NSTX-U Advances in Real-time Deterministic PCIe-based Internode Communication

Keith G. Erickson[a], M. Dan Boyer[b], and D. Higgins[c]

a. kerickso@pppl.gov, Princeton University Plasma Physics Lab, Princeton, NJ 08540, USA, Corresponding author
b. mboyer@pppl.gov, Princeton University Plasma Physics Lab, Princeton, NJ 08540, USA
c. david.higgins@ccur.com, Concurrent Real-Time, Pompano Beach, FL 33069, USA

Abstract

Distributing control mechanisms across modern commercial off the shelf (COTS) GNU/Linux systems often introduces difficult to mitigate non-deterministic behavior. Existing methods to address this problem involve non-real-time technologies such as RDMA over Infiniband or custom Ethernet solutions that trade determinism for ease of use or lower cost. The National Spherical Torus Experiment Upgrade (NSTX-U) is pursuing a new design that allows direct communication between heterogeneous systems with scalable, microsecond latency with one microsecond of jitter on that latency, outside of the constant transmission delay at the physical layer. The future design of the NSTX-U Real-time Communication System will utilize direct PCIe-to-PCIe communication with kernel support tuned for low overhead, allowing two (or more, through a switch) real-time (RT) systems to communicate and share resources as one larger entity. This greatly increases the processing capability of the primary Plasma Control System (PCS), turning previously insurmountable computational challenges into a more manageable divide and conquer parallel task.

Keywords

Real-time Linux; reflective memory; RDMA; PCIe

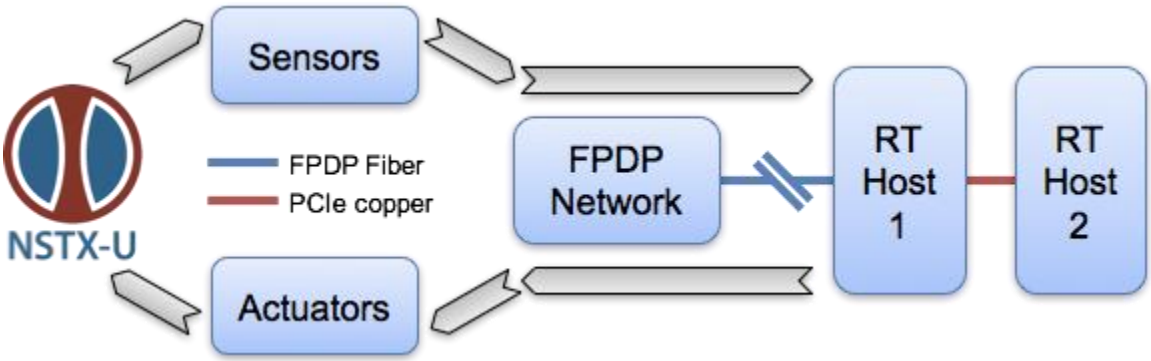
1. Introduction

The National Spherical Torus Experiment Upgrade (NSTX-U) project is a fusion experiment hosted at the Princeton University Plasma Physics Lab whose mission [1] requires advanced real-time control [2]. Recent upgrades to the device have increased the capabilities of the experiment, and new, more sophisticated control solutions will be critical to achieving the physics goals of the experiment and maximizing performance. These future goals for the control systems [3][4][5][14][15] necessitate expanding the computational capacity and flexibility of the existing systems in a scalable fashion. Currently under investigation is the use of remote memory access across external PCIe interfaces as a deterministic, low-latency communication mechanism between heterogeneous commercial off-the-shelf (COTS) computers running a general purpose, real-time capable GNU/Linux operating system.

Historically, the system designs in use on NSTX-U have refrained from distributed computing to avoid the overhead of moving data between nodes, instead opting for an ever increasing core count within a single node. The current Plasma Control System (PCS) computers use 64 AMD Opteron 6386 SE cores in a single host [2], which is enough to meet all current needs. Plans for future computational tasks, however, are exceeding the growth curve of the processing that can fit into a single computer. Outside of special proprietary motherboard designs that allow for eight CPU sockets, there is no practical COTS hardware vendor supporting more than four CPUs for a single system installation. The new real-time communication design will instead enable coupling multiple 64-core nodes into a larger overall system of computers that can grow over time like existing switched fabric solutions, but with the determinism present in the one-monolithic-system approach.

