10GE NIC throughput measurements

30.10.2008 (Revised: 6.11.2008) S.Esenov

1 Introduction

The original, naive, aim was to quickly determine the best TCP and UDP throughput rates between machines hosting the various NIC boards purchased. The immediate requirement for this work was to provide a PC based readout system for HPAD (AGIPD) detector which attaches via a UDP 10GE fibre connection to the front end electronics readout FPGA. This connection is required to run at the highest possible throughput rate without packet loss.

In reality a significant period of time was spent: understanding what factors affect the throughput, what versions of kernels and drivers have to be used with each other, creating and compileing drivers, identifying and modifying (tuning) system and user parameters, finding the tools to monitor and modify parameters, writing additional monitoring tools (xperf), etc. The result of this work, a snapshot of our understanding of how to tune and determine the best throughput, is described in this note. It should be considered as a starting point for a later, more complete, attempt to understand throughput.

2 Terminology

Note that the throughput numbers quoted in the text are decimal, thus 1GB is 1,000,000,000 bytes and not 1,073,741,824 bytes. This convention has been chosen because it is used by most of the performance monitoring tools used.

3 Factors affecting throughput

The following sections describe hardware and software details which are likely to affect the throughput.

3.1 Host PCIe, memory and CPU connectivity

The main boards of the workstations used during NIC testing are based on the Intel 5400 chipset. The main board layout is shown schematically in Figure 1.

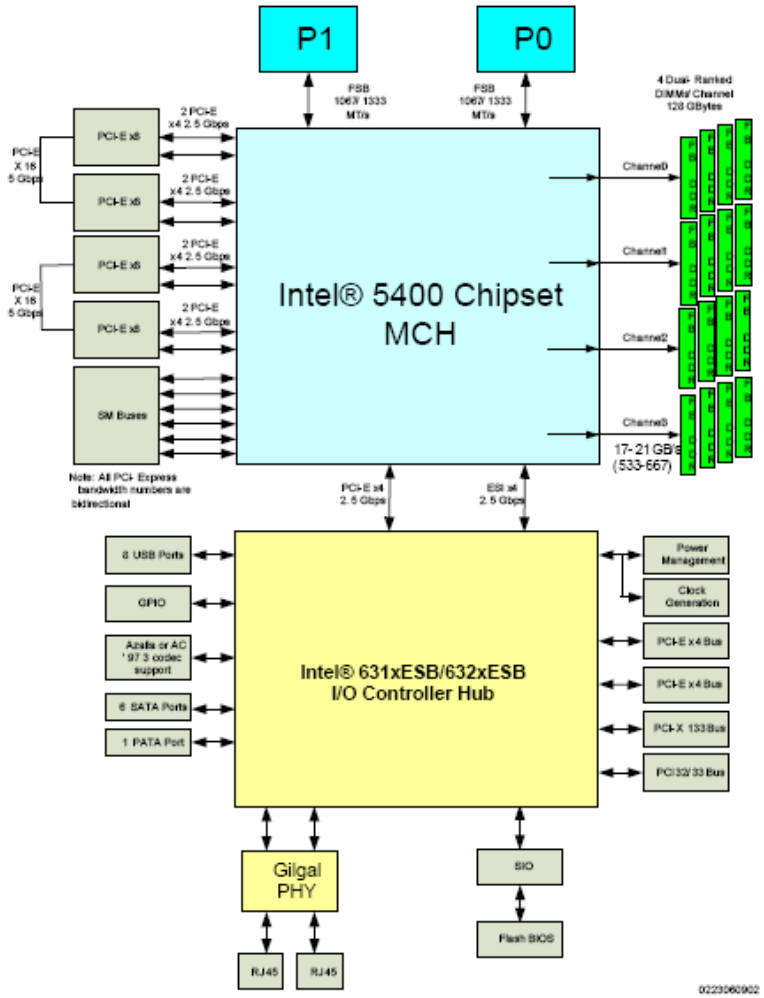

Figure 1 Intel 5400 chipset block diagram

The next paragraphs give brief descriptions of the FSB, Interrupt, memory and PCIe sub systems of the MCH, the text is paraphrased or copied directly from the Intel® 5400 Chipset Memory Controller Hub (http://www.intel.com/Assets/PDF/datasheet/318610.pdf).

