CXL 인터커넥트 기술 연구개발 동향

Trends in Compute Express Link(CXL) Technology

김선영 (S.Y. Kim, seonyoung8436@etri.re.kr) 슈퍼컴퓨팅기술연구센터 연구원
안후영 (H.Y. Ahn, ahnhy@etri.re.kr) 슈퍼컴퓨팅기술연구센터 선임연구원
박유미 (Y.M. Park, parkym@etri.re.kr) 슈퍼컴퓨팅기술연구센터 책임연구원/센터장
한우종 (W.J. Han, woojong.han@etri.re.kr) 슈퍼컴퓨팅기술연구센터 책임연구원/연구위원

ABSTRACT

With the widespread demand from data-intensive tasks such as machine learning and large-scale databases, the amount of data processed in modern computing systems is increasing exponentially. Such data-intensive tasks require large amounts of memory to rapidly process and analyze massive data. However, existing computing system architectures face challenges when building large-scale memory owing to various structural issues such as CPU specifications. Moreover, large-scale memory may cause problems including memory overprovisioning. The Compute Express Link (CXL) allows computing nodes to use large amounts of memory while mitigating related problems. Hence, CXL is attracting great attention in industry and academia. We describe the overarching concepts underlying CXL and explore recent research trends in this technology.

KEYWORDS CXL, Near-Memory Processing, 메모리 확장기술, 차세대 인터커넥트

Ⅰ. 서론

현대의 컴퓨팅 시스템에서 처리되는 데이터의 크기는 기계학습, 대규모 데이터베이스와 같은 데이터 집약적 작업이 널리 사용됨에 따라 기하급수적으로 증가하고 있다. 일례로, 최근 많은 주목을 받는 ChatGPT에서 사용하는 대규모 언어모델인 GPT-3.5에서 사용하는 학습 파라미터의 수는 1,750억 개에 달한다[1]. 이러한 데이터 집약적 작업은 대량의 데이터를 빠르게 처리하고 분석하기 위해 컴퓨팅 시스템에 대용량의 메모리를 요구한다는 특징을 가진다.

하지만, 기존의 컴퓨팅 시스템 구조에서는 대용량 메모리 시스템을 구축하는 데 여러 제약사항이 있다. 우선, 대용량의 메모리를 컴퓨팅 노드에서 사용하기 위해서는 CPU(Central Processing Unit)의 하드웨어적 특성(메모리 컨트롤러 및 채널의 수 등)이 제약사항으로 작용하며[2], 이는 CPU의 면적 및 전력

* DOI: https://doi.org/10.22648/ETRI.2023.J.380503
* 본 연구 논문은 한국연구재단 슈퍼컴퓨터개발선도사업[2021M3H6A1017683, 초병렬프로세서 기반 고집적 컴퓨팅 노드 및 시스템 개발]의 일환으로 수행되었음.

소모 증가 등 치명적인 비용을 요구한다. 또한, 최대치의 메모리 요구량을 기준으로 컴퓨팅 시스템을 구축하는 경우 대부분의 런타임 동안 상당한 양의 메모리가 사용되지 않는 메모리 오버프로비저닝(Memory Overprovisioning)이 발생한다. 이와 관련하여, Microsoft Azure의 연구[3]는 단일 컴퓨팅 노드에서 CPU 코어가 모두 User VM(Virtual Machine)으로 할당된 상태에서 메모리는 그렇지 않은 경우, 최대 25%의 메모리가 사용되지 않고 있음을 보였다.

컴퓨팅 시스템이 이러한 제약사항들을 극복하며 대용량의 메모리를 활용하기 위해서는 1) CPU의 하드웨어적 특성에 의한 한계 이상으로 메모리 용량을 확장하는 방법 및 2) 사용되지 않는 메모리를 유연하게 활용할 수 있는 방법이 필요하다. 1)을 위해서는 컴퓨팅 노드가 하드웨어 인터커넥트를 통해 연결된 메모리를 사용할 수 있도록 하는 메모리 확장기술을 채택할 수 있으며[4], 2)를 위해서는 네트워크를 통해 연결된 메모리를 컴퓨팅 노드들이 접근하여 필요한 만큼 쓸 수 있도록 하는 메모리 분리기술(Memory Disaggregation)을 채택할 수 있다[5].

CXL(Compute eXpress Link)은 확장 메모리 또는 가속기(예: GPGPU) 등 다양한 주변장치를 효율적으로 연결하기 위해 제안된 컴포저블(Composable) 하드웨어 인터커넥트이다[6]. CXL은 1) 주변장치의 확장 메모리에 호스트 프로세서가 load/store 명령을 통해 메모리 시맨틱(Memory Semantic)하게 접근하여 확장 메모리를 자신의 로컬 메모리처럼 사용할 수 있게 하며, 2) CXL 스위치 및 fabric을 통해 메모리 분리기술을 적용하여 여러 노드가 메모리를 유연하게 할당받고 관리할 수 있도록 한다. CXL의 이와 같은 특징은 대용량의 메모리가 필요한 컴퓨팅 시스템에 적합한 하드웨어 인터커넥트로써 지대한 관심을 받고 있다.