1.1. NSTX-U Plasma Control System

Figure 1-1: High level PCS layout with proposed additional RT computer



The NSTX-U Plasma Control System (PCS) shown in Figure 1-1 is a combination of hardware, firmware, and software that performs configuration, data acquisition, transport, processing, actuator output, and archiving of measured and calculated data during each experimental test. The primary physics connection to the experiment's sensors and actuators uses the VITA 17.1 Front Panel Data Port (FPDP) protocol, a highly efficient mechanism developed originally for VME-based systems but now available in many environments.

PCS nominally runs through a 12-second real-time event, or “pulse”, processing inputs and outputs synchronously at a 5 kHz rate. Before the pulse, there is a period of configuration and preparation, followed by a controlled countdown to the start of this real-time event. The actual pulse transitions from plasma initiation, through a ramp-up to target experimental plasma conditions, into a multi-second experimental phase, followed by a plasma ramp-down and ultimately the termination of the sequence. There are approximately 30 different algorithms running on a single real-time computer that control many aspects of the pulse, including the power supplies that drive the magnetic coils, the gas injection system that puffs gas into the machine, and the neutral beams that provide heating. There is an identical twin computer that serves as a development system running in parallel with the live system. In total, the PCS generates about 2 GB of data for each pulse [2].

The PCS algorithms run inside a real-time framework created by General Atomics [6] which is in use on several experiments around the world: DIII-D, NSTX-U, MAST-U, Pegasus, KSTAR [11], and EAST [7]. Its widespread use fosters easy collaboration both in improving the framework and in sharing algorithms developed to run using it. The framework provides many service abstractions that allow a given installation to tailor it to specific applications while utilizing a common API. For example, it supports multiple ways to acquire data, convert it from low-level I/O to physics terms, and eventually archive it in a site-specific format such as PTDATA or MDSplus.

1.2. PCS Expansion

The real-time control computer is a shared system that also houses protection software known as the Digital Coil Protection System (DCPS). “Protection” in this context differs from “control” in terms of the goals and restrictions of the software. The stricter rules guiding protection software are primarily driven by the physical consequences of failure, and they tend towards a cautious approach. Accordingly, changes in control software tend to be more frequent and less rigorously tested to facilitate more aggressive scientific research.

DCPS occupies roughly half of the computational cores, leaving the other half available for PCS [8]. The current set of PCS algorithms utilize most of their allotted cycle time on the remaining cores, motivating the need for expansion beyond the capabilities of a single system. Adding additional nodes as indicated in Figure 1-1 requires some amount of communication between PCS processes, and the framework facilitates this effort in a relatively general way. It has built in functionality to support moving data between processes regardless of their location (local to the computer or remotely on separate host) and independent of the underlying communication method.

There are two higher level abstractions available from the framework for this purpose: a communication buffer and a real-time message (RTM). The backing store for an RTM, while customizable, defaults to System V style shared memory regions. Sharing these between processes on a local host is obviously simple; sharing these between nodes requires some other mechanism to detect and propagate changes.

It turns out that this design maps well to typical implementations of reflective memory systems. Generally, they involve mapping a region of memory and instructing the transport mechanism to handle change propagation automatically. Accordingly, DIII-D, KSTAR, and

MAST-U have successfully used reflective memory in some fashion through this generic RTM interface [9] with different underlying technologies performing the work. NSTX-U is now exploring a new technology to achieve similar functionality with highly deterministic results and peak performance.

2. Technology Background

2.1. Reflective Memory

Reflective memory technology first appeared in the 1980s for use in VME environments, produced by VME Microsystems International Corporation (VMIC). General Electric bought VMIC and later sold it to Abaco Systems, who still produces reflective memory products. Other competitors include Curtiss-Wright, who produces SCRAMNET, and previously (though no longer) Myrinet, which is in use on DIII-D.

Each provider offers potentially different features. There is no specific “Reflective Memory” standard produced by a governing body, but instead a loose set of common, high level traits. A reflective memory system will typically include a vendor specific physical layer, dedicated I/O cards, and software interfaces that hide the underlying communication. The system should handle replicating changes to memory regions between hosts without user involvement and with minimal system overhead. In some cases, hosts have access to knowledge of memory location, whether local or remote, and can use this knowledge to make efficient storage decisions. Reflective memory products typically support mesh-type, many-to-many connections, allowing multiple endpoints to communicate easily. Point-to-point setups do exist, however, and may be advantageous in applications with few hosts and limited budget.

