOPort is busy with transmitting both DMA reads and DMA writes generated by the Intel 82599EB. In our implementation, PCIe post requests and PCIe non-post requests are buffered into a single queue. Thus, the PCIe non-post requests generated by the DMA reads will have interference with the PCIe post requests generated by the DMA writes, somewhat like HOL (head of line blocking). However, such interference only introduces 2.55% reduction in bandwidth.

7 Related Work

In the field of heterogeneous computing, the conventional SMP/ccNUMA is still the dominant server architecture[1-2,8]. In SMP/ccNUMA servers, system buses supporting cache coherence (such as Intel QPI and AMD Hyper-Transport) are used as the fabrics that interconnect multiple homogeneous processors. However, such coherence memory fabrics are not suitable for interconnecting heterogeneous processors, because: 1) many accelerators such as GPGPU do not support cache coherence; 2) the proprietary cache coherence protocols designed by different companies are not open and not compatible with each other.

Therefore, some work proposed to interconnect processors with network fabrics without maintaining the cache coherence between heterogeneous processors. KnightShift[9] is a board-level heterogeneous architecture that interconnects heterogeneous computing components with traditional Ethernet network interface. However, because both the data exchanging and the device sharing between components are implemented in the software level and introduce large overhead, KnightShift is only suitable for a loosely-coupled heterogeneous computing model. SeaMicro Freedom[⑩] interconnects different kinds of processors with its proprietary fabric and provides I/O virtualization technology for I/O device sharing. cZodiac implements similar functions, but there are not enough details from Freedom's published document to judge the differences.

On the inter-server level, there are some studies extending PCIe as interconnection fabric. Dolphin[10-11] announced PCIe Cluster/I/O switch to provide host-to-host interconnection. However, Dolphin's interconnection designed to support TCP/IP is too complicated for intra-server interconnection. In addition, the I/O device attached to the switch cannot be accessed simultaneously by multiple processors. PLX is announcing the PCIe-based ExpressFabric[⑪] as the datacenter fabric. Also, no implementation details are given. Judging from the only available document[12], its I/O sharing mechanism is based on the NTB (non-transparent bridge), which can be different from ours. NEC[13] and NEXTIO[⑫] provide their own multi-root I/O sharing solutions. Both of them are using Ethernet, while we are using raw PCIe and can achieve better performance.

8 Conclusions

Focusing on the heterogeneity inside a single server, we proposed an intra-server interconnect fabric designed for the network-centric server architecture. From two aspects, we meet the system requirements brought by heterogeneity. First, high efficient interconnection between processors is achieved by introducing a lightweight PCIe compliant network. Second, the multi-root I/O resource sharing is achieved by introducing an RID and an address remapping mechanism between processors in hardware level, and there is no modification to original device drivers. In addition, the implementation of the fabric cZodiac is of good scalability. Actually, our proposed fabric is suitable for not only the heterogeneous server but also the high density server.

Acknowledgement We thank anonymous reviewers for their constructive and valuable comments.

⑩ Rao A. SeaMicro technology overview, Oct. 2012. http://www.seamicro.com/sites/default/files/SM_TO01_64_v2.7.pdf, Sept. 2014.

⑪ http://www.plxtech.com/applications/expressfabric, June 2014.

⑫ http://www.nextio.com/products/vnet, June 2014.

Our thanks also go to Dong-Dong Wu and Yong Su at Institute of Computing Technology, Chinese Academy of Sciences, for their generous help on the implementation of cZodiac.

References

[1] Barker K J, Davis K, Hoisie A *et al.* Entering the petaflop era: The architecture and performance of Roadrunner. In *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 2008, Article No. 1.

[2] Sun N H, Xing J, Huo Z G *et al.* Dawning Nebulae: A petaFLOPS supercomputer with a heterogeneous structure. *Journal of Computer Science and Technology*, 2011, 26(3): 352-362.

