OmniXtend: Direct to Caches over Commodity Fabric

Marjan Radi
Western Digital Research
San Jose, USA
marjan.radi@wdc.com

Wesley W. Terpstra
SiFive, Inc
San Jose, USA
wesley@sifive.com

Paul Loewenstein
Western Digital Research
San Jose, USA
paul.loewenstein@wdc.com

Dejan Vucinic
Western Digital Research
San Jose, USA
dejan.vucinic@wdc.com

***Abstract*—There is a dearth of interfaces for efficient attachment of new kinds of non-volatile memory and purpose-built compute accelerators to processor pipelines. Early integrated microprocessors exposed an off-chip front-side bus to which discrete memory and peripheral controllers could attach in a standardized fashion. With the advent of symmetric multiprocessing and deep caches, this direct connection, together with memory controllers, has been implemented primarily using proprietary on-die technology. Proprietary interconnects and protocols hinder architectural innovation and are at odds with the open nature of the rapidly growing RISC-V movement.**

In this paper we introduce OmniXtend, a fully open coherence protocol meant to restore unrestricted interoperability of heterogeneous compute engines with a wide variety of memory and storage technologies. OmniXtend supports a four-hop MESI protocol and is designed to take advantage of a new wave of Ethernet switches with stateful and programmable data planes to facilitate system scalability. Ethernet transport was selected as a starting point for its ubiquity and historic resilience to reduce barriers to entry at modern bandwidths and latencies. Moreover, it allows us to build upon a vibrant ecosystem of hardware and IP, and to provide a boost to architectural innovation through the use of field-reconfigurable networking hardware. We briefly discuss the protocol operation and show performance measurements of the first ever NUMA RISC-V system prototype[1].

***Index Terms*—Cache Coherency, Shared Main Memory, Programmable Switches**

I. Introduction

The memory and computation resources of today's data centers are struggling to keep up with the explosion of memory and bandwidth requirements of ever-increasing big data and machine learning applications [1]. In different fields as varied as artificial intelligence, graph-processing, bioinformatics and in-memory databases, we commonly run into practical limitations dictated by the maximal available size of "main memory" [2]. Growing past these limitations, for instance through the use of RDMA-based architectures, requires the help of software to manage the moving of bits from durable storage into and out of main memory, as well as between the distant copies, which also need to be synchronized through other software mechanisms [3]. With the emergence of byte-addressable non-volatile memories that are denser and less expensive than DRAM, all this software execution overhead becomes excessive when compared with the latency of memory access [4], [5]. Therefore, a more thorough re-architecting is required to take full advantage of new technology and continue scaling. A cost impediment to scaling memory-centric applications is the effectively fixed ratio of memory-to-compute, dictated by the available SoC designs and the market dominance of small to medium sized NUMA systems. Optimizing system architecture to address this problem often requires tilting this ratio in favor of compute or memory, as well as the deployment of purpose-built accelerators instead of general-purpose pipelines. Alas, such undertakings are beyond fiscal reach of most enterprises as they would require reinvention of all the components of the system at once.

OmniXtend was motivated by the desire to break out of the status quo of prevailing system design and fueled by the urgent need of the RISC-V ecosystem for a common scale-out protocol. The auspicious timing of the emergence of P4 language [6] and programmable Ethernet switches [7] made the choice of transport easy, if imperfect. Our hope is that the innovation enabled by the openness will seed a robust ecosystem of components that interoperate easily through an unencumbered and widely available coherence protocol, providing synchronization and consistency in a very efficient fashion, both technically and economically.

The rest of this paper is organized as follows. Section II presents an outline of the OmniXtend protocol and system design. Next, we demonstrate our experimental setup for performance evaluation and discuss the measured performance in Section III. Related work is discussed in Section IV. Finally, we conclude in Section V.

