Jiang T, Hou R, Dong JB *et al.* Adapting memory hierarchies for emerging datacenter interconnects. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(1): 97–109 Jan. 2015. DOI 10.1007/s11390-015-1507-4

Adapting Memory Hierarchies for Emerging Datacenter Interconnects

Tao Jiang [1,2] (江 涛), *Member, CCF, ACM, IEEE*, Rui Hou [1] (侯 锐), *Member, CCF, ACM, IEEE*
Jian-Bo Dong [1] (董建波), *Member, CCF, ACM, IEEE*, Lin Chai [1,2] (柴 琳)
Sally A. McKee [3], *Member, ACM, IEEE*, Bin Tian [4] (田 斌), *Member, CCF*
Li-Xin Zhang [1] (张立新), *Member, ACM, IEEE*, and
Ning-Hui Sun [1] (孙凝晖), *Fellow, CCF, Member, ACM, IEEE*

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

[2] *University of Chinese Academy of Sciences, Beijing 100049, China*

[3] *Computer Science and Engineering, Chalmers University of Technology, Gothenburg 41296, Sweden*

[4] *National High Performance Integrated Circuit Design Center (Shanghai), Shanghai 201204, China*

E-mail: {jiangtao, hourui, dongjianbo, chailin}@ict.ac.cn; mckee@chalmers.se; bin_tian@vip.sina.com
{zhanglixin, snh}@ict.ac.cn

Received July 14, 2014; revised December 15, 2014.

Abstract Efficient resource utilization requires that emerging datacenter interconnects support both high performance communication and efficient remote resource sharing. These goals require that the network be more tightly coupled with the CPU chips. Designing a new interconnection technology thus requires considering not only the interconnection itself, but also the design of the processors that will rely on it. In this paper, we study memory hierarchy implications for the design of high-speed datacenter interconnects — particularly as they affect remote memory access — and we use PCIe as the vehicle for our investigations. To that end, we build three complementary platforms: a PCIe-interconnected prototype server with which we measure and analyze current bottlenecks; a software simulator that lets us model microarchitectural and cache hierarchy changes; and an FPGA prototype system with a streamlined switchless customized protocol Thunder with which we study hardware optimizations outside the processor. We highlight several architectural modifications to better support remote memory access and communication, and quantify their impact and limitations.

Keywords high-speed interconnect, memory hierarchy, time shared memory, datacenter network

1 Introduction

Exploding data volumes and increasingly sophisticated data usage scenarios are beginning to overwhelm existing datacenter interconnection structures. In particular, the north-south traffic (between clients from the Internet and datacenter servers) traditionally accounted for about 80% of the total traffic, but emerging applications are drastically changing communication patterns so that the east-west traffic (within the center) has begun to dominate in newer datacenters, making the latter traffic now constitute up to 80% of the total. In these data centers, about 80% of the traffic flows are smaller than 10 KB in length, and most of the bytes transmitted belong to the top 10% of large traffic flows[1]. Furthermore, common big data workloads such as Hadoop require frequent communication among multiple nodes. Delivering good performance for such traffic patterns requires both high-bandwidth and low-latency interconnects among nodes.

Emerging datacenter interconnects must try to deliver the necessary communication performance while reducing cost and power consumption. For instance, the PCI Express Switch of PLX can be used to create an intra-rack PCIe-based fabric[2]. The CALXEDA EnergyCore SoC integrates an embedded network fabric switch that can efficiently connect thousands of nodes. To reduce queue management in software and to ef-

Regular Paper
Special Section on Computer Architecture and Systems for Big Data
This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401, and the National Natural Science Foundation of China under Grant Nos. 61100010, 61402438, and 61402439.

ficiently support user-level communication, some solutions now integrate separate send and receive queues on the network adapter. InfiniBand (IB)[3] achieves lower latency than Ethernet by employing such QPairs instead of ports for communication.

In addition to supporting explicit communication, datacenter interconnects are increasingly being used to dynamically allocate and flexibly share resources. The high cost of datacenter servers and the disaggregation of resources make it attractive to share inter-node resources like memory. Software-defined datacenter architectures like Intel's Rack Scale Architecture (RSA) re-architect the datacenter design to increase the modularity of components (i.e., CPU, memory, storage) at the rack level①. RSA supports dynamically allocating resources via high-speed Intel Silicon Photonics components. Our own PCIe prototype system supports sharing memory, GPGPUs, and NICs among nodes[4]. For example, when a big data application needs more memory than the nodes it runs on can provide, it can simply "borrow" unused memory from other nodes that are running applications with smaller footprints. Here we focus on optimizing support for sharing global memory; sharing other resources requires similar support, but the details are beyond the scope of this paper.

Another emerging trend is that datacenter interconnects are being integrated into the same chip with processor cores. Designing a new interconnection technology or communication protocol thus requires not only considering the interconnection itself, but also considering the design of the processors that will rely on it. To take better advantage of the performance potential of the entire computer system — including the resource sharing and communication capacities between nodes, we study memory hierarchy implications for the design of high-speed datacenter interconnects.

To that end, we have built three complementary datacenter models: a cluster connected via a PCIe fabric; a software simulator that lets us study advanced functionalities that have not (yet) been implemented directly in commercial chips; and an FPGA-based prototype that integrates an on-chip, switchless customized protocol with a hardware QPair mechanism. We choose PCIe for our prototype interconnect because PCIe enjoys widespread use (almost every major microprocessor includes a PCIe interface, and thus any mainstream processor can be added to our system), and it is non-proprietary (unlike the Intel QPI and AMD HT products we considered). Furthermore, PCIe supports two features absent in Ethernet and IB. First, PCIe allows direct access to the remote memory via normal load/store instructions. Second, PCIe allows "shorter" communication channels: the PCIe-based communication channel removes adapters and additional protocol conversion mechanisms (from PCIe to IB or Ethernet) from the critical path.

