Practical Memory Disaggregation using Compute Express Link

Donghyun Gouk, Sangwon Lee, Miryeong Kwon, Myoungsoo Jung
Computer Architecture and Memory Systems Laboratory,
Korea Advanced Institute of Science and Technology (KAIST)
http://camelab.org

***Abstract*—New cache coherent interconnects such as CXL have recently attracted great attention thanks to their excellent hardware heterogeneity management and resource disaggregation capabilities. Even though there is yet no real product or platform integrating CXL into memory disaggregation, it is expected to make memory resources practically and efficiently disaggregated much better than ever before.**

We propose directly accessible memory disaggregation, DIRECTCXL that directly connects a host processor complex and remote memory resources over CXL's memory protocol (`CXL.mem`). As there is no operating system that supports CXL, we also offer CXL software runtime that allows users to utilize the underlying disaggregated memory resources via sheer load/store instructions. Since DIRECTCXL does not require any data copies between the host memory and remote memory, it can expose the true performance of memory disaggregation to the users.

I. Memory Disaggregation and Its Challenge

Memory disaggregation has attracted great attention thanks to its high memory utilization, transparent elasticity, and resource management efficiency [2, 14, 15]. Many studies have explored various software and hardware approaches to realize memory disaggregation and put significant efforts into making it practical in large-scale systems.

We can broadly classify the existing memory disaggregation runtimes into two different approaches based on how they manage data between a host and memory server(s): i) page-based and ii) object-based. The page-based approach [1, 3, 9, 10, 13, 18, 24] utilizes virtual memory techniques to use disaggregated memory without a code change. It swaps page cache data residing on the host's local DRAMs from/to the remote memory systems over a network in cases of a page fault. On the other hand, the object-based approach handles disaggregated memory access from a remote-side using their own database such as a key-value store instead of leveraging the virtual memory systems [7, 8, 11, 17, 19, 23]. This approach can address the challenges imposed by address translation (e.g., page faults, context switching, and write amplification), but it requires significant source-level modifications and interface changes.

While aforementioned studies try to implement high-performance memory disaggregation system, they rely on network-based data exchange which can significantly deteriorate the performance of memory disaggregation. The network-based data exchange (e.g., RDMA) requires host software intervention and multiple memory copy operations

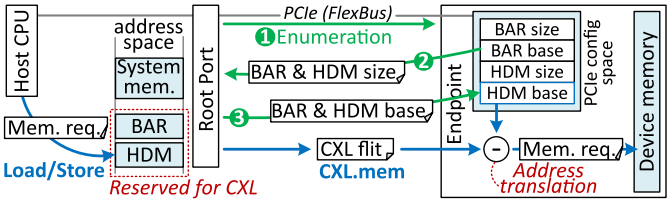


Fig. 1: DIRECTCXL's connection method.

for controlling network interface cards and DMA operations, respectively. Also, the network-based data exchange requires protocol/interface changes between network (e.g., InfiniBand) and host interface (e.g., PCIe). Our evaluation shows that the software intervention and copy operations consume 66% of end-to-end network-based data exchange (Section III).

II. Direct Accessible Memory Disaggregation

Recently, a new concept of open industry standard interconnect, *compute express link* (CXL [4]), is introduced which offering high-performance connectivity among multiple host processors, hardware accelerators, and I/O devices [6]. CXL is originally designed to achieve the excellency of heterogeneity management across different processor complexes, but both industry and academia anticipate its cache coherence ability can improve memory utilization and alleviate memory over-provisioning with low latency [12, 20, 21].

We demonstrate DIRECTCXL, direct accessible disaggregated memory that connects host processor complex and remote memory resources over CXL's memory protocol (`CXL.mem`). As CXL.mem allows the host computing resources directly access the underlying memory of CXL devices through PCIe buses (*FlexBus*), we design and implement CXL devices as pure passive modules, each being able to have many DRAM DIMMs with its own hardware controllers.

