Understanding I/O Direct Cache Access Performance for End Host Networking

Minhu Wang
Tsinghua University
minhuw@acm.org

Mingwei Xu
Tsinghua University
xumw@tsinghua.edu.cn

Jianping Wu
Tsinghua University
wjp@tsinghua.edu.cn

ABSTRACT

Direct Cache Access (DCA) enables a network interface card (NIC) to load and store data directly on the processor cache, as conventional Direct Memory Access (DMA) is no longer suitable as the bridge between NIC and CPU in the era of 100 Gigabit Ethernet. As numerous I/O devices and cores compete for scarce cache resources, making the most of DCA for networking applications with varied objectives and constraints is a challenge, especially given the increasing complexity of modern cache hardware and I/O stacks. In this paper, we reverse engineer details of one commercial implementation of DCA, Intel's Data Direct I/O (DDIO), to explicate the importance of hardware-level investigation into DCA. Based on the learned knowledge of DCA and network I/O stacks, we (1) develop an analytical framework to predict the effectiveness of DCA (i.e., its hit rate) under certain hardware specifications, system configurations, and application properties; (2) measure penalties of the ineffective use of DCA (i.e., its miss penalty) to characterize its benefits; and (3) show that our reverse engineering, measurement, and model contribute to a deeper understanding of DCA, which in turn helps diagnose, optimize, and design end-host networking.

ACM Reference Format:
Minhu Wang, Mingwei Xu, and Jianping Wu. 2022. Understanding I/O Direct Cache Access Performance for End Host Networking. In *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/PERFORMANCE '22 Abstracts), June 6–10, 2022, Mumbai, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3489048.3522662

1 BACKGROUND

Data-intensive workloads, such as software network functions, training deep learning models, and leaf nodes in microservices require end-host networking with hundreds-of-gigabits-scale bandwidth and microseconds-level latency. This has in turn encouraged the quick progress of network hardware in recent years. Since the 2000s, the speed of mainstream network interface cards (NIC) in data centers has increased from 1 Gbps to 100 Gbps [7]. The quickly increasing I/O speed and traffic add to the pressure on other components in the server, especially main memory. Limited by its physical properties and the DRAM market's preference for low cost and large capacity [1], the speed of DRAM falls behind that of both processors and NICs. Memory turns into a bottleneck in I/O operations in which it serves as the primary bridge between the processor and I/O devices. I/O devices hit the same *memory wall* [10] as processors if the performance divergence between NIC and DRAM grows continually.

The fast on-chip processor cache is the key to push beyond the *memory wall*. Direct Cache Access (DCA) extends Direct Memory Access (DMA) to enable I/O devices to also manipulate data directly in the fast on-chip processor cache. DCA has been discussed in academic research [3, 5, 8] and implemented by vendors in widely used commercial hardware [4].

2 DDIO INTERNALS

DCA is critical to scaling-up end-host networking, but it is challenging for applications to utilize it properly first owing to limited knowledge of DCA. We disclose many details of DDIO, which is Intel's implementation of DCA, and use it as an example to emphasize the importance of a hardware-level understanding of DCA.

Specifically, we concentrate on the cache coherence protocol used by DDIO requests and how it fits with the non-inclusive cache hierarchy. We design synthesized micro-benchmarks with varied memory access patterns, thus disclosing complete cache line state transition tables for both inclusive and non-inclusive cache hierarchy. We reveal unpublished hardware details, clarify widespread misunderstandings, and show that insufficient understanding of DCA could lead to inexact optimization or inaccurate performance diagnosis. We also measure the latency of DCA requests to cache lines in different coherence states and unsurprisingly validate that coherence states are critical to cache performance, just like locations of cache lines.

To cite some interesting observations, we find that Intel prepares two special ways (the leftmost two ways) in LLC for cache lines shared between L2 and LLC in its *skylake* processors with a *non-inclusive* cache hierarchy. Cache lines are moved from L2 to these two special ways when accessed by PCIe read (DCA read) instead of the documented and well-known two ways for DDIO allocation (the rightmost two ways). It implicitly extends the available cache for networking data and leads to unexpected cache interference. We also find that access latency to cache lines in private L2 cache maybe even longer than to those in the memory. It's also possible to observe large memory traffic and high cache hit rates at the same time because requests are concurrently sent to the cache hierarchy and the memory controller.

3 A CACHE MODEL FOR DCA

Writing software that makes the most of DCA, i.e., by completing all I/O operations in the cache such that it completely bypasses memory, is challenging. DCAed data that is not processed in time may be evicted back to memory due to cache collisions with subsequent DCAed data (*leaky DMA* problem [9]) or with data from neighbor applications (*latent contender* problem [11]). Evicted data will have to be brought back from memory, unfortunately making DCA useless. Since network I/O requires cooperation among the NIC, PCIe, driver, and application, the effectiveness of DCA operations (i.e., the hit rate) is affected by a wide range of factors in the I/O stack including hardware specifications, system configurations, and workload characterization.

