Using Direct Cache Access Combined With Integrated NIC Architecture to Accelerate Network Processing

Wen Su[1,2,3,4], Longbing Zhang [1,2,4], Dan Tang[1,2,4], Xiang Gao[1,2,4]
[1]Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences
[2]Institute of Computing Technology, Chinese Academy of Sciences
[3]Graduate University of Chinese Academy of Sciences
[4]Loongson Technology Corporation Limited
{suwen,lbzhang,tangdan,gaoxiang}@ict.ac.cn

Abstract—**As network speed continues to grow, new challenges of network processing are emerging. In this paper, we first study the overheads and interactions among the different steps of networking from a hardware perspective and point out that I/O and related memory have become the main bottlenecks of performance promotion. Then based on the analysis, we show that conventional optimizing solutions are insufficient due to architecture limitations. Motivated by the studies, we propose an improved Direct Cache Access (DCA) scheme combined with Integrated NIC architecture, which includes innovative architecture, optimized data transfer scheme and improved cache policy. Experimental results demonstrate that our solution improves about 26.3% network bandwidth and reduces 42.8% and 14.3% cycles on average for receiving and transmitting respectively. Also I/O and memory traffics are significantly decreased. Moreover, we investigate the I/O and cache behaviors for network processing and present some conclusions about the DCA method.**

Keywords-networking; Direct Cache Access; INIC; system architecture

I. Introduction

Recently, many researchers found that I/O system becomes the bottleneck of network performance promotion in modern computer systems [1][2][3]. Aim to support computing-intensive applications, conventional I/O system has obvious disadvantages for fast network processing, in which bulk data transfer is performed. The lack of locality support and high latency are the two main limitations for conventional I/O system, which have been wildly discussed before [2][4].

To overcome the limitations, an effective solution called Direct Cache Access (DCA) is suggested by INTEL [1]. It delivers network packets from Network Interface Card (NIC) into cache instead of memory to reduce the data accessing latency. Although the solution is promising, it is proved that DCA is insufficient to reduce the accessing latency and memory traffic due to many limitations [3][5]. Another widely disused solution is Integrated Network Interface Card (INIC), which has been used in many academic and industrial processor designs [6][7]. INIC is introduced to reduce the heavy burden for I/O registers access in NIC drivers and interrupt handling. But recent report [6] shows that the benefit of INIC is insignificant for the state of the art 10GbE network system.

In this paper, we focus on the high efficient I/O and memory system design for network processing in general-purpose-processor (GPP). Based on the analysis of existing methods, we proposed an improved DCA combined with INIC solution to reduce I/O related data transfer latency.

The key contributions of this paper are as follows:

- A review of the network processing from a hardware perspective and discussions of the main obstacles for performance promotion.
- An improved DCA combined with INIC solution which addresses the low efficiency problem of conventional I/O system.
- Investigation of the I/O and cache behaviors in network processing based on the proposed system implementation.
- Preliminary directions in the co-designs of DCA scheme and system architecture.

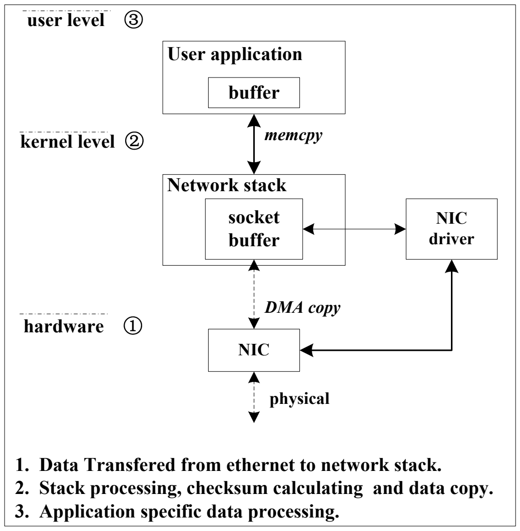
The remainder of this paper is organized as follows. Section 2 presents the background and motivation. In Section 3, we describe the improved DCA combined with INIC solution and give a framework of the proposed system implementation. In Section 4, firstly we introduce the experiment environment and methods, and then analyze the results. We carefully discuss our solutions with many existing technologies in Section 5. Finally we give some related works and conclude our paper in Section 6 and 7, respectively.

