Platform IO DMA Transaction Acceleration

Steen Larsen
Intel Labs
2111 NE 25th Ave
Hillsboro OR 97124
steen.larsen@intel.com

Ben Lee
School of Electrical Engineering and Computer Science
Oregon State University
3013 Kelley Engineering Center
Corvallis, OR 97331
benl@eecs.oregonstate.edu

ABSTRACT
Computer system IO (with accelerator, network, storage, graphics components) has been optimized to use descriptor-based direct memory access (DMA) operations to move data to and from relatively fast addressable system (or main) memory or cache structures. Traditionally, transactions between slower IO sub-systems and system memory have been done using a host bus/bridge adapter (HBA). Each IO interface has one or more separately instantiated descriptor-based DMA engines optimized for a given IO port. As heterogeneous cores multiply in exascale systems, IO traffic can be expected to be more complex and will require more resources. This paper measures the descriptor overhead and analyzes its impact on latency and bandwidth. Based on quantifications of the latency and bandwidth overhead, we propose to improve IO performance using an integrated platform IO accelerator. This IO engine localizes IO transactions to the processor CPU, rather than offloading to various remote platform interface controllers. By simplifying hardware control of IO in complex systems that rely on a central system memory, we conclude there are quantifiable benefits of integrated platform IO transactions in terms of bandwidth-per-pin and latency, and other areas.

Categories and Subject Descriptors
B.4.1 [**Input/Output and Data Communications**]: Data Communications Devices – *processors, channels and controllers, memory.*

General Terms
Performance, Design, Measurement

Keywords
IO latency, memory, DMA, IO bandwidth

1. INTRODUCTION
With IO becoming a peer to processor core (or simply *core*) and memory in terms of bandwidth availability and power requirement, it is important to consider alternatives to existing methods of moving data within a platform. Historically, when a core was simpler and more directly user-focused, it was acceptable to "bit-bang" IO port operations using models such as the Soft-modem [1], where core instructions directly controlled the modem. However, with complex user interfaces and programs using multiple processes, the benefit of offloading data movement to an IO adapter became more apparent. In the case where IO speeds are moderate, it makes sense to move data at a pace governed by the external device, which is much slower than core/memory bandwidth. Data transfer is initiated using a descriptor containing the physical address and size of the data to be moved. This descriptor is then posted (i.e., sent) to the IO adapter, which then processes the direct memory access (DMA) read/write operations as fast as the core/memory bandwidth allows.

The descriptor-based DMA approach makes sense when IO bandwidth requirements are much lower than the core/memory bandwidth. However with increasing use of heterogeneous accelerators, such as PCIe graphics engines, their proximity to system memory leads to a memory latency and a throughput bottleneck. This is coupled with multi-core processors and integrated memory controllers driving higher core performance and memory bandwidth. As a result, IO bandwidth increase is slower than the core/memory bandwidth. Figure 1 shows this trend on typical server platforms [2].

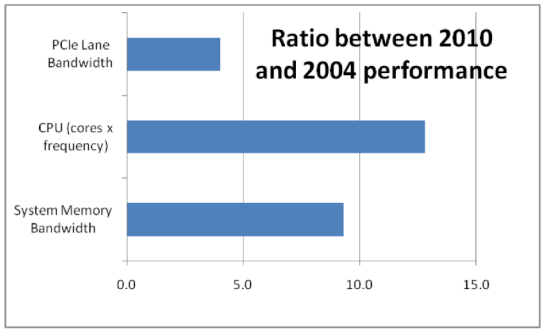

Figure 1. Relative performance increases over time

This paper analyzes the performance impact of the current descriptor-based DMA method and proposes an alternative that integrates DMAs closer to the core/memory complex with a standard/universal interface for communication with memory buffers. The proposed *integrated DMA* (iDMA) removes the overhead of setting up and managing descriptors over a chip-to-chip interface, at a cost of a small incremental core package complexity, and improves bandwidth-per-pin, latency, power, Quality-of-Service (QoS), security, and cost (in terms of silicon die area).

The expected benefits are:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CACHES '11, June 4th, 2011, Tucson, Arizona, USA Copyright © 2011 ACM 978-1-4503-0760-4/11/06 $10.00

- Reduced latency since there is no descriptor traffic across a chip-to-chip interface such as PCIe or system memory.
- Increased bandwidth-per-pin due to absence of descriptor-based traffic. This also implies more efficient and un-fragmented packet transfers to and from addressable memory.
- Other benefits that are less quantifiable are also explored.

In a sense, iDMA can be seen as re-visiting the Soft-modem implementation where the core plays a more active role in moving data within a platform. We expect the research discussed herein to clearly define and demonstrate the benefits of this approach.

2. BACKGROUND

Today's IO-related devices, such as Network Interface Controllers (NIC) (both wired and wireless), disk drives (solid state and magnetic platter), and USB, have traditionally been orders of magnitude lower in bandwidth than the core-memory complex. For example, a modern 64-bit core running at 3.6 GHz compared to a 1.5 Mbps USB1.1 mouse has 153,600 (64×3600MHz/bit/1.5MHz/bit) times higher bandwidth. Multi-cores and Simultaneous MultiThreading (SMT) make this ratio even higher. Therefore, it makes sense to offload the cores by allowing the IO adapters some control over how input/output data is pushed/pulled to/from memory. This allows a core to switch to other tasks while the slower IO adapters operate as fast as they are capable.