2.2. Remote Direct Memory Access

Remote Direct Memory Access (RDMA) is a more formal and comparatively recent refinement of reflective memory. A key difference is the existence of a consortium that defines various protocols in terms of IETF RFCs. Since these protocols are agnostic of the lower layer communication methods, it is possible to use RDMA over an existing InfiniBand or EtherNet network to access memory regions between hosts. The OpenFabrics Alliance promotes these uses of RDMA protocols.

RDMA standards add guarantees that memory access from a remote host will not involve the CPU or the OS on the local host (or vice versa). The “direct” in RDMA signifies the ability of the I/O card to interact with memory regions without first copying data to and from the conceptual CPU core. To be pedantic, a modern CPU includes the memory controller on the die itself and communicates to it via localized high-speed methods. From the perspective of the OS kernel, however, the CPU core itself does not process any instructions to move this data between an I/O device and main memory. In multiprocessor systems, a core accessing an RDMA region may still invoke latencies due to Non-Uniform Memory Access (NUMA) locality.

A typical use of RDMA would be to assemble a cluster of relatively inexpensive computers connected via InfiniBand that are visible to the user as a single entity. Using RDMA,

the user could conceivably distribute processing across the cluster and share data transparently by reading and writing to shared memory regions mapped by the RDMA mechanism. A process on Host A, for example, could then read the output of a process on Host B without the CPU on Host B taking any additional action outside of storing the result in memory.

2.3. Dolphin PCIe Bridge

Dolphin Interconnect Solutions has combined the concepts of RDMA and Reflective Memory with their previous work in both the PCI and PCIe standards to create a product that connects multiple computers using native PCIe itself. They leverage the power and capability of this efficient lower level interface to deliver RDMA compatible technologies with significantly better performance characteristics. While this is a proprietary product in general, the underlying infrastructure is based on open standards that can interact with other compatible products to sustain future growth.

In the Dolphin design, hosts can communicate via a stackable switched fabric or in a direct point to point fashion using copper or fiber. The switch allows many devices to exchange data at full PCIe speeds, while the point to point setup allows up to three in a star configuration. The maximum throughput at PCIe Generation 3 speeds with all 16 links is 128 Gbps with only 1.54% overhead due to the use of PCIe Gen3 128B/130B encoding. This greatly dwarfs the throughput of other technologies and significantly opens the operating space for real-time communications.

The Dolphin products enable more capability than just reflective memory or RDMA. Since the underlying communication mechanism involves a standardized PCIe bus extension, they additionally provide advanced PCIe device sharing features that bypass the CPU similarly to RDMA. Just as separate hosts can access remote memory across the PCIe bus, they can also access remote devices directly without involving the remote CPU. The latency of this device sharing is similar to memory sharing, limited only by physical distance and scaling linearly with data size. It is a possible future scenario where an NSTX-U realtime computer accesses a farm of GPU cards spread across multiple computers for calculating advanced predictive models in real-time as extra inputs for control algorithms. Dolphin-based device sharing would allow housing the GPU cards remotely while accessing them as if they were all housed locally.

3. Testing Platform

All conducted tests were point to point between two systems with identical operating environments. The differentiation in the test computer setups being AMD vs Intel is purely a factual description of the components available for testing and not meant to indicate a branding preference. Where possible, references are instead to the Gen2 and Gen3 PCIe support, which is the largest performance driving limitation.

3.1. Operating Environment

NSTX-U uses a real-time operating system provided by Concurrent Real-Time called RedHawk Linux. It is derived from CentOS, tuned for determinism, and combines a modified kernel and drivers with development tools to provide a complete real-time environment. The kernel modifications deliver better determinism capabilities than those available with the PREEMPT_RT patches provided by http://rtwiki.org/. [2][8][12]

The OS version in use for this testing was an unmodified 64-bit 7.2 release. There were no issues identified during testing that required driver source modifications, thus the drivers for the Dolphin card were also unmodified and are available from the vendor. All real-time applications made use of CPU shielding, a RedHawk feature provided that gives exclusive CPU core access to a specific process, blocking all other processes, interrupts, and timers from running there.

