Running Infiniband on the Cray XT3

Makia Minich
Oak Ridge National Laboratory
Oak Ridge, TN
minich@ornl.gov

Keywords: Cray, XT3, infiniband, Voltaire, linux, OFED, OpenFabrics

Abstract

In an effort to utilize the performance and cost benefits of the infiniband interconnect, this paper will discuss what was needed to install and load a single data rate (SDR) infiniband (IB) host channel adapter (HCA) into a service node on the Cray XT3. Along with the discussion on how to do it, this paper will also provide some performance numbers achieved from this connection to a remote system.

OVERVIEW AND GOALS

System Layout

Since a discussion of the Cray XT3 architecture is beyond the scope of this document, we are going to focus on the overall layout of the systems used in our test. Figure 1 shows the extremely basic overview of the connections between our systems. (If you would like more specifics on the Cray XT3 architecture, visit the Cray website[1] which has a lot of useful marketing media that provides a good overview.)

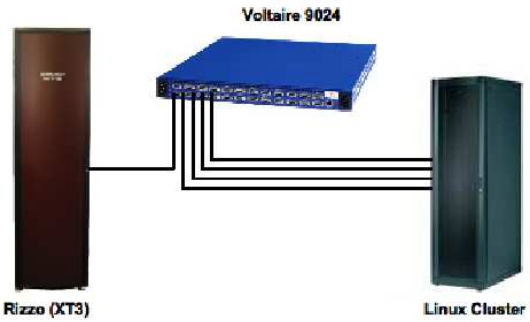

Figure 1. System Layout

Rizzo is a single rack of XT3 hardware comprised of 14 IO nodes (7 IO modules) and 68 compute nodes (17 compute modules). Each of these IO nodes has a single 133MHz PCI-X available for an expansion card. For this testing, we have placed a dual-port, single data rate (SDR), 128MB Voltaire host-channel adapter (HCA) into one of the IO nodes and connected it to a Voltaire 9288 (288-port SDR) switch. While it would be preferable (for testing as well as moving forward) to have more than one HCA in Rizzo, at the time only one was installed.

On the other end, we have Pinto which is an x86_64 based linux cluster. While Pinto has a large number of nodes available, for this testing only four nodes were used. Each of the nodes have a dual-socket single-core 3.4GHz Intel Xeon with an 8-lane PCI-Express based Voltaire 4x SDR HCA.

The Voltaire 9288 and Pinto are co-located, which allows us to use a standard CX4 infiniband cable (15 meter lengths) between the nodes of Pinto and the Voltaire switch. Rizzo happens to be a larger distance away, so we use an active CX4 cable which allowed us to run a longer fiber connection (100m) between Rizzo and the Voltaire 9288.

Normally, when someone talks about infiniband and clusters, they are talking about using it as a high-performance interconnect within a cluster. But, as you can see, in this testing we're using it to bridge two (or more) clusters together so that we can provide a fast data-movement path between the multiple clusters.

Operating System Software

Operating System

To avoid delving too deep into the intricacies of the XT3 software stack, we are going to focus on the two main pieces that we need to be aware of. The IO nodes (and any type of interactive node on the XT3) run diskless with a SuSE-derived base OS (currently based on SuSE Enterprise 9). The compute nodes, on the other hand, run Catamount which allows the compute nodes to boot a micro-kernel and an application.[2] This allows the compute nodes to spend all of their cycles running the application which can help to reduce OS jitter. Pinto, being a standard linux cluster, is running RedHat Enterprise Linux Workstation release 4 update 3. The system breakdowns can be seen in table 1.