GPU accelerators, when used as graphics output devices have been single direction data output. Although graphics require high bandwidth, the output-only characteristic of graphic displays allows for loosely ordered data transactions such as write-combining memory buffers. With more generic accelerators tasks such as sorting on Intel's Many Integrated Core (MIC) architecture [3], data access to system memory is done with descriptors.

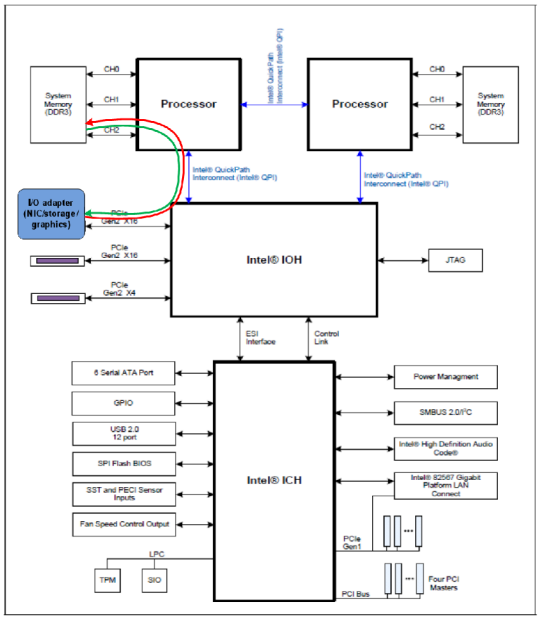

Figure 2. Current DMA method for IO adapters

Figure 2 shows how IO adapters can bypass core interaction (other than snoop transactions to maintain cache coherency) and directly access system memory. The system memory is defined as the entire coherent memory complex, including multi-core caches as well as system DRAM. The diagram is based on the current 5520 Intel server chipset [4] and is similar for AMD interconnects as well. An IO Hub (IOH) is used to connect directly to the CPU sockets over a coherent memory interface using a QPI protocol that supports the MESI(F) coherency states for 64 Byte cache-line units. High-performance IO adapters connect directly to the IOH, while less critical IO adapters and protocol specific interfaces (SATA, USB, 1GbE, etc) connect using an IO Controller Hub (ICH). The ICH communicates to the IOH using an ESI interface that is similar to PCIe, allowing direct media access to BIOS boot flash. A typical Non-Uniform Memory Architecture (NUMA) maintains coherency either through directory tables or other means outside the scope of this discussion. Arbitrary processor scaling can be realized as Intel QPI links scale out, which is analogous to the AMD Hyper-Transport [5] coherent memory interconnect. This 5520 chipset is the platform on which data for our analysis has been collected. Intel's current MIC accelerator implementation is a PCIe ×16 Gen2 interface providing 8GB/s bandwidth, but Section 2.1 demonstrates how any descriptor-based memory coherency would not be a viable consideration since every data snoop would require a descriptor to define the memory location and snoop results.

An optimization to DMA is *Direct Cache Access* (DCA), where an IO device can write to a processor cache by either directly placing data in a cache level or hinting to a pre-fetcher to pull the data from system memory to a higher level. In general, once data is written into system memory DRAM or processor cache, the data is in coherent memory space and the underlying coherency protocol ensures that reads and writes to the memory address space is handled properly. However, as mentioned by Leon *et al.* [6], there can be a pollution aspect to a cache hierarchy with DCA. This will trigger cache evictions and higher memory bandwidth utilization as asynchronous IO transactions occur at different times (also described as reuse distance [7]) relative to CPU utilization. Cache pre-fetchers are also good at hiding the memory page misses of large receive buffers.

2.1 IO flow using descriptors

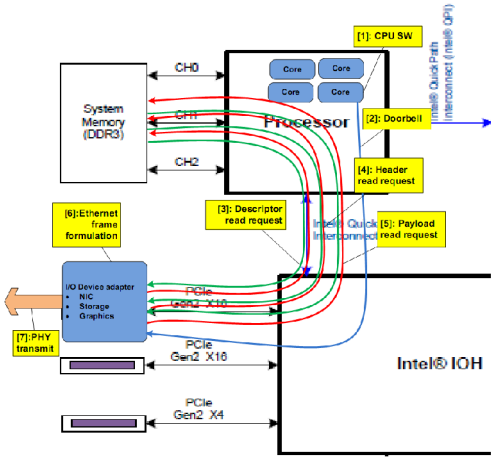

Figure 3. Typical Ethernet Transmit

Figure 3 illustrates a typical IO transmission for an Ethernet NIC (either wired or wireless). Similar descriptor-based transactions

are found on storage and graphics adapters. The following sequence of operations occurs for transmitting an Ethernet message or packet assuming a connection between two systems has already been established (i.e., kernel sockets have been established and opened):

(1) The kernel software constructs the outgoing message or packet in memory. This is required to support the protocol stack, such as TCP/IP, with proper headers, sequence numbers, checksums, etc.

(2) The core sends a write request on the platform interconnect (e.g., PCIe, but also applies to any chip-to-chip interconnect within a platform) to the NIC indicating that there is a pending packet transmission. Since this is a write operation to the memory space reserved by the IO adapter, the write is uncacheable with implications that other related tasks that are potentially executing out-of-order must be serialized until the un-cacheable write completes. The core then assumes the packet will be transmitted, but will not release the memory buffers until confirmed by the NIC.

