## 6.9 Communicating to the Outside World: Cluster Networking

This online section describes the networking hardware and software used to connect the nodes of cluster together. As there are whole books and courses just on networking, this section only introduces the main terms and concepts. While our example is networking, the techniques we describe apply to storage controllers and other I/O devices as well.

Ethernet has dominated local area networks for decades, so it is not surprising that clusters primarily rely on Ethernet as the cluster interconnect. It became commercially popular at 10 Megabits per second link speed in the 1980s, but today 1 Gigabit per second Ethernet is standard and 10 Gigabit per second is being deployed in datacenters. Figure 6.9.1 shows a network interface card (NIC) for 10 Gigabit Ethernet.
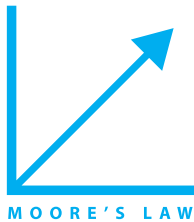
Computers offer high-speed links to plug in fast I/O devices like this NIC. While there used to be separate chips to connect the microprocessor to the memory and high-speed I/O devices, thanks to *Moore's Law* these functions have been absorbed into the main chip in recent offerings like Intel's Sandy Bridge. A popular high-speed link today is **PCIe**, which stands for **Peripheral Component Interconnect Express**. It is called a *link* in that the basic building block, called a *serial lane*, consists of just four wires: two for receiving data and two for transmitting data. This small number contrasts with an earlier version of PCI that consisted of 64
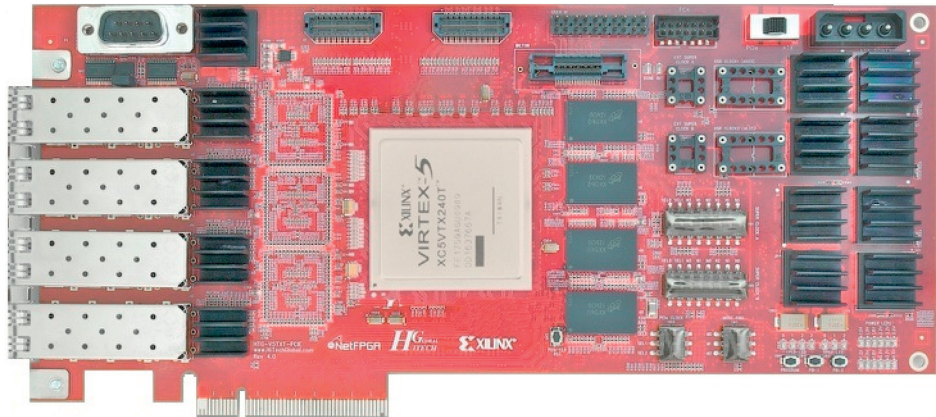
MOORE'S LAW

FIGURE 6.9.1 The NetFPGA 10-Gigabit Ethernet card (see http://netfpga.org/), which connects up to four 10-Gigabit/sec Ethernet links. It is an FPGA-based open platform for network research and classroom experimentation. The DMA engine and the four "MAC chips" in Figure 6.9.2 are just portions of the Xilinx Virtex FPGA in the middle of the board. The four PHY chips in Figure 6.9.2 are the four black squares just to the right of the four white rectangles on the left edge of the board, which is where the Ethernet cables are plugged in.

wires, which was called a *parallel bus*. PCIe allows anywhere from 1 to 32 lanes to be used to connect to I/O devices, depending on its needs. This NIC uses PCI 1.1, so each lane transfers at 2 Gigabits/second.

The NIC in Figure 6.9.1 connects to the host computer over an 8-lane PCIe link, which offers 16 Gigabits/second in both directions. To communicate, a NIC must both send or transmit messages and receive them, often abbreviated as TX and RX, respectively. For this NIC, each 10G link uses separate transmit and receive queues, each of which can store two full-length Ethernet packets, used between the Ethernet links and the NIC. Figure 6.9.2 is a block diagram of the NIC showing the TX and RX queues. The NIC also has two 32-entry queues for transmitting and receiving between the host computer and the NIC.

To give a command to the NIC, the processor must be able to address the device and to supply one or more command words. In **memory-mapped I/O**, portions of the address space are assigned to I/O devices. During initialization (at boot time), PCIe devices can request to be assigned an address region of a specified length. All subsequent processor reads and writes to that address region are forwarded over PCIe to that device. Reads and writes to those addresses are interpreted as commands to the I/O device.