We use the first prototype to investigate the performance ramifications of using remote memory through either direct load/store instructions or the DMA engine. Our results confirm that memory sharing through DMA requests performs consistently well. For using remote memory as block device by DMA, our prototype system has five times bandwidth, 11 times IOPS (input/output per second) and 1/12 latency compared with the system connected by 10 Gigabit Ethernet (GigE) in average for ORION benchmark. However, sharing remote memory through CPU load/store instructions may incur high overhead caused by the modern processors that contain no special optimization for cache line loads/stores through the otherwise considered pure I/O interface. These results highlight opportunities to optimize the execution of direct memory access instructions for sharing memory resources. We use the simulator to study the impact of three specific modifications: increasing the number of read requests that can be processed by the PCIe interconnect controller and bus simultaneously; changing the cache policy from write-through/invalidate to write-back; and modifying the prefetch degree of the prefetcher to accommodate remote memory traffic. For a prefetcher, when the prefetch degree is set to five, it has seven times performance improvement compared with that without prefetcher. Finally, we use the FPGA prototype to study the design space for a QPair implementation that leverages a customized, switchless protocol. Using hardware MMU and on-chip memory (OCM) achieves 25% bandwidth increase compared with using cache without hardware MMU. Moreover, using hardware MMU and OCM has the lowest latency.

In the rest of the paper, Section 2 describes the background of this work. Section 3 presents the modifications of memory hierarchy for resource sharing and communication. Section 4 shows the platforms and methodology. Section 5 contains the evaluation results. Section 6 talks about related work. Section 7 concludes the paper.

① Rack scale architecture for cloud, Oct. 2013. http://www.chinacloud.cn/upload/2013-10/13102200313281.pdf, July 2014.

2 Background

In this work, we plan to build an interconnect system which not only can be used for communication, but also can be used for resource sharing. Fig.1 shows the architecture of our desirable system. Through the support of interconnect, the memory of all the nodes forms a logical memory pool. One node can simply "borrow" unused memory from other nodes.

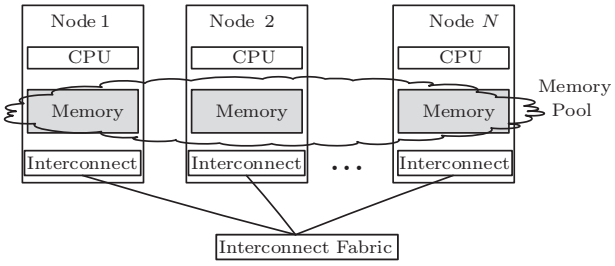


Fig.1. Architecture of resource sharing system.

To keep the cost in check, we use exclusively commodity products for the prototype system. We select x86 processor chips from Intel for the computing nodes and PCIe switch chips from PLX for interconnect. We considered Intel QPI and AMD HT products but decided against using them because they are proprietary and do not have the widespread use as PCIe does. PCIe interface can be found in almost every major microprocessor, thus allowing us to add almost any mainline microprocessor to the system.

The selection of the interconnect technology is the most critical decision for the hardware prototype, because the desired resource sharing will only happen through the selected interconnection links. The conventional choices are Ethernet or IB. To put the raw characteristics of PCIe in perspective, Table 1 compares the key features of PCIe Gen2, 10 Gigabit Ethernet (10 GigE), and IB. PCIe and IB have similar peak bandwidths and the same latency, and 10 GigE has the lowest peak bandwidth and a latency of an order of magnitude more than PCIe and IB. All three support remote DMA (RDMA) and need external switches in real deployments.

PCIe has two features that Ethernet and IB do not have. First, PCIe allows direct access to the remote memory via normal load/store instructions. This feature allows a program to make use of remote memory without modification. It is supported by a PCIe switch with the non-transparent bridge (NTB) functionality. NTB provides isolation among the hosts connected via a PCIe switch while still allowing communications among the hosts. Second, PCIe allows "shorter" communication channel. A typical communication channel with Ethernet and IB goes through the Ethernet and IB adapters connected to the north bridge via a PCIe interface. PCIe-based communication channel is shortened because the adapters and the additional protocol conversion (from PCIe to IB or Ethernet) are cut off from the path.

Table 1. Interconnect Comparisons

	Peak Bandwidth	Application Latency	Remote Memory Access	External Switch
PCIe switch PLX 8648[a]	3 005 MB/s	∼1 µm	RDMA, load/store direct access	Yes
10 GigE	1 175 MB/s[b]	8.9 µs[c]	RDMA	Yes
IB 40 Gb/s Mellanox PCIe Gen2[c]	3 400 MB/s	∼1 µs	RDMA	Yes

Note: [a] The numbers are from PLX PEX 8619 user manual②. [b] The numbers are from our test by using Netperf. [c] The numbers are from Mellanox report③.

Fig.2 shows our prototype system, which is comprised of five computing nodes and a backplane. Each computing node is connected to the backplane via a PCIe adapter card. The backplane contains one transparent PCIe bridge chip and four NTB chips. The transparent bridge chip supports one root node (CPU 0) and four leaf nodes (CPUs 1∼4), giving the system five nodes in total. CPU 0 is in charge of forwarding transactions between the root node and leaf nodes. Each node contains one Intel Core i7 3.4 GHz processor and 8 GB DDR3 DRAM. The NTB chips are used for address isolation and translation. The prototype supports PCIe Gen2 interfaces. Any machine with a PCIe slot can be connected to the backplane.

2.1 Remote Memory Access

In PCIe systems, the NTB includes an address mapping table to translate address spaces between hosts. Once the mapping has been set up, one side can access the memory region of the other side. In our prototype system, the OS on the remote node can no longer see

② PEX 8619 DMA performance metrics. http://www.plxtech.com/8619, Dec. 2014.

③ InfiniBand performance. http://www.mellanox.com/content/pages.php?pg=performance_infiniband, Dec. 2012.