3.2. Existing AMD Systems (Gen2)

NSTX-U has had for several years now a powerful real-time system using 64 AMD Opteron 6386 SE cores across 4 CPUs running at 2.8 GHz. This platform connects Generation 2 PCIe devices to the CPU via separate dual SR5690 northbridge controllers. One northbridge, the “noisy” bus, interfaces with one CPU, 3 PCIe slots, and all other motherboard devices. The other northbridge, the “quiet” bus, interfaces with a different CPU, 3 additional PCIe slots (two x16, one x8), and nothing else. This isolated bus with a dedicated CPU reduces bus contention, which in turn increases determinism by reducing jitter immensely from 30 microseconds to less than 1 microsecond.

In this system, the quiet bus houses a Concurrent-produced Realtime Clock and Interrupt Module (RCIM), a Curtiss-Wright SL240, and the Dolphin PXH830 under test. The SL240 is the FPDP I/O card used for the low level communication with physical hardware, both sensors and actuators. The RCIM is a special card meant specifically for RedHawk systems. It supports multiple features useful for real-time work, such as discrete, high-resolution clocks, internal and external interrupts, internal and external synchronization, and other real-time specific tools [13].

3.3. Potential Intel Systems (Gen3)

Given the vintage of the existing AMD based systems and the emergence of 3rd generation PCIe devices, testing also included two new Intel-based systems. The first used two quad-core Intel Xeon E5-2637 v4 chips running at 3.5 GHz with HyperThreading disabled on a SuperMicro X10DRi motherboard. The other used a SuperMicro X10DAi with two 10-core E5-2687W v3 chips running at 3.10 GHz. These motherboards have similar support for bus separation, though the bus controllers are on the CPU die itself instead of residing on a northbridge. One CPU connects to 3 PCIe slots along with the Platform Controller Hub (PCH) and thus all additional motherboard devices. The other CPU connects to 3 different PCIe slots (also arranged as two at x16 and one at x8) and nothing else. Mirroring the existing systems, it is this “quiet” CPU (and bus) that runs the RT code and interacts with the I/O cards.

Like the AMD system, the quiet bus also houses an RCIM along with a matching Dolphin PXH830. There is no SL240 in this system. In the eventual grand design, only one computer will interface to the outside world; other computers will form a Dolphin-based cluster of sorts. The testing platform attempts to mimic this future deployment.

3.4. Software Harness

Dolphin provides several test programs alongside the drivers with full source code available. This software exercises various aspects of the functionality provided by the cards, and creates datasets for timing evaluation. The main program employed for this testing is called "scipp". The version used has slight modifications such as using `mlockall()` to protect against page faults and additional functions to record data. It follows a very simple pattern for measuring performance. Rather than addressing the inherent difficulty in synchronizing timing mechanisms on separate computers, the software instead measures the round trip time between sending a piece of data to a remote host and receiving the data echoed back over the physical connection. It repeats this for data payload sizes that increase by powers of two. Note that testing revealed that for any given power of two size, such as 64 bytes, the data transferred is actually 4 bytes longer, or 68 bytes in this example, to account for protocol overhead. The test results presented below include payloads ranging from zero through 8 KB by increasing powers of two.

To maintain accuracy, timing functions use the RDTSC/P instruction pair to have the least overhead while accounting for instruction reordering [10]. The RedHawk kernel provides regular and frequent calibration of this cycle counter against the actual clock speed to give better results than simply reading `/proc/cpuinfo` on a RedHat system would provide.

There are two ways to record data. The stock method maintains counters for 200 nanosecond wide time intervals. For every measured time, the program increments the counter corresponding to the interval into which the measured time will fall. For instance, if a measurement is 1300 nanoseconds, it will increment the counter that represents times between 1200 and 1400 nanoseconds. This significantly reduces the amount of data collected for a run. The enhanced custom method uses a "raw times" option to output individual time durations instead of the bucket approach that the stock routines use. This requires a large memory footprint and careful separation between measuring real-time data and outputting the measurements. However, the testing generates gigabytes of data, providing very fine detail that can reveal potential patterns or problem areas.

3.5. Physical Setup

The permutations of configuring computer pairs were AMD Gen2 to AMD Gen2, Intel Gen3 to Intel Gen3, and AMD Gen2 to Intel Gen3. In the latter case, there was no notable difference between the two Intel Gen3 systems, as the bulk of the performance differences came from the change to Gen3 PCIe over Gen2 PCIe. Both computers in each configuration communicated directly through a point to point mini-SAS HD copper cable without an intervening switch.