For example, a write operation can be used to send data to the network interface where the data will be interpreted as a command. When the processor issues the address and data, the memory system ignores the operation because the address indicates a portion of the memory space used for I/O. The NIC, however, sees the operation and records the data. User programs are prevented from issuing I/O operations directly, because the OS does not provide access to the address space assigned to the I/O devices, and thus the addresses are protected by the address translation. Memory-mapped I/O can also be used to transmit data by writing or reading to select addresses. The device uses the address to determine the type of command, and the data may be provided by a write or obtained by a read. In any event, the address encodes both the device identity and the type of transmission between processor and device.

memory-mapped I/O An I/O scheme in which portions of the address space are assigned to I/O devices, and reads and writes to those addresses are interpreted as commands to the I/O device.
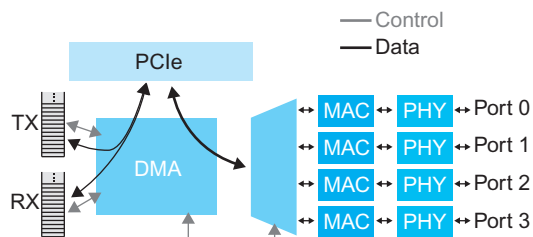

FIGURE 6.9.2 Block diagram of the NetFPGA Ethernet card in Figure 6.9.1 showing the control paths and the data paths. The control path allows the DMA engine to read the status of the queues, such as empty vs. on-empty, and the content of the next available queue entry. The DMA engine also controls port multiplexing. The data path simply passes through the DMA block to the TX/RX queues or to main memory. The "MAC chips" are described below. The PHY chips, which refer to the physical layer, connect the "MAC chips" to physical networking medium, such as copper wire or optical fiber.

While the processor could transfer the data from the user space into the I/O space by itself, the overhead for transferring data from or to a high-speed network could be intolerable, since it could consume a large fraction of the processor. Thus, computer designers long ago invented a mechanism for offloading the processor and having the device controller transfer data directly to or from the memory without involving the processor. This mechanism is called **direct memory access** (DMA).

DMA is implemented with a specialized controller that transfers data between the network interface and memory independent of the processor, and in this case the DMA engine is inside the NIC.

To notify the operating system (and eventually the application that will receive the packet) that a transfer is complete, the DMA sends an *I/O interrupt*.

An I/O interrupt is just like the exceptions we saw in Chapters 4 and 5, with two important distinctions:

1. An I/O interrupt is asynchronous with respect to the instruction execution. That is, the interrupt is not associated with any instruction and does not prevent the instruction completion, so it is very different from either page fault exceptions or exceptions such as arithmetic overflow. Our control unit needs only check for a pending I/O interrupt at the time it starts a new instruction.

2. In addition to the fact that an I/O interrupt has occurred, we would like to convey further information, such as the identity of the device generating the interrupt. Furthermore, the interrupts represent devices that may have different priorities and whose interrupt requests have different urgencies associated with them.

To communicate information to the processor, such as the identity of the device raising the interrupt, a system can use either vectored interrupts or an exception identification register, called the Cause register in MIPS (see Section 4.9). When the processor recognizes the interrupt, the device can send either the vector address or a status field to place in the Cause register. As a result, when the OS gets control, it knows the identity of the device that caused the interrupt and can immediately interrogate the device. An interrupt mechanism eliminates the need for the processor to keep checking the device and instead allows the processor to focus on executing programs.

The Role of the Operating System in Networking

The operating system acts as the interface between the hardware and the program that requests I/O. The network responsibilities of the operating system arise from three characteristics of networks:

1. Multiple programs using the processor share the network.

2. Networks often use interrupts to communicate information about the operations. Because interrupts cause a transfer to kernel or supervisor mode, they must be handled by the operating system (OS).

3. The low-level control of an network is complex, because it requires managing a set of concurrent events and because the requirements for correct device control are often very detailed.

These three characteristics of networks specifically and I/O systems in general lead to several different functions the OS must provide:

Hardware/ Software Interface

- The OS guarantees that a user's program accesses only the portions of an I/O device to which the user has rights. For example, the OS must not allow a program to read or write a file on disk if the owner of the file has not granted access to this program. In a system with shared I/O devices, protection could not be provided if user programs could perform I/O directly.

