Smart Data Cache Injection (SDCI) White Paper

Publication # **58725** Revision: **1.00**
Issue Date: **February 2025**

Advanced Micro Devices

Table of Contents

List of Figures

List of Tables

Revision History

Date	Revision	Description
February 2025	1.00	Initial public release

Chapter 1 Introduction

The importance of high-input/output (I/O) throughput with low latency has grown with the advent of high-bandwidth I/O devices such as storage and networking. The result is an evolution toward chiplet-based scale-out architectures that increase CPU core counts, cache capacity, and DRAM bandwidth.

The steady increase in I/O link speed has dramatically reduced the time needed to process I/O packets. Fetching data from farther system memory (DRAM) can be extremely expensive; thus, for faster application processing, there is a need to optimize packet processing by minimizing memory access latency and DRAM bandwidth. One strategy for reducing DRAM bandwidth is to preload inbound I/O data in processor cache, eliminating the need to fetch that data from DRAM. Reduced DRAM bandwidth also translates to lower usage of DRAM power, which then can be used to power up cores for faster processing. To reduce DRAM bandwidth, incoming I/O data need to be located near cores that process them. CPU cache can be a very efficient alternative to DRAM by significantly reducing processing latencies on CPU cores.

Examples of usage cases where cache injection flow can boost performance:

1. **High-speed networking data processing:** In networking usage cases, data are transferred from network to host buffers. Then a descriptor or flag update is written to indicate data are ready to be consumed. An interrupt is fired to the processor to indicate data are ready for consumption, or the core polls a memory location for indication of a hardware semaphore update. In both usage cases, descriptor and flag data are very useful to have in processor cache so software can examine and move data with minimal delay. In addition to descriptors and flags, packet header data also benefit from faster processing as packet decoding and routing decisions are made faster.

2. **Latency-constrained real-time feedback systems:** In these systems, feedback within a particular latency constraint is critical to finishing work on time and allowing the next cycle to start. Data are usually sent to the host via preferred Quality of Service (QoS) routes to bypass noncritical data. Putting these data into processor L2 cache facilitates quick data processing without incurring the overhead of fetching data from memory.

3. **Financial services applications:** In financial service scenarios, processing information quickly – sometimes by order of nanoseconds – can make the difference between successful order execution and a failed bid. Placing an order delivered via I/O in processor cache and then allowing the processor to execute that order without having to fetch data from memory saves critical time in the processing loop, enabling more transactions to be completed successfully. When cache injection is available, the system should inject only key information to minimize cache pollution.

Chapter 2 SDCI Overview

2.1 What Is SDCI?

Because cache is a critical asset, the data it contains need to be of high utility. Host hardware and processing cores lack the necessary information to determine which data should be cached. Smart Data Cache Injection (SDCI) allows preloading of data in processor caches by steering applicable I/O data directly to a core's L2 cache. This is done using a standards-based approach that allow endpoints to decide what traffic to inject into cache.

Cache residency of critical code is key to overall application performance. Unlike alternative cache injection solutions, SDCI enables endpoints to control which data are injected into caches and thus reduces cache pollution and enhances system performance.

To optimally manage data written into CPU cache, SDCI uses an open-industry standard, PCI Express' TLP Processing Hints (TPH) feature. SDCI allows inbound I/O Direct Memory Access (DMA) writes to be steered directly into L2 cache. Data are not written to DRAM until they age out of cache.

