Direct Cache Access for High Bandwidth Network I/O

Ram Huggahalli
Intel Corporation
ram.huggahalli@intel.com

Ravi Iyer
Intel Corporation
ravishankar.iyer@intel.com

Scott Tetrick
Intel Corporation
scott.tetrick@intel.com

Abstract

Recent I/O technologies such as PCI-Express and 10Gb Ethernet enable unprecedented levels of I/O bandwidths in mainstream platforms. However, in traditional architectures, memory latency alone can limit processors from matching 10 Gb inbound network I/O traffic. We propose a platform-wide method called Direct Cache Access (DCA) to deliver inbound I/O data directly into processor caches. We demonstrate that DCA provides a significant reduction in memory latency and memory bandwidth for receive intensive network I/O applications. Analysis of benchmarks such as SPECWeb9, TPC-W and TPC-C shows that overall benefit depends on the relative volume of I/O to memory traffic as well as the spatial and temporal relationship between processor and I/O memory accesses. A system level perspective for the efficient implementation of DCA is presented.

1. Introduction

Architectural and micro-architectural evaluation of processors, cache hierarchies and system interconnects has often been decoupled from I/O considerations. There are multiple recent trends that require a broader view of computer architecture. Foremost among these trends is the rapid maturity of the internet and the corresponding increase in applications and technologies that aim to make the internet a richer experience. Applications associated with broadband internet such as video and graphics not only involve large volumes of data but are at the same time response time sensitive. I/O technologies such as PCI-Express and multi-gigabit Ethernet are aimed at these emerging applications and enable high data rates within the system and between systems across a network. They represent an unprecedented increase in the amount of raw I/O throughput in mainstream platforms in the near future. Having significantly alleviated I/O throughput limitations, the ability to process high rates of I/O becomes a dominant concern.

The impact of I/O on processor efficiency can be demonstrated using the example of the 10 gigabit Ethernet standard and the processing involved in executing TCP/IP protocol. TCP/IP over Ethernet is the prevalent form of communication used by network-intensive server applications (e.g. web services and e-commerce). The relevance of TCP/IP protocol processing [1, 2, 3, 6, 9] grows stronger as Storage-over-IP starts to become popular with the help of working groups for iSCSI [7], RDMA [13] and DDP [15]. Independent of the volume of data presented by video and graphics data, traditional disk-intensive workloads may also be dependent on Ethernet capability and efficient TCP/IP processing in the future.

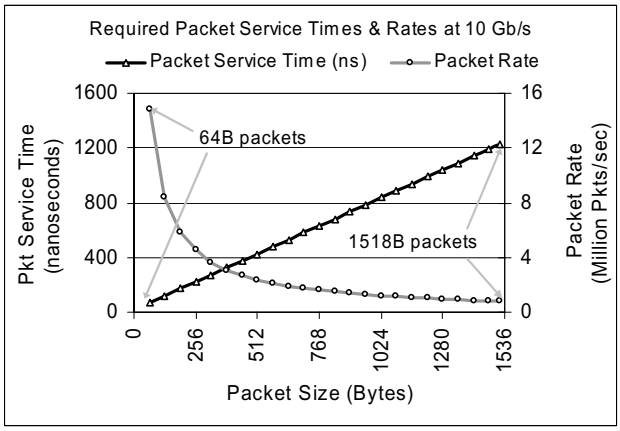


Figure 1. Expected Packet Service Time and Packet Rates at 10 Gb/s versus Packet Size

Figure 1 illustrates that a data rate of 10 Gb/s corresponds to a packet rate of 14.8 million packets per second (MP/s) with 64 byte packets. A system must be able to transmit or receive, on average, a 64 byte packet every 67 ns and at the largest Ethernet packet size of 1518 bytes, a system must be able to transmit or receive once every 1230 ns. When packet data is accessed from system memory, sustaining 10 Gb/s throughput will be very difficult without intervention. In this paper, we present and evaluate a technique called Direct Cache Access (DCA) to minimize memory subsystem dependencies and improve the balance between processing and I/O capabilities.

The rest of this paper is organized as follows. In Section 2, we provide an overview of I/O related data movement and processing. With TCP/IP as our primary I/O centric usage model, typical data structures involved in TCP/IP processing and memory accesses associated with these data structures are reviewed in detail in Section III. We also introduce DCA as an enhancement to the

Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)
1063-6897/05 $20.00 © 2005 IEEE

system interconnect coherency protocol to move inbound I/O traffic directly into processor caches. With inbound network traffic, the current coherency protocol between I/O adapters and processors results in unnecessary memory bandwidth consumption and numerous memory accesses that limit processors from being able to sustain high network I/O rates. In Section 4, the magnitude of the problem is demonstrated using memory access traces and cache simulations for various network intensive and common server benchmarks. Section 5 presents a comprehensive system level perspective for implementing the protocols in mainstream architectures. We observe that successful implementation requires simple but coordinated changes across multiple platform elements such as system software, CPU, chipsets and I/O devices. We conclude with planned future enhancements to DCA as well as other areas of improvement in handling high I/O throughputs.

2. I/O Data Movement and Processing

In this section, we provide an overview of I/O processing and point out the data movement and memory interactions that affect performance.