3.1.1 Front side bus

The 5400 chipset is designed for use with two (P1 & P2) dual and quad core processors. It consists of two principle components, a Memory Controller Hub (MCH) for the host bridge and the I/O controller hub. Each processor socket is connected to the MCH via a separate Front Side Bus (FSB) which supports 1066 MTS, 1333 MTS and 1600 Mega Transactions per Second (MTS), which is a quad-pumped bus running off a 266/333/400 MHz clock, and a point to point DIB processor system bus interface. Each processor FSB supports peak address generation rates of 533/667/800 Million Addresses/second. Both FSB data buses are quad pumped 64-bits which allows peak bandwidths inbound of 8 GB/s (1066 MTS), 10 GB/s (1333 MTS) and 12 GB/s (1600 MTS) and out bound of 17 GB/s, 21 GB/s and 25GB/s, respectively. The MCH supports 38-bit host addressing, decoding up to 128 GB of the processor's memory address space. Host-initiated I/O cycles are decoded to PCI, PCI Express, ESI interface or MCH configuration space. Host-initiated memory cycles are decoded to PCI, PCI Express,

ESI or system memory. A snoop filter eliminates, data cache coherence, traffic on the snooped front side bus of the processor being snooped.

The host CPUs used in the test setup, Table 1, support the 1333 MTS FSB clock and thus have a 10GB/s in and 21 GB/s out bandwidth.

3.1.2 Interrupts

Interrupts are also delivered via the FSB. The legacy APIC serial bus interrupt delivery mechanism is not supported. Interrupt-related messages are encoded on the FSB as "Interrupt Message Transactions." In the Intel® 5400 chipset platform, FSB interrupts may originate from the processor on the system bus, or from a downstream device on the Enterprise South Bridge Interface (ESI). In the later case, the MCH drives the Interrupt Message Transaction onto the system bus. In the Intel® 5400 chipset the Intel 631xESB/632xESB I/O Controller Hub contains I/OxAPICs, and its interrupts are generated as upstream ESI memory writes. Furthermore, PCI 2.3 defines Message Signaled Interrupts (MSI) that are also in the form of memory writes. A PCI 2.3 device may generate an interrupt as an MSI cycle on its PCI bus instead of asserting a hardware signal to the IOxAPIC. The MSI may be directed to the IOxAPIC which in turn generates an interrupt as an upstream ESI memory write. Alternatively, the MSI may be directed directly to the FSB. The target of an MSI is dependent on the address of the interrupt memory write. The MCH forwards inbound ESI and PCI (PCI semantic only) memory writes to address 0FEEx_xxxxh to the FSB as Interrupt Message Transactions.

3.1.3 Memory

Fully Buffered DIMM technology was developed to address the higher performance needs of server and workstation platforms. FB-DIMM addresses the dual needs for higher bandwidth and larger memory sizes. FB-DIMM memory DIMMs contain an Advanced Memory Buffer (AMB) device that serves as an interface between the point to point FB-DIMM Channel links and the DDR2 DRAM devices. Each AMB is capable of buffering up to two ranks of DRAM devices. Each AMB supports two complete FB-DIMM channel interfaces. The first FB-DIMM interface is the incoming interface between the AMB and its proceeding device. The second interface is the outgoing interface and is between the AMB and its succeeding device. The point-to-point FB-DIMM links are terminated by the last AMB in a chain. The outgoing interface of the last AMB requires no external termination.

The four FB-DIMM channels are organized into two branches of two channels per branch. Each branch is supported by a separate Memory Controller (MC). The two channels on each branch operate in lock step to increase FB-DIMM bandwidth. A branch transfers 16 bytes of payload/frame on Southbound lanes and 32 bytes of payload/frame on Northbound lanes. The key features of the FB-DIMM memory interface are summarized in the following list.

