Measurement of IP forwarding performance on complex computer architectures

Olof Hagsand
KTH

Jens Laas
Uppsala U

Robert Olsson
Uppsala U

Bengt Gorden
KTH

Abstract—**Open-source routers on new PC hardware allows for forwarding speeds of 10Gb/s and above. We present detailed performance measurements using Linux on two complex PC hardware platforms. Both platforms use PCIe gen2, dual I/O bridges and have support for non-uniform memory access (NUMA). The AMD platform uses four processors equipped with eight cores and four nodes of local memory. The Intel platform has two quad-core CPUs each with local memory.**

Packets being forwarded through a PC-based router can be separated into three steps: receive-dma, lookup, and transmit-dma. Each step was studied individually. In particular, we studied how varying the CPU core and memory node effects the forwarding speeds.

Our results show a large performance dependency of selecting CPU cores and memory nodes. In particular, DMA works best with memory nodes closest to the I/O bridge where the interface card is connected. Correspondingly, CPU access is most efficient on local memory. Consequently, choosing CPU core and memory nodes badly leads to a significant performance decrease.

I. Introduction

Although the first IP routers were software-based, the forwarding in modern commercial routers are primarily hardware-based, containing applications specific circuits (ASICs), high performance switching backplanes(e.g. cross-bars) and advanced memory systems (including TCAMs). This enables current routers to perform wire-speed routing up to Terabit speeds. The commercial high-end routers of today have little in common with a standard desktop PC.

However, the complexity of the forwarding and routing protocols have increased resulting in more hardware, and more complex software modules, up to a point where hardware cost, power consumption and protocol complexity are important limiting factors of network deployment.

Simultaneously, router development on general-purpose computer platforms (such as PC's) have also advanced. In particular, general purpose hardware combined with open-source software [6], [7], [8] have the advantages of offering a low-cost and flexible solution that is tractable for several niches of networking deployment. Such a platform is inexpensive since it uses off-the-shelf commodity hardware, and flexible in the sense of its openness of the source software and a potentially large development community.

However, many previous efforts have been hindered by performance requirements. While it has been possible to deploy open source routers as packet filterers on medium-bandwidth networks it has been difficult to connect them to high-bandwidth up-links.

In earlier work we have shown [2] how multi-core CPU architectures with NUMA and parallel PCIe buses combined with 10G Ethernet interface cards could be used to speed up packet forwarding of a Linux-based router. In this work we have explored the combinations of memory node, CPU-core and I/O bus configurations. This is a task involving high complexity with many degrees of freedom.

Forwarding in software on a PC-based platform is done by a combination of DMA and CPU processing. The CPU sets up DMA between interface cards and main memory, investigates the packet header and performs lookup, filtering, etc.

However, when multiple, parallell cores are used with NUMA and hardware classification, this process is more complicated. A CPU core allocates memory from a selected memory node. This memory is used for DMA so that interface cards can push packets directly to memory on a specific memory node. By using hardware classifiers, a card can use several queues to transfer data to several locations in parallel.

In this work, we investigate the performance when using different combinations of memory nodes, CPU cores and I/O bridges when performing individual tasks of forwarding. We identified and isolated some tasks and studied each separately. By this, we hope to gain a better understanding of the forwarding behavior and thus be able to utilize the new hardware platforms better in the future.

II. Experimental platform

The experiments used a release of Bifrost [6] 6.1 using the LC-trie forwarding engine [4]. The network interface cards were Intel 10 Gigabit based on the Intel 82599 chipset [1]. The cards have multiple RX and TX queues with multiple interrupt and DMA channels.

The network interface cards have hardware classifiers that compute a hash-value of the packet header. The hash-value is used to select receive queue, and thus selects which memory location the receive DMA transfers the packet to. Load-balancing between CPUs therefore require the traffic header to consist of several flows, enough to make the hash-function evenly distribute the traffic over the different CPUs.