2.1. Basic Processor, Memory and I/O Interaction

The interaction between processors, memory and I/O adapters involves multiple data structures and also multiple system mechanisms. Common today are Direct Memory Access (DMA) techniques that decouple processor involvement during data transfers between I/O devices and the memory subsystem. Typical interactions assuming a DMA capable adapter are shown in Figure 2. The processor, as instructed by software, sets up a system memory based buffer for transmission or reception and provides an appropriate I/O adapter with a descriptor. The descriptor contains a pointer to the buffer and is used by the I/O adapter to read data for transmission or to write data in the case of reception. Upon creating a descriptor, the processor writes to a memory mapped I/O (MMIO) register on the I/O adapter to indicate the presence of a new descriptor. Descriptors are often maintained as a circular ring structure and the MMIO write serves to update a pointer into the descriptor ring. An I/O adapter equipped with a descriptor can independently complete a block transfer and provide a status in memory to the processor. Since the processor is expected to execute other operations (possibly applications) concurrently, an I/O adapter sends an interrupt to bring the processor's attention to any newly reported status information. The processor may also query the NIC directly using MMIO reads to obtain the cause of interrupts. In modern systems and I/O adapters, several optimizations exist to amortize the overhead of MMIO reads, writes and interrupts. These optimizations are implementation specific and are beyond the scope of this paper. The focus of this paper is on system memory based interactions involving status, descriptor and payload data structures. Figure 2 shows the interactions on the 'receive-side' for inbound I/O data.

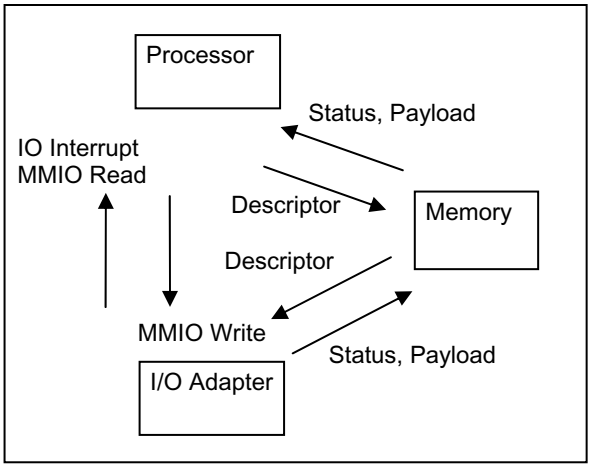


Figure 2. Receive-side Interactions between Processor, Memory and I/O Adapter

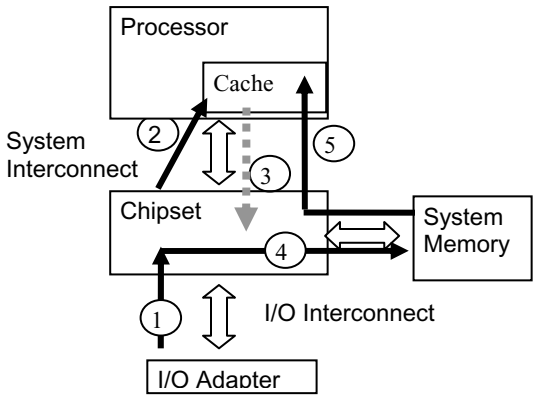
2.2 I/O Interactions From A Cache Perspective

The three data structures that hold status, descriptors and payload are allocated from system memory space that is cacheable in processor caches. It is clear from Figure 2 that cache lines modified by the processor are read by the I/O adapter while cache lines modified by the I/O adapter are read by the processor. Thus accesses from both the processor and the I/O adapter require transactions that maintain coherency using the system interconnect. The sequence of transactions required to maintain cache coherency for I/O to memory write for a single cache line is shown in Figure 3.

For both memory writes and memory read requests from the I/O adapter, the chipset acts as a bridge between I/O, processors and memory, and issues a snoop transaction for each cache line to ensure that the previous copy of the cache lines is invalidated from the processor's cache. In the case of memory writes, the cache line is invalidated, while in the case of a memory read, the cache line is either invalidated or marked as a shared cache line. DCA is concerned primarily with subsequent use of the snooped cache lines by the processor. Depending on processing requirements of inbound data, invalidated cache lines may soon be read by the processor. These memory accesses are compulsory cache misses due to the invalidation protocol implemented by current systems. At the core of DCA is a system interconnect transaction that facilitates data movement directly into the processor's cache. DCA has two benefits: 1) timely availability of

Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)
1063-6897/05 $20.00 © 2005 IEEE

data in cache leading directly to a lower average memory latency and 2) reduction in memory bandwidth requirement. An ideal implementation of DCA would eliminate the need to write data to memory, continuously updating cache lines in the cache with new data.



1. Memory write by I/O Adapter
2. Snoop on system interconnect
3. Probable writeback by processor if snoop finds modified data in cache
4. Merged write to system memory
5. Probable CPU read to process I/O data

Figure 3. Transaction Sequence used to maintain Cache Coherency during Inbound I/O Transfers

3. TCP/IP Example

The magnitude of benefit due to DCA is best demonstrated by using specific data structures used in the TCP/IP software stack and the associated memory bandwidths. One of the well-recognized issues with TCP/IP communication [12] is the significant computational and memory requirements of protocol stack processing and related overheads (such as buffer management, interrupts and scheduling) in a typical O/S environment. Motivated by these issues, several research projects contemplate the potential of using TCP offload engines (TOE) or packet processing engines (PPE) [5, 10] to accelerate TCP/IP packet processing. As a contrast, a comprehensive set of enhancements required in general purpose software and hardware to match high network I/O rates is provided in [14].

Memory accesses associated with TCP/IP processing include the same base components such as status, descriptors and payload (described in Section 2.1). Transfers across the network using Ethernet are, however, packetized where each packet contains a header portion and a payload portion. An adapter moves both header and payload for each packet to and fro with respect to memory. We assume in the following calculations that header and payload are placed in separate memory areas. A typical Ethernet and TCP/IP packet header size is 58 bytes excluding any optional bytes. The packet payload ranges from 0 to 1460 bytes or 0 to 23, 64-byte cache lines. Descriptors are typically partial cache line structures – we assume 16 bytes in the following computations.

