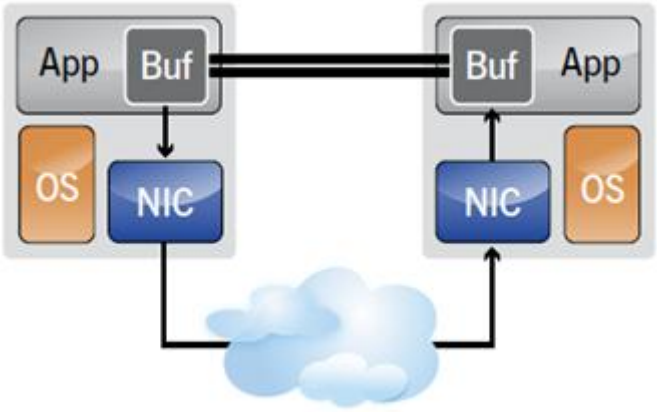
Quick Concepts Part 1 – Introduction to RDMA

Posted on October 8, 2010 | 19 comments

This is the first post in a three post series on getting started. By the end of the getting started series my goal is to get you ready to start coding with a sample program that will demonstrate the performance benefits of RDMA. In this post I will review RDMA concepts in a simple minded way. Later I intend to come back and do more detailed posts on each concept.

What is RDMA



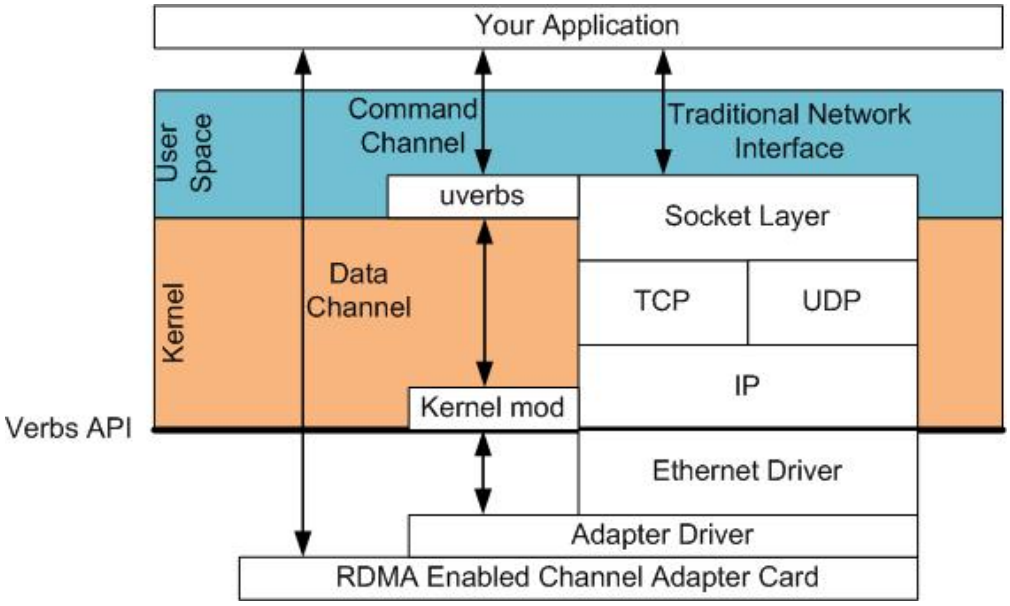
RDMA is Remote Dynamic Memory Access which is a way of moving buffers between two applications across a network. RDMA differs from traditional network interfaces because it bypasses the operating system. This allows programs that implement RDMA to have:

1. The absolute lowest latency
2. The highest throughput
3. Smallest CPU footprint

How Can We Use It

To make use of RDMA we need to have a network interface card that implements an RDMA engine.

We call this an HCA (Host Channel Adapter). The adapter creates a channel from it's RDMA engine though the PCI Express bus to the application memory. A good HCA will implement in hardware all the logic needed to execute RDMA protocol over the wire. This includes segmentation and reassembly as well as flow control and reliability. So from the application perspective we deal with whole buffers.