- Four Fully Buffered DDR (FB-DIMM) memory channels.
- Branch channels are paired together in lock step to match FSB bandwidth requirement.
- Each FB-DIMM Channel can link up to four Fully Buffered - DDR2 DIMMs (FBDIMM1).
- Supports up to 16 dual-ranked FB-DDR2 8GB DIMMs, i.e. 128 GB of physical memory
- The FB-DIMM link speed is at 6x the DDR data transfer speed. A 3.2 GHz FB-DIMM link supports DDR2-533 (FSB@1067 MT/s). A 4.0 GHz FB-DIMM link Supports DDR2-667 (FSB@1333 MT/s) and A 4.8 GHz FB-DIMM link Supports DDR2-800 (FSB@1600 MT/s)
- The MCH will comply with the FB-DIMM specification definition of a host and will be compatible with any FB-DIMM-compliant DIMM.
- Special single channel, single DIMM operation mode (Branch 0, Channel 0, Slot 0 position only).
- All memory devices must be DDR2.

The memory used in the test setup, Table 1, is DDR2-800 clock which could drive 12 and 25 GB/s in and out of a 1600 FSB. The 1333 FSB used lowers these rates to 10 and 21 GB/s, respectively. We are not certain about how the number of DIMM rows affects these numbers !

3.1.4 PCIe

The PCI Express port 1 through port 8 are general purpose x4 PCI Express ports that may be used to connect to PCI Express devices. These x4 ports may be combined into high performance x8 PCI Express ports or up to two optimized high performance x16 graphics interface ports as well. The x16 ports contain several architectural enhancements to improve graphics performance. Port 1and Port 2, Port 3 and Port 4, Port 5 and Port 6, Port 7 and Port 8 are combinable to form single x8 ports. Ports 1-4 and ports 5-8 are combinable to form the two x16 ports. The Intel® 5400 chipset MCH supports up to four high performance x8 ports at on GEN 1 speed or up to two high performance x16 graphics PCI Express ports at revision 2.0 speed. This port contains several architectural enhancements to increase graphics performance.

The NICs used in the test setup, Table 2, both use x8 lanes corresponding to a bandwidth of 2 GB/s.

3.2 NIC hardware

It should be expected that good performance is obtained using hardware which is capable of performing the required task.

The NIC boards chosen implement data transfer over PCIe using x8 lanes, i.e. 2GB/s theoretical bandwidth. In the near future PCIe 3.0 higher rate and x16 lane implementations should further improve the maximum bandwidth.

The TCP Offloading Engine (TOE) of the Chelsio has not been used in the measurements reported here. There are two camps of opinion concerning TOE, supporters (e.g. Chelsio) who envisage offloading TCP stack handling from CPU cores, and those (e.g. Intel) who foreseen the work of stack handling, etc., remaining with dedicated CPU core.

3.3 Kernel

Modern Linux kernels (≥2.6) support various realtime capabilities. It is sometimes necessary to 'deterministically meet scheduling deadlines with specific tolerances' as well as being fast in terms of program execution. In this respect non-realtime kernels have higher latencies, but often performing better than realtime kernels w.r.t. program execution. A realtime kernel has been used whilst making the measurements presented here; it is believed that this should reduce UDP protocol packet loss because of the lower response latencies associated with the realtime system. There are various realtime Linux alternatives: from hard realtime to soft realtime solutions, see Ref 1. Here we use the realtime configuration ("preemptive realtime") of the Linux kernel (2.6.25.16-0.1-rt) from Novell openSUSE 11.0 distribution. What is important is that this kernel is supported by the driver software of NIC vendors.

3.4 Multiple cores

The availability of multiple CPU sockets and multiple cores per socket increases the configuration space for performing tests; which cores should be used on which sockets. In the measurements made here we have always configured the interrupt handling core and the core used for program execution to be on the same CPU (socket), i.e. to be on sister cores. This is generally accepted to produce the best performance.

3.5 Driver software and vendor tuning suggestions

Vendor notes are important for determining: the latest driver software version, which kernels are supported (have been targeted during vendor tests), and what advice exists for tuning performance.

We used the driver cxgb3toe-1.1.022 (TOE/NIC) for the NIC board from Chelsio Communications. It is supposed to work with Linux kernels till versions 2.6.27.xx. It was successfully compiled for our openSUSE realtime kernel (see above). We also have followed the vendor guidelines found in the driver software for tunning the performance of the system. (perftune.sh script).

The ixgbe driver for Intel NIC board came with openSUSE distribution. The tuning guidelines applied were found in /usr/src/linux/Documentation/networking/ixgb.txt

3.6 System and user parameter settings

What are the various parameters, what is their significance?