In addition to these base memory traffic components, TCP/IP processing also involves other processor-to-memory accesses of which the memory accesses related to maintaining the context of network connections (also called TCB or Transport Control Block) are most important. TCBs are typically 512 bytes or greater although not all cache lines are frequently accessed. Unlike other data structures reviewed here, TCB accesses are immune to I/O adapter interactions.

Table 1. Receive-Side Cache Line Transitions and Memory Accesses (payload of N cache lines)

Memory Access Type			Cache Line State Transitions without DCA			Memory Accesses In Cache lines	
Requestor	**Data Structure**	**Direction**	**Descriptor**	**Header**	**Payload**	**Baseline**	**DCA**
Processor	Descriptor	Write	E to M			0	0
NIC	Descriptor	Read	M to I			1	1
NIC	Header	Write		E to I		1	0
NIC	Payload	Write			E to I	N	0
NIC	Status	Write	at I			1	0
Processor	Status	Read	I to E			1	0
Processor	Header	Read		I to E		1	0
Processor	TCB	Read					
Processor	TCB	Write					
Processor	Payload	Read			I to E	N	0
					TOTAL	**2N + 5**	**1**

The core sequence of processor and NIC (Network Interface Controller) memory accesses per packet during TCP/IP protocol processing is shown for packet reception (Receive side) in Table 1. Cache lines associated with descriptors, headers and payload undergo multiple state transitions in response to each memory access. The coherence protocol is assumed to be MESI (Modified, Exclusive, Shared and Invalid states) based. Unknown or don't care states are denoted with an 'x'. We also show cache misses resulting in accesses at memory and account for them in the table.

Key to DCA are the transitions to I state upon the NIC writing data to memory. For now, the accounting of benefits from DCA assume idealized implementations where memory writes can be completely eliminated by keeping cache lines in M state within the cache. The opportunity to reduce memory bandwidth by a factor of 2N+5 where N is the size of the payload within a packet is clearly significant.

We also computed absolute memory bandwidths due to each type of memory access when sustaining 10Gb/s as shown in Figure 4. Two packet sizes: 256 bytes and 1518 bytes were chosen for comparison – these represent control message transfers and large block transfers across the network. In both cases, the dominant source of bandwidth is clearly payload data movement. In the baseline where no DCA is assumed, bandwidths range from 2.6 to 3.8 GB/s on the receive side and 2.6 to 3.0 GB/s on the transmit side. Movement of payload data in any one direction should roughly equate to 10 Gb/s (or 1250 MB/s) save Ethernet specific overheads.

In addition to an idealized DCA, we also show a case where all modified cache lines in the cache due to NIC writes are evicted. Even when accounting evictions, we observe that memory bandwidth reduces from 3.8 GB/s to 2.2 GB/s (71%) in the case of 256 byte packets and from 2.6 GB/s to 1.4 GB/s (92%) in the case of 1518 byte packets. The bandwidth reduced corresponds to processor to memory reads – a more important limiter in a system that has memory bandwidth headroom compared to NIC to memory writes that occur transparently.

DCA has limited applicability in a transmit-intensive workload. It can only eliminate the processor-to-memory read of NIC status when transmit operations complete.

In addition to memory bandwidth reduction, DCA addresses critical latency limitations. To attain 10 Gb/s, each 256-byte packet must be processed at rate of one every 220 ns (Figure 1). From Table 1 we see that there are at least four processor-to-memory accesses that can be exposed to full memory latency: status read, header read, TCB read and payload read. This does not count back-to-back accesses when fetching multiple cache lines of a TCB or payload. Assuming, idle system latencies on recent typical servers of approximately 100 ns, we find that memory latency alone will limit the processor from matching full network I/O rate. The problem is also much more critical on receive side TCP/IP processing.

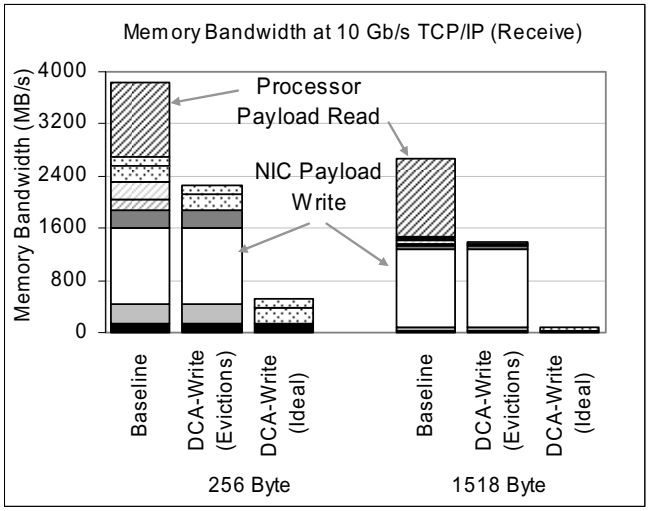


Figure 4. Memory Bandwidth Reduction due to DCA Options (Receive Side, TCB, descriptor and header accesses not labeled)

4. Memory Access Profiles of Benchmarks

We examined four benchmarks with varying amounts of I/O activity to quantify the improvement possible with DCA.

4.1. Benchmarks

The selected benchmarks are briefly described as follows

NTTTCP [20] is Microsoft's command-line tool that exercises the sockets API and is used for measuring network throughput between two end-systems. NTTTCP is used to represent forward looking network intensive applications. The NTTTCP workload is broken up into a set of Transmit (Tx) intensive tests and a set of Receive (Rx) intensive tests. In each case, we also categorize the transferred application buffer size as small, medium and large implying the range of sizes shown in Table 2.