From the OS Comparison table we see that a kernel version is mentioned for the Catamount nodes. While one can't easily type `uname -r` on the command line of the compute node (primarily due to the lack of any user-level interaction), there is an actual kernel version associated with that boot (hence the `-ni` suffix in the table). This kernel is encapsulated in the `stage2.sf` file, which is created by the build process for

[1] http://www.cray.com

[2] http://www.cray.com/downloads/Cray_XT3_Datasheet.pdf

Table 1. OS Comparison

System	OS	Kernel Version
Rizzo	UNICOS 1.4.19 (later 1.5.31)	
• IO Nodes	SuSE Enterprise 9	2.6.5-7.252-ss
• Compute Nodes	Catamount	2.6.5-7.252-ni
Pinto	RedHat Enterprise Linux Workstation 4 update 3	2.6.9-42.EL_lustre.1.4.7smp

the XT3 software stack.

Infiniband Stack

The OpenFabrics Alliance[3] recently began distributing an enterprise version of their stack. Created through a collaboration between different infiniband vendors and opensource community contributors, the OpenFabrics Enterprise Distribution (OFED) is touted as the stable and supported opensource infiniband stack. While development is still ongoing for the main OpenFabrics software branch, the OFED stack takes snapshots in time, to create a supported product for the infiniband community. These releases supply an easy to build and install framework which allows users to start utilizing their infiniband interconnect regardless of what vendor and OS stack is loaded on the system. By unifying the infiniband stack, it has become easier to manage software revisions on multiple platforms as well as provide consistent API's for interconnect development on these platforms.

Our testing is focusing on OFED 1.1 which contains support for all of the kernels involved in our testing. While normally we would build all of the tools associated with the infiniband stack, our system layout precludes us from needing things like MPI. More importantly, we're going to need IP-over-IB (IPoIB) for standard ethernet connections, remote-DMA access to pass large amounts of data across the interconnect, and the sockets direct protocol (SDP) to efficiently encapsulate IP traffic into IB traffic.

Test Suite

The following tests were used to determine not only the functionality of the infiniband connection but also to graphically plot the performance. Because of the nature of our system (which is described in more detail in the System Layout and Operating System Software sections), we can only focus on RDMA and IP based tests. As a side effect, though, this combination will also allow us to test SDP (sockets direct protocol) over the IP interface.

Low-Level Infiniband Tests

Provided as a default functionality test by the OpenFabrics Enterprise Distribution, `ib_send_bw` and `ib_send_lat` (low-level bandwidth and latency tests respectively) allow us to measure the total throughput we could expect from the hardware (removing any constraints that the higher level infiniband protocols would impose).

When the utilities are run with the `-a` option, it performs the tests with message sizes from 2 to 2^{23} bytes for bandwidth and 2 to 2^8 bytes for latency allowing us to see the trend in performance as message size changes. Another advantage of this tool is that it allows you to specify the IB transport that you wish to use for the transfer: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). While normally these three transports would hopefully behave the same, it's good to make sure that they do, as each have their own uses; the Infiniband Trade Association website[4] can provide a good in-depth look at these transports.

NetPIPE

NetPIPE[5] is another bandwidth and latency measurement tool. While it does typically use MPI over Infiniband (or any other high performance interconnect), NetPIPE can also utilize tcp-based connections, which allows us to test IP-over-IB (IPoIB) connections. NetPIPE performs a ping-pong style transfer to measure the transmission rates, and then outputs a table of latency and bandwidth measurements for a range of packet sizes.

Because of the nature of the TCP connections used by NetPIPE, we were easily able to use these same tests to measure the performance of SDP over the infiniband connection. By using `LD_PRELOAD` to load the `libsdp.so` libraries, we were able to use the same NetPIPE binary to test both standard TCP connections as well as SDP connections.

GETTING INFINIBAND ON THE XT3

On a normal cluster, such as Pinto, building and loading the OFED stack is a relatively easy process. You can easily follow the instructions provided by the OFED release documentation to get things up and running. Life is a little bit different on the XT3 though, as there are a few caveats to keep in mind. The first is that we will only be affecting the IO nodes on the XT3, the Catamount nodes will need to rely on routing over Portals to utilize the infiniband connection (such

[3] http://www.openfabrics.org

[4] http://www.infinibandta.org/specs/

[5] See URL http://www.scl.ameslab.gov/Projects/NetPIPE

as for lustre). The second is the limitations set out in the kernel provided by Cray. Because the XT3 is a fully supported platform, Cray makes specific decisions about what is made available in the kernel and what is available in the hardware. This is made painfully obvious when you attempt to build and load the OFED stack only to receive the dreadful `unknown symbol` errors.