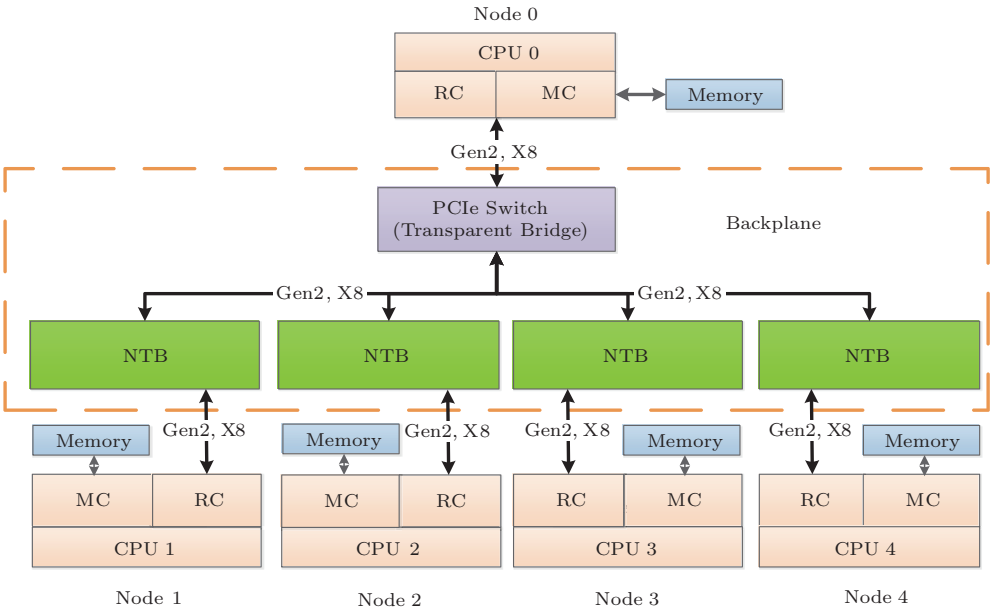


Fig.2. Architecture of PCIe switch-based system. RC: root complex; MC: memory controller; NTB: non-transparent bridge.

the borrowed memory block, hence we term such as time-shared memory (TSM).

Fig.3 shows two optional design methods. One is direct load/store instructions access, and the other is DMA access. Node A can borrow the shared memory provided by node B after they set up their own address mapping tables in the address translation module of NTB. Node A borrows rather than shares remote memory, making it unnecessary to maintain data consistency. Node A can access the remote memory located in node B using either direct load/store instructions or DMA method.

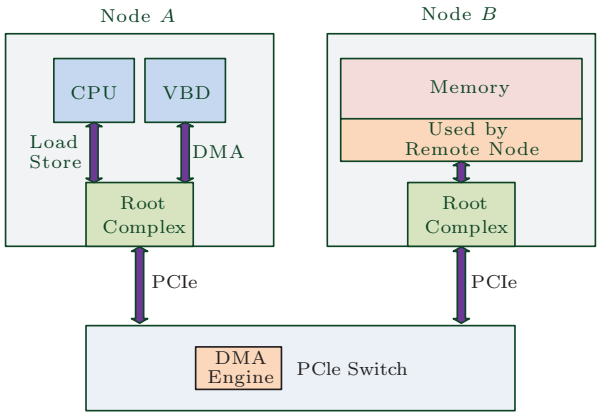


Fig.3. Methods of using TSM.

2.1.1 DMA

PCIe supports DMA access mode by using DMA engine located in the NTB chip. When the client node sends data reading/writing request of remote memory, the request will be first transferred to the NTB chip. The NTB chip reconstructs the request into the DMA descriptor, which contains essential information, such as source address, destination address, and transferred data size, and sends it to the DMA engine. Then, the newly constructed DMA descriptor will be launched to the server side which processes it as its normal DMA request. To simplify the use of the DMA engine, we implement a virtual block driver (VBD) in Linux to emulate a block device backed by the remote memory. The way of using VBD is to make it behave like a local disk. Many traditional databases require strong disk performance, thereby we use ORION (Oracle I/O Calibration Tool)④ for this study. ORION is a standalone benchmark tool for calibrating the I/O performance of storage systems intended to be used for Oracle databases. In addition to the Ethernet virtual disk (Vdisk), IB with SCSI RDMA Protocol (SRP), and VBD implementations, we also measure two more alternatives, 7200rpm SATA disk and SATA3 SSD. Fig.4 shows the measured bandwidth, IOPS and latency. VBD stands out among the competitors. It has the highest bandwidth, highest IOPS, and lowest latency. Its bandwidth is about 50 times of that of SATA disk, and about five

④ Oracle ORION benchmark. http://www.oracle.com/technetwork/cn/topics/index-088165-zhs.html, Dec. 2014.

times of that of SSD. Its IOPS is more than 1 400 times of that of SATA disk and about three times of that of SSD. Its latency is more than 600 times of SATA disk, and more than seven times of that of SSD.

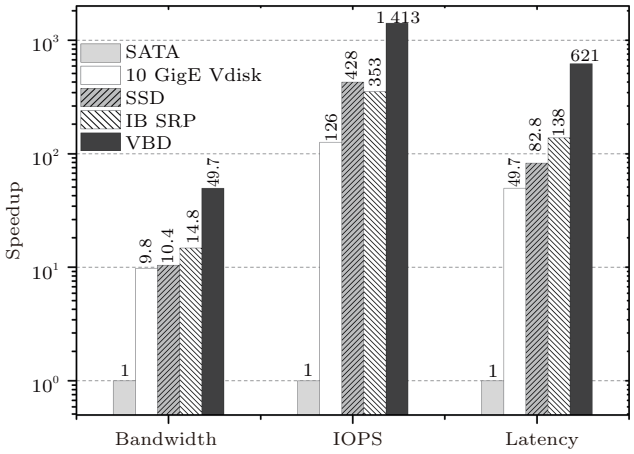


Fig.4. Performance evaluation for Oracle ORION.

2.1.2 Load/Store

In direct load/store mode, with the help of our implemented driver module hooked to the OS, the user application can access remote memory as its local memory. The difference with local memory is that when the request of remote memory load/store misses in caches, it will be delivered to the PCIe Root Complex (RC), forwarded to the PCIe port and transmitted to the corresponding NTB. Finally, the request is handled by the remote node.

Our results show that accessing remote memory through sustained (repeated) direct loads has high overhead, whereas sharing memory through store operations performs well for small transactions. The reasons for this are fairly obvious: PCIe's posted write transactions need not wait for a reply, whereas read transactions must wait for the requested data to arrive. To better understand the architectural interactions underlying our observations, we examine four issues in greater detail: load serialization, impact of the cache write policy, prefetcher behavior, and limitations of atomic instructions.

