ETA: Experience with an Intel® Xeon™ Processor as a Packet Processing Engine

Greg Regnier, Dave Minturn, Gary McAlpine, Vikram Saletore, Annie Foong
Intel Corporation
greg.j.regnier@intel.com

Abstract

The ETA project at Intel Research and Development has developed a software prototype that uses one of the Intel® Xeon™ processors in a multi-processor server as a packet processing engine. The prototype is used as a vehicle for empirical measurement and analysis of a highly programmable packet processing engine that is closely tied to the server's core CPU and memory complex. The usage model for the prototype is the acceleration of server TCP/IP networking.

The ETA prototype runs in an asymmetric multiprocessing mode, in that the packet processing engine does not run as a general computing resource for the host operating system. We show an effective method of interfacing the packet processing engine to the host processors using efficient asynchronous queuing mechanisms.

This paper describes the ETA software architecture, the ETA prototype, and details the measurement and analysis that has been performed to date. Test results include running the packet processing engine in single-threaded mode, as well as in multi-threaded mode using Intel's Hyper-Threading Technology (HT). Performance data gathered for network throughput and host CPU utilization show a significant improvement when compared to the standard TCP/IP networking stack.

1. Introduction

The performance limitations of server-based networking are well documented [1, 2]. A major goal of the Embedded Transport Acceleration (ETA) project is to enable high performance server communication and I/O over standard Ethernet and TCP/IP networks. By doing so, we hope to take advantage of the large knowledge base and ubiquity of these standard technologies. With the advent of 10 gigabit Ethernet, these standards promise to provide the bandwidth required of the most demanding server applications. In addition, by substantially increasing the performance of server networks, we can also enable standard high-volume servers to perform a greater number of storage and communication-centric applications that are commonly served by specialized appliances.

We use the term Packet Processing Engine or *PPE* as a generic term for computing and memory resources that are used for communication-centric processing. There are some desirable attributes for a packet processing engine that we have tried to achieve. These include *scalability*, *extensibility* and *programmability*.

A PPE must scale in terms of communication throughput as well as in its ability to support large numbers of sessions simultaneously. Extensibility is desirable in order to add value to the solution over time in terms of new features, protocols and applications. Programmability is desirable in order for the solution to be adaptable in the face of changing standards and in order to modify its behavior in subtle but important ways. These desired attributes tend to lead us to a solution where the amount of processing and memory resources is not artificially constrained.

Section 2 of this paper describes the base ETA architecture, in particular the architecture of the software that is instantiated on the prototype. Section 3 gives some details about the prototype hardware and the testing environment. Performance and analysis results are presented in section 4 and 5. Section 6 outlines some related work and section 7 summarizes and conclusions are drawn.

2. The ETA Software Architecture

At a high level, the ETA architecture partitions the server software between host and PPE processing resources. The 'host' is where the general purpose operating system and applications reside. The PPE is where the communication-centric tasks, including network protocol processing, are performed. The interface between the host and the PPE is implemented as a set of asynchronous queues in cache-coherent, shared host memory. These queuing structures are used for control, synchronization, and for receiving and transmitting data.

2.1. ETA Host Software

The ETA host software stack allows for multiple paths between host applications and the PPE. Accelerated networking paths are enabled for applications at both the kernel and user levels as well as the non-accelerated path through the operating system's native TCP/IP and driver stack. At both the kernel and user levels, there is a thin layer of software, an adaptation layer that provides an asynchronous programmatic interface to queuing structures that form the interface between the host and the PPE. In addition, legacy sockets applications are enabled as well as new applications that are written directly to the ETA specific interfaces. The high-level view of the ETA host software stack is shown in Figure 1.