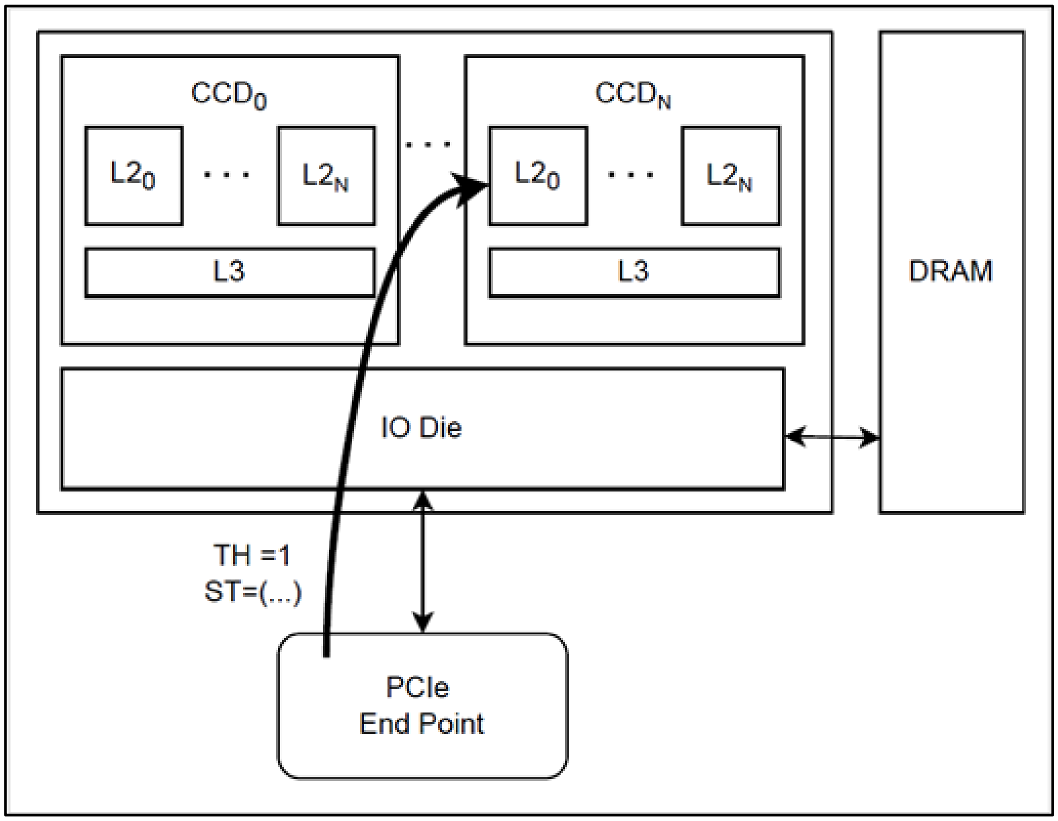


Figure 1. Cache Insertion

Both host and endpoints must support the TPH feature; TLP Hints (TH) in the TLP header identifies data that are candidates for cache insertion. In a typical implementation, the host driver and endpoint firmware collaborate on device settings to direct I/O packets to the correct cache locations.

2.2 SDCI Enablement

AMD EPYC™ 9005 series processors, codenamed “Turin,” enable hardware support for SDCI and the PCIe TPH specification. The AMD Turin platform initialization package provides BIOS and firmware to support SDCI in the operating system’s PCIe subsystem.

Endpoint vendors need to implement hardware support for the PCIe TPH specification as well as software/firmware to enable the TPH hardware. Vendors/OEMs can influence what data are inserted into cache with their software implementations.

Chapter 3 Value Proposition

SDCI will benefit most applications where CPU cores read and write inbound I/O data. This can include both control and data plane aspects of a use case.

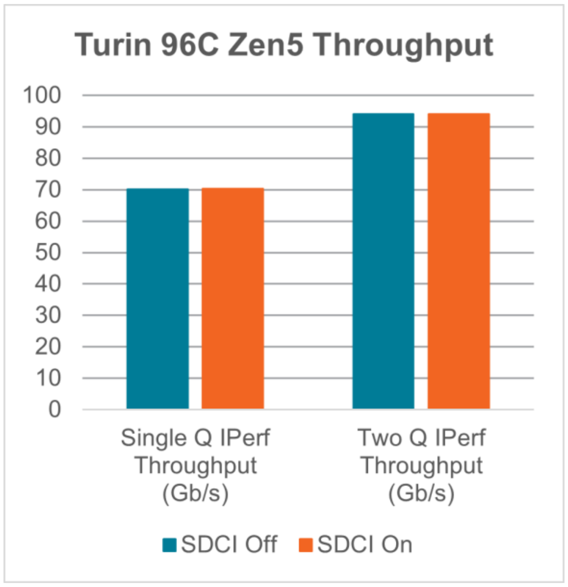
Networking applications are an obvious beneficiary. These applications process inbound packet headers and packets on the CPU. Other examples include storage, CPU artificial intelligence (AI) inferencing, high-performance computing (HPC), or engineering applications where a CPU-based workload performs read/write transactions on inbound I/O write data and is sensitive to latency or DRAM bandwidth.