(3) The NIC state machine is triggered by this doorbell request, which initiates a DMA request to read the descriptor containing the physical address of the transmit payload. The descriptor is not encapsulated in the doorbell write request because there are two descriptors (frame and payload) in an Ethernet packet definition, and a larger network message will require more descriptors (maximum payload for Ethernet is 1460 bytes).

(4) After the memory read request for the descriptor(s) returns with the physical addresses of the header and payload, the NIC initiates a request for the header information (i.e., IP addresses and the sequence number) of the packet.

In the case of an Ethernet descriptor, its size is 16 bytes and it is possible that multiple descriptors can be read from a single doorbell, (i.e., if the cache line size is 64B, 4 descriptors would be read at once). This ameliorates the overall transmit latency and PCIe and memory bandwidth utilization. This approach does not address the fundamental problem where a server may have thousands of connections over multiple descriptor queues that do not allow for descriptor bundling. Transmit descriptor aggregation is beneficial if large multi-framed messages are being processed. For example, a datacenter (where multiple systems are physically co-located) may simply choose to enable jumbo-frames, which would renew the transmit descriptor serialization describe above.

(5) The descriptor also defines the payload memory location, so with almost no additional latency other than the NIC state machine, a request is made to read the transmit payload.

(6) After the payload data returns from the system memory, the NIC state machine constructs an Ethernet frame sequence with the correct ordering for the bit-stream.

(7) Finally, the bit-stream is passed to a PHY (PHYsical layer), which properly conditions the signaling for transmission over the medium (copper, fiber, or radio).

The typical Ethernet receive flow is the reverse of the transmit flow. After the core prepares a descriptor, the NIC performs a DMA operation to transfer the received packet into the system main memory. Upon posting the transfer, the NIC can then interrupt the processor and update the descriptor.

The DMA sequence described above is typical of general IO adapters and makes sense when the IO is significantly slower than the cores. When the IO bandwidths are lower than the processor-system-memory interconnect, there is little conflict for available memory bandwidth.

3. The Proposed Method

The proposed method consists of an integrated DMA (iDMA) engine in the multi-core processor silicon package shown in Figure 5. The iDMA would be implemented as a simple micro-controller or enhanced state machine that manage the data flow between an arbitrary IO device and system memory. It would basically act as a heterogeneous or SoC core to the larger application generic cores. The red arrow shows the iDMA pulling receive traffic from an IO adapter into system memory. The green arrow shows the iDMA reading memory and pushing transmit traffic to the IO adapter. Since IO messages and cache-line memory accesses occur at different times and sizes, basic queue buffers are needed to support arbitration and fragmentation of off-chip interfaces.

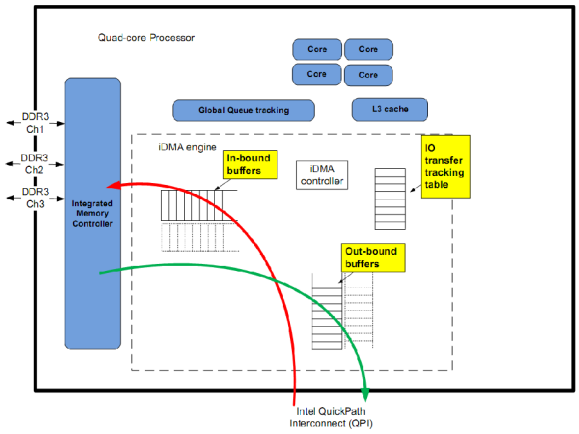

Figure 4. iDMA internal blocks

A driving factor for the iDMA engine proposal is the simple observation that inter-chip communication is complex. The life of an IO transmit activity will make about 20 inter-chip hops based on the typical descriptor-based DMA transfer in figure 3. Not only does this impact latency, but there is frame fragmentation to consider. The memory and QPI transfer sizes are cache-line sized (64 bytes) while the PCIe transactions are usually a maximum of 256 bytes and often bear no relevance to the higher level IO transaction for networking, storage, etc. Additionally off-chip transceivers consume significant energy when active to drive signals at high speed and properly frame the inter-chip communication. PCIe is particularly protocol heavy with a Transaction Layer Protocol (TLP) encapsulated in a Data Layer Protocol (DLP) within a Link Layer Protocol (LLP).

IO offloading engines such as TCP Offload Engine (TOE), Infiniband and iWARP take the approach of supporting an IO connection context at the edge of the platform at the interface controller. This allows low latency in the task of frame formatting of header information that would normally be supported with a generic core. To maintain this connection context, side-RAM and controllers

are needed on the IO adapter increasing cost and power consumption. Rather than repeating these customized offload engines for each IO type and instance in a system, we suggest to place a generic IO controller as close as possible to the cores processing the IO traffic

To support multiple types of IO (storage, networking, graphics, etc) a state machine may need to be replaced with a small microcontroller. Although the main function of the iDMA engine is to move data between memory and the IO interface, supporting multiple priorities and QoS is needed to prevent a large storage transaction from blocking a latency sensitive network operation. An Intel Atom core or ARM core would be appropriate to manage and track the iDMA engine queues over the various types of IO interfaces in the platform.