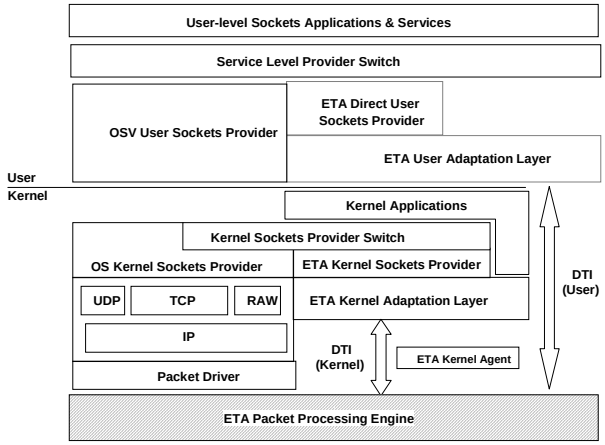

Figure 1 – ETA Host Software Stack

2.2. ETA Host-Engine Interface

The interface between the host processors and the PPE is accomplished through a set of queuing structures called *Direct Transport Interfaces* or DTIs. DTIs are based on the principles of the Virtual Interface Architecture [3] and the Infiniband™ Architecture [7], but differ in that they are optimized for IP networking semantics. In particular, the DTI structures also support the TCP connection commands in addition to data transmission and reception. Anonymous buffer pools are provided in order to support the buffering semantics of TCP streams. Figure 2 shows the structure of the DTI queuing structures.

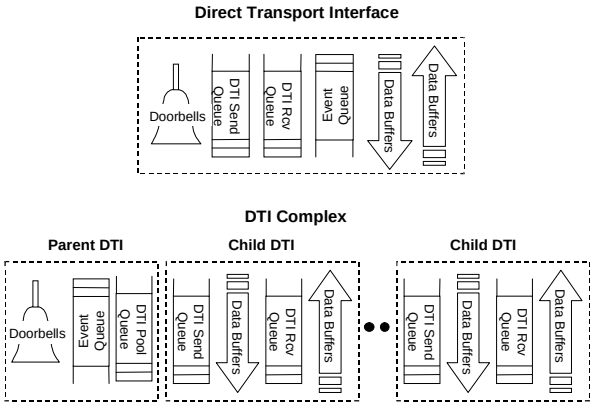

Figure 2 – DTI Queuing Structures

Each DTI may include a send queue, a receive queue, an event queue, doorbells, and data buffers in an associated buffer pool. Individual DTIs include all of these elements. However, groups of DTIs for any given server application can be created such that they share a common event queue and set of doorbells and each child DTI may include only send and receive queues and associated data buffer pools. Parent DTIs are used to listen on new TCP connections. When a new connection is established and accepted, it associates a child DTI with the new connection, thus a child DTI is associated with a TCP session. The following describes the DTI elements:

- The send and receive queues are used to post send and receive descriptors to the ETA packet processing engine. The data can be transferred directly to or from application buffers or the DTI anonymous buffer pool.

- The event queue enables the ETA packet processing engine to post event notices to the host application. Each DTI can be created with a private event queue or have its event notices directed to an event queue shared by multiple DTIs.

- The DTI data buffers enable the ETA packet processing engine to buffer data for the DTI, 1) when the source or target application buffers are not pre-conditioned, 2) when TCP segments are received and there are no receive descriptors posted on the receive queue, or 3) when TCP segments are received out of order.

- The send and receive doorbell addresses are used to write notices or signals directly to the ETA packet processing engine and indicate a context with which each is associated.

2.3. ETA Packet Processing Engine Software

The ETA architecture is largely independent of the implementation of the packet processing engine. The PPE implementation could be a fixed device, a specialized programmable engine, or as in the case of the prototype, a general purpose CPU. An ETA aware PPE needs to support several specific functions.

First, the PPE must support the DTI queuing structures. It must have the ability to be notified of new work posted on the send and receive queues via the doorbell addresses. In addition, the PPE must support the event queue and be able to interrupt the host processors in the event that an application is blocked waiting for a transaction to complete.

The PPE also must be able to execute the actual packet processing functions on behalf of the host, and must of course support an interface to the network itself. In the case of our prototype, the packet processing functions are mainly the termination of TCP/IP connections on behalf of server applications.

3. The ETA Prototype

The ETA prototype uses one of the Intel® Xeon™ Processors in a dual-processor server as the host processor and one as the PPE. The main function of the PPE is the establishment and termination of TCP/IP sessions on behalf of applications running on the host CPU.

This configuration has several advantages. First and foremost, there was no special hardware to develop. Secondly, we could use standard software development tools to develop the software for both the host side and the PPE. We use standard gigabit Ethernet network cards with a modified version of the Ethernet driver. Finally, we can use shared, coherent memory in order to implement the interfaces between the host CPU and the PPE.