[3] Reddi V J, Lee B C, Chilimbi T, Vaid K. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proc. the 37th Annual Int. Symp. Computer Architecture*, June 2010, pp.314-325.

[4] Guevara M, Lubin B, Lee B C. Navigating heterogeneous processors with market mechanisms. In *Proc. the 19th IEEE Int. Symp. High Performance Computer Architecture*, Feb. 2013, pp.95-106.

[5] Zapater M, Ayala J L, Moya J M. Leveraging heterogeneity for energy minimization in data centers. In *Proc. the 12th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, May 2012, pp.752-757.

[6] Suneja S, Baron E, Lara E D *et al.* Accelerating the cloud with heterogeneous computing. In *Proc. the 3rd USENIX Conf. Hot Topics in Cloud Computing*, June 2011, p.23.

[7] Peh L S, Dally W J. A delay model and speculative architecture for pipelined routers. In *Proc. the 7th Int. Symp. High Performance Computer Architecture*, Jan. 2001, pp.255-266.

[8] Ohno Y, Nishibori E, Narumi T *et al.* A 281Tflops calculation for X-ray protein structure analysis with special-purpose computers MDGRAPE-3. In *Proc. ACM/IEEE Conference on Supercomputing*, Nov. 2007, pp.1-10.

[9] Wong D, Annavaram M. KnightShift: Scaling the energy proportionality wall through server-level heterogeneity. In *Proc. the 45th IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp.119-130.

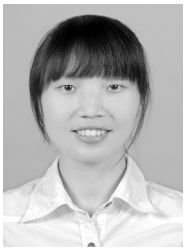
[10] Krishnan V. Evaluation of an integrated PCI express IO expansion and clustering fabric. In *Proc. the 16th IEEE Symp. High Performance Interconnects*, Aug. 2008, pp.93-100.

[11] Krishnan V. Towards an integrated IO and clustering solution using PCI express. In *Proc. IEEE International Conference on Cluster Computing*, Sept. 2007, pp.259-266.

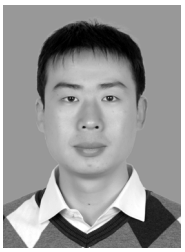
[12] Aswadhati A. Scaling data center services with PCI express. In *Proc. Linley Tech. Data Center Conference*, Feb. 2012.

[13] Suzuki J, Hidaka Y, Higuchi J *et al.* Multi-root share of single-root I/O virtualization (SR-IOV) compliant PCI Express device. In *Proc. the 18th IEEE Symp. High Performance Interconnects*, Aug. 2010. pp.25-31

Zheng Cao received his Ph.D. degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2009. He is an associate professor of ICT, CAS. His main research interests include high performance computer architecture and high performance interconnection networks. He is a member of CCF and ACM.

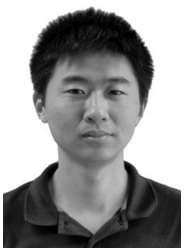


Xiao-Li Liu received her M.S. degree in telecommunication from Beijing University of Posts and Telecommunications in 2011. She is an engineer of ICT, CAS. Her main research interests focus on IO virtualization and high performance interconnection networks. She is a member of CCF and ACM.



Qiang Li received his Ph.D. degree in computer science from ICT, CAS, in 2012. He is an assistant professor of ICT, CAS. His research interests focus on high performance communication. He is a member of CCF and ACM.

Xiao-Bing Liu received his M.S. degree in computer science from Peking University in 2012. He is an associate engineer of ICT, CAS. His main research interests include computer architecture and high performance interconnection networks. He is a member of CCF and ACM.



Zhan Wang is a Ph.D candidate in computer science of ICT, CAS. His main research interests include virtualization and high performance interconnection networks. He is a student member of CCF.

Xue-Jun An received his Ph.D. degree in computer science from ICT, CAS. He is a professor of ICT, CAS. His main research interests include high performance computer architecture and high performance interconnection networks. He is a member of CCF and ACM.