There are a lot of parameters that could/should influence the network performances measured in our tests. Some of them were found to be the main contributors into the final performance.

- Maximum and default OS send/receive buffer sizes for all types of connections: net.core.wmem_max, net.core.rmem_max, net.core.wmem_default, net.core.rmem_default.
- Minimum, default and maximum TCP send/receive buffer sizes: net.ipv4.tcp_mem, net.ipv4.tcp_wmem, net.ipv4.tcp_rmem
- Queue size for pending UDP packages before dropping them: net.core.netdev_max_backlog
- MTU size. The maximum value depends on NIC board: 9600 for Chelsio NIC and 16110 for Intel NIC. The more value was set the better throughput were reached.
- Interrupt handler affinity. Assignment the interrupt handler to run on a particular core in multiprocessor system to avoid the overhead with the context switching.
- Taskset affinity. As above but the task itself to be assigned to a particular core.

Parameters which are not mentioned above or referred to in Table 3 have not been changed from there default values set during driver and system initialization. Uncertainties associated with the values of unseen parameters and possible changes to them remain a significant problem when trying to understand throughput.

3.7 Non transfer related activity on the host

The tests performed here are data transfers between a user process, client, on one host and another user process, server, on a different host. As the hosts used were dedicated to the test being made no other processing load was present. This is not going to be the case in a normal user environment.
As an estimate of the effect of additional loading, the "dd if=/dev/sda of=/dev/null" command was performed in parallel with our tests (dd produces a lot of disk lookups). No visible degradation in the network performance was seen – we assume that this is due to the use of free cores and the high performance of the MCH sub-system.

4 Host hardware and software summary

Two Identical host machines were used during the NIC tests. The relevant hardware configuration details are summarized in Table 1.

Component	Attribute	Type
Main board	Type	Intel D5400XS Skulltrail S771 E54000 EATX
	CPU sockets	2 x XEON or Core2Extreme LGA771
	Front Side Bus (FSB)	1600 / 1333 / 1066 GHz
	Chipset	E5400
CPU	Type	Intel XEON E5420 2.5 GHz
	FSB	1333
	L2 cache / speed	2 x 6MB / 2.5 GHz
Memory	Type	2 x RAM FB-DIMM 8GB-800MHz Kingston KVR800D2D 4F5K2
PCIe	Protocol / Lanes	v 2.3? / 9 x4 configurable as x16, x8, and x4.

Table 1 Host configuration

An openSUSE 64bit operating system,

 # cat /etc/SuSE-release

openSUSE 11.0 (X86-64),

with the 2.6.25.16-0.1-rt realtime kernel,

 # uname –a

Linux hostname 2.6.25.16-0.1-rt #1 SMP PREEMPT x86_64 bit GNU/Linux,

was used during the measurements described here.

5 NIC hardware are software summary

Two NIC boards have been purchased and used in the measurements performed:

1. Intel 10 Gigabit XF SR Server Adapter (product nr. EXPX9501AFXSR, see http://download.intel.com/network/connectivity/products/prodbrf/318311.pdf)
2. Chelsio S310E-SR+ (see http://www.chelsio.com/products_10g_adapters.html).

In this note the vendor name will be used to identify which board is being discussed; Intel when referring to the EXPX9501AFXSR and Chelsio for the S310E-SR+ . The properties of the NIC cards are summarized in the Table 2.

Property	Intel	Chelsio
PCIe protocol	v 2.0	v 1.1
PCIe lanes	x 8	x 8
TCP Offload Engine	No	Yes (can be disabled)
Transceiver from factor	SFP	SFP+
Fibre / connections	MMF / LC	MMF / LC
Max. MTU (Jumbo)	≤16116 Bytes	≤9600 Bytes

Table 2 NIC properties

The NIC drivers used during the tests were the Intel 10 Ggabit PCI Network Driver – version 1.1.18,

 # grep Intel /var/log/boot.msg| grep "10 "

<6>ixgbe: Intel(R) 10 Ggabit PCI Network Driver – version 1.1.18

<6>ixgbe 0000:07:00.0: Intel® 10 Gigabit Network Connection

done eth0 device: Intel Coroporation 82598EB 10 Gigabit AF Network Connection (rev 01)

, and the Chelsio T3 Network Driver – version 1.1.022,

 # grep Chelsio /var/log/boot.msg

<6>Chelsio T3 Network Driver – version 1.1.022

<6>eth2: Chelsio T310 10GBASE-R RNIC (rev 4) PCI Express x8 MSI-X

6 Test setup

The test setup used is show in Figure 2.The 10GE network cards were allocated ip address on the hidden subnet 192.168.1.x and physically connected back-to-back to each other as required using short fibre cables. The 1GE connections were used for host access during testing.