3.1. Prototype Software Environment

We developed the prototype using a standard Linux kernel version 2.4. The PPE software is a loadable Linux module with a stripped down kernel TCP/IP stack, with code added to support the DTI interfaces, and a modified gigabit Ethernet driver. The PPE software module is given affinity to one processor (CPU1) on the dual-processor platform, and never yields the processor. This enables CPU1 usage as a dedicated packet processing engine.

Synchronization is a unique aspect of the PPE software on the prototype. The interface to the network interface controller has been modified so as not to use interrupts for data transfer operations. Instead, the PPE can poll on NIC descriptors in shared host memory in order to detect completed packet transactions. Communication between the host CPU and the PPE via the DTI structures are accomplished through the DTI doorbells in shared host memory as well. Thus, the PPE can poll the doorbell addresses and NIC descriptors for synchronization without causing memory bus traffic until the cache-lines of the associated shared memory is modified. This allows the PPE to run without interrupts and avoid the associated overheads.

3.2. Prototype Hardware Platform

The prototype can run on virtually any multi-processor platform that runs the Linux kernel. The platform used in our testing is a dual-processor Intel® Xeon™ Processor platform. The processors run at 2.4 gigahertz on a 400 megahertz front-side bus. The Network Interface Controllers are standard Intel Pro1000 gigabit Ethernet controllers.

3.3. Prototype Test Environment

Our test environment consists of the server under test (the ETA prototype server) and five client computers connected directly by gigabit Ethernet links. The client computers are standard off-the-shelf servers running the Linux OS and the *ttcp* throughput micro-benchmark.

The tests running on the ETA prototype are kernel-level applications that interface directly to the ETA kernel abstraction layer. Figure 3 shows the basic test environment.

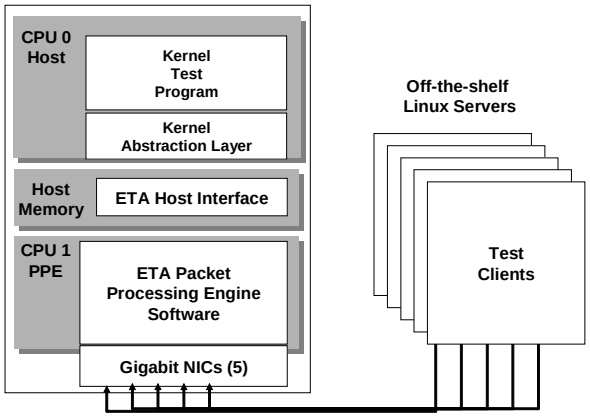

Figure 3 – ETA Prototype Test Environment

4. Measurement Results

Basic throughput tests were performed on the ETA prototype for transmit and receive for several transfer sizes. The ETA test results are compared with a standard Linux dual processor server running the *ttcp* throughput micro-benchmark.

Figure 4 shows transmit performance along with the amount of CPU that is idle and thus available for application use. For transfers of 1024 bytes and less, both CPUs of the standard Linux server were 100% utilized executing the networking stack, thus leaving zero CPUs left idle. For larger sized messages, 20% or less of one of the CPUs was left idle.

For the ETA server, the host CPU was less than 20% utilized across all transfer sizes leaving more than 80% of one CPU idle and available. In addition, the ETA transmit throughput considerably exceeded the standard Linux server for all transfer sizes.