Load Serialization. To evaluate the performance trade-offs of the various approaches, we use each method to transfer 1 GB data between two nodes in block sizes varying from 64 B to 1 MB. The results in Table 2 show that the maximum bandwidth of direct loads is 26 MB/s and that of the DMA loads is 2 980 MB/s. Issuing sustained direct loads performs uniformly poorly for all block sizes. To understand this behavior, we use a PCIe Logic analyzer to collect TLP (Transaction Layer Packet) traces. The traces show that in a 17 μs randomly sampled observation window, the DMA can generate about 400 TLPs, but direct loads generate only 12 TLPs (this happens because a new TLP load request is issued only after the previous load request finishes). Using DMA transfers or issuing sustained writes generates pipelined parallel TLP requests, and thus they both perform significantly better than issuing back-to-back loads. We suspect that the PCIe RC inside the Intel Core i7 processor chip serializes the load requests[5]. Extending the microarchitecture to support parallel load requests is a modification worth exploring.

Table 2. Bandwidth of Load/Store and DMA (MB/s)

Data Size (B)	Load	Store	DMA
64	26	694	282
128	26	1 340	551
512	26	2 480	1 812
1 K	26	2 430	2 300
4 K	26	2 470	2 980
16 K	26	2 480	2 980
64 K	26	2 480	2 980
256 K	26	2 480	2 980
1 M	26	2 480	2 980

Cache Write Policy. The Intel IA-32 architecture allows data loaded from remote memory to be saved in local caches[5]. However, we find that cacheable writes to the remote memory must be carried out in write-through mode. We try using write-back mode on three different IA-32 processors (Intel Core i5, Core i7, and Xeon E5) without success (we are still investigating the issue). In write-through mode, a write always updates (local or remote) memory no matter whether it hits in cache or not.

To analyze the performance of caching remote data locally, we use four microbenchmarks: 1) benchmark 1: one with a read-only working set smaller than the LLC (last level cache); 2) benchmark 2: one with a read/write working set smaller than the LLC; 3) benchmark 3: one with a read-only working set larger than the LLC but with good locality; and 4) benchmark 4: one with a random read/write working set larger than

[5] Intel 64 and IA-32 architectures software developer's manual, Dec. 2012. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, Dec. 2014.

the LLC and without locality. Fig.5 shows the results. For benchmark 1 and benchmark 3, there is little or no performance difference between using local and remote memory. This implies that storing load-intensive data with good locality on the remote node will be unlikely to degrade performance, even on a standard PCIe switch-based system (i.e., without the modifications we make in our FPGA prototype). For benchmark 2 and benchmark 4, using the remote memory performs significantly worse than using the local memory. While the performance of benchmark 4 is in line with our expectations, the performance of benchmark 2 seems anomalous. It turns out that the writes in benchmark 2 are not being handled as expected. The Intel processor manual⑥ states that in write-through mode, writes either refill (option 0) or invalidate (option 1) the matching cache line. The mode can be configured by setting the page table attributes or the memory type range register (MTRR). If we firstly select option 0 and have one host (A) read a remote location and write the value 0 to it, and secondly have the local host (B) write it with the value 1, then when we have A reread the variable, it should get the value 0 from its cached copy. Instead, when A rereads the variable, it gets the value 1, indicating that option 1 is in effect regardless of our setting.

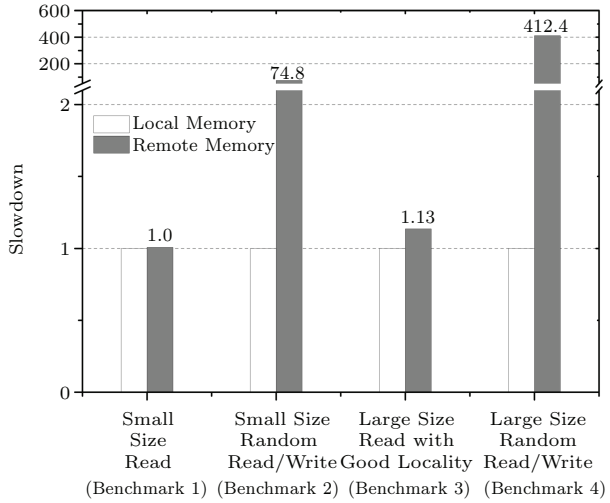


Fig.5. Performance of direct memory access. The lower the slowdown, the better.

Prefetcher Considerations. In order to mask latency and reduce the number of cache misses, many processors use hardware prefetchers to load cache lines before they are requested. The effectiveness of such mechanisms depends on the workload, and most prefetchers only consider local memory characteristics. Given the different latencies and bandwidths, improving the performance of remote memory access in the presence of prefetch engines requires that those mechanisms consider remote memory requirements.

Atomic Instructions. We find that our Intel processors (including Xeon and Core i7 architectures) do not allow lock prefix instructions to be used on remote memory in a PCIe system, which implies that remote memory cannot be used for lock/barrier variables or data structures that require atomic operations. This issue hinders the OS's ability access remote memory transparently.

2.2 Implications for General Communication

The communication performance of all intra-rack interconnects is affected by how well they make use of cache and memory. To better understand how memory hierarchy design influences general communication, we developed a user-level communication library based on remote memory. Before setting up a remote connection, the library allocates a message buffer in the user space and maps it to the remote node. Bypassing the kernel makes it possible to exploit the full speed and bandwidth of PCIe (i.e., applications can see 1-μs transfers). Our library implementation highlights a few limitations of the prototype system, such as address translation. The DMA engine can only use physical addresses to access memory, and thus it must convert virtual addresses before initiating a communication. User-level applications must therefore execute a time-consuming system call to perform this address translation.

One possible communication optimization technique for networks on chip (NoCs) is cache injection[5], which directly sends data from on-chip I/O devices (e.g., an NIC or accelerator) to cache instead of to off-chip memory. Leon *et al.*[5] showed that cache injection mechanisms can improve the performance of several collective communication operations in parallel programs. IBM implemented cache injection in their PowerEN[6] cloud platform. Note that implementing cache injection requires choosing the cache level and cache location to put the injected data, and this becomes increasingly difficult as cache hierarchies grow more and more complex.

⑥ Intel 64 and IA-32 architectures software developer's manual, Dec. 2012. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, Dec. 2014.

3 Memory Hierarchy Modifications

In light of these issues, we design and implement some memory hierarchy modifications to improve communication performance, particularly with respect to remote memory accesses.

