The Development of Myrinet Driver for DP Low Latency Message Passing System

Theewara Vorakosit, Putchong Uthayopas
Parallel Research Group, CONSYL
Department of Computer Engineering,
Faculty of Engineering, Kasetsart University
Bangkok, Thailand.
Email: g4465018@ku.ac.th, pu@ku.ac.th

ABSTRACT -- **Low-latency communication system is crucial for the performance of parallel application on Beowulf cluster systems since its reduced the overhead of important operation such as barrier synchronization. This paper presents our work on the design and development of Myrinet driver for DP, low latency communication system in cluster environment. This driver allows user to exploit DP capability on fast message transmission on Myrinet network.**

This driver is implemented as a loadable kernel module, which can be load or unload without modifying or recompiling the OS kernel. The performance has been measure and clearly shows the good improvement over traditional UDP transmission.

KEYWORDS -- Cluster System, Communication Latency Reduction, Myrinet, Linux Driver

1. Introduction and Related work

High bandwidth, low latency network communication subsystem is essential for cluster system since many important synchronization depends on the fast transmission of short massages. By reducing the latency involved, parallel message passing applications can gain much higher performance on cluster system. Traditional Generic communication protocol such as TCP/IP is too complex for cluster systems since they are not designed to be a “local” protocol in such system. Hence, many communication subsystems are developed to be use in cluster systems. The examples of such works are Directed Point[7], Active Message[2], Fast Message[3], U-Net[4] and Virtual Interface Architecture[5].

DP is one of the implementation that seems to be very interesting in many aspects. This includes a well-protected kernel level implementation, fast and low overhead system call, well define and low overhead architecture. But the problem is that current DP implementation only supports FastEthernet, share memory, and ATM.

One of the most used network switch technology is Myrinet from Myricom. Myrinet is a robust, scalable, and high performance high bandwidth network technology. Myrinet has many useful features to use in cluster such as high bandwidth, ANSI standard, and scalability. Myrinet is fully programmable at the NIC level. This facts is a motivation for the development of a Myrinet driver for DP system on Beowulf cluster so that users can fully exploit the performance of Myrinet-base Beowulf cluster with DP technology.

The design of driver aims at achieving a low latency for short message and high bandwidth for large message. To achieve our goal, the complexity of original Myrinet driver from Myricom [6] has been reduced and other functionality require for the driver such as registering to DP, source route configuration, memory map utility are added. The developing Myrinet driver is still a challenging work due to the programming complexity in kernel level. Hence, this is also the motivation for the selection of GNU/Linux system as an implementation platform since GNU/Linux is a free software and all kernel source code is available.

2. Background

Directed Point (DP) is a communication subsystem for parallel computing that comes from the research project at University of Hong Kong. The DP communication subsystem employs a high-level abstraction to express the interprocess communication in a cluster. In DP model, each node in the cluster is assigned a logical identity called *Node ID (NID)*. Each endpoint of the directed edge is labeled with a unique Directed Pont ID (DPID). Each program can use an association of 3 tuples {*local DPID, peer Node ID, peer DPID*} to identify a communication channel.

The DP subsystem consists of three layers, namely, *application programming interface (API) layer, service layer,* and *network interface layer.* The DP *API layer* provides a way to use DP system. The DP *service layer* is the core of the DP subsystem that provides services for message delivery. This layer is responsible for the delivering of messages from user space to network hardware level. It also helps deliver the incoming packets to the target DP end point. The DP *network interface layer* provides an interface for DP *service layer* to interact with the network hardware. Figure 1 illustrates the DP system architecture.

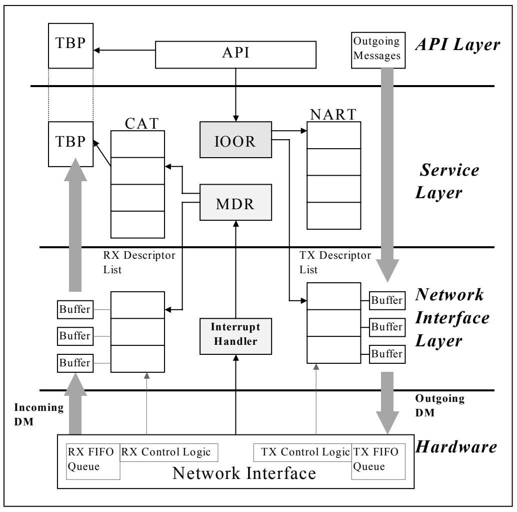

Figure 1. The architecture of DP communication subsystem

Myrinet is a switching technology that is widely used to interconnection for high-performance cluster systems. Some features of Myrinet are:

- Full-duplex 2+2 Gigabit/second links, switch ports, and interface ports.
- Flow control, error control, and "heartbeat" continuity monitoring on every link.
- Low-latency, cut-through, crossbar switches, with monitoring for high-availability applications.
- Scalable to tens of thousands of hosts, with network-bisection data rates in Terabits per second, and can also provide alternative communication paths between hosts.
- Host interfaces has build-in microcontroller called LANai that execute a control program to interact directly with host processes ("OS bypass") for low-latency communication, and directly with the network to send, receive, and buffer packets.
- Support any topology and protocol.
- Conform to American National Standard ANSI/VITA 26-1998

Myrinet card has a memory space of 16 MB. LANai memory is between address 0 to 0x800000. The block diagram of Myrinet card is shown in Figure 2.

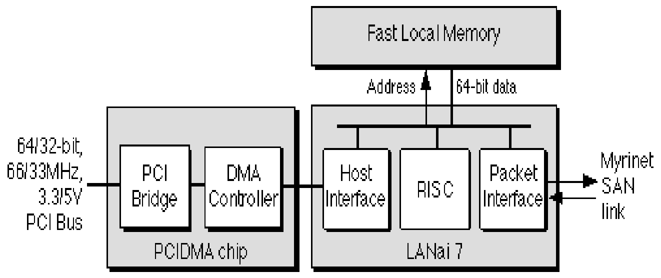

Figure 2. The block diagram of Myrinet NIC

3. Implementation of Myrinet Driver for DP

Myrinet driver is designed to work s a network layer of DP. The driver consists of 2 parts: the LANai control program and Linux Host Driver. LANai control program is a program that execute on LANai processor on Myrinet board. The Linux host driver is a driver execute in Linux Kernel. The Linux host driver and LANai control program communicate using hardware level share memory.

In LANai, the memory available is a 2 MB SRAM (expandable to 8 MB). The LANai memory is divided into 7 sections as follows:

1. Myrinet control program region for Myrinet control program (MCP). The size of this program in our driver is about 256 kB. At the end of this region, it is a base stack pointer. We have to move base stack pointer to this address in order to use the rest of LANai memory. This region if from 0 to 0x3ffff.
2. Blank region that acts as a guard between MCP and other regions. This region is ranging from address 0x40000 to 0x4ffff.
3. Command region. This region allows host and LANai to write and read the commands and status codes such as sending command, busy flags and so more. This region ranges from address 0x50000 to 0x5ffff.
4. DMA control block region. Myrinet NIC contains a DMA controller. The controller uses chains of control blocks stored in LANai memory to initiate DMA-mastering operations. There are 2 chains, one for sending and another one for receiving. This region is located at the address 0x60000 to 0x6ffff.
5. Source route table region. This region is used to maintain the source route table. System administrator has to configure source route table for each node statically. When sending a packet, LANai will search for a source route for target host from DP header. This region is from 0x70000 to 0x7ffff. In this version, the driver supports 6 bytes source route. That means cluster can span to maximum of 6 switches or a few hundred nodes. This table can contain up to 21845 hosts.
6. Send buffer. This region is used to store the outgoing packet to be sent. Only one packet can be stored in this region at a time. MCP supports up to 65536 bytes of

data. However, the current version of DP supports only 1500 bytes. This region is ranging from address 0x9000 to 0x9ffff.

7. Receive buffer. This region is used to store the incoming packet. This region ranges from address 0x10000 to 0x10ffff.

For the driver to work, the LANai controller code has been developed using C language. This C code is the compiled to LANai machine code using cross compiler available from Myricom. The LANai is programmed logic is a state machine. Figure 3 shows the flowchart that explains the LANai controller code logic.

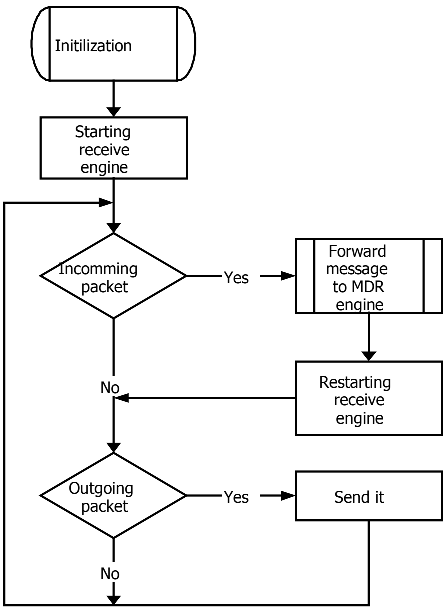

Figure 3. Flow chart for LANai Send Receive Operation

For the Linux host driver, the driver consists of 4 main functions: sending function, receiving function, memory-mapped function, and source route configuration and registration function. The operations of each function can be briefly summarized as follows:

- Sending function responsible for the sending of outgoing packet. This module is called directly by DP service layer. The transmission of a message works as follow:
    1. Driver checks whether LANai is free.
    2. Driver creates DP header in kernel memory.
    3. Copy the header and data to LANai memory.
    4. Wait until the send operation in LANai completed.
- Receiving function receives the incoming packet. LANai will receive all incoming packet at all time. If there is an incoming packet, the routine works the following way:
    1. LANai checks that there is an incoming packet and whether the incoming packet is a DP packet or not.
    2. LANai set the size of incoming packet in DMA control block and raises the interrupt to host driver.
    3. Driver triggers DMA transfer of incoming packet into host memory.
    4. Driver calls DP service layer to dispatch incoming packet.
- Memory-mapped function provides the mapping of all 16MB of LANai memory space into a part of the user space. To use this function, user have to create a character device file with predefine major number and minor number to be 0. The function is used to debug Myrinet.
- Source route configuration function provides a way to set source route for each node. Source route is required for the sending of packets to other node. In order to keep the overhead low, the source route table are configure statically. Source route configuration is registered in a file /proc/net/myri_route. User can configure source route using cat command to /proc/net/myri_route. This file can also be read.

The driver itself is implemented as a loadable kernel module. Hence, the driver can be loaded or unloaded from memory without modifying or recompiling the kernel. In order to support the listed functionally, driver composes of 7 major modules, which are:

- Initialization module – this module responsible for locating Myrinet NIC from a system using PCI function called in Linux and gets the pointer to configuration space if a Myrinet NIC is located. Next, the memory in Myrinet NIC is mapped as a part of kernel memory, so kernel can directly access the card. Finally, the routine will initialize Myrinet hardware, clear, and check all content of LANai memory.
- Loading module – this module load MCP program to LANai memory. MCP code is programmed in C language, which is compiled by LANai cross compiler to LANai machine code. This machine code is then converted into C array definition in order to simplify the loading task. LANai executable file can be converted to C array using "*gendat*" utility from Myricom and then included into a driver source code. Loading task is simplified to be only the copy of array content to LANai memory.
- Sending module. This module is a major part of this driver. Sending module is called by DP service region to send a packet to another node.
- Interrupt service routine. This routine is another major part of the driver. This routine is called when LANai interrupts host. LANai will interrupt host only when an incoming packet is a DP packet. This routine also help copy the data from LANai memory to DP service layer buffer.
- Character device and /proc Routine. This routine provides a convenient way to directly access the NIC. User can directly access a whole NIC as though the NIC is in user space using *mmap* system call as well as generic read/write system call. /proc provide a

convenient way to configure source route for that node. This module will create /proc/net/myri_route.

- Post-initialization module. The main function of this module is to register components such as interrupt service routine to a kernel. After finish the registration process, this routine will start the LANai operation.
- Clean up module. The driver can be unloaded from kernel using the *rmmod* command. The *rmmod* command will call clean up module to free all resources that are allocated during the work. Hence, this module has to free all resources before the driver are unloaded from kernel. The resource cleaned are things such as memory, interrupt line, /proc and so on.

4. Performance

The driver has been tested using 2 nodes from a Beowulf cluster called AMATA. These two system has the following setup: 1 GHz Athlon processor with 512 MB RAM connect through Myrinet switch, Linux Kernel 2.2.16, and Myrinet card model: M2M-PCI64A-2-50973 with LANai version of 7.2 and 2 MB board memory.

A ping-pong program has been developed to measure the round-trip time for message size ranging from 8 bytes to 1024 bytes. The comparison has been made between the newly developed driver and the performance of code with UDP over FastEthernet, UDP over Myrinet, and GM driver over myrinet. The results are as shown in Table 1 and the graph are plotted as illustrated in Figure 4. The time given in Table 1 is in the unit of Microsecond.