The size of these queue buffers depends on several factors, but is based on the largest IO size required and the "drain-rate" supported by the memory interface and the IO interface. Since the possibility of IO contention within the iDMA is highly platform dependent and increases as the number of IO devices increases, we do not make any specific claims on the buffers sizes at this time. NUMA architectures tie memory closely to related cores, so an optimal environment would have an iDMA per multi-core and memory pair. This silicon expense could be mitigated by having IO-oriented multi-core processors (each with iDMA) and less IO oriented multi-core processors (without iDMA) in the same NUMA platform.

For the IO adapter, we assume there are small transmit and receive queue buffers. This could be as small as two frames (e.g., 1,500 bytes for Ethernet and 8KB for SCSI) allowing a frame to transmit out of the platform or into the iDMA buffers while the second frame is filled either by receive PHY or the iDMA transmit. For the receive flow described in Section 3.2, a small DMA engine is suggested. This would lead to a much smaller IO adapter implementation than current IO adapters (e.g. 25mm×25mm silicon dissipating 5.1W for a dual 10GbE [8]).

There are several reasons that suggest system architecture should move away from current discrete DMA IO adapter engines and become a more centralized data mover engine. First, this leads to simpler IO adapters, which can be viewed as merely a FIFO and PHY since there is no need for a DMA engine with descriptor related logic in each IO adapter within the platform. There are also other advantages including latency, bandwidth-per-pin, QoS, power, security, silicon complexity, and system cost that are quantified in Section 4.

The following two subsections present transmit and receive flows to illustrate the advantages of iDMA.

3.1 iDMA Transmit Flow

Figure 6 illustrates the transmit sequence for iDMA, which consists of the following steps:

(1) The kernel software in a core constructs the transmit data structure (i.e., payload and header information) similar to the legacy method described in Section 2.

(2) The core writes a doorbell to the iDMA engine. The format of this doorbell is exactly the same as the legacy doorbell, but it does not traverse the inter-processor or PCIe interface.

(3) The doorbell triggers the iDMA engine to request the descriptor from the system memory. As in the previous step, since the iDMA engine is on the same silicon or package as the multi-core processor, latency is significantly reduced and no chip-to-chip bandwidth is utilized as compared to the descriptor fetch over PCIe.

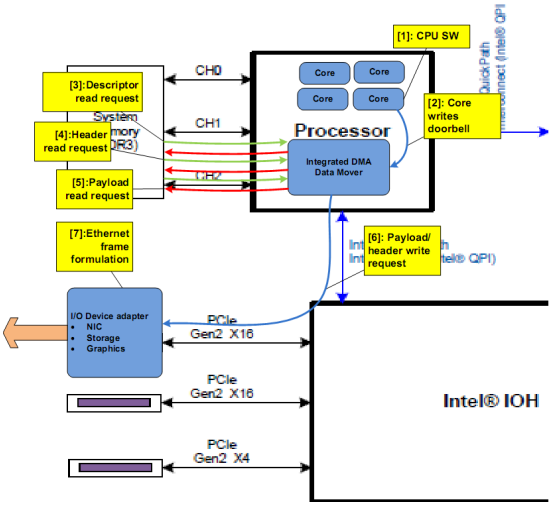

Figure 5. iDMA transmit flow

(4) Based on the descriptor content, a read request is made by the iDMA for the header information. Note that the descriptor fetch and contents do not traversing the PCIe interface impacting latency and bandwidth-per-pin quantified in Section 4.

(5) A read request is made for the payload data with almost no additional latency.

(6) The iDMA sends a write request to the IO adapter. This is the only transmit payload related task on the PCIe interface, but relies on the underlying PCIe Data Layer Protocol (DLP) and Link Layer Protocol (LLP) for flow control credit and link integrity.

(7) The simplified IO adapter constructs the transmit packet and the PHY performs DSP signal conditioning for optical, copper or wireless media.

3.2 iDMA Receive Flow

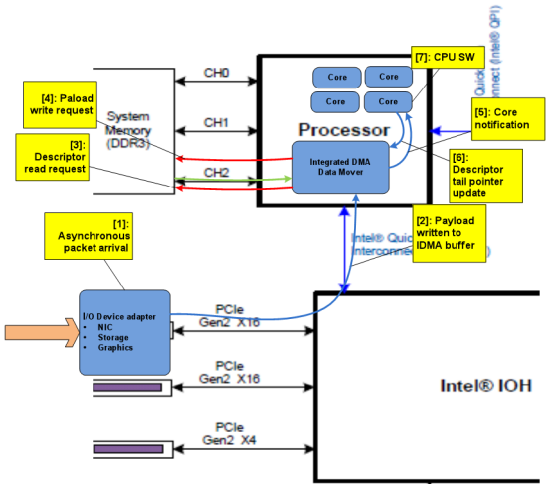

Figure 6. iDMA receive flow

Figure 6 shows the proposed iDMA receive sequence consisting of the following steps:

(1) The PHY asynchronously receives the packet and some basic link level processing is done to verify that the packet is for the intended platform, such as MAC address filtering. The receive packet will possibly target a particular core, such as current RSS mapping of multiple connections to multiple cores.