3.1 Remote Memory Access

Since device configuration registers are rarely accessed and their accesses exhibit poor locality, a write-through/invalidate cache policy works well with the memory mapped I/O (MMIO) operations used to set them. But unlike MMIO, TSM treats remote locations like local memory. Due to the long latency of remote memory access, remote data with good locality of reference should be kept in cache as much as possible. If the cache uses a write-through/invalidate policy for remote memory (as required for MMIO of current x86 architectures), each write operation will cause an invalidation, and subsequent loads of the same address will miss in cache. A write-back policy would thus be more suitable.

Since remote memory accesses have different latencies and bandwidths compared to local memory, it is difficult for a single prefetch strategy to perform uniformly well. A more robust scheme would intelligently issue prefetches based on the latency of memory access.

For example, for remote memory, the prefetch degree (or prefetch distance, which indicates the number of cachelines per prefetching) of a stride prefetcher should be adjusted to account for the longer access latency. We can use a register to statically configure the "prefetch degree". TLB entries could be modified to identify remote pages, or the prefetcher could distinguish the physical addresses of local versus remote memory. When the prefetcher detects the remote memory, it uses the configured "prefetch degree" of the register. For local memory, the prefetcher uses original "prefetch degree" to prefetch.

The results in Section 2 show that the serial processing of read requests severely limits the performance of remote memory direct read operations. Obviously, improving concurrency for these read operations will greatly improve performance. In addition, if the remote memory space can be used for lock/barrier variables or atomic operations, unmodified applications could benefit from remote memory. For instance, the ARM processor uses exclusive loads and stores to execute atomic operation. Therefore, we can try to use ARM processor instead of x86 processor.

3.2 Communication

DMA controllers only recognize physical addresses, but user-level communication uses virtual addresses. Two common ways to solve that problem are: 1) using a system call to perform the translation and transmit the physical address to the DMA controller before each send operation; and 2) copying the message to a pre-allocated memory buffer for which the physical address is known. The former imposes a higher latency, and the latter results in higher CPU utilization. To avoid such performance penalties, we integrate a memory management unit (MMU) into the interconnect adapter to allow user-level applications to trigger transmissions with virtual addresses, as Pfister describes for Infiniband[3].

Like many modern processors, our FPGA platform has an integrated on-chip memory, which has similar performance with L2 cache (the LLC of our platform). To avoid the complexity of deciding where to place injected data in cache, data injection can, instead, be applied to the on-chip memory. Storing frequently accessed small messages and descriptors in the on-chip memory can improve communication performance and reduce L2 cache interference.

4 Platforms and Methodology

Current processors lack efficient support for remote memory access, and the PCIe standard does not support hardware offloading of some functions (such as queue management). These limitations led us to study architectural optimizations in our software simulator and on our customized interconnect system. For the latter, we implement a customized interconnect protocol in a Zynq7045 FPGA platform⑦. We add QPair hardware to better support communication and remote resource sharing mechanism in the protocol. Our simulator also models an ARM processor and our customized interconnect protocol: this let us validate our baseline simulator before adding processor modifications to support TSM access.

4.1 Simulator

Our simulation platform is based on gem5[8]. We develop the remote memory controller and the lo-

⑦ Zynq-7000 silicon devices. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/silicon-devices/index.htm, Mar. 2014.

cal/remote address map module. The remote memory controller includes a DMA engine which can transmit data between remote memory and local memory. We choose the ARMv7 processor for its support for exclusive loads and stores for atomic operations. We model a write-back cache that treats remote and local memory as the same, rather than treats the remote memory as I/O space.

In addition, the simulator can be configured to continuously send load requests to remote memory. The simulator configuration parameters are presented in Table 3.

Table 3. Simulator Configuration

Component	Parameter
Local memory latency (mean)	100 ns
Remote memory latency (mean)	600 ns
L1 DCache size	64 KB
L1 ICache size	32 KB
L2 Cache size	2 MB

4.2 Customized Protocol FPGA Prototype

The FPGA prototype overall structure and the protocol stack are shown in Fig.6. To overcome the limitations of the PCIe standard, we insert a network layer between the transaction layer and the data link layer in the customized protocol. The protocol is comprised of a control center and four logic layers, including the transport layer, the network layer, the data-link layer, and the physical layer. Here, we introduce our protocol layers down-top.

Data-Link and Physical Layers. The multiple instances of the data-link layer and the physical layer comprise I/O ports of the embedded switch. The data-link layer is responsible for reliable data transmission from point to point. That is, we design the error detection and recovery mechanisms to avoid packet data from transient faults. Besides, we provide sequence checking and flow control mechanisms to prevent packet loss. The physical layer is used for parallel/serial data conversion, and high speed serial data transmission.

Network Layer. The network layer is mainly composed of an embedded switch and a configurable routing table. Before the interconnect could be used, the routing table should be configured in advance through the provided APIs. Unless it is filled with valid data, the routing table cannot be used to direct packets forwarding from the ingress port to egress port of the embedded switch. To trade off the hardware cost and network efficiency, the embedded switch is designed with a 7×7 crossbar, which removes the need for external switches for node-level interconnections.

Transport Layer. There are three different channels integrated in the transport layer, namely instruction cut-through (ICT) channel, remote direct memory access (RDMA) channel, and queue pair (QPair) channel. The ICT channel is designed for instruction-level/cacheline-level remote memory accessing, which transmits load/store instructions or cacheline fill/write-back requests from processors to their remote counterparts. The RDMA channel could be used for large volume data transportation, e.g., page-level remote memory accessing. And the QPair channel is suitable for user-level socket-based communication. Besides the three channels, we also provide two user configurable tables for addresses translation, i.e., the local/remote address mapping table and the virtual/physical address translation table (hardware MMU). Here, the lo-