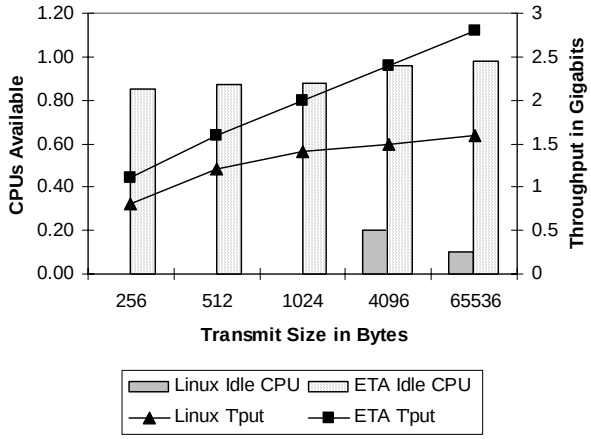

Figure 4 – Transmit Performance

Figure 5 shows throughput and available CPU for the receive path. For all cases, both CPUs of the standard Linux server were 100% utilized running the networking stack, thus leaving zero CPUs left for other applications (that is why the dark gray bars don't show on the graph). The Host CPU of the ETA prototype was less than 20% utilized for all receive transaction sizes, leaving 80% of a CPU idle and available. ETA receive throughput exceeds the Linux server by a relatively small margin. This smaller improvement is partly due to memory-to-memory copy performance. Our ETA implementation uses a one-copy receive path due to the fact that we use off-the-shelf network interface controllers that place packets directly into a pre-allocated packet buffer. These packet buffers must then be copied to destination buffers by the PPE.

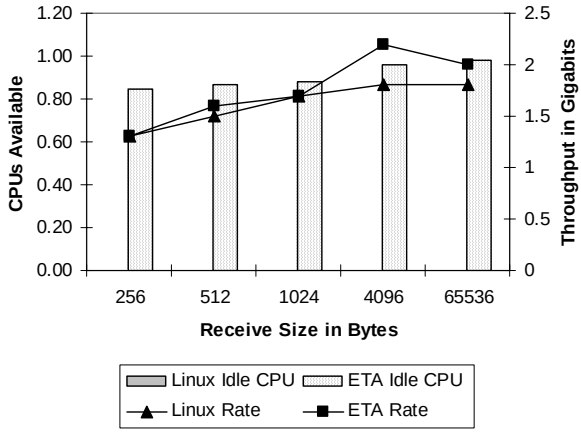

Figure 5 – Receive Performance

Figure 6 compares the transmit performance of the standard Linux server (2P SMP), the ETA prototype in single-threaded mode (ETA ST), and the ETA prototype with Intel's Hyper-Threading Technology [5] enabled on the packet processing engine (ETA HT). With HT enabled, the PPE runs on two hardware threads and provides a degree of parallelism, thus hiding the memory latency which we have found to be a performance limiter of TCP/IP processing on servers. The results show an approximately 50% increase in transmit performance on the ETA prototype with HT enabled, achieving over four gigabits of throughput.

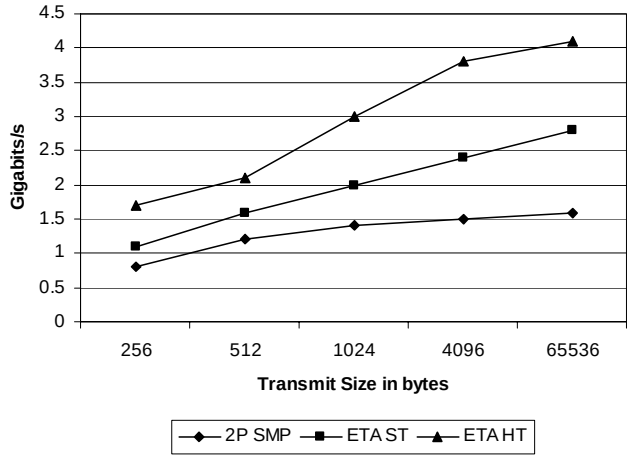

Figure 6 – Transmit Performance with HT

Figure 7 shows receive throughput for the standard Linux server (2P SMP), the ETA prototype with the PPE running in single-threaded mode (ETA ST) and the ETA prototype with the PPE running with HT enabled (ETA HT). In addition, we added a test path where we enabled HT, but did not execute the data copy on the PPE (ETA HT NoCopy). For receive, we see that enabling HT improved the ETA performance about 20 percent. As

noted before, this relatively small improvement is partly due to the copy performance on our prototype implementation. When we omit the copy by the PPE into the test application buffer, we see significant performance increase, similar to the performance of the transmit case (nearly four gigabits per second).