(2) The iDMA engine could poll the IO adapter for receive traffic, but since memory bandwidth usually greatly exceeds a particular IO interface bandwidth, having a small buffer in the iDMA engine for receive traffic is suggested. Another argument is that the IOH currently has a write cache allowing buffering of in-bound IO related traffic as it enters memory coherency. The clear benefit is that as soon as the IO adapter receives a frame, it can be pushed into the iDMA receive buffer.

This task does need a simple DMA engine in the IO adapter to cause a PCIe write transaction, but does not need descriptors if the receive iDMA memory buffer is pinned to a particular location in physical memory. This physical memory address indicating the address to the available IO adapters can be initialized on kernel software boot. The iDMA receive control needs to properly manage and increment the receive queue as new receive messages arrive. Since the IOH write cache depth is 128 cache lines [4], an initial size of the iDMA receive buffer should be 128×64 = 8KB. This is also referred to as the *push-push* (or hot-potato) model.

(3) This triggers the iDMA engine to fetch the descriptor as in the legacy case. The descriptor defines how the iDMA should process the received packet and can be pre-fetched. Multiple receive descriptors can be fetched to optimize memory bandwidth utilization since 4 NIC descriptors can fit in a 64-Byte cache line. Again, it should be noted the descriptor fetch does not cross the PCIe interface, but only the memory interface.

(4) The iDMA engine processes the headers and writes to the system memory. This step does not preclude writing directly to a processor's cache. The simplest direct cache writing would be in the L3 cache that is shared by all cores in a multi-core processor on the socket at the cost of adding another write port to the physical cache structure. If the receive traffic can be targeted to a specific core, the iDMA could write directly into the L1 or L2 of the target core.

(5) The iDMA engine notifies a core based on the traffic classification and target core information found in the receive packet. This mapping is done effectively by a hash function, such as Toeplitz found in receive side scaling (RSS) [9]. If none is specified, the iDMA engine can be configured to treat the packet as low priority and notify for example core0 or some preconfigured core.

It can be seen that descriptors still exist in the receive flow on the memory interface. There are three reasons why the use of descriptors on the memory interface should be preserved.

- There would be a significant software change requiring the device driver to handle the control of the iDMA data movement. This has been explored with Intel's IOAT DMA engine [10], which resides on a PCIe interface and provides benefits only on larger (> 4KB) IO sizes.
- The IO adapter remains a simple FIFO + PHY structure. Such a simple topology saves silicon area and cost over the entire platform.
- Without descriptors, an application would need to manage the data movement directly at the cost of core cycles. It is unclear the cycle overhead would warrant this memory descriptor tradeoff.

The iDMA would need sufficient buffering to support the platform IO demands, but future work could show that buffer sharing for platform IO between networking, storage, and graphics could be an efficient use of CPU socket die area.

4. Experiments and Analysis

In order to test our hypothesis, a descriptor-DMA based network IO adapter is placed in an IOH slot of an existing platform based on the hardware configuration shown in Figure 2. This was to determine the potential impact of iDMA on an existing platform without actually building an entire system. By observing the PCIe transactions using a PCIe protocol analyzer, we could measure the impact descriptor-based IO adapter DMA engines had in terms of several factors. Two primary factors are discussed below and summarized in Table 1.

Table 1 Quantified benefits of iDMA

Factor	Measurement unit	Descriptor DMA	iDMA	Estimated Improvement	Comment/justification	Section
Latency	microseconds to transmit a TCP/IP message between two systems	8.8	7.38	16%	Descriptors are no longer latency critical	4.1
Bandwidth-per-pin	Gbps per serial lane link	2.1	2.5	17%	Descriptors no longer consume chip-to-chip bandwidth	4.2

4.1 Latency

Latency is a critical aspect in network communication that is easily masked by the impact of distance. However, when inter-platform flight-time of messages is small, as in a datacenter of co-located systems, the impact of latency within a system is much more important. An example is seen with automated stock market transactions (arbitrage and speculation) as demonstrated by Xasax [11] claiming 30µs latency to the NASDAQ trading floor. A second example is high performance computing (HPC) nodes where LinPack benchmark (used to define the Top500 supercomputers) share partial calculations of linear algebra matrix results among nodes. A third example is a Google search request, which may impact tens of systems between the page request and response as the results, advertisements, and traffic details are calculated on multiple networked or virtualized systems, in effect multiplying any system related latency. Certainly, intercontinental distances and multiple switches/routers will add millisecond-scale latencies, but for the scope below, let us consider a typical datacenter.

Figure 7 shows the typical 10Gb Ethernet (GbE) latency between a sender (TX) and a receiver (RX) in a datacenter type of environment where the fiber length is on the order of 3 meters. These results are based on PCIe traces of current 10GbE Intel 82598 [12] NICs (code named Oplin) on PCIe ×8 Gen1 interfaces. The latency benchmark NetPIPE is used to correlate application laten-

cies to latencies measured on the PCIe interface for 64-byte messages. The 64-byte size was used since it is small enough to demonstrate the critical path latencies, but also large enough to demonstrate a minimal message size that may be cache-line aligned.