- The OS provides abstractions for accessing devices by supplying routines that handle low-level device operations.

- The OS handles the interrupts generated by I/O devices, just as it handles the exceptions generated by a program.

- The OS tries to provide equitable access to the shared I/O resources, as well as schedule accesses to enhance system throughput.

The software inside the operating system that interfaces to a specific I/O device like this NIC is called a **device driver**. The driver for this NIC follows five steps when transmitting or receiving a message. Figure 6.9.3 shows the relationship of these steps as an Ethernet packet is sent from one node of the cluster and received by another node in the cluster.

device driver A program that controls an I/O device that is attached to the computer.

First, the transmit steps:

1. The driver first prepares a packet buffer in host memory. It copies a packet from the user address space into a buffer that it allocates in the operating system address space.

2. Next, it "talks" to the NIC. The driver writes an *I/O descriptor* to the appropriate NIC register that gives the address of the buffer and its length.

3. The DMA in the NIC next copies the outgoing Ethernet packet from the host buffer over PCIe.

4. When the transmission is complete, the DMA interrupts the processor to notify the processor that the packet has been successfully transmitted.

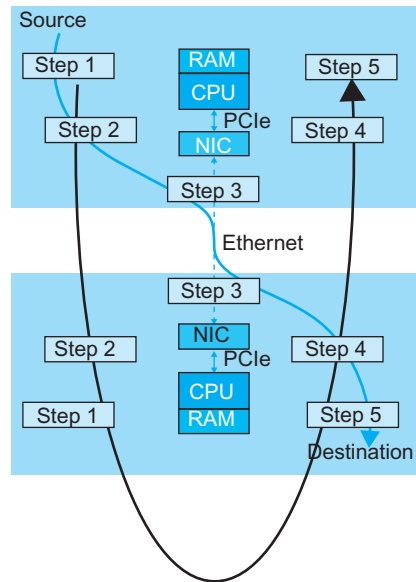5. Finally, the driver de-allocates the transmit buffer.

FIGURE 6.9.3 Relationship of the five steps of the driver when transmitting an Ethernet packet from one node and receiving that packet on another node.

Next, the receive steps:

1. First, the driver prepares a packet buffer in host memory, allocating a new buffer in which to place the received packet.

2. Next, it "talks" to the NIC. The driver writes an I/O descriptor to the appropriate NIC register that gives the address of the buffer and its length.

3. The DMA in the NIC next copies the incoming Ethernet packet over PCIe into the allocated host buffer.

4. When the transmission is complete, the DMA interrupts the processor to notify the host of the newly received packet and its size.

5. Finally, the driver copies the received packet into the user address space.

As you can see in Figure 6.9.3, the first three steps are time critical when transmitting a packet (since the last two occur after the packet is sent), and the last three steps are time critical when receiving a packet (since the first two occur before a packet arrives). However, these non-critical steps must be completed before individual nodes run out of resources, such as memory space. Failure to do so negatively affects network performance.

Improving Network Performance

The importance of networking in clusters means it is certainly worthwhile to try to improve performance. We show both software and hardware techniques.

Starting with software optimizations, one performance target is reducing the number of times the packet is copied, which you may have noticed happening repeatedly in the five steps of the driver above. The *zero-copy* optimization allows the DMA engine to get the message directly from the user program data space during transmission and be placed where the user wants it when the message is received, rather than go through intermediary buffers in the operating system along the way.

A second software optimization is to cut out the operating system almost entirely by moving the communication into the user address space. By not invoking the operating system and not causing a context switch, we can reduce the software overhead considerably.

In this more radical scenario, a third step would be to drop interrupts. One reason is that modern processors normally go into lower power mode while waiting for an interrupt, and it takes time to come out of low power to service the interrupt as well for the disruption to the pipeline, which increases latency. The alternative to interrupts is for the processor to periodically check status bits to see if I/O operation is complete, which is called **polling**. Hence, we can require the user program to poll the NIC continuously to see when the DMA unit has delivered a message, and as a side effect the processor does not go into low power mode.

polling The process of periodically checking the status of an I/O device to determine the need to service the device.