II. OmniXtend System Design

OmniXtend aims to scale-out coherent caches for memory-centric architectures. Fig. 1 shows a high-level demonstration of a memory-centric architecture with OmniXtend. According to Fig. 1, OmniXtend allows large numbers of RISC-V compute nodes to connect to a shared and coherent memory pool. OmniXtend extends the SiFive's TileLink coherence protocol [8] to work beyond the processor chip. OmniXtend uses the programmability of modern switches to enable processors' caches to exchange coherence messages directly over the Ethernet fabric. In the following subsections, we first overview the characteristics of programmable switches used in our work.

[1]A sample implementation of OmniXtend for FPGA is available for download from https://github.com/westerndigitalcorporation/omnixtend

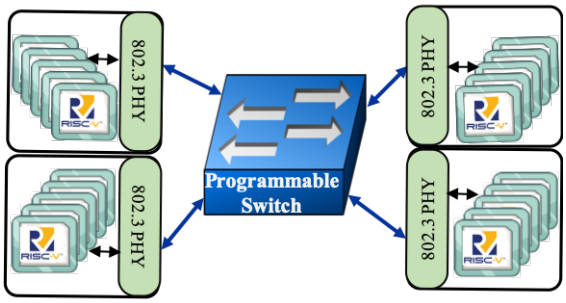

Fig. 1. Memory-centric architecture with OmniXtend

Then we present the details of the coherence protocol, as well as the switch data plane design.

A. Programmable Network Switches

Every programmable switch has fully programmable match-action pipelines to provide a reconfigurable data plane and customize packet processing capability [6]. The data plane program is responsible for expressing the packet processing behavior. In general, an incoming packet goes to a programmable parser to extract the custom headers required for match operation, then extracted header fields transit through multiple pipeline stages including match-action tables to perform matches based on predefined rules and to execute specific actions. Finally, a deparser unit assembles the packet to a stream of bytes and sends it out. This protocol-independent switch architecture of the programmable network devices enables us to push different networking/memory management algorithms to the programmable switches by defining customized packet formats and processing behavior (i.e., an off-the-shelf switch instead of NoC for coherence).

B. Providing Cache Coherence

In this section, we present how the coherence protocol in OmniXtend works. All of the read (i.e., get), write (i.e., put), and read-modify-write (i.e., atomic) operations on a shared address space are completed through two-stage request and response transactions between active participants called agents. There are two types of agents defined in the system: Master Agent, and Slave Agent. Master Agents can request Slave Agents for specific permission and performing read and/or write operations on different addresses. Slave Agents are responsible for managing the access permissions to different memory addresses, communicate with main memory if required, and respond to the original requester. The coherence policies are composed of permission transfer operation on cached copies. A Master Agent (i.e., cache controller) must first obtain necessary permissions on a specific memory block through transfer operations to perform read and/or write operations on that specific memory block. None, Read or Read+Write are the possible permissions that an agent can have to work on a copy of a memory block. As a result, a memory block may have different per-node states as follows:

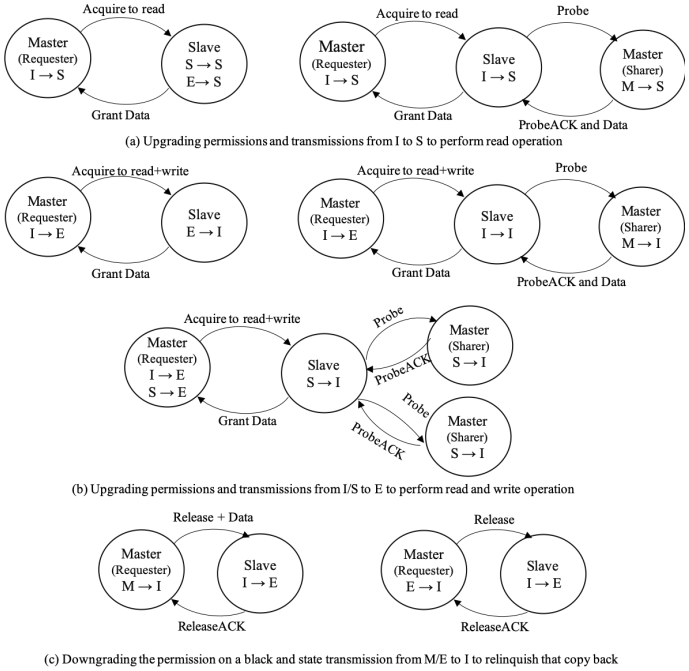

Fig. 2. High-level description of the coherence protocol.