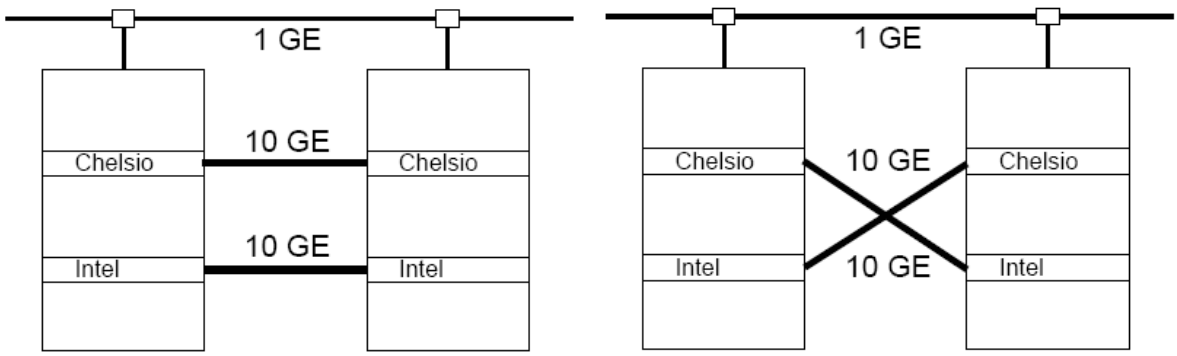

Figure 2 Test system network connections

7 Starting point settings and performance measurements

In view of the large choice of parameter available, the *baseline* settings summarized in Table 3 have been used for initial performance measurements. Note that IPV4 routing has been used throughout.

Parameter	value	comment
tcp_window_scaling	1	i.e. enabled
net.ipv4.tcp_sack	0	No visible influence
net.ipv4.tcp_mem	1546176 / 2061568 / 3092552	Default
net.ipv4.tcp_wmem	4096 / 10000000 / 16777216	
net.ipv4.tcp_rmem	4096 / 10000000 / 16777216	
net.core.wmem_max	16777216	
net.core.rmem_max	16777216	
net.core.wmem_default	10000000	
net.core.rmem.default	10000000	
net.core.netdev_max_backlog	300000	Crucial for UDP packet loss
net.core.optmem_max	524288	For scatter/gather interface
tcp_nodelay	off	
tcp_sndbuffer	256k (= msg len)	
tcp_rcvbuffer	256k (= msg len)	
TOE	disabled	Chelsio only parameter
MTU	9600	Jumbo packet size
Irq_balancing	0	disabled
Interrupt core	6	Starting from 0
Send core	4	
Recv core	5	

Table 3 Starting point parameter settings

The network performance measured with the baseline settings are summarized in Table 4.

The TCP and UDP throughputs are remarkably similar, ~7.5Gb/s, and independent of performance testing tool, NIC used and transmission direction.

Differences between the two NICs are apparent when the interrupt rates are compared, the Chelsio system appears to generate a higher interrupt rate.

Protocol	What	Chelsio to Intel		Intel to Chelsio		Chelsio to Chelsio xperf/netperf		Intel to Intel	
		43	44	43	44	43	44	43	44
TCP	bw (Gb/s)	7.61	7.61	7.33	7.33	7.61/7.50	7.61/7.50	7.34	7.34
	prog core %	63	70	63	100	74/70	100/90	60	49
	Interrupts/s	12k	14k	12k	91k	30/52k	94/43k	12k	14k
UDP	bw (Gb/s)	7.51	7.51	7.26	7.26	7.51	7.51	7.26	7.26
	Packet loss	0	0	0	0	0	0	0	0
	prog core %	100	43	100	85	100	90	100	40
	Interrupts/s	3.2k	8.7k	4k	81k	3k	80k	4k	9k

Table 4 Baseline setting network performance (xperf runs on same core as interrupts), 43 = sender, 44 = receiver.