II. Background and Motivation

In this section, firstly we review the progress of network processing and point out main performance bottlenecks nowadays. Then from the perspective of computer architecture, a deep analysis of network system is given. Also the motivation of this paper is presented.

A. Network Processing

Fig. 1 illustrates the steps of network processing. At the beginning, Packets from physical line are sampled by NIC which performs the address filtering and stream control. Then the DMA engine in NIC sends the packets to the socket buffer and notifies OS to invoke network stack processing by interrupts. When OS receives the interrupt, the stack accesses the data in socket buffer and calculates the checksum. Protocol specific operations are executed layer by layer in stack processing. Finally, data is transferred from socket buffer to the user buffer allocated by applications. This operation is usually done by a function called *memcpy* in OS kernel.



user level ③
User application
buffer
kernel level ②
memcpy
Network stack
socket buffer
NIC driver
DMA copy
hardware ①
NIC
physical
1. Data Transfered from ethernet to network stack.
2. Stack processing, checksum calculating and data copy.
3. Application specific data processing.

Figure 1. Network Processing Flow

The time cost of network processing can be mainly broken down into following parts: Interrupt handling, NIC driver, stack processing, kernel routine, data copy, checksum calculation and other overheads. The first 4 parts are considered as packet cost, which means the cost scales with the number of network packets. The rests are considered as bit cost (also called data touch cost), which means the cost is in proportion to the total I/O data size. The proportion of these costs highly depends on the hardware platform and applications. There are many measurements and analysis about network processing costs [9][10]. Generally, the kernel routine cost ranges from 10% - 30% of the total cycles; the driver and interrupt handling costs range from 15% - 35%; the stack processing cost ranges from 7% - 15%; and data touch cost takes up 20% - 35%. With the development of high speed network (e.g. 10/40 Gbps Ethernet), an increasing tendency for kernel routines, driver and interrupt handling costs is observed [3].

B. Motivation

To reveal the relationship among each parts of network processing, we investigate the corresponding hardware operations. From the perspective of computer architecture, network system performance is determined by three domains: CPU, Memory and I/O. Fig. 2 depicts the relationship.

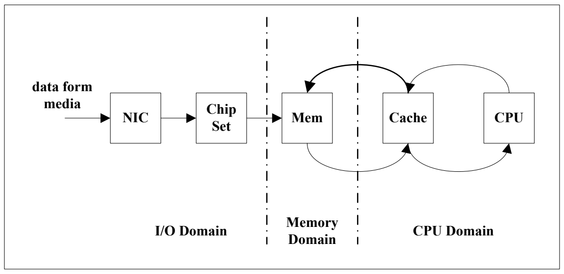


Figure 2. Hardware Perspective of Network System

Obviously, the network system can achieve its maximal performance only when the domains above are in balance. It means that the throughput or bandwidth of each hardware domain should be equal with others. Actually this is difficult for hardware designers, because the characteristics and physical implementation technologies are quite different for CPU, Memory and I/O system (chipsets) fabrications. The speed gap between memory and CPU (a.k.a "the memory wall") has been paid special attention for more than ten years, but still it is not well addressed. Also the disparity between data throughput in I/O system and computing capacity provided by CPU has been reported in recent years [1][2].

Meanwhile, it is obvious that the major time costs of network processing mentioned above are associated with I/O and memory speeds (e.g. driver processing, interrupt handling, and memory copy costs). The most important nature of network processing is the "producer-consumer locality" between every two consecutive steps of the processing flow. That means the data produced in one hardware unit will be immediately accessed by another unit (e.g. the data transferred from NIC will be accessed by CPU soon). However for conventional I/O and memory systems, the data transfer latency is high and the locality is not exploited.