Table 2: Application buffer size categories

Case	Size
Small	64B to 256B
Medium	512B to 4KB
Large	8KB to 64KB

SPECweb99 [17] is a benchmark that attempts to mimic a web server environment. The benchmark setup uses multiple client systems to generate an aggregate load

Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)
1063-6897/05 $20.00 © 2005 IEEE

on the system under test (a web server). Each client (mimicking browsers) initiates TCP connections to the web server and makes HTTP requests for static or dynamic web pages.

TPC-W [18] is a multi-tier benchmark that attempts to model an e-commerce environment (based on an on-line bookstore like amazon.com). Since our interest is in packet-processing, we focused on the front-end tier that handles incoming connections, processes HTTP requests, generates queries to the back-end and finally puts together a web page that is sent back to the client.

TPC-C [19] is an online-transaction processing benchmark that simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is based on the primary transactions in an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. The nature of I/O in the TPC-C benchmark is very different from the other benchmarks that we studied. It is predominantly disk I/O compared to network I/O. TPC-C, however, makes an interesting contrast demonstrating how disk I/O traffic can be different compared to network I/O. Database applications are also interesting in that much of the disk I/O, in future, may translate into network I/O if the a remote disk subsystem over IP networks is used.

4.1 Evaluation Methodology

Our evaluation methodology consists of an extensive set of cache simulations fed by memory access traces collected on an existing system running the workloads of interest. Cache simulations were performed using the CASPER simulation framework [8]. CASPER enables the analysis of a spectrum of cache structures and hierarchies including UP & MP cache hierarchies, shared caches, chipset caches and snoop filters. Prior research using a hardware cache emulator for TPC-H and cache coherence protocols represents an alternative approach [4].

For NTTTCP, each trace was 30M instructions long while in all other cases, the traces were 100M instructions long. Since only about 10K instructions are executed per Ethernet packet, and the memory access patterns are repetitious, the 30M instructions per NTTTCP trace should be sufficient. In the case of SPECWeb99, TPC-C and TPC-W, each trace corresponds to 2 to 5M bus transactions. The traces and their lengths are expected to be sufficient to demonstrate a contrasting set of characteristics which can guide system design. We caution that the trace based analysis results presented here should not be applied directly to any official statement of benchmark scores.

The traces were collected on the system bus (also known as Front-Side Bus or FSB) on a dual processor, Pentium® III Xeon™ system with a 2M 2nd level cache. For SPECWeb99 and TPC-W, memory subsystems were configured to 4-8GB in order to keep the disk traffic at negligible levels. We simulated 4MB, 8MB and 16MB caches, all configured to a 64-byte line size and 8-way set associativity.

4.2 Usage of I/O Writes by Processor

I/O write addresses and processor reads to the same address were tracked in terms of both spatial and temporal aspects of their relationship. In Table 3, we show the portion of all processor references due to I/O write traffic – these processor references (cache misses) occur only because memory has been updated by new data from the NIC or disk controller. We also show how much of the I/O write traffic is ever used by the processor.

The first column in Table 3, indicates that most processor references are unrelated to the I/O write traffic in the case of transmit intensive workloads including SPECWeb99 and TPC-W. We will show in a later section that as cache sizes increase, a higher percentage of processor references to system memory will be related to I/O writes.

Table 3: Importance of Incoming I/O Data

	% of All CPU references due to I/O Writes	% of I/O Writes referenced by CPU
Tx-small	3.1%	100.0%
Tx-medium	4.1%	100.0%
Tx-	4.7%	100.0%
Rx-small	34.3%	100.0%
Rx-medium	55.8%	100.0%
Rx-large	66.9%	99.7%
SPECWeb99	2.5%	99.7%
TPC-W	1.1%	99.7%
TPC-	2.9%	7.1%

As shown in the second column, the percentage of incoming data subsequently read by the processor is nearly 100% whenever the source of the I/O write traffic is the NIC. TPC-C is a contrasting case where only 7.1% of the I/O writes are touched by the processor. Unlike NIC I/O writes, processing of data from a disk controller I/O is highly application dependent. Inbound disk traffic is not subjected to protocol processing upon arrival in memory unlike the packetized data moved by a NIC.

4.3 Distance between I/O Writes and Processor Reads

In order to benefit from direct placement of the I/O data in the cache, the processor must read the data from the cache in a timely manner. Otherwise, the modified cache line may be evicted to memory before it is read. In addition, the cache is subjected to two replacements instead of one – first, when placing the I/O data in the

cache and second, due to untimely placement, when the processor misses the cache anyway.

We characterize the temporal distance between incoming data that is placed in memory and the subsequent CPU access to this data. This distance is measured in terms of system bus clocks (66 MHz) between these two events and is shown in Figure 5. Figure 5 illustrates the fundamental characteristic of packetized workloads that must undergo protocol processing before the data portion of the packet is delivered to the application. In most cases, nearly 100% of the I/O writes are touched within 20K bus cycles. The packets are also processed in a batched fashion by current stacks. Upon receiving an interrupt from the NIC, the driver and the OS process one or more packets as they arrive across the network. The status of a packet in the descriptor structure and the packet header and processed as quickly as possible. While the application's use of the data itself is highly dependent on application's characteristics, we observe that the data portion of the packet is also touched very early in most cases. For very large application buffer sizes (32KB and 64KB), we observed that stack optimizations to remove intermediate copies of the data were used. In this case, only the header portion of the packet is used within a short distance, and payload data movement is deferred until the entire payload arrives across the network. At this point, the payload may be directly transferred to the application buffer. Note that for Rx-large, only 80% of the I/O writes are touched within 20K bus cycles.