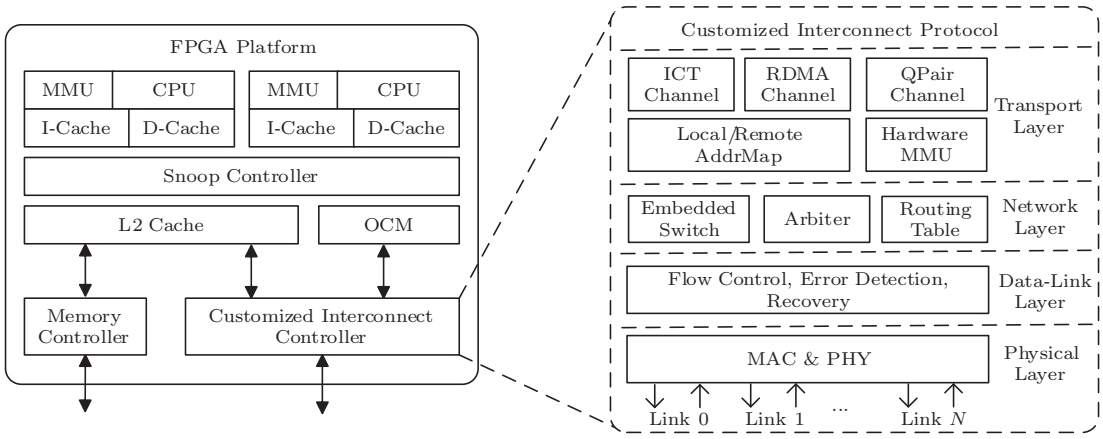


Fig.6. FPGA prototype system architecture.

cal/remote address mapping table is used by the ICT channel and the RDMA channel to achieve the address translation from local to remote. While, as it is known, the hardware MMU is used by the QPair channel to translate virtual address into physical one.

The QPair channel can support up to 64k connections, which are isolated from each other. Therefore, it is enough and safe, though we can use it in a way of sharing. Each connection consists of a send queue (SQ), a receive queue (RQ), and a completion queue (CQ). SQ is made up of send descriptors which contain data address, size, and some control signals. RQ and CQ are similar to SQ except that RQ's descriptors point to the receive buffer, and CQ's descriptors just contain completion information. It is convenient to transmit data that the sender only needs to post a send descriptor and wait for the completion information from CQ. Relatively, the receiver only needs to post a receive descriptor, wait for completion and read data from the receive buffer. It is easy to use and can support user-level applications with high quality.

Our FPGA-based prototype platform is based on Xilinx ZC706[8] development board, as shown in Fig.7. It contains an ARM dual-core Cortex-A9 MPCore processor, 1 GB of DDR3 DRAM and 256 KB on-chip memory. The link speed of our protocol is 2.5 Gbps, with 8 b/10 b encoding, and the latency is about 0.6 μs. Table 4 lists the major configuration parameters of the prototype system.

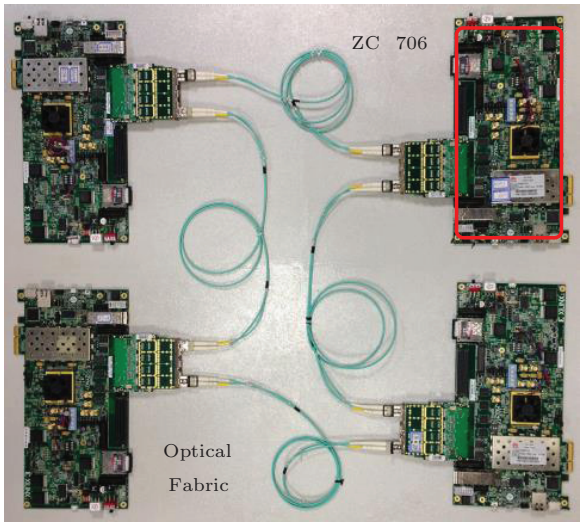


Fig.7. 4-node FPGA prototype system.

Table 4. Customized Protocol Prototype System Configuration

Component	Parameter
CPU	ARM Cortex-A9 667 MHz
L2 cache	512 KB
Memory	1 GB DDR3 1 066 MHz
On-chip RAM	256 KB
Link of our protocol	2.5 Gbps, 8b/10b encoding

5 Performance Evaluation

Fig.8 shows simulation results for our microbenchmarks when using local memory, remote memory with a write-through cache, and remote memory with a write-back cache and three different outstanding request queue (ORQ) sizes. Increasing the size of the bus' ORQ allows us to send read requests in larger batches. The figure shows that write-back mode (bars 3∼5) performs better than write-through/invalidate mode for benchmark 2 and benchmark 4 and that a larger ORQ can improve the performance of loading remote memory. For instance, for benchmark 4, ORQ16 is five times better than ORQ1. ORQ16 performs as well as ORQ1024, and thus modest hardware additions suffice to deliver good performance.

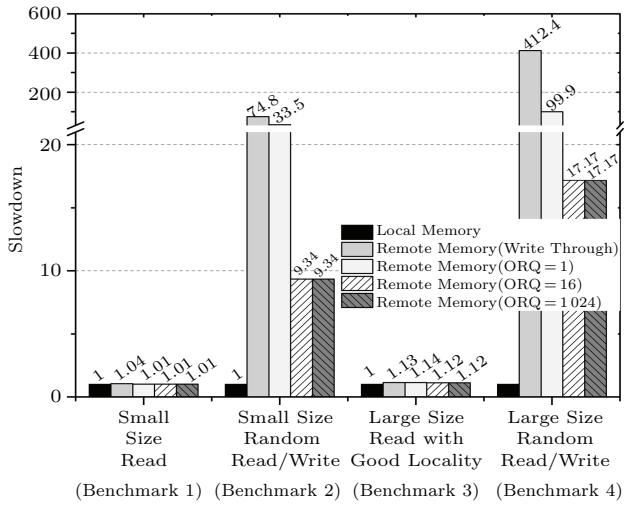


Fig.8. Performance of using TSM. The lower the slowdown, the better.

To evaluate prefetcher effectiveness, we run benchmark 3 and four memory-intensive applications from SPECCPU 2000 (gzip, gcc, mcf, gap). Fig.9 shows that increasing the prefetch degree improves performance.

[8] Zynq-7000 silicon devices. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/silicon-devices/index.htm, Mar. 2014.

Unsurprisingly, with a prefetch degree of 5, the microbenchmark runs seven times faster than the baseline with prefetching disabled.

Next we evaluate the effectiveness of using QPair instead of ports. Our initial customized protocol does not support cache coherence between DMA engine and the processor, and thus we use an uncached memory region for the DMA buffers. In a second implementation, we modify the hardware to support cache coherence. Note that the Zynq board's 256 KB on-chip memory[8] has higher bandwidth and lower latency than off-chip memory. Since the on-chip memory and the cache hierarchy are independent, the CPU must bypass cache to access on-chip memory. We use the on-chip memory to hold QPair configuration data and small communication data packets, which provides good access latency and reduces cache interference.