Based on the analysis above, we get the observation that I/O and memory systems have become the limitations for network processing nowadays. Conventional DCA or INIC solutions cannot successfully address this problem, because they are inefficient in either I/O transfer latency or locality utilization (discussed in section 5). To diminish these limitations, we present an improved DCA combined with INIC solution. The solution not only takes the advantages of both methods but also makes many improvements in I/O and memory policies and software scheme.

III. Design Methodologies

In this section, we describe the proposed DCA combined with INIC solution and give a framework of the system implementation. Firstly, we present the improved DCA technology and discuss the key points of incorporating it into I/O and memory designs. Then, important data structures and details of the DCA scheme are given. Finally, we suggest a system interconnection architecture with the NIC integration.

A. *Improved DCA*

With the purpose of reducing data transfer latency and memory traffic, we present an improved Direct Cache Access solution. Differing from conventional DCA scheme, our solution carefully considers the following points.

The first one is cache coherence. Conventionally, data sent from device by DMA is stored in memory only. And for the same address, a different copy of data is stored in cache which usually needs additional coherent unit to perform snoop operation [11]; but when DCA is used, I/O data and CPU data are both stored in cache with one copy for one memory address. Although snoop operation is needless for DCA based transfer, conventional method still performs that operation due to the difficulty of distinguishing DCA and non-DCA requests. Our solution successfully addresses this problem by modifying cache policy and system architecture (discussed later), which separates the data paths of these two types of requests. The scheme is shown in fig. 3.

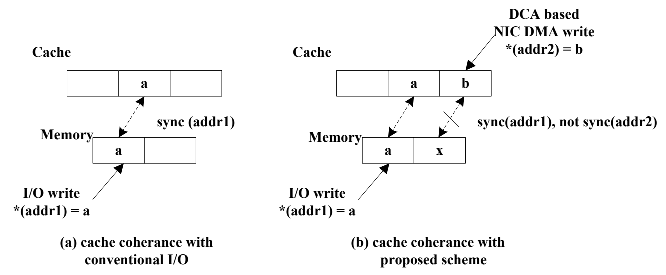


Figure 3. Two Cache Coherence Models

The second one is cache pollution. DCA is a mixed blessing to CPU: On one side, it accelerates the data transfer; on the other side, it harms the locality of other programs executed in CPU and causes cache pollution. Cache pollution is highly depended on the I/O data size, which is always quite large. For example, one Ethernet packet contains a maximal 1492 bytes normal payload and a maximal 65536 bytes large payload for Large Segment Offload (LSO). That means with a common network buffer (usually 50 ~ 400 packets size), a maximal size ranging from 400KB to 16MB of data is sent to cache. Such big size of data will cause cache performance drop dramatically. In this paper, we carefully investigate the relationship between the size of I/O data sent with DCA and the size of cache system (in Section 4). Scheduling of the data sent with DCA is an effective way to improve performance, but it is beyond the scope of this paper.

The third one is DCA policy. DCA policy refers to the determination of when and which part of the data is transferred with DCA. Obviously, the scheme is application specific and varies with different user targets. In this paper, we make a specific memory address space in system to receive the data transferred with DCA.

B. *DCA Scheme and details*

To accelerate network processing, many important data structures used in NIC driver and the stack are coupled with DCA. NIC Descriptors and the associated data buffers are paid special attention in our solution. The former is the data transfer interface between DMA and CPU, and the later contains the packets. For further research, each packet stored in buffer is divided into the header and the payload. Normally the header is accessed by protocols frequently, but the payload is accessed much fewer with modern network stack and OS. The details of the related data structures and network processing steps can be found in previous works [13].