Based on industry standards, SDCI enables endpoint devices to efficiently redirect only important data packets to CPU cache.

3.1 SDCI Performance with iPerf

SDCI's effectiveness can be seen in data collected using iPerf, a popular cross-platform tool that can measure Transmission Control Protocol (TCP) network performance. iPerf reports the maximum achievable throughput between two network endpoints.

Results were collected with and without SDCI to observe net gains. System configuration details are included in Appendix A.



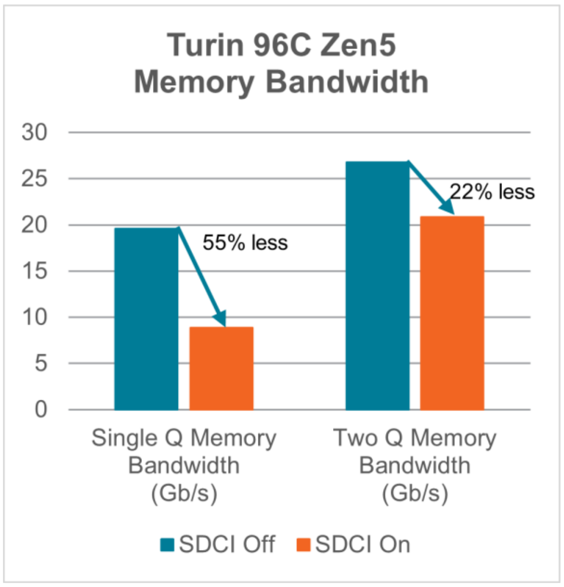
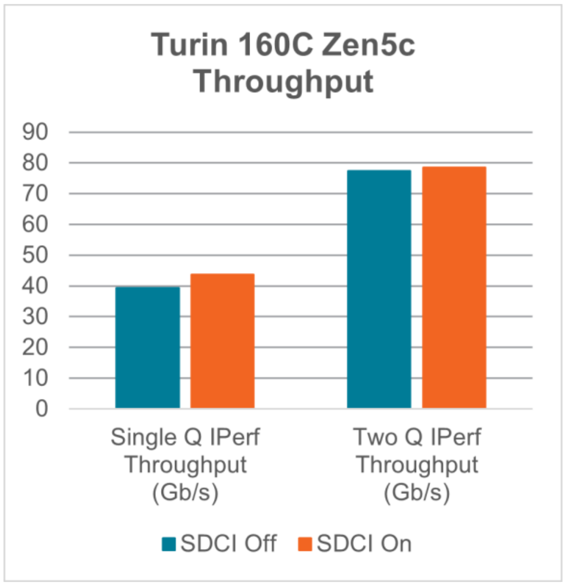


Figure 2. iPerf Throughput and Memory Bandwidth with Turin 96C Zen5



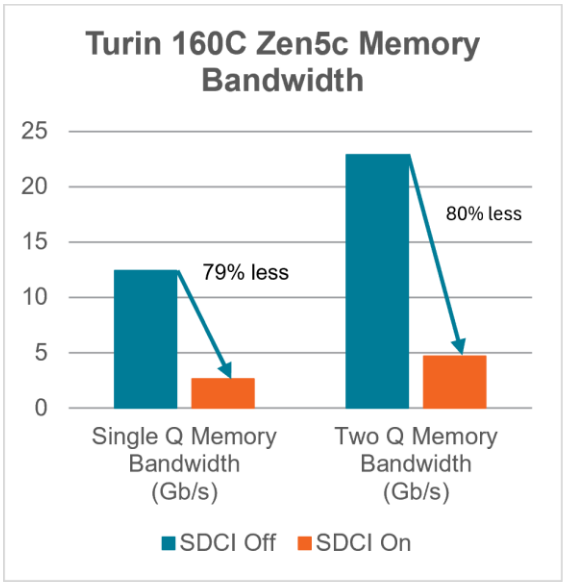


Figure 3. iPerf Throughput and Memory Bandwidth with Turin 160C Zen5c

3.2 SDCI Performance Observations

3.2.1 Turin 96C Zen5

iPerf performance saturates NIC throughput limits and remains unchanged whether SDCI is enabled or disabled. At the same performance level, a significant reduction in total memory bandwidth is observed with SDCI enabled. With iPerf performance around 70 Gb/s using a single queue, enabling SDCI results in a 55 percent reduction in total memory bandwidth.

