Network Interface Cards as First-Class Citizens*

W. Feng[1] P. Balaji[2] A. Singh[1]

[1]Dept. of Computer Science,
Virginia Tech,
{feng, ajeets}@cs.vt.edu

[2]Mathematics and Computer Science,
Argonne National Laboratory,
balaji@mcs.anl.gov

Abstract

Network performance has improved by over an order-of-magnitude in the past decade. While, the overall system architecture itself has made modification attempts to match such growth, these changes are mostly "incremental enhancements, tweaks and adjustments" that try to keep the hardware control hierarchy away from the network communication path. However, we are rapidly reaching a stage where these adjustments are no longer sufficient to allow the network to realize its full capacity without being bottlenecked by the hardware control hierarchy in the system. Thus, rather than a "band-aid" fix to high-performance networking and I/O at the compute node, in this paper we re-visit a more radical approach that elevates the network interface card from a second-class citizen that resides out on an I/O interconnect to a first-class citizen that resides on the system bus. In this architecture, the network adapter would have its own cache, its own memory, its own processing units and follows the overall cache coherency and memory management protocols, much like what a regular CPU does. This architecture takes a step beyond existing system architectures and allows for direct communication data management without having to coordinate with the north bridge on every access.

1 Introduction

As we move to multi-petascale and exascale computing systems, massive-scale computing systems, consisting of hundreds of thousands to millions of processing units, are being assembled. For these processing units to efficiently communicate with each other and present themselves as one large supercomputer, the network communication infrastructure should be fast, efficient, and scalable.

In order to meet such growing demands, network performance has improved by more than an order of magnitude over the past decade. While the overall system architecture itself has made modified to try to match the growing capabilities of these networks, these changes are mostly incremental enhancements or adjustments that try to keep the hardware control hierarchy away from the network communication path. However, as communication latencies approach a microsecond and bandwidths reach tens of gigabits per second, these adjustments are no longer sufficient. As such, rather than further "band-aid" fixes to high-performance networking and I/O, we need more radical approaches to re-design the overall system architecture for improved communication capability.

Stage 1: System Architecture (Shared Bus Networks): Traditional system architectures consist of network adapters being connected to I/O buses that are shared with other devices. These I/O buses connect to an I/O controller hub (i.e., Southbridge) which, in turn, would be connected to the Northbridge through a high-speed channel. Finally, the Northbridge with an integrated memory controller could direct data to memory. That is, data has to traverse three steps before it could reach memory. In this architecture (Figure 1(a)), network data has no direct access to the CPUs themselves before going into memory.

Stage 2 System Architecture (Dedicated I/O Interconnects): As network speeds grew beyond a gigabit per second, system designers realized that the aforementioned architecture was too restrictive for high-speed network devices to perform effective communication. Thus, they upgraded the architecture to get rid of the last step in the hardware control hierarchy and connect network adapters directly to the Northbridge using dedicated point-to-point links, much like the I/O controller hub used to be traditionally connected (Figure 1(b)). Technologies such as PCI Express (PCIe) and cave-mode Hypertransport (HT) made these architectures possible, allowing network adapters to connect to the Northbridge with speeds matching those of current memory bandwidths. However, even this upgraded system architecture lagged behind the demands of rapidly increasing system sizes in two areas: (i) network data still had to pass through the Northbridge before it could be transferred to memory, which means that memory access was still based on an access negotiation and hardware direct memory access (DMA) model, and (ii) network data still did not have access to go directly to the processors, much like the stage 1 architecture.

Stage 3 System Architecture (Direct Cache Access): Recently, network hardware vendors and system architects have been pushing for the next upgrade to system architectures to better support network devices in the form for **direct cache access (DCA)** architectures. While the hardware layout of the DCA architecture is very similar to that of the previous

*This work was supported in part by the Virginia Tech Foundation and the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