Table 1. Receive-Side Transitions and Memory Accesses

Data transfer type			**request**	**Cost evaluation (cycles)**		
Req.	Op.	data structure	Cache lines	DCA with INIC	DCA without INIC	DNIC without DCA
1 CPU	write	descriptor	1	15	400	400
2 NIC	read	descriptor	1	15	400	1200
3 NIC	write	header	1	15	400	1200
4 NIC	write	payload	< 40	400	10500	48000
5 NIC	write	descriptor (status)	1	15	400	1200
6 CPU	read	descriptor (status)	1	15	400	400
7 CPU	read	header	1	15	400	400
8 CPU	read	payload	< 40	600	16000	16000
		Summary		1090	28900	68800

*status in descriptor displays whether DMA successful done the transfer.

The process of transferring one packet from NIC to the stack with proposed solution is illustrated in Table 1. All the accessing latency parameters in Table 1 are based on a state of the art multi-core processor system [3]. One thing should be noticed is that the cache accessing latency from I/O is nearly the same with that from CPU. But the memory accessing latency from I/O is about 2/3 of that from CPU due to the complex hardware hierarchy above the main memory.

We can see that our solution saves above 95% CPU cycles in theory and avoids all the traffic to memory controller. In this paper, we transfer the NIC Descriptors and the data buffers including headers and payloads with DCA to achieve the best performance. But when cache size is small, only transfer the Descriptors and the headers with DCA is an alternative solution.

DCA performance is highly depended on cache policy. Obviously for cache system, write-back with write-allocate policy can help DCA achieves better performance than write-through with write non-allocate policy. Based on the analysis before, we do not use the snooping cache technology to maintain cache coherence for DCA accesses.

C. *On-chip network and integrated NIC*

To implement the proposed solution, we present a system interconnection architecture with improved I/O and memory subsystems. The system is derived from a general purpose processor (GPP) called Godson3 [14]. The proposed architecture is shown in fig. 4.

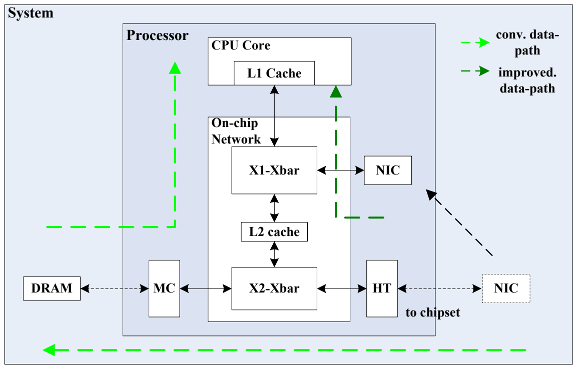


Figure 4. System Interconnection Architecture

In fig. 4, two cross-bars are employed in the on chip network. The X1-Xbar stores and arranges the accessing requests from L1 Cache and NIC, and then it sends the requests to L2 cache. X2-Xbar controls the conventional I/O requests sent to DRAM. Differing from many conventional INIC solutions, we attach the NIC to the nearest place to CPU core instead of the place below I/O controller (HT in fig.4) to diminish hundreds of cycles spent on the data transformation in I/O Bridge.

Since the DCA and non-DCA requests are mapped into different addresses, we modify the buffer and descriptor allocations in NIC driver. Packets sent from NIC are routed in X1-Xbar, which decides the target of the request is either L2 cache (cacheable) or memory (uncacheable).

IV. EVALUATION

We evaluate the proposed system with benchmarks in silicon chip and simulator, and then we compare the network bandwidth and latency of two systems using and not using the improved DCA scheme with INIC. Then we deeply analyze I/O traffic and cache performance to reveal the benefits obtained from the solution.

A. *Experiment Environment and Method*

Since the proposed architecture has been successfully used in a Godson3 single-core processor, we evaluate the network performance basing on the silicon chip. Details of the test environment are shown in Table 2. A 1GbE NIC intellectual property (GMAC-IP) developed by Synopsys [17] is incorporated as the INIC. A quad-core 2.66GHz Dell computer is used as the stressor which is connected to the system under test (SUT) through a directly linked Giga Ethernet.