Some of the measurements appear to be limited by the core running the testing tool reaching 100% cpu usage. It was not possible to increase the throughput significantly (>5%) by running two sets of senders and receivers on different cores, the cpu usage simply reduced to 50% on the two cores used.

With the settings used no packet loss was observed using the UDP protocol. Packet loss is most likely minimized by using a real time kernel and setting the backlog parameter to a large value. The packet loss rate as a function of backlog size for Intel-to-Intel transmission is shown in XXX.

8 Conclusions

Sustained TCP and UDP throughputs of ~7.5Gb/s (940MB/s) have been measured with both NIC boards tested on the host hardware used.

No UDP packet loss was seen, which we attribute to the use of a real time kernel and the correct setting baseline parameters like the backlog.

The cpu usage on the cores running the test programs is seen to reach 100%, but cannot be identified currently as a bottleneck. The documented PCIe, FBDIMM and FSB bandwidths are significantly higher than the rates we think are reached during testing, which suggests that this is also not a bottleneck. More work is required to understand possible bottlenecks. The CPU's frequency, 2.5GHz, and FSB bandwidth, 1333MHz, could be increased by replacing with higher performance CPUs.

Our understanding of UDP throughput management and monitoring has improved to the point where tests with the HPAD (AGIPD) detector front end interface should be made.

9 Follow up work and open questions

This section lists points that should be looked at.

- Why does UDP = TCP performance, should it? Look at the error rates on the TCP are they the same as the packet losses with UDP.
- RDMA how to use and other features of the NICS.
- TOE has not been used on the Chelsio board, what does it promise, what does it achieve.
- Multiple NICs usage as a single logical unit. Bonding.
- Find the reason for seeing only 75% of the maximum throughput. Multiple versions of xperf (2 senders & 2 receivers) 7.5Gb/s increased to 7.9Gb/s i.e. with two program cores at 50% - where is the bottleneck bus, memory, PCIe, elsewhere ? BUS/logic analyzer on PCIe buses.
- Finish "not yet complete" sections.

10 Glossary

Terms used and what they mean (not yet complete):

MSI-X = Message Signalled Interrupts. This is an alternate form of interrupt differing from the traditional pin-signalled system; instead of asserting a given IRQ pin, a message is written to a segment of system memory. Each device can have from 1 to 32 unique memory locations (MSI, up to 2048 with the newer MSI-X standard) in which to write MSI events to. An advantage of the MSI system is that data can be pushed along with the MSI event, allowing for greater functionality

11 Parameters

In this section describes the parameters which may have been modified during measurements. Not yet complete.

tcp_window_scaling = controls whether the size of the sliding window is allowed to be dynamically modified. (disable: sysctl –w net.ipv4.tcp_window_scaling=0; enable: =1;)

12 Performance testing tools used

This section describes the tools used for performance measurement. Not yet complete.

12.1 Netperf

Throughput server client test program. Note that the –T5,5 client option determines which cores will be used by the server and client programs.

Server:
netserver

Client:
netperf -P1 -l 60 -H 192.168.1.143 -T 5,5
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.1.143 (192.168.1.143) port 0 AF_INET : cpu bind
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec
10000000 10000000 10000000 60.02 7336.60

12.2 Iperf

Throughput server client test program.

Server:
iperf -s -i5

Client:
iperf -c 192.168.1.143 -l256K -i5 -t60 -fM

Client connecting to 192.168.1.143, TCP port 5001
TCP window size: 9.54 MByte (default)
---
[3] local 192.168.1.44 port 35506 connected with 192.168.1.143 port 5001
[ID] Interval Transfer Bandwidth
[3] 0.0- 5.0 sec 4368 MBytes 874 MBytes/sec
…
[ID] Interval Transfer Bandwidth
[3] 55.0-60.0 sec 4360 MBytes 872 MBytes/sec
[ID] Interval Transfer Bandwidth
[3] 0.0-60.0 sec 52309 MBytes 872 MBytes/sec

12.3 Xperf

Throughput server client test program. Written for the tests performed here. During the initial measurement stage very different results were determined for the Chelsio and Intel NICs, which made us suspicious of the netperf and iperf test programs – needless to say this was unfounded and the observed effects were due to using incorrect kernels with NIC drivers, etc.