- Modified: The node has the only valid cached copy which contains dirty data with read and write permission.
- Shared: The node has read-only permission on a cached copy. The block is valid, clean and other nodes may have a read-only copy.
- Exclusive: The node has read and write permission on a cached copy which is valid and clean.
- Invalid: The node has no permission on that block.

The following operations are available for performing read and/or write operation on the shared memory:

- Get: Read the data from a specified address.
- Put: Write the data at a specified address.
- Atomic: Atomically read a data value and write a new value that is the result of some arithmetic or logical operations.

Operations listed above are executed through exchanging coherence messages between Master and Slave Agents:

- Acquire: A request message from a Master Agent to obtain appropriate permissions on that address along with a copy of that data block.
- Grant Data: A response message from a Slave Agent to acknowledge the reception of an acquire and grant permission to the original requesting Master Agent along with a copy of the data block.
- Release: A request message from a Master Agent to downgrade its permissions on a cached copy.
- Probe: A request message from a Slave Agent to query, modify or revoke the permissions of a cached copy stored by a particular Master Agent.

Fig. 2 demonstrates the transactions which take place when a cache controller issues a read, write or read+write request

format	opcode	param	size	domain	source
3 bits	3 bits	3 bits	4 bits	3 bits	16 bits

Fig. 3. OmniXtend packet header format.

in order to acquire the correct permission and perform the requested operation. Transactions presented in Fig. 2 are relatively straightforward. According to Fig. 2(a) and Fig. 2(b), a read/write request issued by a cache controller (i.e., Master Agent) is composed of multiple transactions to acquire the correct permission to work on the target memory block. The requesting Master Agent sends an Acquire message to a Slave agent to obtain necessary permissions to read/write a memory block. The Slave Agent then probes other Masters to forcibly release their permissions on that block and write back any dirty cached copy. Upon reception of all the probe acknowledgment messages by the Slave Agent, it responds to the requesting Master with a Grant Message. Moreover, Fig. 2(c) shows the transactions which take place when a node issuing a release request to evict a cached block containing dirty data and release its related permissions in order to replace it with a new block.

C. Switch Data Plane Implementation

Fig. 3 shows the header format of the OmniXtend packets. This packet header includes the fields required for coherence protocol operations. The combination of these OmniXtend header fields encodes in every message the necessary information for coherence operations like the operation type, permission, memory address and data. OmniXtend messages are encoded as Ethernet packets, along with a standard preamble followed by a start frame delimiter. OmniXtend keeps the standard 802.3 L1 frame to interoperate with Barefoot Tofino [7] and future programmable switches, by replacing Ethernet header fields with coherence messages' fields. We have implemented a custom switch parser using p4 [6] to parse the OmniXtend packets' header. We also programmed the match-action units of the data plane to enable the switch to perform custom packet processing on OmniXtend messages. As a result, when the switch receives an OmniXtend packet, the customized parser will parse the packets and perform the match-action table lookup based on the extracted header fields. Finally, the packet is emitted to a specific output port according to the match-action unit's output.

III. Performance Evaluation

This section presents our experimental setup for performance evaluations, followed by the results achieved from the system evaluations.

A. Experimental Setup and Performance Evaluation Scenarios

Fig. 4 shows the experimental setup for system evaluations. According to this figure, we used two Xilinx VCU118 [9] with SiFive U54-MC Standard Core as the memory fabric [10]. All the memory and cache controllers' functionalities