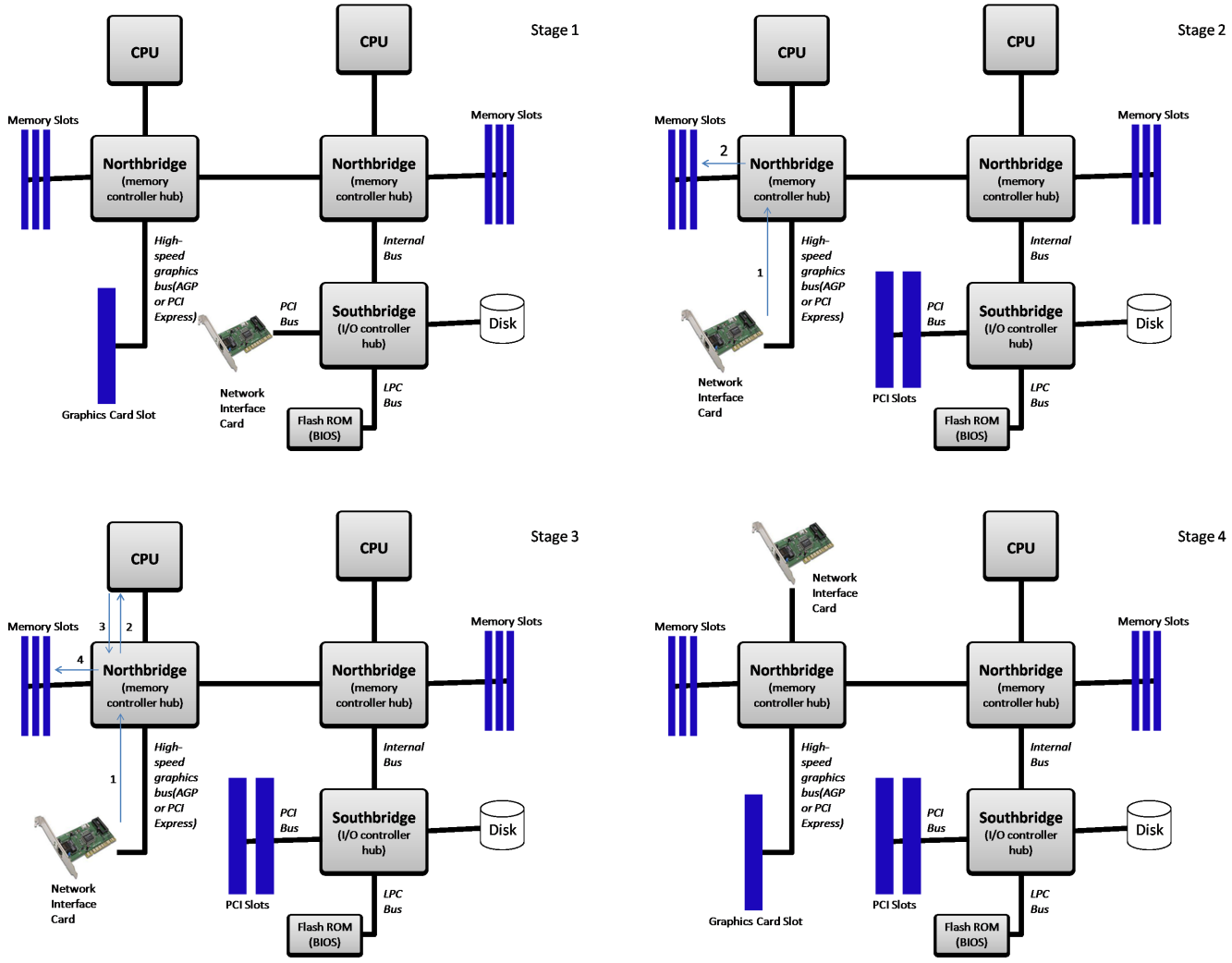

Figure 1: Four System Architectural Stages: (a) Shared Bus Networks; (b) Dedicated I/O Interconnects; (c) Direct Cache Access; (d) Networks as First-Class Citizens

generation architecture, it makes one valuable addition—in this architecture, network adapters can directly access CPUs and directly write to their caches instead of having to write to memory (Figure 1(c)). This improves access latency, as data movement is not limited by memory bandwidth and data does not have to be fetched to the CPU cache after it arrives over the network.

Stage 4 System Architecture (Networks as First-Class Citizens): In this generation of architecture, which is not currently available, we envision that a network adapter would sit on par with the other CPUs in the system; the network adapter would have its own cache, its own memory, its own processing units and would follow the overall cache coherency and memory management protocols, much like what a regular CPU does. This architecture takes a step beyond direct cache access and allows for a similar mechanism but without having to coordinate with the Northbridge on every access (Figure 1(d)). Such a mechanism would allow network communication to be truly bounded only by the network protocol and communication stack rather than the coordination overhead involved in current architectures' hardware control hierarchy.

To date, there have been several projects that have looked at memory-integrated network interfaces. Of particular note is the research of Mukherjee et al. [10] and Feng [2].