Table 2. Receive-Side Transitions and Memory Accesses

CPU	
Frequency	800MHz
Issue rate	4 instructions per cycle
Instruction set	MIPS compatible
Memory & Cache	
Cache	64KBL1+512KBL2, four-way set-associative
Memory	250MHz, DDR2
IO	
Intern Bus	800MHz, 128 bits AXI
I/O Bus	200MHz, HT 1.0
NIC	Giga Ethernet

We evaluate the SUT in Linux 2.6.21 OS with Netperf [15] benchmarks. Packets with different payload size are tested to reveal the relationship between different applications and the system performance.

To deeply evaluate our system, we also use the Cadence Xtreme-3 simulation environment [16] to monitor system detail behaviors. Xtreme can highly simulate the behavior of VLSI with cycle accuracy, which is widely adopted by many IC companies as their design verification environment. There are no differences in system architecture between silicon chip and simulator, but the software and micro-benchmarks in simulator is simplified.

To investigate cache and I/O system behaviors, we trace the following process in simulator: on the transmitting side, CPU stores all the I/O data to L2 cache and then NIC fetch the data with DCA and transmit it to the physical line. A Packet Generator (PG) module is connected to the physical line to collect information. On the receive side, the PG produces packets to the NIC receive line under the 1Gbps network protocol. That is to produce enough pressure on the proposed network system. Then NIC samples and sends the data to L2 cache with DCA. The process under test is shown in fig. 5.

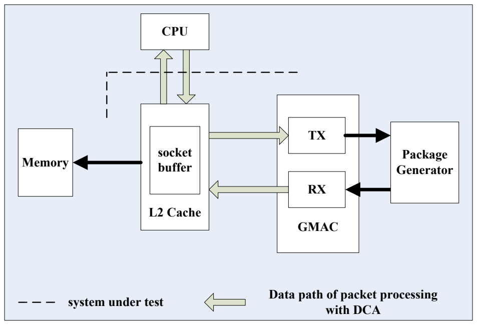


Figure 5. The Architecture of System under Test in Simulator

B. *Network System Performance Evaluation*

We measure the network bandwidth of the SUT under TCP protocol, and results are shown in fig.6. We can see that our scheme improves about 26.3% bandwidth on average, and achieves a peak value of 723Mbps.

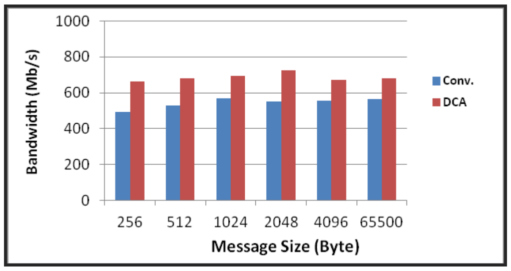


Figure 6. TCP bandwidth with different message size

Based on the simulator, we measure the Receiving (Rx) and transmitting (Tx) processes separately and the results are shown in fig.7 and 8. The two figures show that our solution saves about 42.8% CPU cycles on average for Rx and about 14.3% for Tx. The benefit decreases with I/O data increasing, especially when data size is larger than 128KB. This is mainly because the large amount of I/O data sent to L2 cache causes dramatic increment in cache misses and evictions, which significantly degrades cache performance.