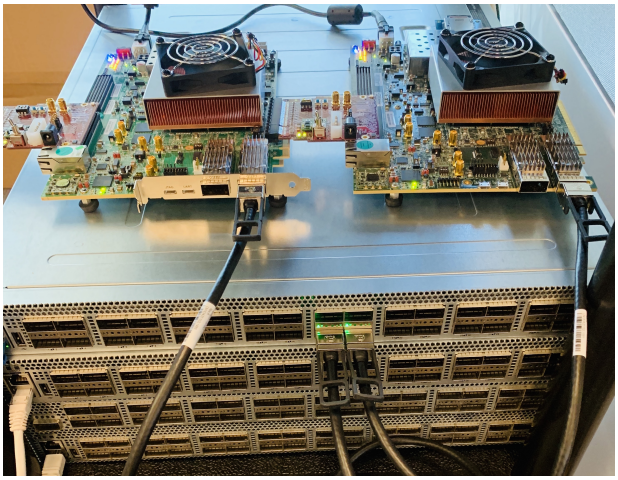

Fig. 4. Experimental Setup.

are implemented in the Xilinx VCU118 FPGAs. A 64-port ToR Barefoot Tofino switch [7], that forwards coherence messages, is located between memory endpoints. The two QSFP+ connectors of the VCU118 boards and Tofino ports are configured to run at 10Gb. We evaluated the OmniXtend performance in four different scenarios:

- Single Socket: The cores on a single VCU118 board issue random read and write requests in the incoherent mode for different memory block sizes.
- NUMA local memory: The cores on a single VCU118 board issue random read and write requests in the coherent mode for different memory block sizes.
- NUMA remote memory: The cores on each VCU118 board issue random read and write requests in the coherent mode for different memory block sizes hosted on the other board. In this case, two boards are directly connected using a QSFP+ cable.
- NUMA local memory through the switch: The cores on a single VCU118 board issue random read and write requests in the coherent mode for different memory block sizes. However, the cores on a single board exchange coherence messages through the Tofino switch.
- NUMA remote through the switch: The cores on each VCU118 board issue random read and write requests in the coherent mode for different memory block sizes hosted on the other board. In this case, two boards are communicating through the Tofino switch.

B. Experimental Results

Fig. 5 shows the memory read and write latencies. According to this figure, increasing the working set size increases the memory access latency, because at some point the data is fetched from L2 and DRAM. Moreover, local and remote memory-access latencies are almost the same, as the system wants to make sure about the coherency of all the copies of the requested block. This means that even for local accesses the coherence protocol runs between the cores of the single board. As it can be seen from Fig. 5, the Tofino switch adds

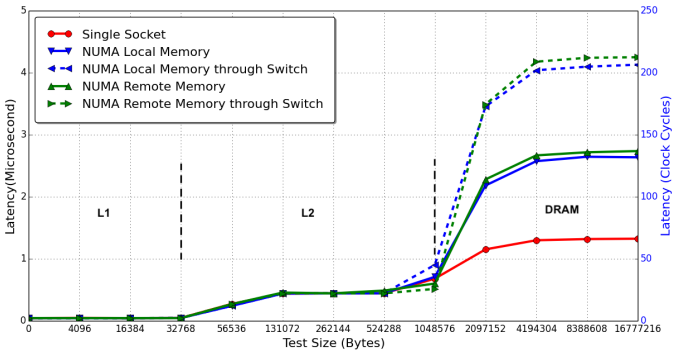

Fig. 5. Memory access latency.

about 1.2 μs latency for coherent memory accesses. These performance results present a great promise for the feasibility of scalable main memory systems.

IV. Related Work

The growing gap between CPU and memory performance demands efficient techniques to hide main memory access latency and full utilization of computation resources in order to support high-performance data processing for modern applications (e.g., distributed machine learning and big data analysis). Caching frequently used data is the most common technique to hide the performance gap between storage and computation device [11], [12]. However, caching has its challenges like maintaining cache coherence to ensure the coherence of cached copies at different machines under concurrent accesses [13]. Although, there have been a host of cache coherence protocols over the past few decades, none of them manage to provide a scalable coherent cache memory pool to a large number of nodes in a distributed data center environment [14], [15]. In other words, cache coherence is a well-researched area, but it is reviving once again in the age of fast and programmable networks. Providing distributed cache coherence over Ethernet fabric is feasible now, as memory and network latency are converging [16]. Unfortunately, the few existing coherent memory fabrics like CCIX [17] and Intel Omni-path [18] are designed for specific system architectures, and they do not provide a fully open coherence protocol meant to restore unrestricted interoperability of heterogeneous compute engines with a wide variety of memory technologies and hardware components. To the best of our knowledge, OmniXtend is the first attempt towards enabling a large number of compute nodes connected directly to a coherent shared cache pool by having processors' caches to exchange coherence messages directly over commodity fabric. OmniXtend is the first cache coherent memory technology providing an open-standard interface for memory access and data sharing across a large number of processors, FPGAs, GPUs, machine learning accelerators, and other components. Moreover, the programmability of OmniXtend switches allows any modification to coherence domains or protocols immediately deployable without requiring new ASIC design and manufacturing.