A. Intel platform

The motherboard of the Intel platform is a TYAN 7025 with two physical Quad-Core XEON Processors E5620 at 2.4 GHz, each with 4 cores plus 4 hyper-threading cores giving a total

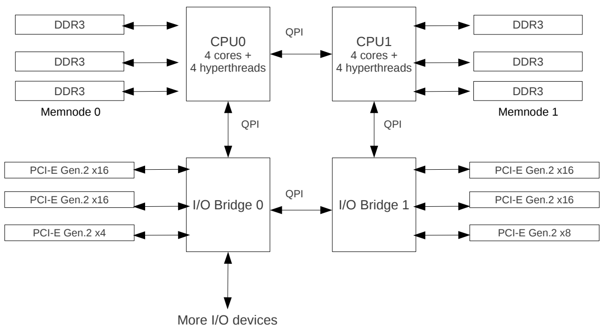

Figure 1. Block diagram of the Intel XEON platform.

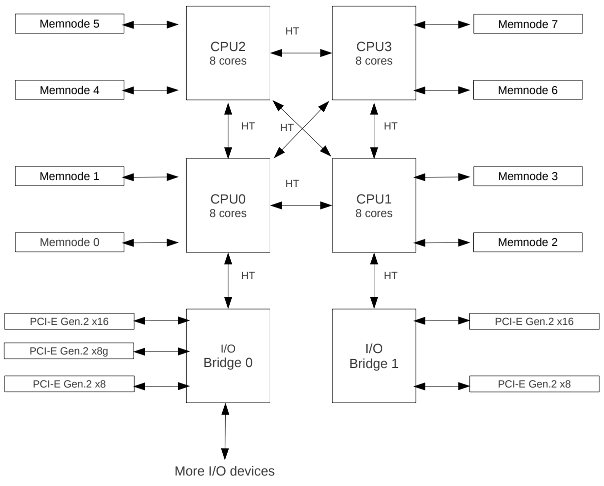

Figure 2. Block diagram of the AMD Opteron platform.

of 16 CPU-cores, see Figure 1. The hyper-threads are seen as individual cores by the operating system.

Each physical CPU has has its own memory node with three memory channels, thus forming a simple NUMA architecture. Internal buses are QPI.

B. AMD platform

The motherboard of the second platform is a SuperMicro H8QG6/H8QG6i with four physical eight-core AMD Opteron Processor 6140 with 2.6 GHz giving a total of 32 CPU-cores, see Figure 2. Each physical CPU has two local memory nodes. The AMD internal cross-connect structure is schematically shown in the block diagram.

An Opteron processor is actually sub-divided into two blocks of four cores each. To each such quad is assigned a separate memory node. It is shown in the experiments that from a performance point-of-view, it may be more correct to regard the system as eight CPUs with four quad cores each, rather than four CPUs with eight cores each.

C. Tools

We used several existing tools, and developed some of our own. We also made modifications to the Linux kernel.

The traffic was generated using a modified version of pktgen [3]. In particular, by carefully selecting flows we could control which RX queue was used by the hash-based classification on the interface cards [5], and thus control which CPU core received the traffic.

We developed the tool `eth_affinity` to control interrupts, device queues, and CPU cores, while user space CPU core and memory nodes were controlled by the `taskset` and `numactl` commands.

CPU memory latency and bandwidth tests were made using two tools from the `lmbench` package: `lat_mem_rd` to measure memory latency and `bw_mem` to measure memory bandwidth.

III. RESULTS

A simple setup was used where two PC routers generated and received traffic [2].

The experiments were divided into three basic steps of packet forwarding on a software-based router:

- *RXDMA* Receive DMA performance.
- *MEMCPU* Memory node bandwidth and latency by CPU read access from different CPU cores.
- *TXDMA* Transmit DMA performance.