Looking at hardware optimizations, one potential target for improvement is in calculating the values of the fields of the Ethernet packet. The 48-bit Ethernet address, called the *Media Access Control address* or *MAC address*, is a unique number assigned to each Ethernet NIC. To improve performance, the "MAC chip"—actually just a portion of the FPGA on this NIC—calculates the value for the preamble fields and the CRC field (see Section 5.5). The driver is left with placing the MAC destination address, MAC source address, message type, the data payload, and padding if needed. (Ethernet requires that the minimum packet, including the header and CRC fields but not the preamble, be 64 bytes.) Note that even the least expensive Ethernet NICs do CRC calculation in hardware today.

A second hardware optimization, available on the most recent Intel processors such as Ivy Bridge, improves the performance of the NIC with respect to the memory hierarchy. *Direct Data IO (DDIO)* allowing up to 10% of the last level cache is used as a fast scratchpad for the DMA engine. Data is copied directly into the last level cache rather than to DRAM by the DMA, and only written to DRAM upon eviction from the cache. This optimization helps with latency, but also with bandwidth; some memory regions used for control might be written by the NIC repeatedly, and these writes no longer need to go to DRAM. Thus, DDIO offers benefits similar to those of a write back cache versus a write through cache (Chapter 5).

Let's look at an object store that follows a client-server architecture and uses most of the optimizations above: zero copy messaging, user space communication, polling instead of interrupts, and hardware calculation of preamble and CRC. The driver
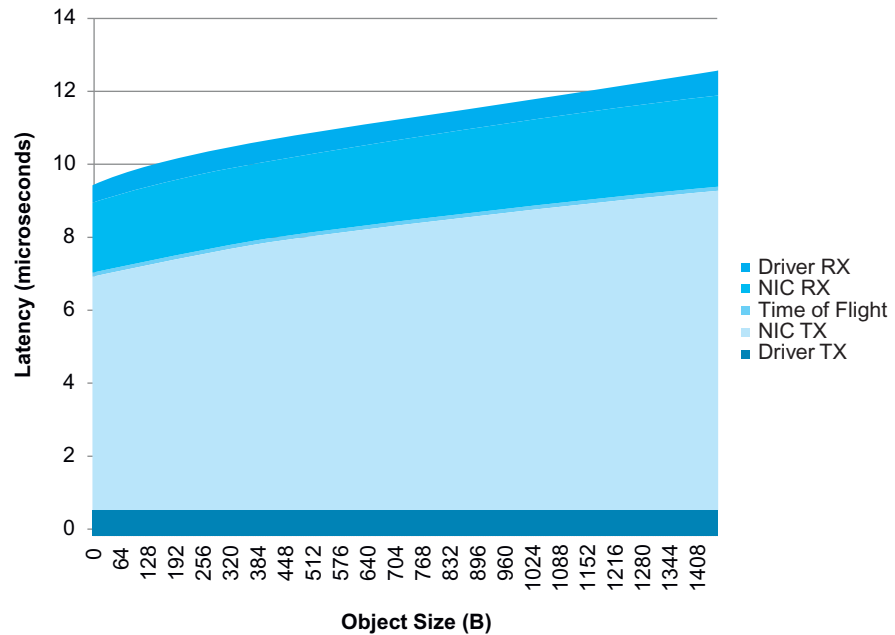
FIGURE 6.9.4   Time to send an object broken into transmit driver and NIC hardware time vs. receive driver and NIC hardware time. NIC transmit time is much larger than the NIC receive time because transmit requires more PCIe round-trips. The NIC does PCIe reads to read the descriptor and data, but on receive the NIC does PCIe writes of data, length of data, and interrupt. PCIe reads incur a round trip latency because NIC waits for the reply, but PCIe writes require no response because PCIe is reliable, so PCIe writes can be sent back-to-back.

operates in user address space as a library that the application invokes. It grants this application exclusive and direct access to the NIC. All of the I/O register space on the NIC is mapped into the application, and all of the driver state is kept in the application. The OS kernel doesn't even see the NIC as such, which avoids the overheads of context switching, the standard kernel network software stack, and interrupts.