Kernel Changes

On the initial attempts to getting infiniband on the XT3 (primarily against Unicos version 1.4.19), we required a couple changes to the default running kernel on the IO-nodes. Two symbols which OFED relies on were not being exported by the kernel: `bad_dma_address` and `dev_change_flags`. Applying the following patch to the IO-node kernel source, addresses this problem:

Listing 1. Kernel Patches

```
# Patch to arch/x86_64/kernel/pci-nommu.c
@@ -10,6 +10,9 @@
  * Dummy IO MMU functions
  */

+dma_addr_t bad_dma_address;
+EXPORT_SYMBOL(bad_dma_address);
+
 void *pci_alloc_consistent(struct pci_dev *hwdev,
                            size_t size,
                            dma_addr_t *dma_handle)
 {

# Patch to net/core/dev.c
@@ -3482,10 +3482,7 @@
 #if defined(CONFIG_BRIDGE) || \
     defined(CONFIG_BRIDGE_MODULE)
 EXPORT_SYMBOL(br_handle_frame_hook);
 #endif
-/* for 801q VLAN support */
-#if defined(CONFIG_VLAN_8021Q) || \
-    defined(CONFIG_VLAN_8021Q_MODULE)
 EXPORT_SYMBOL(dev_change_flags);
-#endif
 #ifdef CONFIG_KMOD
 EXPORT_SYMBOL(dev_load);
 #endif
```

With the application of this patch, we were able to rebuild and boot this kernel to make sure that everything was working properly. In order to build the OFED modules (covered in the next section), we used this modified source (rather than utilizing the kernel headers provided by the installed XT3 software).

Later versions of the XT3 kernel (starting with Unicos release 1.5) actually now have these patches incorporated. This makes building and running the OFED stack much easier in the long run. No longer do we need to rebuild the kernel, nor do we actually need the source code to build the OFED modules (instead we are able to utilize the header files located in `/opt/xt-os/default`).

Building and Loading OFED

Once we had a working kernel, we could then build the OFED stack. To avoid kernel versioning mismatch errors, it is important to keep an eye on the gcc versions throughout this process. First, we need to make sure and build OFED with the same gcc version that the running kernel was built with (e.g. if the kernel was built with gcc-3.2, you need to build the modules with gcc-3.2). As an added bonus, a lot of the OFED tools fail to compile with gcc-3.2 and would prefer to be built with gcc-3.4 or higher. For this reason, we build the modules first and then the rest of the stack later. This is easily done by creating two separate `ofed.conf` files, the first to build the modules and the second to build the userspace tools.

Depending on the version of the kernel being used, we found that a change might be needed to the OFED source code due to the OFED stack seeming to not recognize the XT3's kernel as a proper kernel value, and therefore not apply any of the needed patches to get things working. So, it was needed to decompress the `openib-1.1.tgz` source file, apply the patch in listing 2 and then re-compress.

Listing 2. OFED Patches

```
# Patch to configure
@@ -259,7 +259,7 @@
    done
    # Apply default patches
    case ${KVERSION} in
-       2.6.5-7.244*)
+       2.6.5-7.*)
          printf "\nApplying␣patches␣for␣${KVERSION}␣\
kernel:\n"
          if [ -d ${CWD}/patches/2.6.5-7.244 ]; then
             for patch in ${CWD}/patches/2.6.5-7.244/*
```

When all was said and done, we ended up with a few RPM's that we could then install into our IO node image and be off and running, configuring the system just like any other SuSE based image.