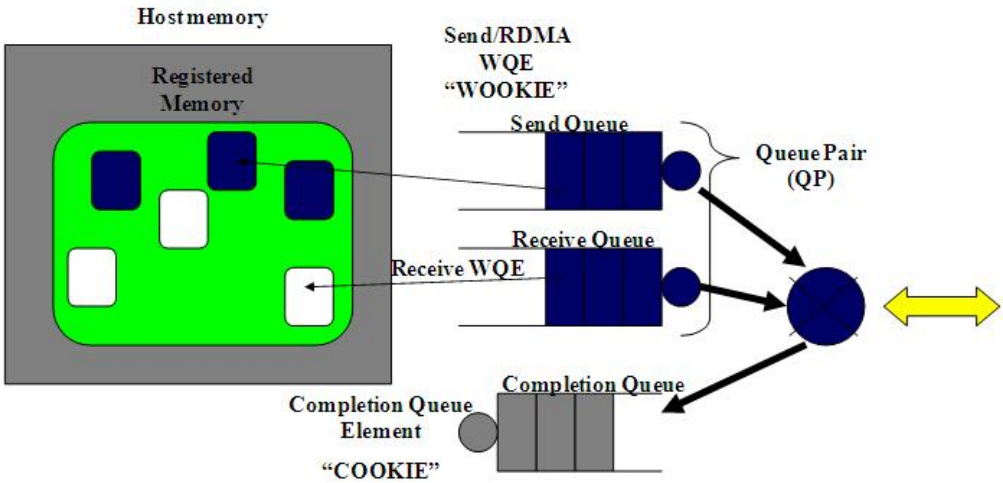
In RDMA we setup data channels using a kernel driver. We call this the command channel. We use the command channel to establish data channels which will allow us to move data bypassing the kernel entirely. Once we have established these data channels we can read and write buffers directly.

The API to establish these the data channels are provided by an API called "verbs". The verbs API is a maintained in an open source linux project called the Open Fabrics Enterprise Distribution (OFED). (www.openfabrics.org). There is an equivalent project for Windows WinOF located at the same site.

The verbs api is different from the sockets programming API you might be used to. But once you learn some concepts it is actually a lot easier to use and much simpler to design your programs.

Queue Pairs

RDMA operations start by "pinning" memory. When you pin memory you are telling the kernel that this memory is owned by the application. Now we tell the HCA to address the memory and prepare a channel from the card to the memory. We refer to this as registering a **Memory Region**. We can now use the memory that has been registered in any of the RDMA operations we want to perform. The diagram below show the registered region and buffers within that region in use by the communication queues.



RDMA communication is based on a set of three queues. The send queue and receive queue are responsible for scheduling work. They are always created in pairs. They are referred to as a **Queue Pair(QP)**. A Completion Queue (CQ) is used to notify us when the instructions placed on the work queues have been completed.

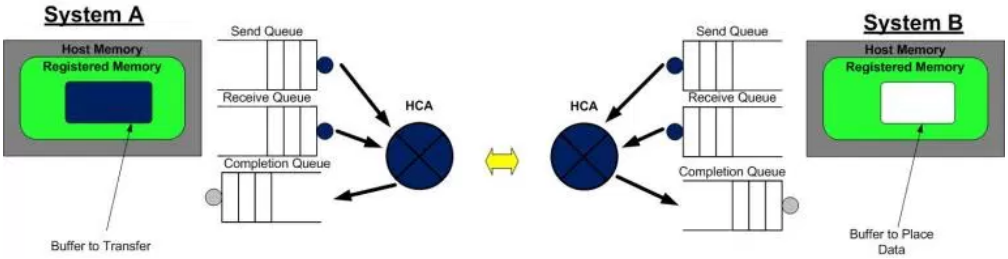
A user places instructions on it's work queues that tells the HCA what buffers it wants to send or receive. These instructions are small structs called work requests or **Work Queue Elements (WQE)**. WQE is pronounced "WOOKIE" like the creature from starwars. A WQE primarily contains a pointer to a buffer. A WQE placed on the send queue contains a pointer to the message to be sent. A pointer in the WQE on the receive queue contains a pointer to a buffer where an incoming message from the wire can be placed.

RDMA is an asynchronous transport mechanism. So we can queue a number of send or receive WQEs at a time. The HCA will process these WQE in order as fast as it can. When the WQE is processed the data is moved. Once the transaction completes a **Completion Queue Element (CQE)** is created and placed on the **Completion Queue (CQ).** We call a CQE a "COOKIE".