Moreover, networking applications have different performance-related objectives and different resource constraints. DCA reduces the latency of I/O operations and memory traffic at the cost of a larger cache footprint, which may not be the goal of all applications. The available cache size and memory bandwidth also vary according to the execution environment. Current research gives many practical suggestions on how to tune DCA to meet the workload-specific requirements: *ResQ* [9] reduces the number of available network buffers to restrict the cache usage of a single Network Function (NF), thus ensuring performance isolation between multiple NFs sharing a physical machine. *Shenango* [6] chooses to copy packets once instead of applying a zero-copy design to encourage buffer recycling. *PacketMill* [2] reserves most of the cache for DCA for extremely high-performance packet forwarding. These empirical suggestions are specific to their contexts (e.g., multi-tenancy, dedicated machine) and targets (e.g., minimum interference, maximum performance) and thus challenging to generalize to arbitrary DCA optimization problems (e.g., generate less than 1 GB/s memory traffic while using no more than 20% of cache).

To help network-intensive applications in a broad spectrum to take full advantage of DCA, we build an analytical framework to model the impacts of hardware specifications, system configurations, and workload characterization on the effectiveness of DCA, i.e., to answer the question: *how does the cache hit rate of DCA operations change if we tune X?* We also systematically measure the benefit of optimizing DCA, i.e., to answer the question: *what is the penalty of a miss by a DCA operation?* An application can determine its tuning target of DCA hit rate using the penalty measurement and can then learn tuning advice from the DCA model to utilize DCA according to its need.

We build our model using four observations of the current network stack. (1). We always pre-allocate finite packet buffers and use them circularly. (2). All buffers experience the same life cycle in which it traverses the network processing pipeline and gets touched by NIC and processor alternately. (3). States of buffers could be described by their locations in the memory hierarchy (physical state, e.g., in L2, L3, or memory) and their position in the network processing pipeline (logical state, e.g., in the RX queue or the TX queue). (4). The volume of memory access during a period could be estimated using the number of packets processed by the system. Combining the four observations, we build a Markov model describing the transition of the physical and logical state of a representative cache line in network buffers. The transition when the cache line is accessed by NIC or processor has been revealed in our measurement. The transition during the cache line waiting in network queues could be estimated using the lengths of the queues. We could solve the Markov model and transform its stable probability to obtain the expected cache hit rates at certain logical states.

Our model achieves good precision and provides some insight into the parameter tuning of the networking stack. It gives an average of 2.7%, and a maximum of 5.4% prediction errors on the PCIe write hit rate, and less than 1% errors on the PCIe read hit rate on a benchmark consisting of 5 widely used virtual network functions and realistic traces. We find that crux of the matter is the lengths of queues in the processing pipeline. How queues are utilized by network devices and drivers also matters. We show that our model could be used to decide the minimum number of cache ways required to meet given memory traffic constraints.

Our model could serve as a reference for networking software and hardware designs. We need to monitor and control the lengths of queues in the processing pipeline to meet the cache constraints and tune the cache coherence protocol to benefit from the cyclic pattern of buffer usages.

ACKNOWLEDGMENTS

We thank our shepherd Daniel S. Berger and anonymous SIGMETRICS reviewers for their valuable comments. The research is supported by the National Natural Science Foundation of China under Grant 61832013 and 61872426. Mingwei Xu is the corresponding author.

REFERENCES

[1] Shekhar Borkar and Andrew A. Chien. 2011. The future of microprocessors. *Commun. ACM* 54, 5 (2011), 67–77.

[2] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostic. 2021. PacketMill: toward per-Core 100-Gbps networking. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021.* ACM, Virtual Event, USA, 1–17.

[3] Ram Huggahalli, Ravi R. Iyer, and Scott Tetrick. 2005. Direct Cache Access for High Bandwidth Network I/O. In *32st International Symposium on Computer Architecture (ISCA 2005), 4-8 June 2005, Madison, Wisconsin, USA.* IEEE, Madison, Wisconsin, USA, 50–59.

[4] Intel. 2012. Intel® Data Direct I/O Technology (Intel® DDIO): A Primer. https://web.archive.org/web/20210225132434/https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/data-direct-i-o-technology-brief.pdf

[5] Amit Kumar and Ram Huggahalli. 2007. Impact of Cache Coherence Protocols on the Processing of Network Traffic. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA.* IEEE, Chicago, IL, USA, 161–171.

[6] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, USA, February 26-28, 2019.* USENIX, Boston, MA, USA, 361–378.

[7] Ron Stein. 2021. Key drivers of 100Gbps network adoption. https://web.archive.org/web/20210416000644/https://www.datacenterdynamics.com/en/opinions/key-drivers-100gbps-network-adoption/

[8] Dan Tang, Yungang Bao, Weiwu Hu, and Mingyu Chen. 2010. DMA cache: Using on-chip storage to architecturally separate I/O data from CPU data for improving I/O performance. In *16th International Conference on High-Performance Computer Architecture (HPCA-16 2010), 9-14 January 2010, Bangalore, India.* IEEE, Bangalore, India, 1–12.

[9] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina J. Argyraki, Sylvia Ratnasamy, and Scott Shenker. 2018. ResQ: Enabling SLOs in Network Function Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018.* USENIX, Renton, WA, USA, 283–297.

[10] William A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News* 23, 1 (1995), 20–24.

[11] Yifan Yuan, Mohammad Alian, Yipeng Wang, Ren Wang, Ilia Kurakin, Charlie Tai, and Nam Sung Kim. 2021. Don't Forget the I/O When Allocating Your LLC. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021.* IEEE, Valencia, Spain, 112–125.