The experiments were devised as follows:

- *Single*. In these experiments, the performance of the combination of single core and memory node was measured.
- *Multiple*. In some experiments, several cores were used. In this cases, local memory node was used.

In all experiments, a single 10Gb/s interface card was attached to the x8 PCIe slot on I/O bridge 0. Therefore, the card can be said to be 'closest' to CPU0 in both cases.

A. RXDMA

In the receive DMA case, the packet generator sent different flows in order to control which CPU processed the data using the technique described in Section II-C. The packet length was 64 bytes.

Packets were received and classified by the interface card on the router and transmitted to local memory using DMA. The associated CPU processed the packets but simply dropped them before lookup and forwarding was made using a kernel patch in `ip_forward`.

1) RXDMA on single cores, Intel: In the first experiment, single CPU cores and memory nodes were used to receive traffic on the Intel platform. We observed that the performance of memory node 0 was around 2.2Mpps, while only 2.0 Mpps for memory node 1, independent of CPU core. The interface card was attached via I/O bridge 0 which is closer to memory node 0 in QPI hops.

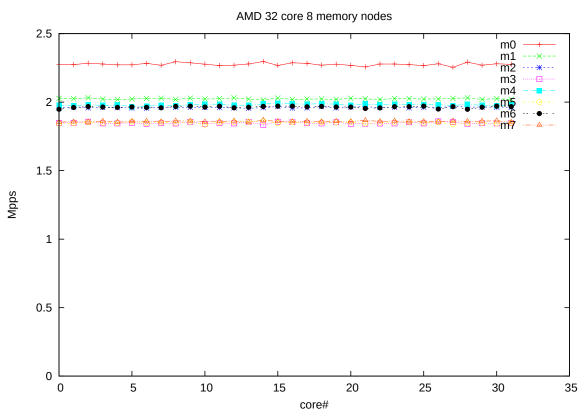

Figure 3. *RXDMA single AMD*: Packets per second of eight memory nodes.

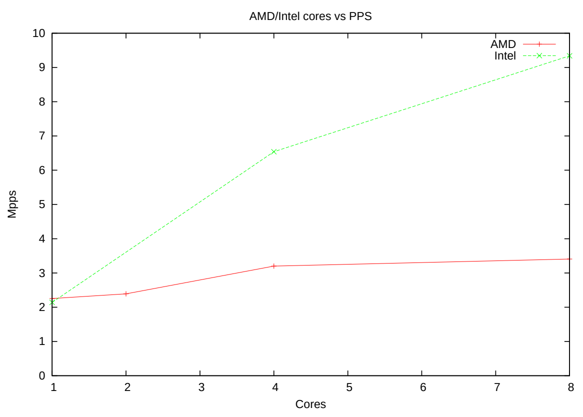

Figure 4. *RXDMA multi*: Packets per second on Intel and AMD when several cores were used in parallell.

2) RXDMA on single cores, AMD: The same experiment was repeated for the AMD platform. As can be seen in Figure 3, the performance of memory node 0 is higher than the others, around 2.3Mpps. One can also see that memory nodes 3, 5 and 7 have a lower performance than the others, and memory node 1 has slightly better than 2, 4 and 6.

The AMD is more intricate than the Intel platform, but the main result is the same, that the memory node closest to the active I/O bridge has the best performance. The other results can be explained if one sees the AMD system as an eight CPU system, where the distance from cores 0-3 to memory nodes 1, 2, 4, and 6 is distance one, and distance to memory nodes 3, 5 and 7 is distance two.

3) RXDMA on multiple cores: In the next experiment, flows were tuned so that several cores received traffic in parallel. CPU cores used only local memory. It can be seen in Figure 4 that the performance of the Intel platform increases to over 9 Mpps when 8 cores were used, while the AMD levels off at 3.4 Mpps. Although the performance of a single AMD core is better, the Intel scale better with the same driver and software being run in the two cases.