Integrating CXL devices into system memory. Figure 1 shows how CXL devices' internal DRAMs are mapped (exposed) to a host's memory space over CXL. The host CPU's system bus contains one or more CXL root ports (*RPs*), which connect one or more CXL devices as endpoint (*EP*) devices. Our host-side kernel driver first enumerates CXL devices by querying the size of their base address register (BAR) and their internal memory, called host-managed device memory (*HDM*), through PCIe transactions. Based on the retrieved sizes, the kernel driver maps BAR and HDM in the host's reserved system memory space and lets the underlying CXL devices

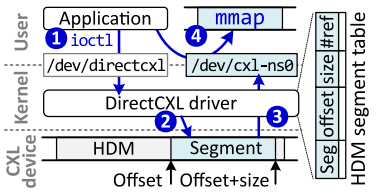
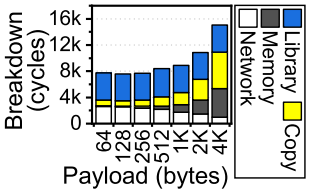
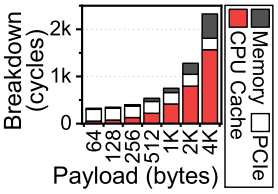


Fig. 2: Software runtime/driver of DIRECTCXL.



(a) RDMA.



(b) CXL.

Fig. 3: Performance comparison of load latency.

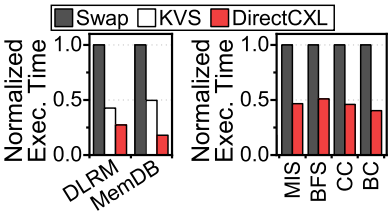


Fig. 4: Real workload performance.

know where their BAR and HDM (base addresses) are mapped in the host's system memory. When the host CPU accesses an HDM system memory through load/store instruction, the request is delivered to the corresponding RP, and the RP converts the requests to a CXL flit. Since HDM is mapped to a different location of the system memory, the memory address space of HDM is different from that of EP's internal DRAMs. Thus, the CXL controller translates the incoming addresses by simply deducting HDM's base address from them and issues the translated request to the underlying DRAM controllers. The results are returned to the host via a CXL switch and FlexBus. Note that, since HDM accesses have no software intervention or memory data copies, DIRECTCXL can expose the CXL device's memory resources to the host with low access latency.

Software Runtime for DIRECTCXL. While applications running on the host can directly access the CXL device by referring to HDM's memory space, it requires software runtime/driver to manage the underlying CXL devices and expose their HDM in the application's memory space. We thus support DIRECTCXL runtime that simply splits the address space of HDM into multiple segments, called *cxl-namespace*. DIRECTCXL runtime then allows the applications to access each CXL-namespace as memory-mapped files (`mmap`).

Figure 2 shows the software stack of our runtime and how the application can use the disaggregated memory through cxl-namespaces. When a CXL device is detected (at a PCIe enumeration time), DIRECTCXL driver creates an entry device (e.g., `/dev/directcxl`) to allow users to manage a cxl-namespace via `ioctl`. If users ask a cxl-namespace, the driver checks a (physically) contiguous address space on an HDM by referring to its HDM segment table whose entry includes a segment's offset, size, and reference count (recording how many cxl-namespaces use this segment). Once DIRECTCXL driver allocates a segment based on the user request, it creates a device for `mmap` (e.g., `/dev/cxl-ns0`) and updates the segment table. The user application can then map the cxl-namespace to its process virtual memory space using `mmap`.

III. EVALUATIONS AND CONCLUSION

We built all DIRECTCXL IPs from the ground and configure many customized FPGA add-in-cards and high-performance datacenter accelerator cards to implement CXL network topology for memory disaggregation. As yet there is no processor architecture supporting CXL, we also build our own in-house host processor using RISC-V ISAs, which employs four out-of-order cores whose last-level cache (*LLC*) implements CXL RP. Our in-house softcore processors work at 100MHz while CXL and PCIe IPs (RP, EP, and Switch) operate at 250MHz.

Microbenchmark. Figure 3a decomposes RDMA latency into essential hardware (`Memory` and `Network`), software (`Library`), and data copy latencies (`Copy`). `Library` is the primary performance bottleneck in RDMA when the size of payloads is smaller than 1KB (53.3%, on average). As the payloads increase, `Copy` gets longer and reaches 37.3% of total execution time. This is because users must copy all their data into RNIC's MR, which takes extra overhead in RDMA. The `Library` and `Copy` consumes 46.2% and 19.8% of end-to-end latency, on average, respectively. In contrast, as shown in Figure 3b, the breakdown analysis for `DirectCXL` shows a completely different story. As DIRECTCXL allows host to access remote memory resources using sheer load/store instruction, there is neither software nor data copy overhead.