4. Results

Test results ultimately showed very promising performance metrics. The datasets described below represent summaries of close to a billion test runs in varying configurations. Larger datasets generated with the “raw times” option were downsampled for display purposes in a way designed to capture and highlight, or show the absence of, important outlier data. Simple averaging would hide these key data points. Instead, the procedure involved carving data sets into fixed size chunks, keeping the highest and the lowest sample in that chunk, and discarding the rest. This procedure is purely for visualization purposes; the data sets are available in their original form. Data generated without the “raw times” option still involves a very high sample count, but the equivalent data reduction occurred at creation time as described earlier. All times are round-trip; the expectation is that one-way times are nominally half of this value.

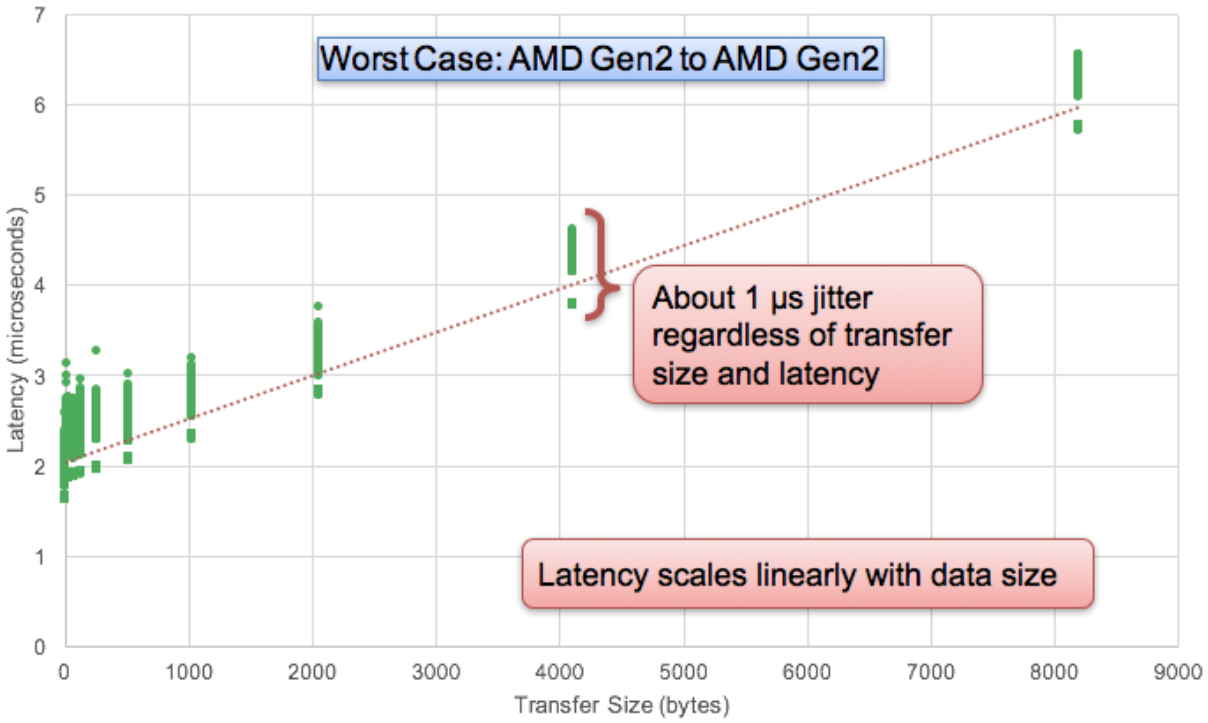


Figure 5-1: Linear Latency With Constant Jitter

Figure 5-1 shows completely linear scaling of total latency with constant jitter when communicating between two identical AMD Generation 2 systems. While the data sets consist of millions of samples (the shortest run was 10 million, the longest was 100 million), it is important to note that the first data point in each size bucket results in an abnormal spike due to caching and other extraneous, non-real-time applicable reasons. That single data point is not present in the figure. There were zero spikes aside from that first point, which the figure accurately reflects.

There is an obvious gap through which the trendline tracks relatively consistently. The gap is small -- less than 100 nanoseconds -- and studying this is both non-trivial and not inherently beneficial to the ultimate goal at hand. Anything under 1 microsecond meets the

demands of NSTX-U and serves as an appropriate lower bound for optimization opportunities. Still, future planned endeavors may revisit this as a curiosity.