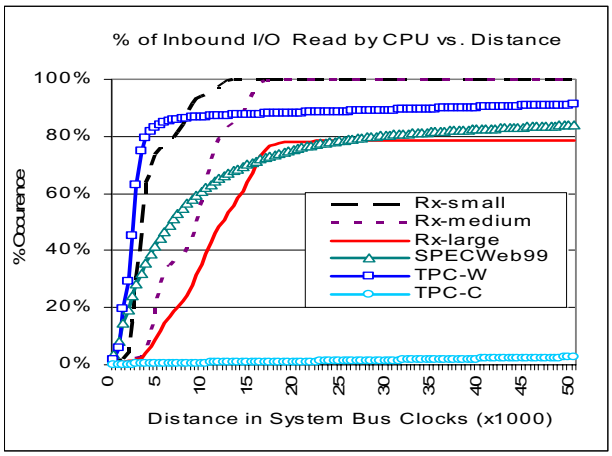


Figure 5: Distance between I/O Write & CPU Read

SPECWeb99 and TPC-W exhibit very similar temporal behavior to NTTTCP. 80% of the I/O writes are read within 30K bus cycles in the two cases. TPC-C data is consistent with our expectations of disk related I/O traffic – the distance is too long for the benchmark to benefit much from DCA.

Data in this section clearly illustrates the opportunity to reduce a significant portion of CPU cache misses using DCA. A high volume of incoming network I/O traffic is a pre-cursor to substantial performance improvements with DCA. Most network I/O traffic is touched by the processor and is touched within a reasonably short interval. This characteristic allows for a direct reduction in compulsory misses in the case of network I/O if the I/O write data is placed directly in the processor's cache.

4.4 Traffic Profiles

Figure 6 shows the profile of cache misses in conjunction with the I/O read and write snoop traffic that the cache is exposed to. These profiles may be thought of as 'address bus' profiles and are based on simulations of a 4MB cache. Each profile shows code reads, data reads, reads-for-ownership (RFOs) and write-backs from the processor (or CPU as labeled in the following figures). RFO's are issued by the processor when a store instruction is executed and all cache levels in the processor incur a miss. A cache line is read into the cache, marked exclusive (E state) and updated in the cache subsequently (M state).

The traffic profiles for the Tx cases indicate a significant amount of I/O read traffic. This property is expected since the NIC reads data from memory using DMA before transmitting the data to the network. The Tx cases, however, do contain a small amount of I/O write traffic corresponding to the processing of ACKs received for any packets sent and also for descriptor updates by the NIC. The ACKs are often 'coalesced' such that the number of explicit ACK packets required is a small fraction of the transmitted packets.

A large portion of the Rx traffic is due to I/O writes from the NIC. A related difference compared to the Tx cases is that there are a high number of CPU data read misses. The overall reduction in traffic is directly related to the reduction in data read misses.

SPECWeb99 and TPC-W (front-end) respond to client requests that are typically small packets by sending web pages (constituting multiple packets) across the network. Both workloads have a much smaller I/O write component compared to the I/O read component. Web servers that redirect most of the requests to other servers or that aggregate large amounts of content from application or back-end servers are likely to have different profiles. The small percent of I/O write traffic can be sufficient to demonstrate visible performance gains for SPECWeb99 and TPC-W since it affects performance critical data read misses.

TPC-C has a significantly different profile compared to either TPC-W or SPECWeb99 with a much larger portion of the total traffic being I/O writes. Since the data placed in the cache is marked as Modified, it is evicted as a write-

Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)
1063-6897/05 $20.00 © 2005 IEEE

back at a later point. In TPC-C, since a majority of the data is not used, we note that the write-back traffic has increased relative to the base case without a noticeable change in data read misses. It is thus important to apply DCA selectively to different types of traffic.

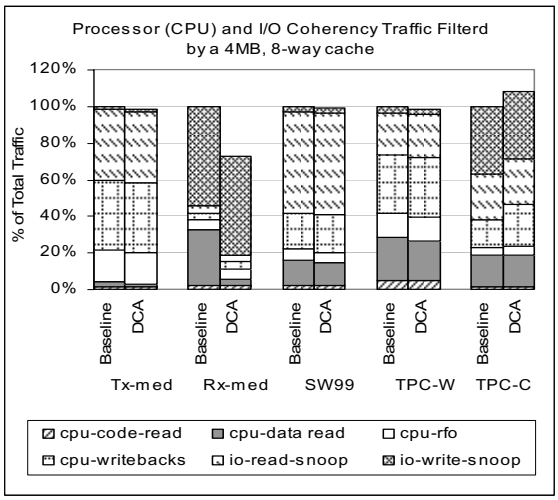


Figure 6 Traffic Profile with a 4MB cache

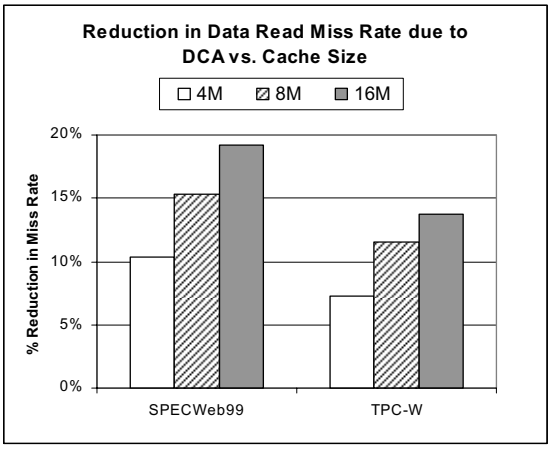


Figure 7. DCA Impact on Data Read Miss Rates

Among the constituents of each profile, we expect that processor data reads and write-backs are the most directly impacted due to DCA. When a data structure is expected to be modified by an I/O agent, typical driver programs ensure that, once modified, the data is read and interpreted before being re-used (or read for ownership). Processor data reads are not only an important bandwidth component but it is the second significant source of memory latency stalls apart from code reads. Hence in Figure 7 we show processor data read miss rates with and without DCA. In the case of SPECWeb99 and TPC-W, the reduction in data read misses is 10% and 7% respectively for a 4MB cache.