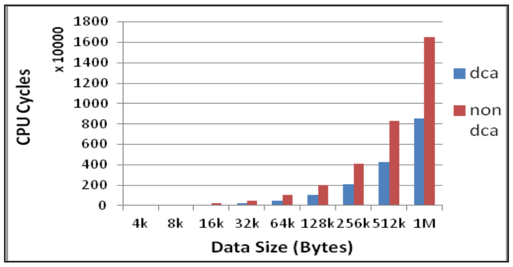


Figure 7. Measured Receiving Latency

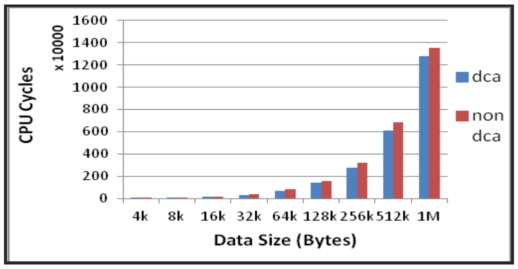


Figure 8. Measured Transmitting Latency

Obviously, the performance improvement is more significant for Rx than Tx. Considering the network processing steps, DCA can only help the data transfer between Memory and NIC (I/O transfer) for Tx; but for Rx, DCA not only reduce the I/O data transfer latency but also accelerates the stacking and memory copy. So we get the observation that DCA method is more promising in reducing memory accessing cost than improving the I/O transfer.

C. *I/O Traffic Profiling*

Based on the analysis before, our solution can effectively reduce I/O traffic and completely eliminate the overheads of accessing I/O Bridge. So we measure the latency of transmitting and receiving packets between CPU (data buffer) and NIC with different packet size. The result is presented in fig. 9.

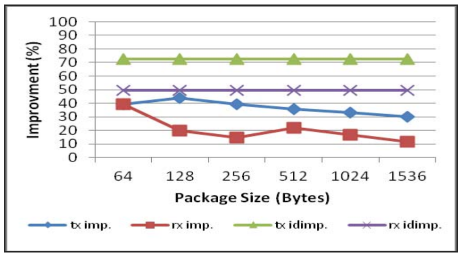


Figure 9. I/O Traffic Improvement

In fig. 9, the tx-idimp and rx-idimp refer to the theoretical improvements using DCA in transmitting and receiving. The value is calculated from the ratio of the last level cache (L2 cache in this paper) accessing latency to memory accessing latency, since DCA do not change the program behavior but just reduces the data accessing latency. We can see that the real improvements (tx imp. and rx imp.) are lower than the theoretical ones and decrease with the increasing of packet size. This is mainly because the overheads produced by DCA hardware rise with the increment of data size, e.g. the transformation overheads between I/O and cache accessing requests scales with data size.

D. *Cache Performance Profiling*

We also investigate the cache system behavior in the experiment. The benefits of the proposed solution are highly associated with the Cache Miss Rate, which is affected by the reuse distance of the transferred data in cache [2]. So we measure the Cache Miss Rate and Data Reuse Distance (DRD) on the receiving side with I/O data ranging from 64 bytes to 1024 bytes. Fig. 9 and 10 depict the results.

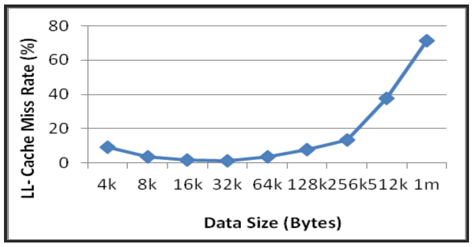


Figure 10. LL-Cache Miss Rate

From fig. 9, we see that Last Level Cache (LL-Cache) Miss Rate remains lower than 10% most of the time and slightly rises with data size increasing. But when data is larger than 256KB, the miss rate dramaticlly rises up to over 30%. The miss rate is over 70% for 1MB data, in which the later 512KB causes over 50% Compulsory Misses due to the exceeding size of data to the L2 cache. From the experiment, we get the conclusion that the data size sent to cache with DCA should not be larger than 1/4 of the LL-Cache size, otherwise much cache pollutions will be produced and system performance will decrease dramatically.