Figure 6.9.4 shows the time to send an object from one node to another. It varies from about 9.5 to 12.5 microseconds, depending on the size of the object. Here is the time for each step in microseconds:

0.7 – for the client "driver" (library) to make the request (Driver TX in Figure 6.9.4).

6.4 to 8.7 – for the NIC hardware to transmit the client's request over the PCIe bus to the Ethernet, depending on the size of the object (NIC TX).

0.02 – to send object over the 10 G Ethernet (Time of Flight). The time of flight is limited by speed of light to 5 ns per meter. The three-meter cables used in this measurement mean the time of flight is 15 ns, which is too small to be clearly visible in the figure.

1.8 to 2.5 – for the NIC hardware to receive the object, depending on its size (NIC RX).

0.6 – for the server "driver" to transmit the message with the requested object to the app (Driver RX).

Now that we have seen how to measure the performance of network at a low level of detail, let's raise the perspective to see how to benchmark multiprocessors of all kinds with much higher level programs.

Elaboration: There are three versions of PCIe. This NIC uses PCIe 1.1, which transfers at 2 gigabits per second per lane, so this NIC transfers at up to 16 gigabits per second in each direction. PCIe 2.0, which is found on most PC motherboards today, doubles the lane bandwidth to 4 gigabits per second. PCIe 3.0 doubles again to 8 gigabits per second, and it is starting to be found on some motherboards. We applaud the standard committee's logical rate of bandwidth improvement, which has been about $2^{version\ number}$ gigabits/second. The limitations of the Virtex 5 FPGA prevented the NIC from using faster versions of PCIe.

Elaboration: While Ethernet is the foundation of cluster communication, clusters commonly use higher-level protocols for reliable communication. Transmission Control Protocol and Internet Protocol (TCP/IP), although invented for planet-wide communication, is often used inside a warehouse scale computer, due in part to its dependability. While IP makes no deliver guarantees in the protocol, TCP does. The sender keeps the packet sent until it gets the acknowledgment message back that it was received correctly from the receiver. The receiver knows that the message was not corrupted along the way, by double-checking the contents with the TCP CRC field. To ensure that IP delivers to the right destination, the IP header includes a checksum to make sure the destination number remains unchanged. The success of the Internet is due in large part to the elegance and popularity of TCP/IP, which allows independent local area networks to communicate dependably. Given its importance in the Internet and in clusters, many have accelerated TCP/IP, using techniques like those listed in this section [Regnier, 2004].

Elaboration: Adding DMA is another path to the memory system—one that does not go through the address translation mechanism or the cache hierarchy. This difference generates some problems both in virtual memory and in caches. These problems are usually solved with a combination of hardware techniques and software support. The difficulties in having DMA in a virtual memory system arise because pages have both a physical and a virtual address. DMA also creates problems for systems with caches, because there can be two copies of a data item: one in the cache and one in memory. Because the DMA issues memory requests directly to the memory rather than through the processor cache, the value of a memory location seen by the DMA unit and the processor may differ. Consider a read from a NIC that the DMA unit places directly into memory. If some of the locations into which the DMA writes are in the cache, the processor will receive the old value when it does a read. Similarly, if the cache is write-back, the DMA may read a value directly from memory when a newer value is in the

cache, and the value has not been written back. This is called the *stale data problem* or coherence problem (see Chapter 5). Similar solutions for coherence are used with DMA.

Elaboration: Virtual Machine support clearly can negatively impact networking performance. As a result, microprocessor designers have been adding hardware to reduce the performance overhead of virtual machines for networking in particular and I/O in general. Intel offers *Virtualization Technology for Directed I/O* (*VT-d*) to help virtualize I/O. It is an I/O memory management unit that enables guest virtual machines to directly use I/O devices, such as Ethernet. It supports *DMA remapping*, which allows the DMA to read or write the data directly in the I/O buffers of the guest virtual machine, rather than into the host I/O buffers and then copy them into the guest I/O buffers. It also supports *interrupt remapping*, which lets the virtual machine monitor route interrupt requests directly to the proper virtual machine.

Check Yourself

Two options for networking are using interrupts or polling, and using DMA or using the processor via load and store instructions.

1. If we want the lowest latency for small packets, which combination is likely best?

2. If we want the lowest latency for large packets, which combination is likely best?