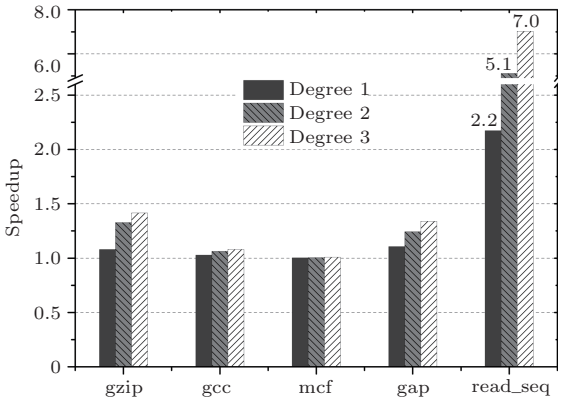


Fig.9. Performance of prefetching TSM. The higher the speedup, the better.

We implement user-level communication library on top of the QPair and again use our microbenchmarks for evaluation. To study the effectiveness of the cache, MMU, and on-chip memory, we conduct experiments on four configurations: 1) placing data in uncached memory space to avoid the coherence problem; 2) placing data in cached space with cache coherence maintained by hardware; 3) using a hardware MMU to accelerate address translation; and 4) placing data in on-chip memory and using the hardware MMU. In addition, we also evaluate the Gigabit Ethernet (GigE) of the FPGA platform. Fig.10 and Fig.11 compare the latency and bandwidth for different packet sizes.

In these experiments, the latency is calculated in the socket application layer, including time elapsed from hardware to software. Due to the low frequency (667 MHz) of the CPU clock, software has the biggest influence on latency. These two figures show that placing data in cache has lower latency and higher bandwidth than using uncached memory, and the MMU improves performance, as well. Using on-chip memory and MMU has the lowest latency — about 13 µs when transferring 64B packets, which is six times lower than GigE. We find the QPair latency to be a little large (7 µs) in our FPGA platform, because it needs both accessing main memory and a handful of software supports from communication library in order to translate socket requests into QPair operations.

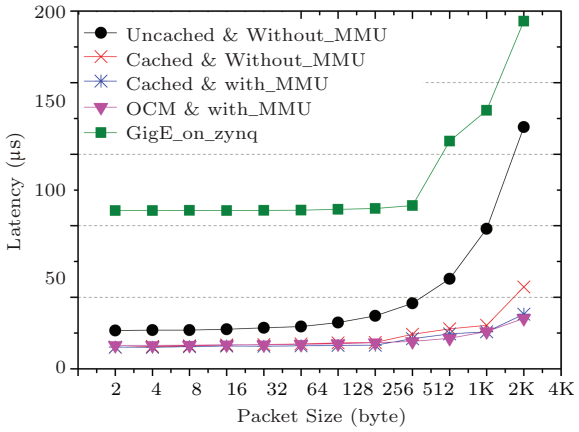


Fig.10. Latency of QPair.

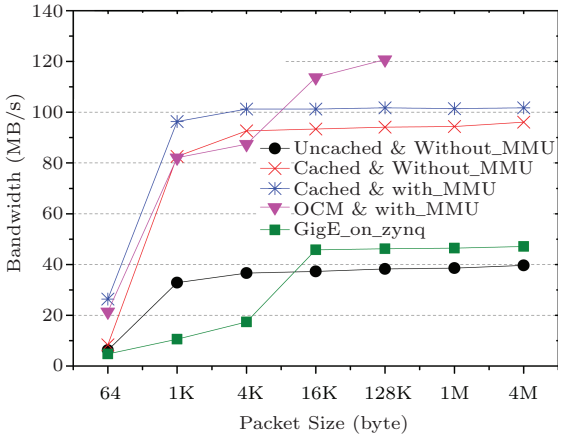


Fig.11. Bandwidth of QPair.

When the packet size is larger than 16 KB, using on-chip memory and MMU has the highest bandwidth — more than twice of that of GigE (120 MB/s). The on-chip memory has similar access latency and bandwidth to the L2 cache. However, in the case of using cache, only when the address hits the L2 cache will Qpair put data in L2 cache. Otherwise, data would be

put in main memory resulting in the performance reduction. Additionally, in the transmission of big data packets (larger than 16 KB), it is difficult to ensure all of these addresses will hit the L2 cache. Therefore, the performance of using on-chip memory is better than that of using cache. Besides, the performance of using hardware MMU is higher than that of using software to do address translation. In conclusion, using on-chip memory and MMU could gain the best performance. Note that the maximum packet size that can use the on-chip memory optimization is 128 KB (because the rest of the on-chip memory holds QPair configuration information).

6 Related Work

Datacenters require highly efficient, low-cost, flexible interconnects to manage the rapidly growing internal traffic generated by an increasingly diverse set of applications. Early datacenters often employed standard high performance computing (HPC) networking solutions like 10 GigE[8], and InfiniBand[3]. Newer designs increasingly include interconnects that better match the communication requirements of modern datacenter workloads. For instance, Freescale, IDT, Mobiveil, and Prodrive are promoting the use of ARM servers connected by Rapid-IO[9], and AMD SeaMicro is marketing ultra low-power, small-footprint datacenters connected by their proprietary Freedom^{TM} Supercomputer Fabric[10]. Even PCIe — once viewed as inappropriate for use as a general-purpose fabric — is being used within small-scale, tightly coupled data-center racks once connected by more traditional HPC network technologies.

The ability to share datacenter resources over these high-performance interconnects improves utilization, lowers cost, and increases flexibility. To that end, Deshpande *et al.* proposed MemX, which allows Xen virtual machines to transparently access cluster-wide memory resources over 1 Gigabit Ethernet[9]. Lim *et al.*[10] disaggregated server memory, moving a portion of main memory to separate memory blades optimized for both capacity and power usage. Their design both expands memory capacity and supports dynamic capacity sharing across multiple servers. Like the designs we analyzed here, their implementation also demonstrates PCIe's potential as a datacenter interconnect technology.

Recently, many industry leaders and researchers have proposed the concept of software-defined datacenter server. For instance, the rack scale architecture (RSA) has been proposed by Intel, which provides the ability to provision pooled memory resources[11]. Novakovic *et al.* proposed remote memory accessing through QPair mechanism based on special APIs[11], while we propose to do remote resources accessing through LD/ST instructions (not special API) and OS controlled RDMA channel, which presents less software overhead and good application-level transparency.