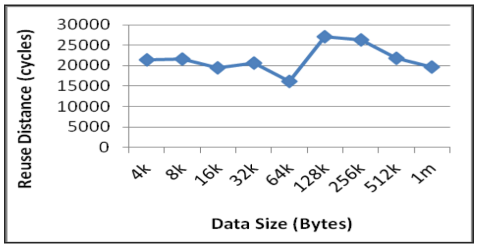


Figure 11. Reuse Distance of the I/O Data Stored in Cache

DRD is defined as the time gap between two consecutive write and read operations to the same address in cache, which evaluates the minimal time data should stay in cache [2]. Smaller the DRD is, lower the cache miss rate is. From fig. 10 we see that DRD nearly keeps a line with data size increasing, except minor fluctuations when data size is 128KB. It is confirmed with the early analysis that DRD is only affected by the nature of program and hardware architecture. We think the fluctuation is caused by the periodical interrupts handling in OS, which is proved to be a burden to network packets processing.

V. Discussions

Network processing involves CPU, Memory, I/O, stack, driver and OS designs, and each of them interacts with others. It is widely accepted that there are no solutions which can fully address the bottlenecks of network performance promotion so far [5][13]. We hold the opinion that the bottlenecks will consistently change alone with hardware fabrication technology improvements, architecture innovations and software optimizations.

DCA and INIC are two widely discussed solutions which address the inefficient I/O transfer problem for network processing. In this paper, we have the following considerations to DCA and INIC methods:

1) Only DCA cannot substantially reduce I/O and memory traffics. DCA is a data-prefect like solution and still will produce data transfers (cache invalid or cache refill operations) between cache and memory system under the existing cache policy; meanwhile compared with the less than 150 cycles saved by DCA, more than 1000 cycles are still spent on the I/O Bridge accessing [18] for each data transfer.

2) If well scheduled, DCA can effectively accelerate the stacking, driver processing and buffer maintaining processes. When I/O data and important data structures are stored in cache by DCA, CPU can access them in fewer cycles and exploits the locality to improve performance. That is very important for memory intensive operations in network processing such as data copy and buffer maintaining. Scheduling is necessary to avoid the Cache blocks be polluted by large amount I/O data.

3) INIC helps the data transfer. Due to the system architecture innovation, we observe that INIC brings more benefits by decreasing data transfer latency and diminishing the I/O traffic crossing the I/O Bridge than reducing the register accessing cost.

The proposed solution takes the advantages of both DCA and INIC method and overcomes the defects. It mainly facilities the intensive data transfer among CPU, Memory and NIC, meanwhile the I/O Bridge accessing traffic is eliminated. One thing should be paid special attention is that data transferred into cache should be accessed in a short time or else it will probably be evicted.

The software in simulator is little different with real scenario, in which stack processing and user space operations are not included. We think the conclusions about I/O and cache behavior are still available because of these processes have little effects on I/O data transfer and cache access (discussed in section 2).

VI. Related Work

Many promising solutions have been suggested to accelerate the network processing. Recently, INIC and DCA related schemes are widely discussed. Binkert e.t. [19] carefully investigated the details of INIC technology and gave a framework of V-SINIC, which allows packet-level parallelism both on transmit and receive. Liao e.t. [20] evaluated a 10GbE INIC system with Sun Niagara 2 multi-core processer and presented that INIC improves the performance of cache system and helps OS Context Switch, but the register accessing latency is not obviously reduced. Meanwhile, INTEL introduced Direct Cache Access (DCA) technology and incorporated it into their commercial CPU designs [21]. A research team of INTEL deeply analyzed the characters of DCA in both single-core and multi-core systems [1][5]. The results show that DCA is promising in reducing data transfer latency and memory traffic, but the benefits are limited by system architecture and existing policies. Our previous work [2] showed that DCA will cause cache pollutions and an alternative solution for cache injection for I/O system is also presented. In addition, Liao e.t. presented anther work on the new I/O system architecture, which shifted the NIC Descriptor management from NICs to an on-chip network engine [6].