Figure 7 also shows the impact of DCA for 8MB and 16MB caches. Larger caches present two opportunities for DCA: a) the proportion of data read misses due to I/O writes increases since it does not depend on the cache size and b) DCA will have a relatively smaller effect on pollution given that application working set sizes do not generally scale proportional to the cache size. When the cache size is increased from 4MB to 16MB, a 19% reduction in data read miss rate is observed for SPECWeb99 and a 14% reduction for TPC-W.

5. System Considerations

High variations in workload characteristics imply that a flexible model for DCA must be adopted for implementation. The scope of DCA should generally be limited to workloads that exhibit a strong producer-consumer relationship. A high percentage of data from the source must used within a short temporal distance between arrival of data from the source and processor usage. When attempting to push data into the cache, if a new cache line must be allocated (upon a miss), it can potentially replace existing cache lines and also cause evictions if the lines are dirty. The relative working set size of data structures also plays an important role when enabling DCA. The data structures for which a producer-consumer relationship exists may be large compared to working sets of other processes that are using caches. In I/O device to processor shared memory communication, control structures such as descriptors often have a much smaller working set than the data buffers that they control. While these descriptors are accessed frequently by the processor, their impact on other working sets in the cache can be expected to be minimal. Finally, there must be robust support to provide specific knowledge of which processor in multiprocessor (MP) systems will first process the data.

In this section, we examine the following major architectural options: selection of cache in hierarchy, replacement policy and end line state, system interconnect protocol, selection of write stream and target processor determination. In order to adapt to various types of workloads, DCA implementation should emphasize a flexible strategy that permits fine grained selection of the data that must be sent directly to cache. At the source of the data (the producer), it is important to distinguish streams based on the above characteristics and provide mechanisms that tag these streams for DCA. The streams must also be deterministically associated with processors where they will be processed.

5.1. Cache Selection within a Hierarchy

Analysis of workloads with network I/O content such as NTTTCP, SPECweb99 or TPC-W showed that there is a net gain from using DCA with a 4MB, 8-way cache.

Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)
1063-6897/05 $20.00 © 2005 **IEEE**

The smaller the cache, the smaller the I/O related component of the total cache misses and hence lower the benefit due to DCA. We also showed that the distance between I/O writes to memory and processor reads is reasonably short. The distance is fundamentally determined by when the processor is notified regarding the arrival of new data using an event such as an interrupt. In typical, gigabit network controller implementations, interrupts are moderated to reduce processing overhead by collecting several packets and interrupting once for all the packets. The number of packets accumulated provides a direct indication of the amount of cache occupied in anticipation of processor usage.

Table 4. Cache Occupancy due to Batching of Packets (10 Gb/s, 4000 interrupts/sec)

Pkt Size (Bytes)	*Pkt Rate (MPls)*	*Cache Lines per Pkt*	*Pkts per Int*	*with Data (MB)*	*w/o Data. (MB)*
64	14.9	3	3720	0.28	0.26
256	4.5	6	1132	0.29	0.08
512	2.4	10	587	0.30	0.04
1024	1.2	18	299	0.30	0.02
1518	0.8	25	203	0.30	0.01

In Table 4, we assume continuous arrival of packets (shown at different sizes) at a data rate corresponding to 10 Gb/s. We assume that a typical rate of 4000 interrupts per second is used in order to amortize interrupt processing overheads. The number of packets accumulated (including packet header, descriptor and payload) before the processor is interrupted corresponds to a total of 0.28 to 0.30 MB or 7.1 to 7.4% of a 4MB cache across the selected packet sizes. Note that the quantity well exceeds a 256K cache and is more than half of a 512K cache. In an environment where other non-streaming working sets are sharing the same cache, the last level cache of a typical server cache hierarchy will be strongly preferred. Unless the cache belongs to a dedicated processing core for network I/O, a smaller cache in the range of 256K to 512K cannot be recommended for 10 Gb/s rates.

Just as network I/O and disk I/O were observed to have different characteristics, data types within network I/O may also be different. In typical TCP/IP processing, a minimum of one copy operation is incurred where payload from NIC's buffers is copied to the application buffer. If the copy operation is eliminated by swapping pages or offloading to a copy engine, then using DCA for payload is unnecessary. Consequently, capability to turn DCA on or off dynamically by the NIC will be important.

5.2. Cache Replacement Policy

The benefits of DCA described so far assume the commonly used pseudo-LRU policy for replacing lines in processor caches. However, in this context, since I/O data is not reused before being invalidated by new data, we are investigating mechanisms to limit allocation of ways to this type of data. Consider a hypothetical scenario where TPC-C is executed with storage across the network (IP storage). In this case the traditional database code and data working sets must contend with the streaming network I/O data that is brought into caches for TCP/IP and related protocol processing. Figure 8 shows data read miss rate for TPC-C for four cases: a) No IO Stream: all interference from I/O is ignored to obtain a hypothetical best case b) IO Stream Base: I/O stream is turned on and allowed to interfere with the TPC-C memory accesses c) I/O Stream with DCA-LRU: Using DCA for the I/O stream and d) I/O Stream with DCA-W1: limiting I/O writes to a single way while using DCA. The number 10000 is a simulated, fixed spacing between I/O writes and processor reads for the I/O stream. Preliminary simulations indicate that when cache space is a constraint, DCA may incur I/O interference effects in cache by bringing data in too early. This negative effect can be managed by limiting I/O-related allocations to one cache way.