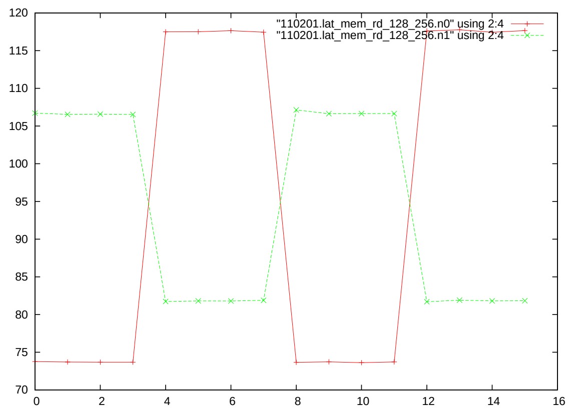

Figure 5. *MEMCPU single Intel:* Memory latency in ns as a function of CPU cores.

When studying software profiles we could see a software spinlock having much higher contention in the AMD profiles. This needs to be further analyzed but could possibly indicate that the cost of cache coherency is higher on AMD with its eight memory nodes compared with Intel's two.

B. MEMCPU, Memory latency and memory latencies

The second class of experiments illustrates the case where packets reside in memory (RAM), and a single CPU core accesses the packet header to make IP lookup, filtering, queueing, etc.

A CPU core loads a packet from memory. If this memory is not local to the core due to NUMA, this initial load (and store) may have varying latency and bandwidth. After the initial load, the processing of the packet header is most probably made in the L1 cache.

1) MEMCPU Intel latency: Memory latency for the Intel platform is shown in Figure 5 for the two memory nodes. The results show that local memory is faster (e.g., 74ns) for cores in CPU 0, while slower when accessing the other memory node (e.g., 118 ns).

The plots are asymmetrical for the two memory nodes. This was due to slightly different DIMMs in the memory nodes.

One can also observe the hyper-threads (cores 8-15), which behave in a similar way to the real cores (0-7).

2) MEMCPU AMD latency: Figure 6 shows the memory latency for the first three memory nodes in the AMD system. There are four levels in the graph: around $47ns$, $90ns$, $95ns$ and $132ns$. These illustrate: local memory node ($47ns$); same CPU other quad ($90ns$); remote CPU same quad ($95ns$); remote CPU other quad ($132ns$). This illustrates the internal HT layout of the AMD architecture.

The AMD platform was further investigated in terms of memory bandwidth, the results are similar to the latency

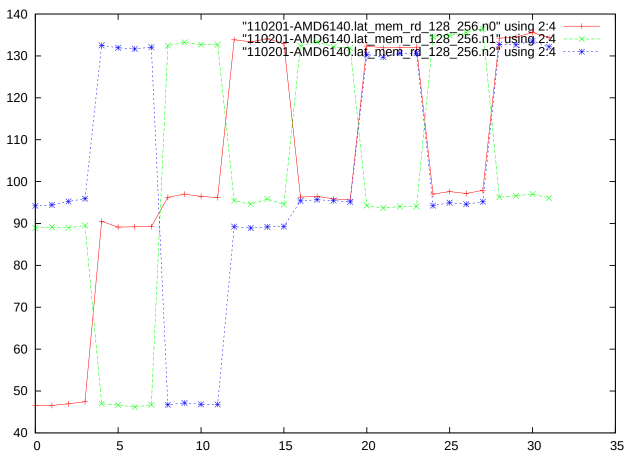

Figure 6. *MEMCPU single AMD:* Memory latency in ns as a function of CPU cores (first three memnodes).