V. Conclusion

We present OmniXtend, a new open coherent memory technology which enables the exchange of coherence messages directly with processor caches over Ethernet fabric. This memory-centric architecture provides an open-standard interface for data access and sharing across processors, machine learning accelerators, GPUs, FPGAs, and other components. Moreover, OmniXtend offers potential support of future fabrics that connect I/O, storage, memory and compute components. OmniXtend has potential to scale-out with the number of agents by partitioning the global coherence domain to flexible coherence domains. Our system experiments show a great promise for scalable memory-centric systems.

References

[1] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17), Boston, 27-29 March 2017, pp. 649-667.

[2] B. Van Essen, R. Pearce, S. Ames and M. Gokhale, "On the Role of NVRAM in Data-intensive Architectures: An Evaluation," In IEEE 26th International Parallel and Distributed Processing Symposium, Shanghai, 2012, pp. 703-714

[3] A. Dragojevic, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast Remote Memory", In 11th USENIX Symposium on Networked Systems Design and Implementation, Seattle, Apr. 2014, pp. 401–414.

[4] D. Ustiugov, A. Daglis, J. Picorel, M. Sutherland, E. Bugnion, B. Falsafi, D. Pnevmatikatos, "Enabling Storage Class Memory as a DRAM Replacement for Datacenter Services", CoRR, 2018.

[5] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," Proc. IEEE, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," SIGCOMM Comput. Commun. Rev, vol. 44, no. 3, July 2014, pp. 87–95.

[7] Barefoot Tofino 2019.https://barefootnetworks.com/products/brief-tofino

[8] SiFive TileLink Specification, SiFive, Inc., Dec 2018

[9] VCU118 Evaluation Board, User Guide, Xilinx.

[10] SiFive U54-MC Core Complex Manual, SiFive Inc.

[11] S. Kumarand P. K. Singh, "An overview of modern cache memory and performance analysis of replacement policies," In 2016 IEEE International Conference on Engineering and Technology (ICETECH), 17-18 March 2016, Coimbatore, India, pp.210–214.

[12] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," In 26th Symposium on Operating Systems Principles (SOSP '17), Shanghai, China, 2017, pp. 121–136.

[13] Sorin, Daniel J., Hill, Mark D., and Wood, David A, "A Primer on Memory Consistency and Cache Coherence," Synthesis Lectures on Computer Architecture, vol. 6, pp. 1–212, 2011.

[14] Z. Al-Waisi and M. O. Agyeman, "An overview of on-chip cache coherence protocols," In 2017 Intelligent Systems Conference (IntelliSys), London, 7-8 Sept, 2017, pp. 304-309.

[15] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, "A Survey on Cache Management Mechanisms for Real-Time Embedded Systems", ACM Computing Surveys, vol. 48, no. 2, November 2015, pp. 32:1–32:36.

[16] M. K. Aguilera, N.bAmit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, "Remote memory in the age of fast networks," In 2017 Symposium on Cloud Computing, Santa Clara, California, 2017, pp. 121–127.

[17] CCIX Consortium, An Introduction to CCIX: White Paper, https://docs.wixstatic.com/ugd/0c1418_c6d7ec2210ae47f99f58042df0006c3d.pdf

[18] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, R. C. Zak, "Intel® Omni-path Architecture: Enabling Scalable, High Performance Fabrics," In 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, Santa Clara, CA, 2015, pp. 1-9.