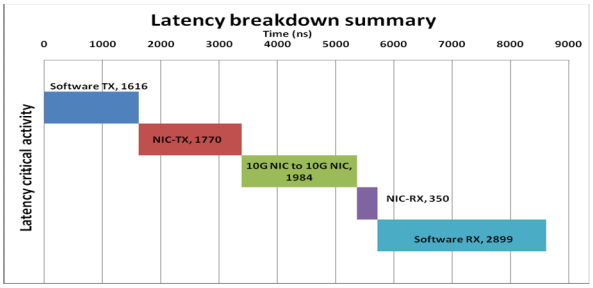

Figure 7. Critical path latency between two systems

End-to-end latency consists of both hardware and software delays, and depends on many aspects not directly addressed in this paper, such as core frequency, memory frequency and bandwidth, and cache structure. The software component is not related to iDMA since it relates to how the core processor core handles IO traffic only in terms of residency in main memory. Software latency is described in more detail by Larsen *et al.* [13]. The hardware latency can be split into three portions. First, the TX NIC performs DMA reads (NIC-TX) to pull the data from the system memory to the TX NIC buffer. This is followed by the flight latency of the wire/fiber and the TX/RX NIC state machines (NIC to NIC). Finally, the RX NIC performs DMA writes (NIC-RX) to push the data from the RX NIC buffer into the system memory and interrupts a processor for software processing. The total latency can be expressed by the following equation:

$$Total\ critical\ path\ latency = Tx_{SW} + Tx_{NIC} + fiber + Rx_{NIC} + Rx_{SW}$$

The latency for the NIC-TX portion can be further broken down as shown in Figure 8 using PCIe traces. A passive PCIe interposer was placed between the platform PCIe slot and the Intel 82598 NIC. PCIe traces were taken from an idle platform and network environment. These latencies are averaged over multiple samples and show some variance, but is under 3% min to max. The variance is generated by a variety of factors such as software timers and PCIe transaction management. Based on a current 5500 Intel processor platform with 1066MB/s Double Data Rate (DDR3) memory, the doorbell write takes 230 ns, the NIC descriptor fetch takes 759 ns, and the 64B payload DMA read takes 781 ns. The CPU frequency is not relevant since the DMA transactions are mastered by the PCIe NIC adapter.

If DMA transfers were "pushed" by the iDMA engine close to the core and memory instead of being "pulled" by the slower NIC device, the transmit latency would be significantly reduced. This is summarized in Figure 9 with each bar described below:

(1) A core writes to the iDMA to trigger the transmit flow as in a legacy PCIe doorbell request. Since this is on-die propagation, this latency is well within 20 ns.

(2) The descriptor fetch would not be needed to determine physical addresses of the header and payload. Instead, the iDMA running at (or close to) core speeds in the 2-3 GHz range essentially reduces the descriptor fetch to a memory fetch of about 50 ns for current Intel 5500 systems.

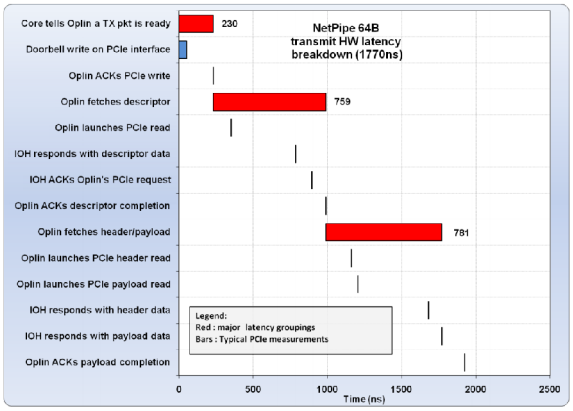

Figure 8. NIC TX Latency breakdown

(3) Similarly, this replaces the header/payload information fetch by the NIC adapter with a memory read request of about 50 ns.

(4) The iDMA transfers the header/payload information to the IO adapter, which is expected to be similar to the legacy doorbell latency of 230 ns.

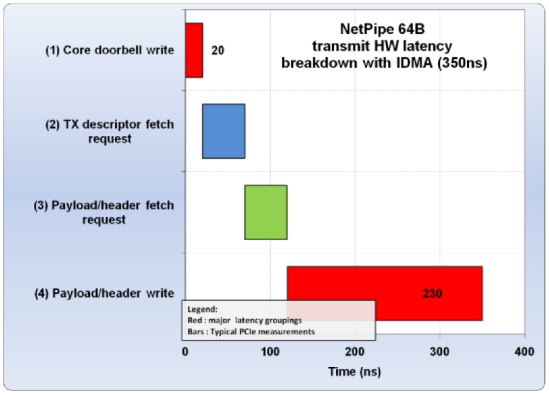

Figure 9. iDMA HW transmit breakdown

These improvements reduce the ideal NIC-TX transmit latency from 1,770 ns to 350 ns or 80%. While 80% latency reduction in moving a small transmit message to an IO adapter may be impressive, a more realistic benchmark is the end-to-end latency impact. The total end-to-end delay as seen by an application reduces from 8.8 µs to 7.38 µs, or 16% reduction in latency. 16% improvement by itself may not be impressive, but there are other advantages such as:

- Smaller bandwidth-delay product reduces buffering requirements at higher levels of hardware and software, e.g., 10Gbps × 8.8µs requires 11KB of buffering, while 10Gbps × 7.38 µs requires only 9KB. This bandwidth-delay product is per connection (or flow), so in the case of web servers the impact can be increased several thousand-fold.
- Less power consumed as the PCIe link is not as active and can transition to lower power states more quickly.