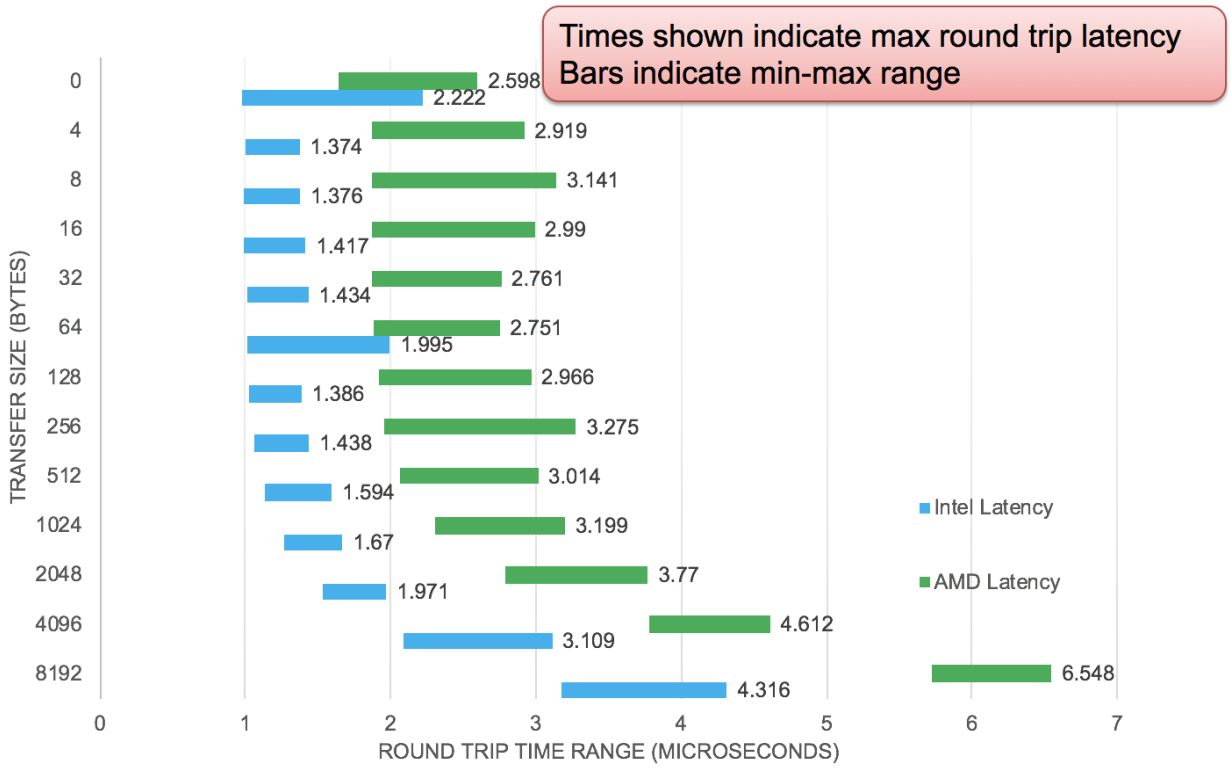


Figure 5-2: Generation 3 PCIe Improvements

Size	Min	Max
0	0.982	2.222
4	1.000	1.374
8	0.988	1.376
16	0.992	1.417
32	1.013	1.434
64	1.014	1.995
128	1.030	1.386
256	1.061	1.438
512	1.132	1.594
1024	1.266	1.670
2048	1.530	1.971
4096	2.082	3.109
8192	3.174	4.316