7 Conclusions

In addition to supporting explicit communication, emerging datacenter interconnects are increasingly being used to dynamically allocate and flexibly share resources. Local datacenter interconnects are now integrated into the same chip with the processor cores. Designing a new interconnection technology or communication protocol thus requires considering not only the interconnection itself but also the design of the processors and the local memory hierarchies on which they rely.

In this paper, we studied memory hierarchy implications for the design of high-speed datacenter interconnects. We built three complementary datacenter server models: a hardware PCIe prototype system for investigating the performance and optimization space of access remote memory and communication; a software simulator for evaluating specific memory hierarchy modifications that we cannot (yet) implement directly in hardware; and an FPGA-based customized interconnect system for studying the design space of hardware optimizations that leverage queue pair (QPair) design together with a customized, switchless protocol. Based on these platforms, we identified problems in existing memory hierarchies and discussed optimizations to better support remote memory access, including increasing the number of requests that can be issued concurrently via the interconnect interface, adapting the cache write policy to better exploit the locality of remote memory accesses, and increasing the prefetch degree to compensate for the increased latencies of remote memory accesses.

[9] Merritt R. RapidIO nudges ARM into servers, Jul. 2013. http://www.eetimes.com/document.asp?doc_id=1318957, Dec. 2014.

[10] Amd — SeaMicro technology overview, Oct. 2012. http://www.seamicro.com/sites/default/files/SM_TO01_64_v2.7.pdf, Dec. 2014.

[11] Rack scale architecture for cloud. http://www.chinacloud.cn/upload/2013-10/13102200313281.pdf, Oct. 2013.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] Benson T, Akella A, Maltz D. Network traffic characteristics of data centers in the wild. In *Proc. the 10th ACM SIGCOMM Conf. Internet Measurement*, Nov. 2010, pp.267-280.

[2] Regula J. Integrating rack level connectivity into a PCI Express switch. In *Proc. Hot Chips: A Symposium on High Performance Chips*, Aug. 2013, pp.259–266.

[3] Pfister G. An introduction to the InfiniBand^{TM} architecture. In *High Performance Mass Storage and Parallel I/O*, Cortes T, Jin H, Buyya R (eds.), John Wiley & Sons, 2001, pp.617-632.

[4] Hou R, Jiang T, Zhang L, Qi P, Dong J, Wang H, Gu X, Zhang S. Cost effective data center servers. In *Proc. the 19th IEEE Int. Symp. High Performance Computer Architecture*, Feb. 2013, pp.179-187.

[5] Léon E, Riesen R, Ferreira K, Maccabe A. Cache injection for parallel applications. In *Proc. the 20th ACM Int. Symp. High Performance Distributed Computing*, Jun. 2011, pp.15-26.

[6] Brown J, Woodward S, Bass B, Johnson C. IBM power edge of network processor: A wire-speed system on a chip. *IEEE Micro*, 2011, 31(2): 76-85.

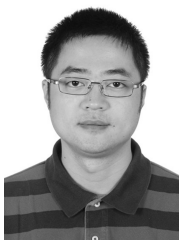
[7] Binkert N, Beckmann B, Black G *et al*. The gem5 simulator. *ACM SIGARCH Comput. Archit. News*, 2011, 39(2): 1-7.

[8] Hurwitz J, Feng W. End-to-end performance of 10-Gigabit Ethernet on commodity systems. *IEEE Micro*, 2004, 24(1): 10-12.

[9] Deshpande U, Wang B, Haque S, Hines M, Gopalan K. MemX: Virtualization of cluster-wide memory. In *Proc. the 39th International Conference on Parallel Processing*, Sept. 2010, pp.663-672.

[10] Lim K, Chang J, Mudge T, Ranganathan P, Reinhardt S, Wenisch T. Disaggregated memory for expansion and sharing in blade servers. In *Proc. the 36th International Symposium on Computer Architecture*, Jun. 2009, pp. 267-278.

[11] Novakovic S, Daglis A, Bugnion E, Falsafi B, Grot B. Scale-out NUMA. In *Proc. the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Feb. 2014, pp.3-18.



Tao Jiang received his M.S. degree in computer architecture from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2007. He is an assistant professor of ICT, CAS. His main research interests include computer architecture and operating system. He is a member of CCF, ACM, and IEEE.

Rui Hou received his Ph.D. in computer architecture from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2007. He is an associate professor in ICT, CAS. His research interests include data center system, CPU architecture and chip design, interconnect and workload analysis & optimization. He is a member of CCF, ACM, and IEEE.

Jian-Bo Dong received his Ph.D. degree in computer architecture from Graduate School of Chinese Academy of Sciences in 2012. He is now an assistant professor in Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests include datacenter architecture, high-speed interconnect, and reliability issues.

Lin Chai received her B.S. degree in computer science and technology from Beijing University of Posts and Telecommunications, in 2012. Now she is pursuing her M.S. degree in computer architecture at Institute of Computing Technology, Chinese Academy of Sciences, Beijing. Her research interests include computer architecture and operating system.

Sally A. McKee received her Ph.D. degree in computer architecture from the University of Virginia in 1995. She is an associate professor in Chalmers University of Technology, Gothenburg. Her research interests include analyzing application memory behavior and designing more efficient memory systems together with the software to exploit them. She is a member of ACM and IEEE.

Bin Tian received his M.S. degree in computer engineering from Zhejiang University, Hangzhou. He is a senior engineer of National High Performance Integrated Circuit Design Center (Shanghai). His main research interest includes VLSI design & verification methodology, electronic design automation and SoC design. He is a member of CCF.

Li-Xin Zhang received his Ph.D. degree in computer science from the University of Utah in 2001. He is a professor in Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests include data center computing systems, advanced cache/memory systems, architectural simulators, distributed/parallel computing, performance evaluation, and workload characterization. He is a member of ACM and IEEE.

Ning-Hui Sun is a professor and the director of Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). He received his B.S. degree from Peking University in 1989 and M.S. and Ph.D. degrees both in computer science from ICT, CAS in 1992 and 1999, respectively. He is the architect and main designer of the Dawning series high performance computers, from Dawning2000 to Dawning Nebulae. His research interests include computer architecture, operating system, and parallel algorithm. He is a fellow of CCF and a member of ACM and IEEE.