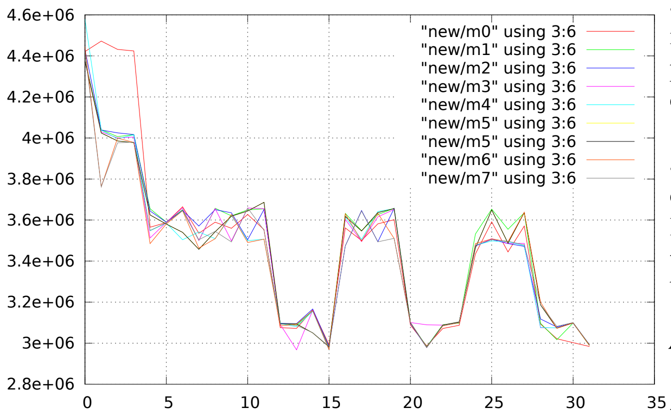

Figure 7. *TXDMA single AMD:* Packets per second of eight memory nodes.

measurements. In particular, local memory performs more than three times better than the most 'remote' memory.

C. Transmit DMA

When studying transmit DMA, packets were generated from one CPU core at a time using different memory nodes, and the packet-per-second performance was measured.

1) TXDMA AMD: The TXDMA measurement for AMD is shown in Figure 7 and shows a somewhat different behaviour than RXDMA. Better performance can be observed for memory node 0, but only for the first quad of CPU0. A dependency of CPU core is also evident which was not present for RXDMA. The same pattern visible in the MEMCPU case is also seen here: the number of inter-CPU hops is visible in the performance graphs, and performance is better for the CPU closest to the I/O bridge where the interface card was attached.

IV. Conclusions and Future Work

We have made detailed performance measurements related to packet forwarding on two complex PC platforms. We have studied DMA and memory behaviour when using different CPU cores and memory nodes.

Our results show the significance of selecting a good combination of CPU cores and memory nodes. Essentially, memory nodes should be close to the corresponding I/O bridge or CPU performing the task. At the same time however, high forwarding speeds require parallelized processing by using all CPU cores.

Further, the outgoing I/O bridge may not be known until the lookup has been made which means that TXDMA may not always be made locally.

This leads to an optimizing problem where our results provide some guidance in how the forwarding load should be balanced among processors and memory nodes within a given architecture.

Several of the results presented in this paper are preliminary and we do not yet know where the bottlenecks in each case are. The Linux networking community have done a good job in making the forwarding code scale with the number of processors, but the software is still a limiting factor in some cases.

We have studied RXDMA, TXDMA and memory latency and bandwidth separately. We will continue the study by combining the individual measurements in a complete parallelized forwarding. Optimal forwarding performance may lead to trade-offs between parallel processing and location as shown by these studies.

Acknowledgements

This paper is a result of a study sponsored by .SE, Stiftelsen for Internetinfrastruktur. Interface boards were provided by Intel; CPUs by AMD. Uppsala university and KTH provided experimental networking infrastructures.

References

[1] *Intel 82599 10GbE Controller Datasheet*, Revision 2.4, Intel Corporation, September, 2010.

[2] O Hagsand, R.Olsson., B. Gorden *Towards 10Gb/s open source routing*. In Proceedings of the Linux Symposium, Hamburg, October, 2008

[3] R.Olsson. *Pktgen the linux packet generator*. In Proceedings of the Linux Symposium, Ottawa, Canada, volume 2, pages 11 - 24, 2005.

[4] S. Nilsson, and G. Karlsson, *Fast address look-up for Internet routers*, In Proc. IFIP 4th International Conference on Broadband Communications, pp. 11-22, 1998.

[5] Microsoft Corporation, "Scalable Networking: Eliminating the Receive Processing Bottleneck-Introducing RSS", WinHEC 2004 Version - April 14, 2004

[6] R. Olsson, H. Wassen, E. Pedersen, "Open Source Routing in High-Speed Production Use", Linux Kongress, October 2008.

[7] Kunihiro Ishiguro, et al, "Quagga, A routing software package for TCP/IP networks", July 2006

[8] M. Handley, E. Kohler, A. Ghosh, O. Hodson, P. Radoslavov, "Designing Extensible IP Router Software", in Proc of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI) 2005.