Scaling to two queues, iPerf performance is around 94 Gb/s and enabling SDCI results in a 22 percent reduction in total memory bandwidth. Memory bandwidth reduction is lower with two queues because the additional bandwidth and overhead in processing Rx buffers for two queues increases overall cache utilization and results in more evictions before the CPU can consume the in-cache copy. The evicted data need to be read back when the CPU finally requests them.

3.2.2 Turin 160C Zen5c

With iPerf performance around 43 Gb/s using a single queue, enabling SDCI results in a 79 percent reduction in total memory bandwidth. Scaling to two queues, iPerf performance is around 78 Gb/s and enabling SDCI results in an 80 percent reduction in total memory bandwidth. The eviction rate is observed to be much lower in this case, and the data continue to be available in cache for the CPU to consume as soon as it is ready. This has a direct impact on overall memory bandwidth savings compared with the earlier test.

3.3 Monitoring Memory Bandwidth with AMD uProf

Memory bandwidth used by an application can be monitored using AMDuProfSys, which is included as part of the AMD uProf toolkit. The tool can be downloaded from *https://www.amd.com/en/developer/uprof.html*. The current Linux version is 5.0-1479. Download and install the package of your choice. The description below assumes that tar.bz2 is installed.

Once AMD uProf is installed, the AMDuProfSys program is available from AMDuProf_Linux_x64_5.0.1479/bin.

A special driver is needed for the tool to work. This can be installed using the following commands:

- `$ cd AMDuProf_Linux_x64_5.0.1479/bin`
- `$ sudo ./AMDPowerProfilerDriver.sh install`

To measure memory bandwidth when running iPerf on the Turin server, invoke AMDuProfSys as follows:

```
./AMDuProfSys --config <CONFIG> -o <OUTPUT_DIR> <options> <WORKLOAD>
<workload-specific-args>
```

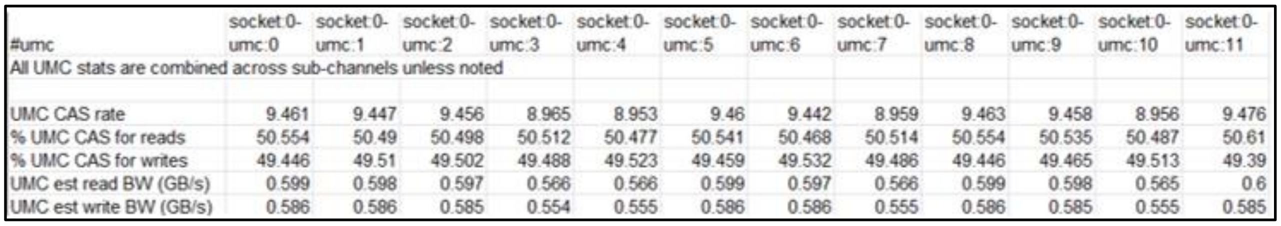
For example:

```
./AMDuProfSys --config umc -o /tmp/test taskset -c 0 iperf -s -p 9000
```

Start the iPerf client from the remote system. Once the test is complete, kill the AMDuProfSys command invocation (the iPerf server does not terminate by itself) to stop iPerf and prepare the report. The report will be available as a .csv file within the OUTPUT_DIR.

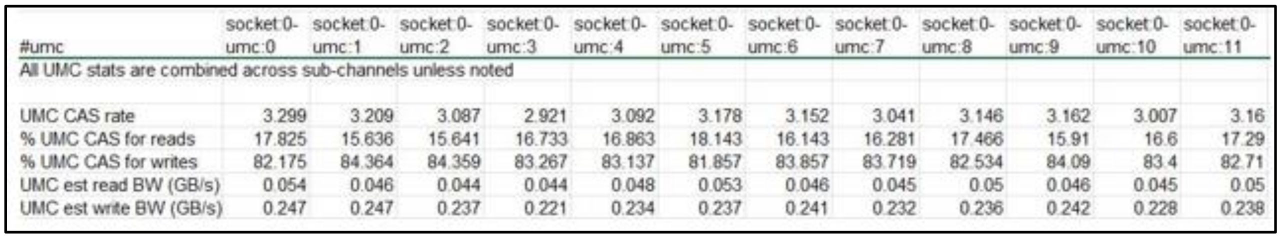
Memory bandwidth can be monitored by observing the “UMC est read BW” and “UMC est write BW” fields in the report.