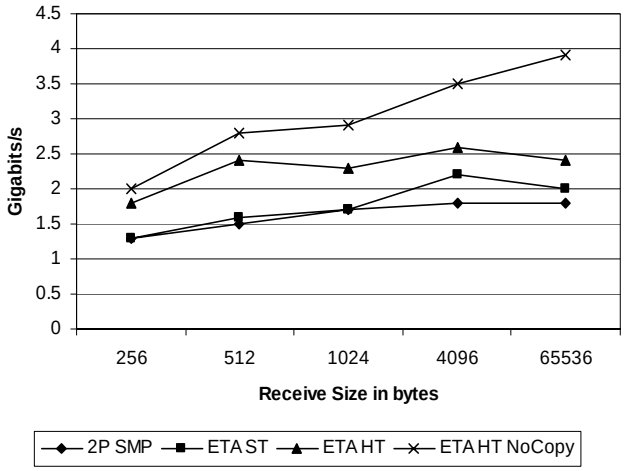

Figure 7 – Receive Performance with HT

5. Analysis

To understand the differences in performance between the SMP system and the ETA prototype, we used the VTune™ [11] performance analysis tool to profile their execution. Figure 8 and 9 show a high-level execution profile for the SMP and ETA systems respectively.

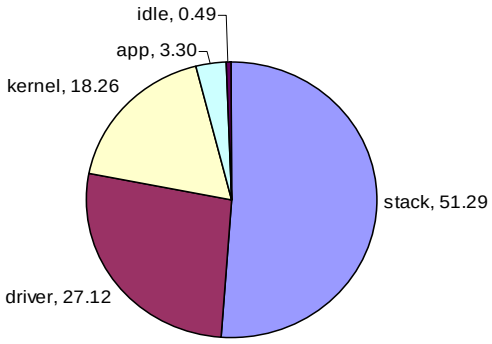

Figure 8 – Execution profile of SMP system

Each pie-chart contains the measured execution profile for the 1KB transmit test-case. Each slice of a pie represents the total amount of CPU that is being used for a certain function, e.g. a value of 10 equates to 10% utilization of 2 CPUs (or 20% usage of 1 CPU). To explain the terms within the charts, the term 'stack' equates to execution attributed to TCP/IP processing; 'driver' equates to the driver for the Ethernet controller; 'kernel' equates to kernel execution other than TCP/IP; and 'app' is the test application.

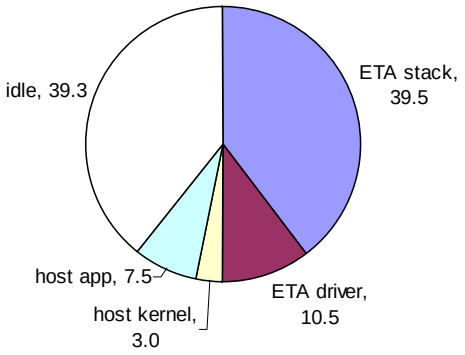

Figure 9 – Execution profile of ETA system

The execution profiles for the two machines are quite different. First, TCP/IP related processing on the SMP system requires significantly more CPU utilization than on the ETA prototype (51% vs. 39%). If we also assume that much of the kernel slice on the SMP machine is executed on behalf of the TCP/IP stack, the disparity is even larger. Much of this difference can be attributed to the efficiency of the interface between the test application and the TCP/IP stack.

Another major difference in the comparison is the efficiency of the drivers (27% vs. 10%). This efficiency can largely be attributed to the fact that the ETA PPE does not need to share resources with the OS and application, and can poll the devices descriptor queues without incurring an interrupt. This avoids not only costly interrupt processing, but also reduces the number of device register accesses required over the (relatively) slow I/O bus. Additional and more detailed performance analysis is required, and planned.

6. Related Work

Recent commercial efforts to increase server network performance have centered on specialized TOE (TCP/IP Offload Engine) devices [4, 8]. TOE devices generally offload varying amounts of the TCP/IP protocol stack on a device that attaches to the I/O subsystem of a server. TOE devices generally utilize separate, specialized processing and memory resources. The ETA prototype described in this paper differs from these devices in that it utilizes processing and memory resources of the server itself, making the packet processing engine a first class citizen of the core CPU and memory complex. The software is partitioned in a manner that is much more efficient than in standard symmetric multiprocessing systems.

Other related research efforts include the *QPIP* [9] work at Berkeley that showed the effectiveness of interfacing IP protocols implemented on an intelligent network adapter using the Queue Pair model of the Infiniband™ Architecture. The *TCP Servers* project [10] at Rutgers University showed a framework where the network processing could be partitioned onto a dedicated node, processor or an intelligent adapter and interface to the host applications through lightweight communication mechanisms.