VII. CONCLUSIONS

Since the high speed network becomes a standard feature of server and desktop platforms nowadays, numerous methods are proposed to improve the network efficiency. In this paper, we presented a solution that using improved DCA scheme combined with INIC architecture to achieve an efficient, low latency and high throughput network system. We measured the benefits obtained from our solution and investigate the I/O and Cache behaviors of network processing. The experiment showed that our solution increases over 26% network bandwidth and reduces 42.8% and 14.3% time costs on average for receiving and transmitting respectively. I/O and memory traffics are significantly decreased. Based on the results, we got the conclusion that data sent to cache with DCA should not larger than 1/4 of the last level cache size. Also we showed that DCA and INIC methods are more promising in reducing memory accessing latency than decreasing I/O latency.

I. REFERENCES

[1] R. Huggahalli, R. Iyer, S. Tetrick, "Direct Cache Access for High Bandwidth Network I/O", *ISCA*, 2005.

[2] D. Tang, Y. Bao, W. Hu et al., "DMA Cache: Using On-chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance", HPCA, 2010.

[3] Guangdeng Liao, Xia Zhu, Laxmi Bhuyan, “A New Server I/O Architecture for High Speed Networks,” HPCA, 2011.

[4] E. A. Le´on, K. B. Ferreira, and A. B. Maccabe. Reducing the Impact of the MemoryWall for I/O Using Cache Injection, In 15th IEEE Symposium on High-Performance Interconnects (HOTI’07), Aug, 2007.

[5] A.Kumar, R.Huggahalli, S.Makineni, “Characterization of Direct Cache Access on Multi-core Systems and 10GbE”, HPCA, 2009.

[6] Sun Niagara 2, http://www.sun.com/processors/niagara/index.jsp

[7] M. Ohmacht et al, “Blue Gene/L compute chip: Memory and Ethernet subsystem,” IBM Journal of Research and Development, 49(2/3):255–264, March/May 2005.

[8] Guangdeng Liao, L.Bhuyan, “Performance Measurement of an Integrated NIC Architecture with 10GbE”, 17th IEEE Symposium on High Performance Interconnects, 2009.

[9] A.Foong et al., “TCP Performance Re-visited,” IEEE Int’l Symp on Performance Analysis of Software and Systems, Mar 2003

[10] D.Clark, V.Jacobson, J.Romkey, and H.Saalwen. “An Analysis of TCP processing overhead”. IEEE Communications,June 1989.

[11] J.Doweck, “Inside Intel Core microarchitecture and smart memory access”, Intel White Paper, 2006

[12] Amit Kumar, Ram Huggahalli., Impact of Cache Coherence Protocols on the Processing of Network Traffic

[13] Wenji Wu, Matt Crawford, “Potential performance bottleneck in Linux TCP”, International Journal of Communication Systems, Vol. 20, Issue 11, pages 1263–1283, November 2007.

[14] Weiwu Hu, Jian Wang, Xiang Gao, et al, “Godson-3: a scalable multicore RISC processor with x86 emulation,” IEEE Micro, 2009. 29(2): pp. 17-29.

[15] Rick Jones. NetPerf: a network performance benchmark. http://www.netperf.org

[16] Cadence Incisive Xtreme Series. http://www.cadence.com/products/sd/ xtreme_series.

[17] Synopsys GMAC IP. http://www.synopsys.com/dw/dwtb.php?a=ethernet_mac

[18] D.J.Miller, P.M.Watts, A.W.Moore, "Motivating Future Interconnects: A Differential Measurement Analysis of PCI Latency", ANCS, 2009.

[19] Nathan L.Binkert, Ali G.Saidi, Steven K.Reinhardt. Integrated Network Interfaces for High-Bandwidth TCP/IP. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (ASPLOS). 2006

[20] G.Liao, L.Bhuyan, "Performance Measurement of an Integrated NIC Architecture with 10GbE", HotI, 2009.

[21] Intel Server Network I/O Acceleration. http://download.intel.com/technology/comms/perfnet/download/ServerNetworkIOAccel.pdf