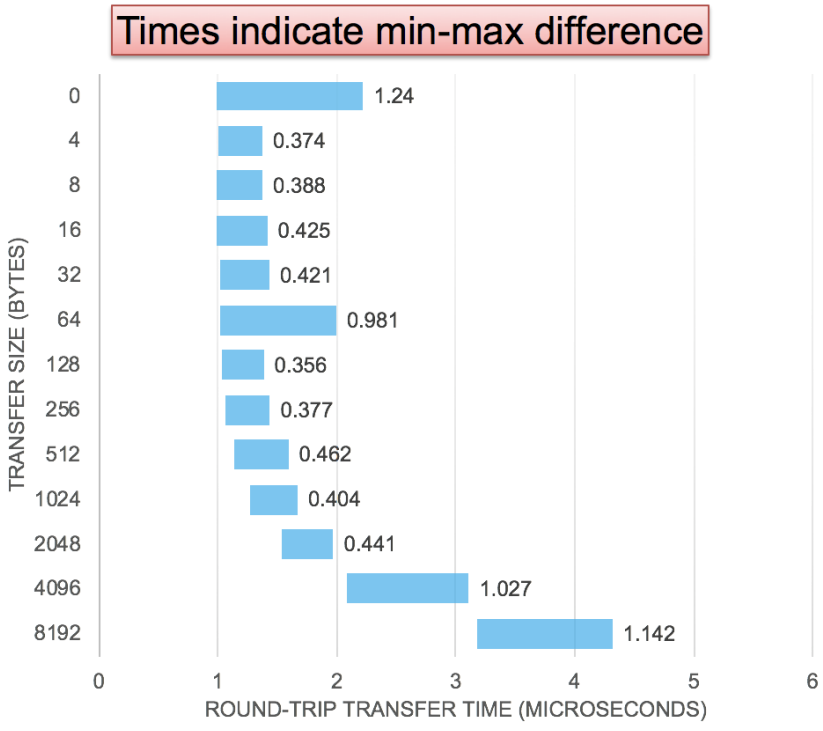


Figure 5-3: Jitter range across payload sizes

Figures 5-2 and 5-3 highlight the significant differences encountered moving from Generation 2 PCIe to Generation 3 PCIe. Latency still scales linearly (note the exponential Y axis), and jitter is mostly constant. The jitter improvements are obvious and consistent with expectations based on the general infrastructure improvements of Generation 3 PCIe, such as the overhead reduction from 8B/10B encoding to 128B/130B encoding [16]. There is a strange, unexplained oddity around several key payload sizes, however, that warrants further exploration. This oddity either does not appear in Generation 2 PCIe or is masked by the overall latency and jitter increase. Of special note is that the payload size of zero represents only the driver overhead, indicating that some potential improvements may be possible on the driver side. Nevertheless, the performance improvements indicate a clear overall win.

5. Conclusion and Future Work

The results presented above serve as a solid proof of concept for using Dolphin products in a deterministic, real-time safe way. The measured jitter was well within acceptable limits for the NSTX-U timing requirements, and in all cases, latency scaled linearly with payload size.

The main differences between systems under test were centered around the generation of PCIe bus support, despite the different hardware vendors. Ultimately, when AMD releases the next generation of server platforms [17] with support for Gen3, these tests will likely show that the platforms are comparable. Considering that the measured throughput is near the theoretical maximum, it is clear that the bottleneck is not the CPU.

As NSTX-U moves toward a future centered around clusters of real-time control computers, advanced predictive control algorithms become a reality. Eventually, the experiment could leverage a network of relatively inexpensive processing nodes to provide real-time predictions of plasma shape and profile evolution based on proven simulations to augment the capabilities of existing methods. Additionally, this can serve as an easy platform to spin up new real-time diagnostics and integrate them into the control system without significant timing analysis and engineering effort.

6. References

[1] J. Menard, et al., https://doi.org/10.1088/0029-5515/52/8/083015
[2] K. G. Erickson, et al., https://doi.org/10.1016/j.fusengdes.2014.04.069
[3] M. D. Boyer, et al., https://doi.org/10.1088/1741-4326/aa68e9
[4] I. R. Goumiri, et al., https://doi.org/10.1088/0029-5515/56/3/036023
[5] S. P. Gerhardt, et al., https://doi.org/10.1063/1.4889781
[6] B. G. Penaflor, et al., https://doi.org/10.1016/j.fusengdes.2004.04.009
[7] B. G. Penaflor, et al., https://doi.org/10.1016/j.fusengdes.2007.11.012
[8] K. G. Erickson, et al., https://doi.org/10.1109/sofe.2015.7482291
[9] B. G. Penaflor, et al., https://doi.org/10.1016/j.fusengdes.2009.01.034
[10] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf
[11] Sang-hee Hahn, et al., https://doi.org/10.1016/j.fusengdes.2008.12.082

[12] K. G. Erickson, et. al., https://doi.org/10.1109/tps.2014.2321106
[13] http://www.concurrent-rt.com/wp-content/uploads/2016/11/RCIM-Data-Sheet-vJan2016.pdf
[14] M. D. Boyer, et al., https://doi.org/10.1088/0029-5515/55/5/053033
[15] I. R. Goumiri, et al., http://dx.doi.org/10.1063/1.4976853
[16] http://pcisig.com/specifications/pciexpress/
[17] http://www.amd.com/en/products/epyc