Server:
./srv

Client:
./clnt -H 192.168.1.143 -i5 -t100 -l256K
Remote host: 192.168.1.143
Remote port: 12345
Network protocol: TCP
Message length: 262144
Test time: 60
Report time: 5
Connections: 1
Some network related parameters are ...
net.core.wmem_max = 16777216
net.core.rmem_max = 16777216
net.core.wmem_default = 10000000
net.core.rmem_default = 10000000
net.core.max_backlog = 300000
net.core.optmem_max = 524288
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_sack = 0
net.ipv4.tcp_mem = 1546176 2061568 3092352
net.ipv4.tcp_wmem = 4096 10000000 16777216
net.ipv4.tcp_rmem = 4096 10000000 16777216
T 5s: 4.6 GB 916.22 MB/s (7.33 Gb/s) Pkg# 0 PLoss# 0 (0.0000%), Msg# 0 MLoss# 0 (0.0000%) 1m:0.08 5m:0.09 15m:0.09
...
T 55s: 50.3 GB 913.66 MB/s (7.31 Gb/s) Pkg# 0 PLoss# 0 (0.0000%), Msg# 0 MLoss# 0 (0.0000%) 1m:0.44 5m:0.18 15m:0.12
*********** A L A R M **********
T 60s: 54.8 GB 913.63 MB/s (7.31 Gb/s) Pkg# 0 PLoss# 0 (0.0000%), Msg# 0 MLoss# 0 (0.0000%) 1m:0.49 5m:0.19 15m:0.12
11/05/08 17:48:18 TCP connection '192.168.1.44:60516' <--X--> '192.168.1.143:12345'
T 60s: 54.8 GB 913.63 MB/s (7.31 Gb/s) Pkg# 0 PLoss# 0 (0.0000%), Msg# 0 MLoss# 0 (0.0000%) 1m:0.49 5m:0.19 15m:0.12

13 Monitoring tools used

This section lists the monitoring tools used.

13.1 Mpstat

Used to monitor core cpu usage and interrupt rates.

mpstat -P ALL 5

05:30:18 PM	CPU	%user	%nice	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
05:30:23 PM	all	0.00	0.00	0.02	0.00	0.00	0.00	0.00	99.98	10079.40
05:30:23 PM	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1260.00
05:30:23 PM	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1259.80
05:30:23 PM	2	0.00	0.00	0.20	0.00	0.00	0.00	0.00	99.80	1260.00
05:30:23 PM	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1260.00
05:30:23 PM	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1259.80
05:30:23 PM	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1260.00
05:30:23 PM	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1260.00
05:30:23 PM	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	1259.80
05:30:23 PM	8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

13.2 Interrupt inspection

grep eth2 /proc/interrupts

4336: 10 11 14 12 11 15 10 13 PCI-MSI-edge eth0-lsc

4337: 99710 98249 102951 99198 99512 101887 100899 10569021 PCI-MSI-edge eth0-rx0

4338: 49776 50553 49905 50514 49538 50609 3670785 49162 PCI-MSI-edge eth0-tx0

13.3 Setting interrupt affinity

echo 1 > /proc/irq/4337/smp_affinity
echo 2 > /proc/irq/4338/smp_affinity

13.4 Other monitoring tools

Not yet complete.

vmstat =

netstat =

lspci =

oprofile =

modinfo =

sar =

14 Configuration tools

The tools used to modify configurations are listed. Note yet complete.

14.1 sysctl

sysctl -w net.core.wmem_max=16777216

14.2 ifconfig

ifconfig eth2 down

ifconfig eth2 mtu 9600 txqueuelen 10000 up

14.3 ethtool

ethtool -k eth2

14.4 Other configuration tools

setpci =

/proc =

15 Packet handling

The motivation for including this section is to give an understanding of what happens to a packet, from its arrival at the NIC until it appears in the user buffer. The current, stolen, implementation of the section is a placeholder.

The following paragraphs are stolen from the Interrupt Swizzling Solution for Intel® 5000 Chipset Series-based Platforms - Application Note (http://www.intel.com/Assets/PDF/appnote/314337.pdf). and should apply to Linux kernel's ≥ 2.4.