Table 1. Comparison of UDP over Fast Ethernet, Myrinet and DP over Myrinet

Message size	UDP Fast Ethernet	UDP Myrinet	DP Myrinet	GM Myrinet
1	52.3	62.8	26.25	15.99
2	47.8	47.8	26.75	15.98
4	48.5	47.8	27	16.02
8	48.0	47.0	27	16.02
16	46.8	47.0	27	16.02
32	48.5	48.0	27.25	15.96
64	51.8	49.0	28.25	16.78
128	53.5	52.5	29.75	19.98
256	58.0	61.8	36.5	30.18
512	68.5	68.3	40.25	38.42
1024	88.3	88.5	51.5	55.16

For the new driver, the graph in Figure 4 clearly shows a much lower latency compared to usual UDP over both Myrinet and Fast Ethernet. When compared with GM driver from Myricom, for small message size, GM driver is a little bit faster (about 10 Microsecond) that our DP driver. The DP driver has slightly lower latency when message size is larger than 512 bytes. The time different may cause by many factors. One of the most important factor is how fine tune the code has been done. The logical step to further fine tune the code is to profile them in more detail and look at the code tuning at assembly language level.

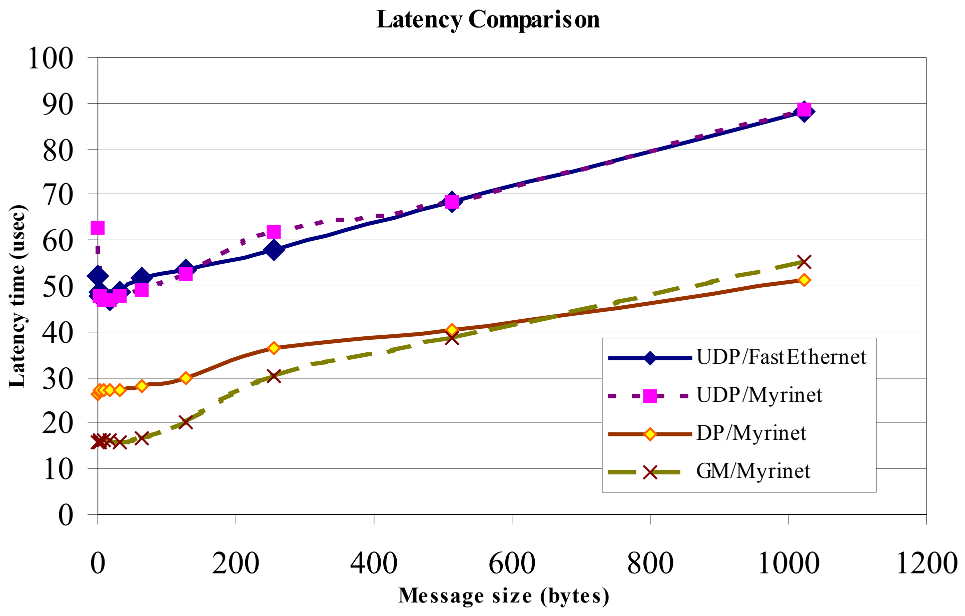

Figure 4. Performance comparison between Myrinet over DP and UDP

5. Conclusion and Future Work

In this paper, an implementation of Myrinet driver for DP system has been discussed with the experimental results. The techniques used in developing the driver have been explained. From the experimental results, the newly developed driver shown a satisfactory reduction of message latency although, there seems to be slight problem that slow down the Myrinet hardware transmission.

In the future, this driver will be used as a communication subsystem under a planned Pico-MPI that will be developed later. This implementation aims to explore the full optimize of message passing implementation from user space level to kernel level which has a high potential to generate fast and compact experimental MPI implementation.

6. Acknowledgement

This research has been partly supported by KU research and Development Institute under SRU grant. Most of the equipment used in the developed is sponsored by AMD Far East Inc.

7. References

[1] Chun-Ming Lee, Anthony Tam, and Cho-Li Wang, *Directed Point: An Efficient Communication Subsystem for Cluster Computing,* The University of Hong Kong, 1997

[2] T. von Eichken, D. E. Culler, S. C. goldstein and K. E. Schauser, Active Message: a Mechanism for Integrated Communication and Computation," *The 19th Int'l on Computer Architecture, 1992*

[3] S. Palin, M. Lauria and A. Chien, "High Performance Messages (FM) for Myrinet," *Supercomputing,* 1995

[4] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels, *U-Net: A user-level network interface for parallel and distributed computing*. In Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995. Available from

http://www.cs.cornell.edu/Info/Projects/ATM/sosp.ps

[5] D. Dunning and G. Regnier. *The virtual interface architecture*. InHot Interconnects V, pages 47--58, 1997

[6] Myricom Inc, *M2M PCI64A Product Specification*, http://www.myri.com/myrinet/PCI64/m2m-pci64a.html, July 2000

[7] Myricom Inc, *PCI64 Programmer's Documentation*,

http://www.myri.com/myrinet/PCI64/
programming.html, July 2000