PERFORMANCE OF INFINIBAND ON THE XT3

After successfully getting the infiniband connection up and running on the XT3, we were able to measure the actual performance of the infiniband link. Because of our system architecture, there is a limit in the types of tests that could be run. In then end, though, we should be able to get a clear picture on what kind of performance we can achieve.

For each of these tests, we used the infiniband node on Rizzo as the server node. In addition, we will be using two nodes on Pinto, one as a server (to show us the results of two like nodes), and the other as a client which can connect to both the Rizzo node and the other Pinto node. While it isn't preferential to mix the architectures (really, we should have two Rizzo nodes interacting) these tests give us a hint of what can actually be done with infiniband on Rizzo.

Low-Level Infiniband Tests

First, we'll start with the low-level infiniband tests, which should give us a good baseline of the overall performance of the infiniband interconnect. For this test, we will be running a server on Rizzo to connect to one Pinto node as well as testing the two Pinto nodes with each other. Figure 2 shows the results of this test.

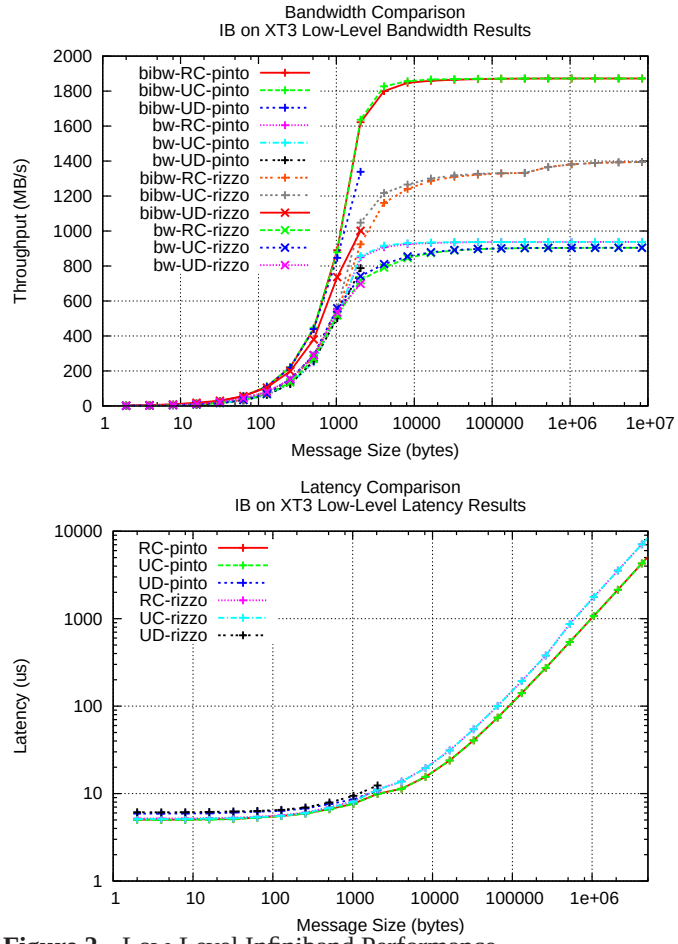

Figure 2. Low-Level Infiniband Performance

We can first see the performance hit delivered to the system due to the added complexity created by using the 100m active cable from Rizzo to the Voltaire 9288 (specifically with the bidirectional tests), with about a 400MB/s (bi-direction) difference between Rizzo and Pinto. But, there is something very interesting to pay attention to. The 133MHz PCI-X bus has a theoretical peak of 1GB/sec, and Rizzo is performing at about 900MB/s, so uni-directionally we are able to achieve close to the same rates as the PCI-e links (many PCI-X based systems are still utilizing the 100MHz PCI-X bus, which would have caused a peak limit at about 800MB/sec).

The latency difference between the two systems may be a function of the bus differences, but more than likely this is due to 100m cable.