For example, see excerpts in Figure 4 and Figure 5 from reports collected from iPerf test runs on a Turin Zen5 server with 12 channels of DDR5 and both SDCI Off and SDCI On:



#umc	socket:0-umc:0	socket:0-umc:1	socket:0-umc:2	socket:0-umc:3	socket:0-umc:4	socket:0-umc:5	socket:0-umc:6	socket:0-umc:7	socket:0-umc:8	socket:0-umc:9	socket:0-umc:10	socket:0-umc:11
All UMC stats are combined across sub-channels unless noted												
UMC CAS rate	9.461	9.447	9.456	8.965	8.953	9.46	9.442	8.959	9.463	9.458	8.956	9.476
% UMC CAS for reads	50.554	50.49	50.498	50.512	50.477	50.541	50.468	50.514	50.554	50.535	50.487	50.61
% UMC CAS for writes	49.446	49.51	49.502	49.488	49.523	49.459	49.532	49.486	49.446	49.465	49.513	49.39
UMC est read BW (GB/s)	0.599	0.598	0.597	0.566	0.566	0.599	0.597	0.566	0.599	0.598	0.565	0.6
UMC est write BW (GB/s)	0.586	0.586	0.585	0.554	0.555	0.586	0.586	0.555	0.586	0.585	0.555	0.585

Figure 4. Turin Zen5 Memory Bandwidth with SDCI Off



#umc	socket:0-umc:0	socket:0-umc:1	socket:0-umc:2	socket:0-umc:3	socket:0-umc:4	socket:0-umc:5	socket:0-umc:6	socket:0-umc:7	socket:0-umc:8	socket:0-umc:9	socket:0-umc:10	socket:0-umc:11
All UMC stats are combined across sub-channels unless noted												
UMC CAS rate	3.299	3.209	3.087	2.921	3.092	3.178	3.152	3.041	3.146	3.162	3.007	3.16
% UMC CAS for reads	17.825	15.636	15.641	16.733	16.863	18.143	16.143	16.281	17.466	15.91	16.6	17.29
% UMC CAS for writes	82.175	84.364	84.359	83.267	83.137	81.857	83.857	83.719	82.534	84.09	83.4	82.71
UMC est read BW (GB/s)	0.054	0.046	0.044	0.044	0.048	0.053	0.046	0.045	0.05	0.046	0.045	0.05
UMC est write BW (GB/s)	0.247	0.247	0.237	0.221	0.234	0.237	0.241	0.232	0.236	0.242	0.228	0.238

Figure 5. Turin Zen5 Memory Bandwidth with SDCI On

Chapter 4 Summary Conclusion

High-speed I/O device performance is highly sensitive to access latency, making it crucial to keep data close to CPU cores. With the advent of scale-out chiplet technologies, the number of cores has increased, leading to nonlinear DRAM memory access latency. To address the complexities of scale-out architectures, better I/O management is needed.

SDCI aims to enhance system and application performance by lowering latency and reducing DRAM bandwidth required for applications. Without SDCI, inbound I/O writes data must be written to DRAM and then read before use by the CPU cores. Injecting data directly into cache allows it to be read there, significantly reducing memory bandwidth usage and latency for some applications. SDCI hardware and BIOS firmware support is enabled by default on AMD Turin products for easy endpoint integrations. By reducing overall DRAM bandwidth, system-level tradeoffs can be made, such as reducing costs by not using DRAM channels or increasing performance by lowering power usage, increasing DRAM bandwidth, and/or reducing latency.

AMD’s solution uses PCIe standards to enable PCIe endpoints, drives, and applications to collaboratively decide which data to cache. This optimizes cache usage, minimizes cache pollution, and ensures cache is used effectively. AMD is following a step-wise roadmap that will continue to optimize SDCI and broaden applicable use cases.