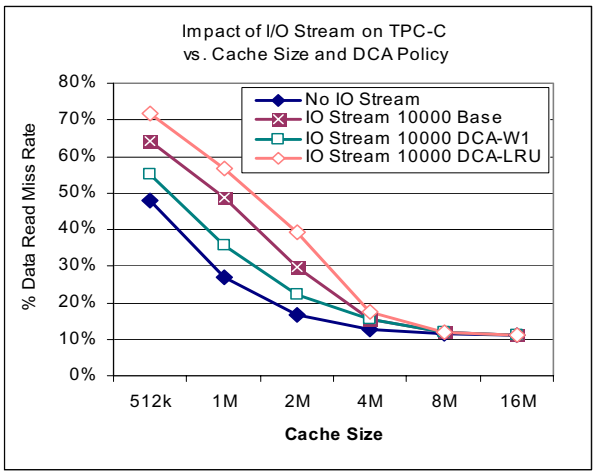


Figure 8. Impact of Limiting I/O Stream Allocation to a Single Way (8-way caches assumed)

5.3. System Interconnect Protocol

Simulation data presented so far assumed that 'Write-Allocate' transactions are used to replace other existing cache lines within the target cache. A 'Write-Update' protocol offers a good trade-off when the processor is accessing control structures or buffers that are frequently recycled. If a data structure due to some form of temporal locality remains in cache, then a Write Update protocol can largely succeed. Figure 9 summarizes a comparison between the Write-Allocate and Write-Update for the three server benchmarks. For the same working set (as

determined by the selected traces), as cache size increases, the greater the opportunity to update directly in cache.

Table 5. State Transition in DCA Protocols

	Initial State	Final State
Update	I	I
	M, E or S	M or E
Allocate	M, E, S or I	M or E

Write-Update also does not incur any cache pollution regardless of the amount of data actually touched by the processor. This is why TPC-C does not suffer an increase in data read misses as in the case of Write-Allocate.

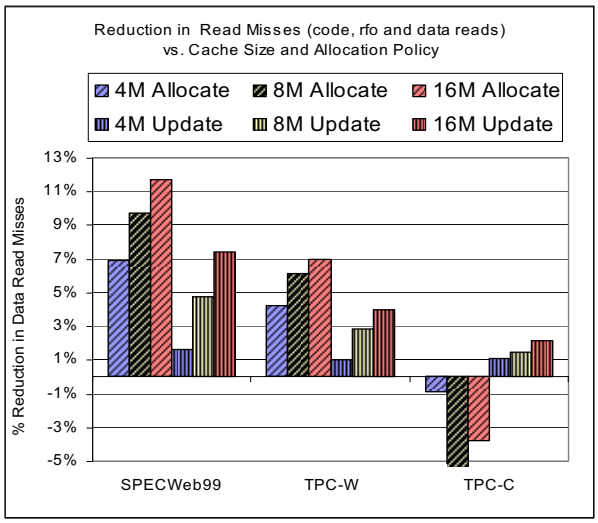


Figure 9. Impact of DCA on Data Read Misses

DCA may be viewed as an I/O adapter initiated prefetching mechanism. In the idealized implementation of DCA, data is directly brought into cache without touching memory. However, the act of bringing data into cache ahead of processor's demand requests is critical in decoupling the processor from memory. Recognizing this key characteristic enables us to propose a significantly simpler mechanism called Prefetch Hint. In one version of this proposal, the snoop associated with the I/O write is tagged to trigger a hardware prefetch within the processor. A hardware prefetcher can use the snoop address to issue a prefetch request to memory. This approach has the benefit of mapping to existing micro-architectural implementations that do not expect unsolicited data from external sources. The hint may also be decoupled from snoops for greater flexibility in triggering prefetches.

5.4. Identify the Target Processor

Since implementation of DCA can intricately depend on the architecture of the base processor and platform, we provide a reference model identifying the basic components involved in the correct yet flexible operation of DCA. In this model (Figure 10), we recognize a Write Agent as the source of new data. The Write Agent is responsible for issuing DCA transactions for routing in the platform. A Requesting Agent is the processor running system software that sets up the Write Agent with DCA preferences. DCA preferences primarily include the choice of enabling/disabling DCA and the target identifier. A Target Agent is the target CPU and cache that receives data from the Write Agent. System software (O/S or Virtual Machine Monitor) must provide a reliable environment for sending data from Write Agents directly into the cache of the processor that will first touch the data. When data is sent into a processor, the task that will use the data must be running or should run within a short interval on that processor. In some usage models such as network protocol processing, we have shown workload characteristics indicating that the interval between cache placement and the use of the same data by the processor is relatively short. Thus, with the aid of the system software, it is only required to ensure that data is sent to the correct Target Agent in multi-cache systems.