7. Conclusions

We can see from the results that software partitioning can significantly increase the overall communication performance of a standard multi-processor server. Specifically, partitioning the packet processing onto a dedicated set of compute resources allows for optimizations that are otherwise not possible when time-sharing the same compute resources with the OS and applications. For example, our prototype PPE does not need to incur the overhead of interrupts and system calls because it can poll shared memory for new work. Polling can be done without placing load on the memory subsystem or front-side bus of the platform because we can rely on the cache coherence protocols to allow the PPE to poll internally in cache. Memory load is only incurred when the associated memory location is updated by either a network device or one of the host processors. Cache interference is also largely avoided because we are not sharing caches with the OS and applications except through the ETA host interface. Other optimizations are possible, such as strategic pre-fetching of control and packet header information.

We have seen that threading, Intel's Hyper-Threading Technology in particular for our prototype, is an important factor in achieving greater performance. Multi-threading allows parallelism that is useful for hiding memory latency. Networking workloads tend to have poor locality and thus poor cache behavior. For example, when a device receives a new packet, it never lands in the cache so that when it is referenced by the PPE, significant memory latency is incurred. Given the growing disparity between processor speeds and memory latency, multi-threading becomes more important over time.

This paper has shown the capabilities of a general purpose Intel® Xeon™ Processor for server-based packet processing. Specialized packet processors with support for specific networking functions have the potential for providing even greater absolute performance [6]. The type of processing and memory resources that are used for packet processing involve a set of trade-offs that include absolute performance, flexibility, extensibility and cost.

The ETA host-PPE interface allows for low-overhead, asynchronous interaction between the host processors and the packet processing engine. The DTI queuing structures presented in this paper are built on proven concepts [3, 7], and extend those concepts for an optimized solution for IP-based networks.

We have plans for additional ETA development and analysis along multiple vectors. We currently are performing analysis on the capabilities of our PPE to accelerate TCP connection establishment. This capability is important for web servers that have to deal with a great number of short-lived connections. We also are in process of developing and analyzing an iSCSI storage stack over ETA and are investigating techniques to minimize data copies through the use of the DDP and RDMA protocols that are currently under definition. Additionally, we are working to enable legacy user-level sockets applications on ETA that once complete will allow us to run and analyze a large number of network applications.

8. References

[1] J. Kay and J. Pasquale, "The Importance of Non-Data Touching Processing Overheads in TCP/IP", In *Proceedings of ACM SIGCOMM*, 1993.

[2] Annie Foong, Thomas Huff, Jaidev Patwardhan, and Greg Regnier, "TCP Performance Re-Visited", 2003 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, Texas, March 2003.

[3] Don Cameron and Greg Regnier, *The Virtual Interface Architecture*, Intel Press, 2002.

[4] L. Gwennap, "Count on TCP offload engines", EETimes article at: http://www.eetimes.com/semi/c/ip/OEG20010917S0051.

[5] W. Magro, P. Peterson, and S. Shah, "Hyper-Threading Technology: Impact on Compute-Intensive Workloads", *Intel Technology Journal*, Feb 2002.

[6] Y. Hoskote, V. Erraguntla, D. Finan, J. Howard, D. Vangal, V. Veeramachaneni, H. Wilson, J. Xu, N. Borkar, "A 10GHz TCP offload accelerator for 10Gbps Ethernet in 90nm dual-VT CMOS", IEEE International Solid-State Circuits Conference, 2003.

[7] Infiniband™ Trade Association, http://www.infinibandta.org.

[8] Prasenjit Sarkar, Sandeep Uttamchandani and Kaladhar Voruganti, "Storage over IP: When Does

Hardware Support Help?", 2nd USENIX Conference on File and Storage Technologies, March 2003.

[9] P. Buonadonna, D. Culler, "Queue-Pair IP: A Hybrid Architecture for System Area Networks", *Proceedings of ISCA 2002*, Anchorage, AK, May, 2002.

[10] M. Rangarajan, A. Bohra, K. Banerjee, E. Carrera, R. Bianchini, L. Iftode and W. Zwaenepoel, "TCP Servers: Offloading TCP Processing in Internet Servers", Rutgers University Technical Report, DCS-TR-481, March 2002.

[11] VTune™ Performance Analyzer, http://www.intel.com/software/products/vtune/vpa/index.htm.

Disclaimer