Appendix A System Configuration

A.1 System Configuration

A.1.1 BIOS Settings

- `DF Common options → Memory Addressing: NPS 1`
- `DF Common options → ACPI → ACPI SRAT L3 Cache as NUMA Domain: Disable`
- `NBIO Common options → SMU Common Options → Determinism Control: Manual`
- `NBIO Common options → SMU Common Options → Determinism Enable: Performance`
- `NBIO Common options → SMU Common Options → APB Disable (APBDIS): 1`
- `NBIO Common options → SMU Common Options → DFPstate: 0`
- `NBIO Common options → SMU Common Options → Power Profile Selection: High Performance Mode`
- `NBIO Common options → SMU Common Options → DF C-States: Disabled`
- `NBIO Common options → PCIe ARI Support: Enable`
- `NBIO Common options → PCIe Ten bit tag support: Enable`
- `CPU Common options → Local APIC Mode: X2APIC`
- `CPU Common options → Core Performance Boost: Enabled`

To enable SDCI settings:

- `DF Common options → SDCI: Enabled`

A.1.2 CPU and Other Information

Tests were run on the following platforms:

Table 1. CPU and Other Information

Processor	DDR5 Frequency (MT/s)	TDP (W)
Turin Zen5 96 Cores	6000	400
Turin Zen5c 160 Cores	4800	400

A.1.3 Ethernet Adapter Information

The Ethernet adapters used in the tests were 2 × 100G model P2100G Broadcom NICs. Please contact Broadcom support for a firmware version with TPH support enabled.

Broadcom firmware settings: The firmware should be set up for steering-tag support for PCIe memory writes for Rx buffers and completion queue writes. It is recommended to have PCIe relaxed ordering turned on in the firmware. Optionally, turn off firmware support for RDMA and DCBX.

A.1.4 iPerf Version

```
$ iperf --version
iperf version 2.1.5 (3 December 2021) pthreads
```

A.1.5 OS and Kernel Version

AMD has upstreamed support for TPH enablement in the Linux kernel. The 6.13 Linux kernel release in January 2025 includes this support. The tests for the performance numbers reported here were run earlier on an Ubuntu 22.04.2 LTS system with a 6.10+ custom kernel that had identical kernel patches for TPH support.

A.1.6 Kernel Boot

No special kernel boot parameters were used. It is recommended to have "processor.max_cstate=0" enabled in the boot-command line.

A.1.7 Linux System Configuration

iPerf performance tests typically also tune some kernel parameters like net.core.rmem_max and net.core.wmem_max, among others, for best performance. However, those were not applied to leave the system configuration closer to defaults.

A.1.8 Broadcom NIC Configuration

- Limit queue count to 1, 2 or 4 queues
- Queue sizes were limited to 511 entries, also tested with 255 entries per queue
- NIC IRQ lines were pinned to CPU cores IRQ 0 on CPU 0, IRQ 1 on CPU 1, and so on.

A.1.9 Testing iPerf

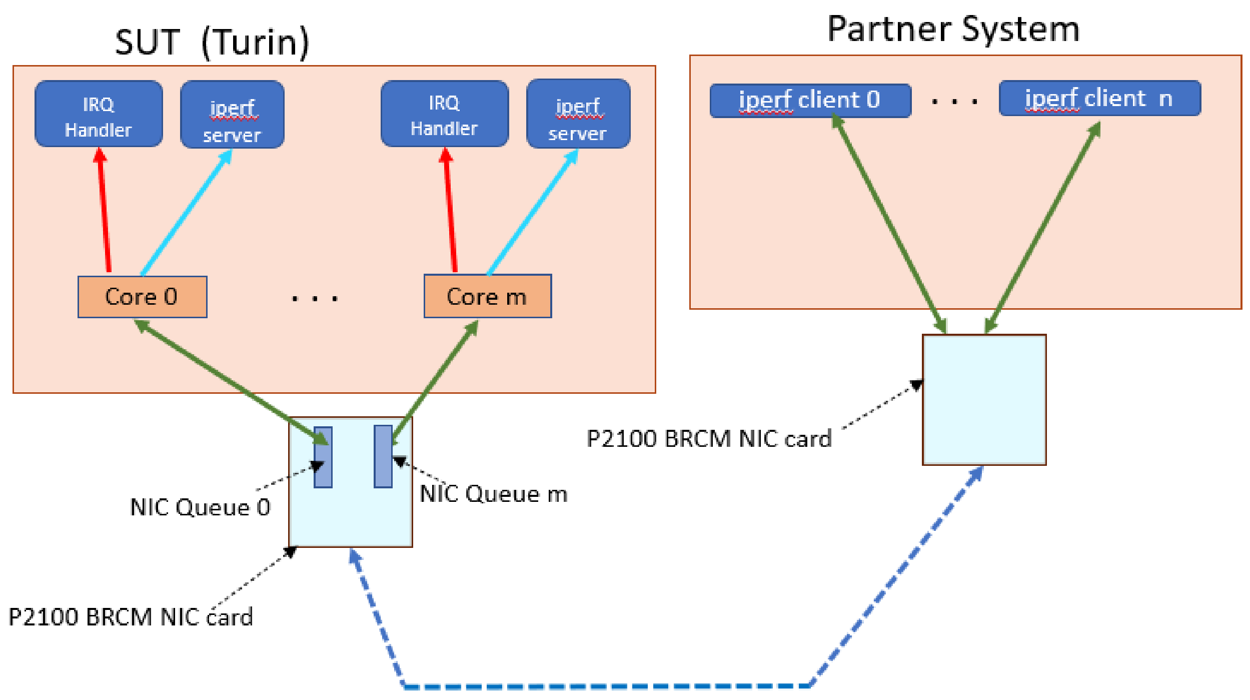
To test SDCI bandwidth savings, iPerf can be used in one of two modes in relation to the IRQ pinning:

1. Run iPerf on the same set of CPU SMT threads to which IRQ is pinned.
2. Run iPerf on the sibling CPU SMT thread to the SMT thread to which IRQ is pinned.

A.1.10 System Profiling

The system was profiled to observe the behavior of cache accesses and the corresponding savings in memory bandwidth using an interval version of AMD's MultEvent tool. AMD has since enabled a publicly available version of its AMD uProf profiling tool as a substitute for MultEvent. Refer to Section 3.3 for more information.

A.2 Test Configuration



- Test is run with fixed queue count. Tested with 1 and 2 queues.
    - IRQ for each queue is pinned to separate and distinct CPU cores.
- iPerf server threads run on the same set of cores servicing the IRQs.
- Partner system runs iPerf clients.
    - Single queue tests use a single client
    - Client count (n) for multiple queue tests (m server NIC queues, where m ≥ 2), is 2*m;

A.3 Testing with iPerf – Single Queue

Start iPerf server and bind it to the core based on the approach selected in the A.1.9 Testing iPerf section above. Start the client to send traffic to the iPerf server.

On Server:

```
$ sudo taskset -c 0 sh -c "iperf -s -p 9000"
```

On Client:

```
$ numactl -C 0-15 iperf -c 192.168.1.100 -t 420 -i 1 -P1 -N -p 9000 &
```

- Confirm that only a single CPU core is busy on the server.
- Confirm that only a single Rx queue in the NIC receives all the packets.
- Check the Ethernet traffic throughput and packet rate (using a tool like vnstat).
- Ensure that iPerf is running only on selected CPU SMT threads.

iPerf performance should be in the 68–70 Gbps range. Run iPerf client multiple times, each time for 30 seconds or more and make sure the packet rate and iPerf throughput are consistent. There may be a variation of about 3–5% between runs in the iPerf reported numbers, which is acceptable for the single-queue test.

A.4 Testing with iPerf – Two (or More) Queues

Start iPerf server and bind it to the core based on the approach selected in the A.1.9 Testing iPerf section above. Start the client to send traffic to the iPerf server.

On Server:

```
$ sudo taskset -c 0-1 sh -c "iperf -s -p 9000"
```

On Client:

```
$ numactl -C 0-15 iperf -c 192.168.1.100 -t 420 -i 1 -P4 -N -p 9000 &
```

- Confirm that only two CPU cores are busy on the server.
- Confirm that only two queues are receiving packets. The packet rate on both queues should be fairly close to each other.
- Check the Ethernet traffic throughput and packet rate (using a tool like vnstat).
- Ensure that iPerf is running only on selected cores.

iPerf performance should be in 93–94 Gbps range. Run iPerf client multiple times, each time for 30 seconds or more, and make sure the packet rate and iPerf throughput are consistent. There may be a variation of about 2–3% between runs in the iPerf reported numbers, which is acceptable for the two-queue test.