However, in the commercial sector, such work does not exist. The most closely related work is the Communciation Streaming Architecture (CSA) from Intel [4, 8], which elevated the NIC to a first-class citizen by connecting it directly to the memory controller hub, i.e., Northbridge, and which was later subsumed by PCI Express. However, in both cases — CSA and PCI Express, neither *treat* the NIC as a first-class citizen as neither are integrated with the memory subsystem and access architecture. Instead, both CSA and PCI Express hold onto their network-I/O access roots.

We believe that this may be the time to re-visit these ideas once again. In addition to integrating the NIC into the memory subsystem, we want to introduce more powerful, network-specific, processing capabilities to the NIC and treat the NIC as a network co-processor. However, we will avoid the trap of putting an actual CPU on the NIC as on-card CPUs typically add overhead, increase latency, and in many cases, decrease peak bandwidth available to the application, because an actual CPU is a generalized compute processor rather than a task-specific network processor.

In general, the advantages gained by moving the NIC to the system bus include lower latency, higher bandwidth, and the ability to run a cache-coherency protocol over the NIC so that data movement is provided automatically by the memory management system. This would allow application programmers to focus on parallelizing programs (by hiding the details of managing parallelism and data locality from the programmer) rather than burdening them with managing explicit data movement into and out of a computing node. The disadvantages include the loss of explicit control over data movement to/from the network and the non-standard interface to the NIC. (With the NIC on the I/O bus, there are several standard interfaces, e.g., PCI for PCs.) However, bridges from the system bus are possible. An example of such a bridge is Intel's Accelerated Graphics Port, which provides a fast and dedicated pipe to a graphics accelerator.

2 Design of a Memory-Integrated NIC

Here we study different aspects of the network interface card (NIC) in order to better understand what design decisions need to be made to support a memory-integrated NIC, as summarized in Table 1.

Virtualize User Addressing of the NIC. This issue has been recently addressed by research into OS-bypass protocols [13, 3, 9, 11, 14, 5]. In order to virtualize the NIC to a user process, address translation and protection must be provided. Address translation allows a user process to access the NIC through virtual addresses while protection isolates user processes from each other. But rather than virtualizing the NIC via the operating system (OS), it is virtualized through virtual memory hardware for better performance. Thus, we can dramatically improve the performance of accessing NIC memory via virtual memory by having the OS map NIC memory pages directly into user space and then having the virtual memory hardware translate these memory-mapped virtual addresses to appropriate physical address in the NIC memory.

Cached NIC Registers. Network messages can be cached in CPU and NIC caches (like regular cacheable memory), thereby, in general, increasing the effective host-to-network bandwidth, decreasing the effective host-to-network latency, and reducing traffic on the system bus by taking advantage of spatial and temporal locality. Currently, NIC registers are not cached because no coherency protocol is run over the I/O bus. Furthermore, CPU access to NIC memory often have side effects unlike normal cacheable memory.

Cache-Block Transfers. Network messages can be transferred between the CPU cache, NIC cache, and main memory via cache-block transfers rather than DMA transfers. This allows messages to be moved in a single cache-block transfer and avoids incurring the DMA initiation and teardown overhead of multiple (and bursty) DMA transfers of a few bytes at a time.

Memory-Based Queue. The API to the NIC can be designed as a memory-based queue rather than an explicit data-movement primitive to/from the NIC. The current practice

I/O Access	Memory Access
Device on I/O bus	Memory on system bus
Indirect via OS	Direct via protected user access
Uncached NIC regsters	Cached NIC registers
Ad-hoc data movement	Cache block transfers
Explicit data movement via API	Memory-based queue
Notification via interrupts	Notification via cache invalidation
Limited device memory	Plentiful memory
No out-of-order access and speculation	Out-of-order access and speculation

Table 1: Transforming NIC Access from I/O Access to Memory Access

of using data-movement primitives to move information to/from the NIC couples CPU involvement with the NIC. By using a memory-based queue API, we can decouple the CPU from the NIC, thus freeing the CPU to do useful work while network communication is occurring; and sending/receiving packets simply amounts to writing/reading queue memory. In addition, we avoid the side effects of explicit data movement by treating NIC queue accesses as side-effect-free memory accesses. The potential disadvantage of this approach is that the user no longer has knowledge of how data is actually moved between the CPU and the NIC (although a counter-argument would be that the user does not have knowledge of how data is actually moved between the CPU and memory either).

Notification via Cache Invalidation. The CPU can be notified of NIC events indirectly via cache invalidation rather than directly via heavyweight interrupts which pollute the cache or directly via polling which wastes CPU cycles which could be better spent on computation.