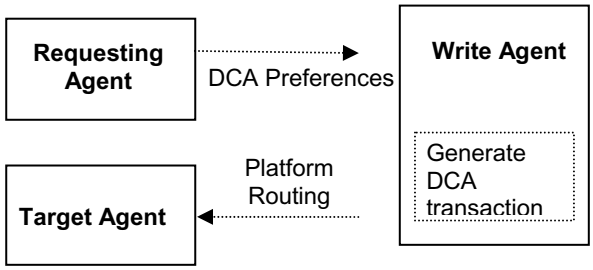


Figure 10. A Reference Model for DCA

Two affinitization methods that are expected to become prevalent in operating systems can be leveraged for DCA: 1) NUMA memory affinity and 2) connection based affinity. Both Microsoft and Linux support NUMA-aware memory allocation schemes so that the memory requested by a task is allocated from the memory that is local to the processor that it is running on. If the requested memory space is not available, then it is allocated from the remote processor's memory. For a NUMA memory optimized driver, we expect that a large majority of memory write transfers from I/O devices can be routed to the cache of the processor that owns the memory space indicated in the DMA write addresses. Since there is no guarantee that a packet will be processed on a processor that owns the memory containing the packet buffer, NUMA memory based affinity provides a coarse guidance regarding where the data will by processed. Connection based affinity [16] involves the balancing of packet processing across multiple processors in the system while maintaining in-order delivery of the data. In addition, cache thrashing can be minimized by ensuring that a single TCP connection is always processed by one processor. As

1063-6897/05 $20.00 © 2005 **IEEE**

packets arrive from the network, NICs can be enabled to examine packet headers and assign processors to these packets in a deterministic manner. The availability of the target processor at the NIC directly helps DCA implementation. The target processor is used as part of DCA preferences passed into the chipset via the I/O interconnect.

5.5. Summary of Architectural Guidelines

We reviewed a number of system-wide issues that must be addressed for proper DCA implementation. A brief summary of the guidelines that were developed is provided in Table 6. It is important to interpret these guidelines considering that the focus of our research was on network intensive workloads and server processors / platforms.

Table 6 Summary of DCA System Considerations

Cache in Hierarchy	Limit to Last-level cache except when the cache is not used for other purposes.
Replacement Policy	Normal LRU-like policy is sufficient. Way limits are worth investigating
Cache Line State after DCA	Exclusive sufficient in the network I/O usage model.
System Interconnect Protocol	Write-Allocate provides the highest opportunity. Ability to accommodate Write-Update in same design can be useful. Prefetch hint approach offers a simple, highly flexible alternative
Write Stream	Dynamic, per transaction selectivity
Target CPU Determination	Dynamic, per transaction selectivity supporting a scheme such as RSS

6. Conclusions

By alleviating the I/O interconnect bottleneck in low cost, high volume platforms, using PCI-Express and gigabit Ethernet technologies, we anticipate that the processor and in particular processor to memory interactions can severely limit the attainment of high network I/O rates. In this paper, we have reviewed scenarios such as TCP/IP on Ethernet where a strong relationship may exist between I/O traffic and processor efficiency. The result of extensive characterization of memory access traces is an enhancement called Direct Cache Access. We note that due to different type of I/O characteristics, system level implementation must be based on a reference model that emphasizes flexibility. Higher rates of I/O, larger caches and efficiency improvements in other parts of network protocol processing will continue to increase the importance of DCA and similar I/O centric enhancements.

Further characterization of I/O and processor relationship is ongoing. Coherency protocol inefficiencies in the outbound direction include the eviction of modified data from cache to memory upon an I/O read. The efficient use of cache in the presence of heterogeneous workloads is under investigation. Apart from memory dependencies, I/O related computation will continue to receive attention due to additional layers of processing involving data integrity, security and flexibility offered by XML-like representations.

REFERENCES

[1] J. Chase et. al., "End System Optimizations for High-Speed TCP", IEEE Communications, June 2000.

[2] D. Clark et. al., "An analysis of TCP Processing overhead", IEEE Communications, June 1989.

[3] D. Clark et. al., "Architectural Considerations for a new generation of Protocols", ACM SIGCOMM, September 1990.

[4] M. Dubois et al, "Evaluation of Shared Cache Architectures for TPC-H", 5th Workshop on Comp. Architecture Evaluation using Commercial Workloads (CAECW-2002).

[5] A. Earls, "TCP Offload Engines Finally Arrive", Storage Magazine, March 2002.

[6] A. Foong et al., "TCP Performance Analysis Re-visited," IEEE Int'l Symp on Performance Analysis of Software and Systems, Mar 2003

[7] iSCSI, IP Storage Working Group, Internet Draft, available at http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt

[8] R. Iyer, "On Modeling and Analyzing Cache Hierarchies using CASPER", 11th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), April 2003.

[9] S. Makineni and R. Iyer, "Performance Characterization of Packet Processing in Commercial Server Workloads," 6th Workshop on Workload Characterization (WWC), Oct 2003.

[10] J. Mogul, "TCP offload is a dumb idea whose time has come," Symposium on Hot Operating Systems (HOT OS), 2003.

[11] E. Nahum, D. Yates , J. Kurose and D. Towsley, "Cache behavior of network protocols, ACM SIGMETRICS Performance Evaluation Review, v.25 p.169-180, June 1997

[12] J. B. Postel, "Transmission Control Protocol", RFC 793, Information Sciences Institute, Sept. 1981.

[13] RDMA Consortium. http://www.rdmaconsortium.org

[14] G. Regnier et al, "TCP Onloading For Data center Servers", IEEE Computer. November 2004.

[15] Remote Direct Data Placement Working Group. http://www.ietf.org/html.charters/rddp-charter.html

[16] "Scalable Networking: Eliminating the Receive Processing Bottleneck – Introducing RSS". Microsoft WinHEC April 2004.

[17] "SPECweb99 Design Document", available online at http://www.specbench.org/osg/web99/docs/whitepaper.html

[18] "TPC-W Design Document", www.tpc.org/tpcw/

[19] "TPC-C Design Document", www.tpc.org/tpcc/

[20] "TTTCP Benchmark", http://ftp.arl.mil/~mike/ttcp.html

[21] D. Yates., "Connection-Level Parallelism for Network Protocols on Shared-Memory Multiprocessor Servers," Ph.D. Dissertation, University of Massachusetts, Amherst, 1997.

1063-6897/05 $20.00 © 2005 **IEEE**