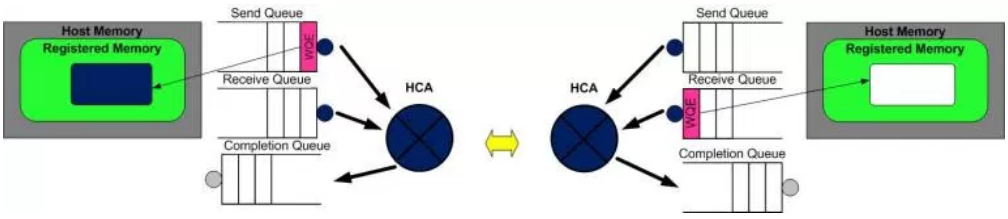
A Simple Example

Lets look at a simple example. In this example we will move a buffer from the memory of system A to the memory of system B. This is what we call **Message Passing** semantics. The operation is a SEND, this is the most basic form of RDMA.

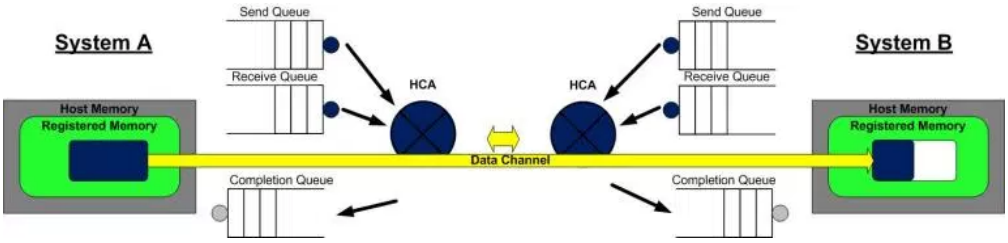
Step 1 System A and B have created their QP's Completion Queue's and registered a regions in memory for RDMA to take place. System A identifies a buffer that it will want to move to System B. System B has an empty buffer allocated for the data to be placed.



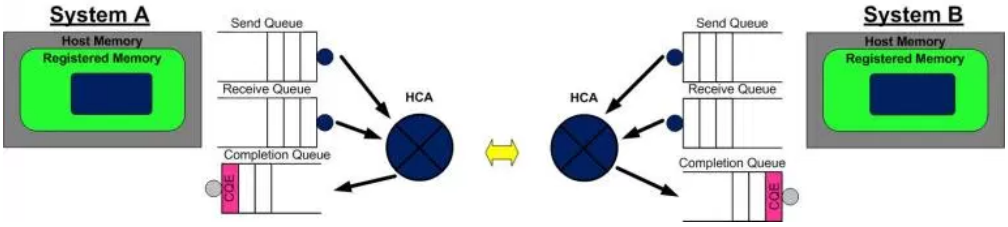
Step 2 System B creates a WQE “WOOKIE” and places in on the Receive Queue. This WQE contains a pointer to the memory buffer where the data will be placed. System A also creates a WQE which points to the buffer in it’s memory that will be transmitted.



Step 3 The HCA is always working in hardware looking for WQE’s on the send queue. The HCA will consume the WQE from System A and begin streaming the data from the memory region to system B. When data begins arriving at System B the HCA will consume the WQE in the receive queue to learn where it should place the data. The data streams over a highspeed channel bypassing the kernel.



Step 4 When the data movement completes the HCA will create a CQE “COOKIE”. This is placed in the Completion Queue and indicates that the transaction has completed. For every WQE consumed a CQE is generated. So a CQE is created on System A ‘s CQ indicating that the operation completed and also on System B’s CQ. A CQE is always generated even if there was an error. The CQE will contain field indicating the status of the transaction.



The transaction we just demonstrated is an **RDMA SEND operation**. On Infiniband or RoCE the total time for a relatively small buffer would be about 1.3 µs. By creating many WQE’s at once we could move millions of buffers every second.

Summary

This lesson showed you how and where to get the software so you can start using the RDMA verbs API. It also introduced the Queuing concept that is the basis of the RDMA programming paradigm. Finally we showed how a buffer is moved between two systems, demonstrating an RDMA SEND operation.