Virtual Memory as an Automatic Overflow Buffer. The memory on a NIC may overflow when bursts of network messages arrive at the NIC. However, with the NIC being treated as a network co-processor and CPU peer, a cache replacement policy to main memory could buffer these messages automatically without any CPU intervention. However, the gain of having plentiful buffer space comes at the cost of additional run-time overheads.

Out-of-Order Accesses and Speculative Loads. Out-of-order accesses and speculative loads on a CPU's accesses to a NIC (like side-effect-free regular memory accesses) are possible. This is of exceptional importance to the run-time software and hardware community as it not only allows for the potential re-ordering of compute-based instructions but also allows for the potential re-ordering of network-based instructions in a dynamic pipeline. Currently, this cannot be done because I/O buses do not adequately support multiple outstanding transactions whereas the system bus does; side effects in NICs often force NIC accesses to be performed in-order; and NICs do not provide any rollback mechanisms if the CPU's speculation is incorrect.

3 Related Work

There are several research directions that relate to networks as first class citizens. We discuss some of them here.

Point-to-Point Interconnects for Network Device Connectivity: In some ways, there have already been some research directions that try to alleviate network interface cards to a pseudo-first-class-citizen like status by connecting to them through dedicated point-to-point links instead of shared I/O buses. PCI Express (PCIe) [12] was introduced by Intel in 2004 as a structured point-to-point links that connected either directly to the memory controller or through a series of cross-bar switches. This allowed network adapters to plug directly at the end of the PCIe link and use the fabric in a dedicated manner. The hypertransport consortium [1] went a step further to utilize a similar idea, but to have a unified point-to-point fabric that not only connected the process/memory system to the network adapter, but also to the other processors. Thus, communication between processors was as fast as that with the network adapter. Intel QuickPath [7] utilizes similar ideas for the Intel-based platforms.

While all of these research directions attempt to alleviate the network adapter's position to that of the CPU with respect to the communication overhead between the CPU and the NIC, there are still several issues in which a NIC is not considered on-par with the CPU, primarily with respect to access rights to memory and cache regions, as identified in this paper.

Direct Cache Access Capabilities for Networks: Another technology that allows NICs to have an improved access to CPU cache is *direct cache access* [6]. The short version of this technology is that instead of writing data to memory, network adapters would be allowed to directly write data into a processor cache. This allows applications to avoid fetching from memory for small and critical data that just arrived from another remote node. While this is a step-up for network adapters, they still have to follow regular access-request pro-

tocols unlike CPUs which follow more sophisticated cache coherence logic between themselves.

4 Concluding Remarks

As network performance continues to increase, small tweaks and adjustments to the system architecture to allow the network to sustain its performance potential are no longer sufficient. In this paper we revisited a more radical approach to system architecture that elevates the network interface card from a second-class citizen that resides out on an I/O interconnect to a first-class citizen that resides on the system bus. We presented various details of this architecture, including issues, pitfalls, and various solutions.

References

[1] HyperTransport Consortium. http://www.hypertransport.org.

[2] W. Feng. Network interface cards as first-class citizens. Invited Talk, The Ohio State University, 1999.

[3] http://www.hippi.org/cST.html. Scheduled transfer protocol (st), (st is also being commercially promoted as part of gsn), 1996–present.

[4] http://www.intel.com/design/network/events/idf/csa.htm. Communication streaming architecture (csa), 2003–2004.

[5] http://www.viarch.org. Vi architecture, 1998–1999.

[6] R. Huggahalli, R. Iyer, and S. Tetrick. Direct Cache Access for High Bandwidth Network I/O. In *ISCA*, 2005.

[7] Intel. http://www.intel.com/technology/quickpath.

[8] Intel Corporation. Communication streaming architecture — reducing the pci network bottleneck. White Paper, Intel Corporation, 2003. Available online (4 pages).

[9] M. Lauria and A. Chien. High-performance messaging on workstations: Illinois fast messages (fm) for myrinet. In *Proceedings of Supercomputing '95*, November 1995.

[10] S. S. Mukherjee and M. D. Hill. Making network interfaces less peripheral. *IEEE Computer*, 31(10):70–76, October 1998.

[11] S. Pakin, V. Karamcheti, and A. Chien. Fast messages (fm): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, 5(2):60–73, April-June 1997.

[12] PCI-SIG. http://www.pcisig.com/specifications/pciexpress/base2.

[13] I. R. Philp and Y. Liong. The scheduled transfer (st) protocol. In *Proceedings of Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, January 1999.

[14] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. Pm: An operating system coordinated high-performance communication library. In *Proceedings of High-Performance Computing and Networking '97*, pages 708–717, April 1997.