4.2 Throughput/Bandwidth efficiency

Figure 10 shows a breakdown of transaction utilization in receiving and transmitting data on a PCIe interface for a dual 10GbE

NIC. The iperf bandwidth benchmark was used on a dual 10GbE Intel 82599 [8] (codename Niantic) Ethernet adapter on an Intel 5500 server. The primary difference (in this context) to the earlier latency capture with the Intel 82598 is that PCIe bandwidth has doubled. The internal state-machine driven NIC and core frequency remains the same.

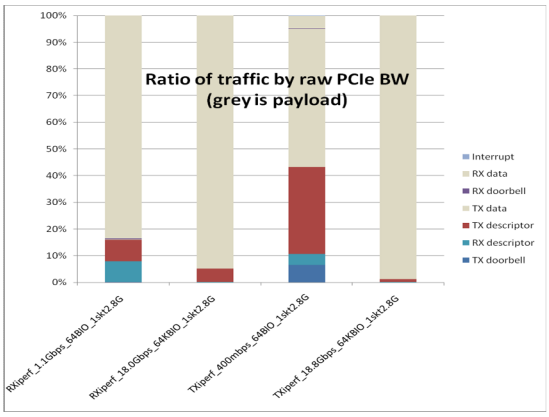

Figure 10. Raw PCIe link bandwidth utilization

PCIe captures consisted of more than 300,000 PCIe ×8 Gen2 packets, or 10 ms of real-time trace. This gives a statistically stable capture for analysis. The four bars in Figure 10 show extreme cases of TCP/IP receive traffic (`RXiperf_`) and transmit traffic (`TXiperf_`) for small (`_64BIO_`) and large (`_64KBIO_`) IO sizes. Receive traffic performance is important for such applications as backup and routing traffic, while transmit traffic performance is important in serving files and streaming video. Small IO is representative of latency sensitive transactions and large IO is representative of storage types of transactions. TCP/IP is also a bi-directional protocol where an ACK message is required to complete any TCP communication. Descriptors and doorbell transactions consume a significant amount of bandwidth. The ratio of payload related transactions to non-payload transactions show a 50% inefficiency of PCIe transactions for small IO messages. This includes PCIe packet header and CRC data along with PCIe packet fragmentation. The bits-on-the-wire is shown in Figure 10, which shows that transmitting small payload sizes utilizes 43% of the PCIe bandwidth for descriptor and doorbell traffic that would otherwise be removed with iDMA.

In the case of IO receive for small payload sizes, the benefit reduces to 16% since 16B descriptors can be pre-fetched in a 64B cache-line read request. For large message IO sizes, the available PCIe bandwidth is efficiently utilized with less than 5% of the bandwidth used for descriptors and doorbells.

In general, network traffic has bi-modal packet size with concentrations of small Ethernet frames and large Ethernet frames. Thus, we can average overall efficiency of TCP/IP traffic and see that without descriptors and associated interrupts and doorbell transactions on the PCIe interface, available bandwidth on the PCIe interface would increase by 17%.

There are other benefits beyond physical bandwidth-per-pin on the PCIe interface. The current 4-core Intel 5500 system arbitrates memory bandwidth between IO requests and multi-core processor requests. As a multi-core processor starts to contain more cores and SMT threads, memory bandwidth becomes more valuable. The memory subsystem is not capable of distinguishing priority of requests between a generic IO adapter and a core. For example, the current arbitration cannot distinguish between a lower latency IO priority (e.g. storage request) and a higher latency IO priority (e.g. network IO request). Instead, the bandwidth is allocated in a round-robin fashion that can be modified during system initialization. By having arbitration logic for memory bandwidth between cores and IO adapters in the processor socket, the level of conflicting priorities between cores and IO for system memory access is reduced with the iDMA.

4.3 Other benefits

Other benefits that were not quantified with measurements are described below:

- Reduced system software complexity in small accelerator cores, where the descriptor rings or queues need to be maintained on a per core basis.
- Arbitration between accelerator cores for DMA engine transfer resources becomes important as core count increases. The Intel MIC, with 32 cores can have up to 32 generators of DMA transaction requests. The Nvidia GTX 280[3], with 30 shared multiprocessors each with up to 32 separate threads of execution helps indicate how the arbitration of DMA transfers for proper service becomes important. Additionally, PCIe devices usually support only a certain number of outstanding transactions, so the overhead of descriptor processing can become significant.
- Increased system bandwidth scalability since IO interfaces are more efficient at moving data. Fewer PCIe lanes are required and smaller form-factors required for future designs.
- Power management control has better visibility of power state transitions to allow for better power reduction heuristics. Total system power is reduced by removing descriptor transactions on the IO adapter interface, not only from the point-of-view of a bits-on-the-wire, but also the associated framing overhead (e.g., TLP/DLP/LLP framing on PCIe packets [14]).
- Improved QoS since various traffic flows can be moderated and controlled more carefully by the multi-core processor.
- Reduced cost and silicon area since the complex segmentation and reassembly tasks of IO operations can be centralized into a single hardware implementation.
- Increased security since arbitrary devices can be prevented from directly accessing physical memory.