NetPIPE

Now that we've seen the raw bandwidth that should be available, we can now shift our focus to TCP based bandwidth. If we were to use something standard, such as NFS or SCP, over the infiniband connection, we would be relegated to the IP-over-IB interface. It is no secret that current IPoIB performance leaves a lot to be desired, but it is still useful to see what we can expect from IPoIB if we need to use it. As a side effect of this, we are also able to take a look at SDP performance (which basically encapsulates the TCP traffic and re-routes it to the RDMA level). While still in heavy development, SDP is already proving to be quite useful. Figure 3 shows the results of our testing.

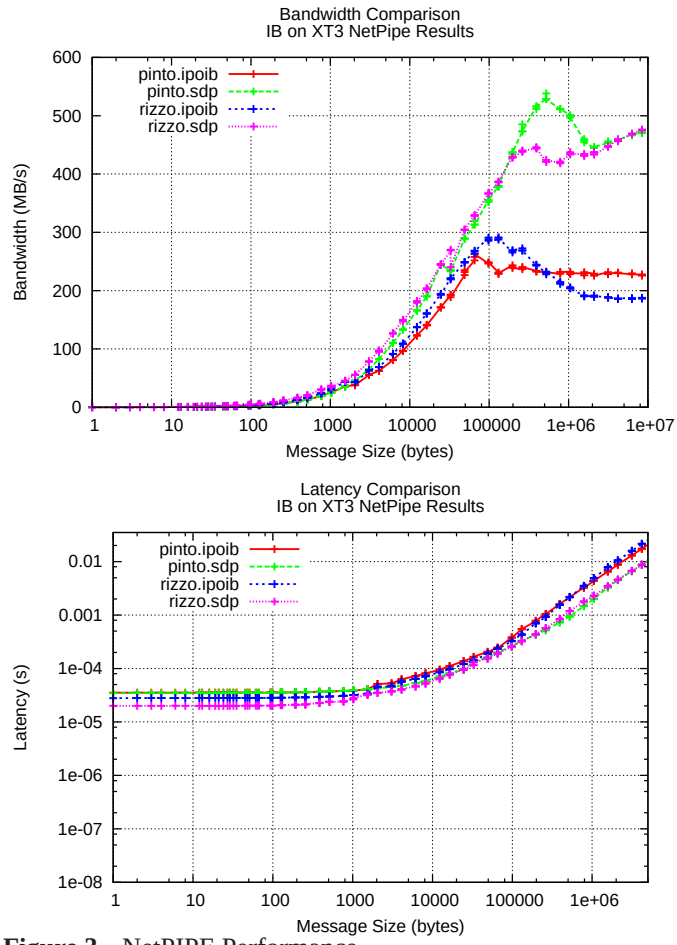

Figure 3. NetPIPE Performance

On first glance, one can clearly see the primary bene-

fit of SDP over the standard IPoIB interface. While still 400MB/s below the full RDMA level, SDP is providing about a 200MB/s increase over IPoIB. As more development continues on the SDP drivers, we should hopefully see these numbers improve. The results from latency aren't overly surprising.

CONCLUSIONS

After a few failures in the initial attempts at bringing up infiniband on the XT3, we think that this shows quite well that it can be done both functionally and effectively. These first steps open the door to being able to provide another high-speed data movement path off the XT3. With this new ability, we could effectively, for example, utilize the infiniband interconnect as a storage network using SRP, iSER or even using Lustre[6] over infiniband to move large data sets from the XT3 to a centralized Lustre filesystem. Though there is a large amount of work still to be done to raise all performance closer to low-level performance levels, we can now look at infiniband as a viable option for the XT3.

ACKNOWLEDGMENTS

The author would like to thank Don Maxwell from ORNL for putting up with inappropriately timed full system crashes and a ton of incessant XT3 questions. Also, we'd like to thank Andrew Lehtola from Cray for pushing our kernel changes through the management chain and into the real release.

[6] http://www.lustre.org