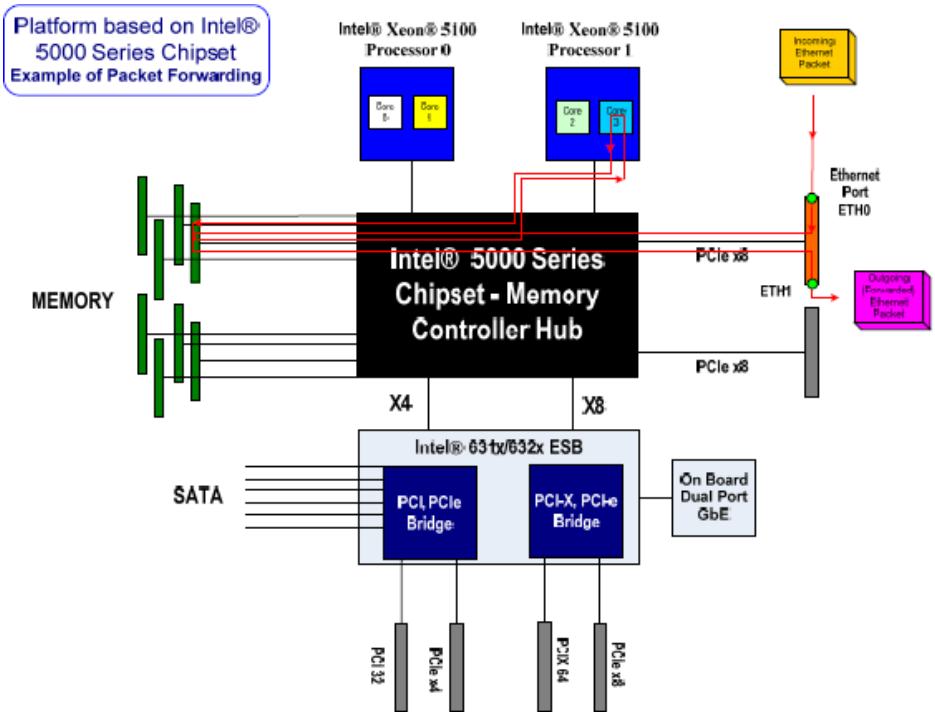

Figure 3 Intel E5000 handbook showing bus connectivity of host main board

A typical network processing scenario, a network packet is received over an Ethernet port. The NIC controlling that port deposits the packet in the memory and signals the CPU through a HW interrupt. The CPU runs an interrupt service routine (ISR) to attend the HW interrupt and update the HW status of the network device. At the end of the ISR, the CPU queues the follow-up work for processing the received packet(s) by signaling a SoftIRQ. SoftIRQs, being highly privileged threads in Linux, are closely guarded resources. The networking stack has one SoftIRQ permanently assigned in the architecture of Linux kernel. SoftIRQs are a per CPU resource, hence there is one networking SoftIRQ per available core in SMP platforms.

As mentioned, all packet processing, that is, queueing and dequeing packets from the network interface, IPv4/IPv6 processing and TCP/UDP processing in Linux happens in the SoftIRQ context. In fact, several layer 3 and layer 4 networking functions such as firewall, VPN, proxy and intrusion detection are processed in the same SoftIRQ context upon receiving a packet.

Since SoftIRQs are CPU specific and are triggered through ISRs for specific HW devices, functions such as routing (or layer 3 forwarding) have affinity at the software level to a specific CPU core. This is the same core that receives the hardware interrupt upon receipt of a packet.

16 References

Additional information sources which were found to be useful whilst pursuing the aim of this note are listed below.

1. **10 Gb Ethernet, M.Wagner, Red Hat. (18-20th June 2008).** A slide show presentation about 10Gb Ethernet performance, tuning and functionality on RHEL5.X (Red Hat Enterprise Linux release 5.X).

2. **Anatomy of real-time Linux architectures. From soft to hard realtime. (15 Apr 2008).** M.Tim Jones (mtj@mtjones.com), Consultant Engineer, Emulex Corp. http://www.ibm.com/developerworks/linux/library/l-real-time linux/index.html?S_TACT=105AGX99&S_CMP=CP