5. RELATED WORK

Sun Niagara2 NIC integration [15] showed a method of how descriptors can be removed from a PCIe interface, but this requires a complete integration of a fixed type of IO that is an inflexible approach to generic descriptor based IO. Schlansker *et al.* [16] and others proposed bringing IO adapters into a memory coherent domain. This adds complexity to support the often slower IO adapters response to coherency protocol. Intel has implemented QuickData Technology [17] focused on storage interfaces, but still needs descriptors on a PCIe interface to define the offloaded DMA transactions.

6. FUTURE WORK

We plan to explore further the memory and processor overhead of minimizing descriptor processing. Proper quantification of the core cycles and memory bandwidth is expected to further strengthen the iDMA proposal. The current results described in Section 4 are based primarily on measurements of the PCIe interface. We hope to confirm these findings using other descriptor-based interfaces, such as GPU accelerators, Light Peak [18], and SuperSpeed USB.

Reducing or removing descriptor-based traffic can lead to higher-level improvements in the area of sockets structure handling and driver implementation. Cases for improved QoS can be proposed with an iDMA engine that is peer to the application cores.

Further exploration of instantiation of the iDMA is needed. A state machine driven approach, such as Intel's IOAT [10], can be compared in terms of die area and power to a larger, but more adaptable small heterogeneous core.

IO offloading schemes such as TCP Offload Engines (TOE), iWARP and Infiniband reduce latency to some extent by formulating headers on the adapter, but still utilize descriptors to move payload larger than a cache-line size to and from system memory. We plan to characterize the benefit in typical offload mechanisms as well.

7. CONCLUSIONS

With accelerator data movement complexities, especially in a heterogeneous core environment, there is a need to address unscalable inefficiencies. As system core cycles become less valuable due to MCP and SMT, and IO requirements increase, we proposed an IO accelerator to consider in future platform architectures. The current IO approach of descriptor-based DMA engines to entirely off-load from the CPU can be considered "pull-push" for TX and RX transactions. In this paper, we explored the impact of the descriptor related overhead and proposed a "push-pull" method with a dedicated iDMA engine in the CPU. Adapting write-combining memory type to generic IO and offloading IO data movement to a generic platform requirement (networking, storage, and graphics) suggests quantifiable latency and bandwidth-per-pin advantages and potential benefits in power, QoS, silicon area, and security for future datacenter server architecture.

8. ACKNOWLEDGMENTS

Discussions with Steve McGowan on IOAT and USB descriptors and Justin Teller on MIC descriptor details.

9. REFERENCES

[1] Softmodem description, [cited 2010]; Available from: http://en.wikipedia.org/wiki/Softmodem.

[2] I/O Bottlenecks: Biggest Threat to Data Storage, [cited 2010]; Available from: http://www.enterprisestorageforum.com/technology/features/article.php/3856121

[3] Fast Sort on CPUs, GPUs, and Intel MIC Architectures; Available from: http://techresearch.intel.com/spaw2/uploads/files/FASTsort_CPUsGPUs_IntelMICarchitectures.pdf.

[4] Intel 5520 chip-set datasheet, [cited 2010]; Available from: http://www.intel.com/assets/pdf/datasheet/321328.pdf.

[5] HypterTransport IO Link Specification Rev3.10, [cited 2010]; Available from: http://www.hypertransport.org/docs/twgdocs/HTC20051222-00046-0028.pdf.

[6] Leon, E.A., Ferreira, K.B., and Maccabe, A.B. 2007. Reducing the Impact of the MemoryWall for I/O Using Cache Injection. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*.22-24 Aug. 2007.

[7] Dan, T., Yungang, B., Weiwu, H., and Mingyu, C. 2010. DMA cache: Using on-chip storage to architecturally separate I/O data from CPU data for improving I/O performance. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*.9-14 Jan. 2010.

[8] Intel 82599 10GbE NIC; Available from: http://download.intel.com/design/network/prodbrf/321731.pdf.

[9] Introduction to Receive Side Scaling, [cited 2010]; Available from: http://msdn.microsoft.com/en-us/library/ff556942.aspx.

[10] IOAT description; Available from: http://www.intel.com/network/connectivity/vtc_ioat.htm.

[11] Banks and Outsourcing: Just Say 'Latency'; Available from: http://www.hpcwire.com/features/Banks_and_Outsourcing_Just_Say_Latency_HPCwire.html.

[12] Intel 82598 10GbE NIC; Available from: http://www.intel.com/assets/pdf/prodbrief/317796.pdf.

[13] Larsen, S., Sarangam, P., and Huggahalli, R. 2007. Architectural Breakdown of End-to-End Latency in a TCP/IP Network. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*.24-27 Oct. 2007.

[14] PCIe Base 2.1 Specification; Available from: http://www.pcisig.com/specifications/pciexpress/base2/.

[15] Guangdeng, L. and Bhuyan, L. 2009. Performance Measurement of an Integrated NIC Architecture with 10GbE. In *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*.25-27 Aug. 2009.

[16] Schlansker, M., Chitlur, N., Oertli, E., Stillwell, P.M., Rankin, L., Bradford, D., Carter, R.J., Mudigonda, J., Binkert, N., and Jouppi, N.P. 2007. High-performance ethernet-based communications for future multi-core processors. In *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*.10-16 Nov. 2007.

[17] Intel QuickPath Interconnect; Available from: http://en.wikipedia.org/wiki/Intel_QuickPath_Interconnect.

[18] [cited 2010]; Available from: http://www.intel.com/go/lightpeak.