Real workloads. For the real workload evaluation, we used DLRM [16], in-memory database (MemDB [11]) and four graph analysis from Ligra [22]. Figure 4 shows the execution latency of `Swap` (FastSwap [2]), `KVS` (HERD [11]), and `DirectCXL`. For Ligra, we exclude `KVS` because Ligra's graph processing is not compatible with a key-value structure.

As shown in the figure, `Swap` shows worst performance as it does not understand workload's data access characteristics. `KVS` can reduce the latency of `Swap` as it can fine-control where the data is placed and removes the overhead imposed by page-based memory management. However, it has two major issues: First, it requires significant modification of the application's source codes, which is often impossible (e.g., MIS, BFS, CC, BC). Second, `KVS` requires heavy computation such as hashing at the memory node, which increases monetary costs. In contrast, `DirectCXL` without having a source modification and remote-side resource exhibits 3× and 2.2× better performance than `Swap` and even `KVS`, respectively.

We propose DIRECTCXL that connects host and remote memory resources over CXL's memory protocol. The results of our real system evaluation show that DIRECTCXL exhibits 3× better performance than conventional memory disaggregation, on average, for real-world workloads.

IV. FUTURE WORK AND ACKNOWLEDGEMENT

We are extending both software and hardware parts of this work to: i) Integrating remote memory exposed by CXL to NUMA subsystem, so that user can use CXL memory without source code modification. ii) Extending our in-house CXL IPs towards CXL 3.0, supporting new features such as dynamic

capacity [5], and iii) SoC silicon fabrication. Myoungsoo Jung is the corresponding author (mj@camelab.org).

V. Demo and Original Publication

Demo video. https://youtu.be/6a5NSMH-7hY

Original publication. D. Gouk, S. Lee, M. Kwon and M. Jung. *Direct Access, High-Performance Memory Disaggregation with* DirectCXL, in USENIX ATC 2022. https://www.usenix.org/system/files/atc22-gouk.pdf

References

[1] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, S. Novakovic, A. Ramanathan, P. Subrahmanyam, L. Suresh, K. Tati *et al.*, "Remote regions: a simple abstraction for remote memory," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 775–787.

[2] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, "Can far memory improve job throughput?" in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[3] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 79–92.

[4] CXL Consortium, "Compute Express Link Specification Revision 2.0."

[5] ——, "Compute Express Link Specification Revision 3.0."

[6] ——, "Compute Express Link™ 2.0 White Paper," https://www.computeexpresslink.org/_files/ugd/0c1418_14c5283e7f3e40f9b2955c7d0f60bebe.pdf.

[7] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[8] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro, "No compromises: Distributed transactions with consistency, availability, and performance," in *Proceedings of the 25th symposium on operating systems principles*, 2015, pp. 54–70.

[9] 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)*, 2017, pp. 649–667.

[10] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 417–433.

[11] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using rdma efficiently for key-value services," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 295–306.

[12] P. Kennedy, "Compute Express Link or CXL What it is and Examples," https://www.servethehome.com/compute-express-link-or-cxl-what-it-is-and-examples/.

[13] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee, "Mind: In-network memory management for disaggregated data centers," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 488–504.

[14] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 2012, pp. 1–12.

[15] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu, "Memory disaggregation: Research problems and opportunities," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1664–1673.

[16] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: https://arxiv.org/abs/1906.00091

[17] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-tolerant software distributed shared memory," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 291–305.

[18] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, "Thymesisflow: a software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 868–880.

[19] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "Aifm: High-performance, application-integrated far memory," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 315–332.

[20] D. D. Sharma, "CXL: Coherency, Memory, and I/O Semantics on PCIe Infrastructure," https://www.electronicdesign.com/technologies/embedded-revolution/article/21162617/cxl-coherency-memory-and-io-semantics-on-pcie-infrastructure.

[21] N. Shenoy, "A Milestone in Moving Data," https://newsroom.intel.com/editorials/milestone-moving-data.

[22] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2013, pp. 135–146.

[23] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 33–48.

[24] C. Wang, H. Ma, S. Liu, Y. Li, Z. Ruan, K. Nguyen, M. D. Bond, R. Netravali, M. Kim, and G. H. Xu, "Semeru: A memory-disaggregated managed runtime," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 261–280.