본고에서는 CXL에 대한 전반적인 내용 및 CXL 관련 최신 연구개발 동향을 소개한다. 본고의 구성은 다음과 같다. Ⅱ장에서는 CXL에 대한 전반적인 내용을 살펴본다. Ⅲ장에서는 CXL을 지원하는 프로세서 및 장치들을 소개하고, CXL 관련 최신 연구동향을 소개하며 CXL 연구개발 동향을 살펴본다. 끝으로, Ⅳ장에서 전체적인 내용을 요약하며 본고를 마무리한다.

Ⅱ. CXL

CXL은 CPU와 주변장치 간 직렬 통신을 위한 PCIe 기반 하드웨어 인터커넥트이다. CXL은 2019년에 인텔로 대표되는 x86 진영에서 발족한 CXL 컨소시엄에 의해 처음 제안되었으며, 현재 3.0 버전까지 공개되었다[7]. CXL 이전에 ARM 진영의 CCIX[8]와 같이 유사한 인터커넥트들이 먼저 제안되었지만, CXL은 x86 아키텍처의 서버 시장 우위를 바탕으로 빠르게 생태계를 구축해 나가며 차세

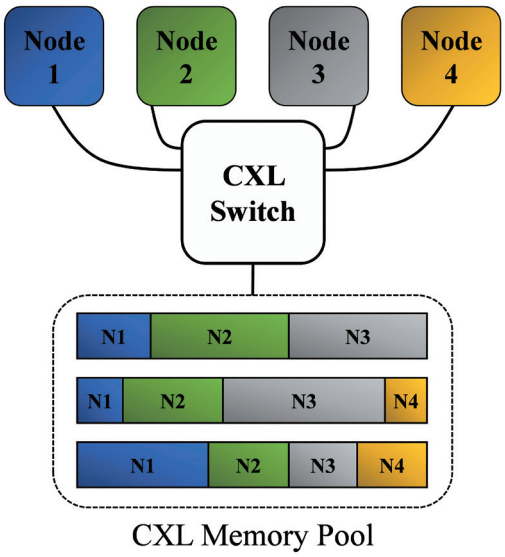

그림 1 CXL Memory Pooling

대 인터커넥트 시장에서 선두를 달리고 있다.

CXL은 호스트 프로세서와 주변장치 간 PCIe 대비 유연한 메모리 접근 메커니즘 제공을 통해 컴퓨팅 노드의 메모리 용량 및 대역폭을 효율적으로 확장하여 구조적 한계를 극복하도록 한다. 예를 들어, CXL은 서버 혹은 데이터 센터 내 메모리 자원에 대해 코어 간 또는 그림 1과 같이 컴퓨팅 노드 간 공유 환경을 제공하여 메모리 자원의 효율적 사용을 가능하게 한다. 이 장에서는 CXL 표준 문서[6]를 기반으로 CXL 프로토콜의 구조, CXL 장치의 유형, 그리고 CXL 버전별 특징을 설명하며 CXL에 대한 전반적인 내용을 소개한다.

1. CXL 프로토콜 구조

CXL 프로토콜은 다음과 같은 세 가지 서브 프로토콜로 이루어져 있다(그림 2): CXL.io, CXL.cache, CXL.mem. 각 서브 프로토콜에 대한 설명은 다음과 같다.

- CXL.io: CXL.io 서브 프로토콜은 장치 탐색(Device Enumeration), DMA(Direct Memory Access) 등 PCIe와 유사한 기능을 제공하는 서브 프로토콜이다. 또한, PCIe의 기능 외에도 CXL.io에서 추가된 기능을 통해 CXL 장치 접근 시 PCIe 대비 낮은 접근 지연 시간을 보장한다. CXL.io는 모든 CXL 장치 유형에서 공통으로 사용되는 서브 프로토콜이다.
- CXL.cache: CXL.cache 서브 프로토콜은 호스트의 캐시 메모리와 CXL 장치의 캐시 메모리 간 일관성을 유지하는 데 사용된다. CXL.cache는 비대칭형 프로토콜로서 호스트 프로세서가 캐시 일관성에 대한 책임을 지고 캐시 일관성이 훼손되는 경우 snoop 트랜잭션을 통해 일관성을 유지하는 구조이다. 이때, 일관성 유지 프로토콜은 MESI(Modified, Exclusive, Shared, Invalid)를 사용한다.
- CXL.mem: CXL.mem 서브 프로토콜은 호스트 프로세서가 캐시 라인 단위(i.e., 64byte)로 메모리 접근법(load/store)을 통해 주변장치의 메모리에 접근할 수 있게 하는 기능을 제공한다. 이를 통해 호스트 프로세서는 주변장치의 메모리를 자신의 로컬 메모리처럼 사용할 수 있다.

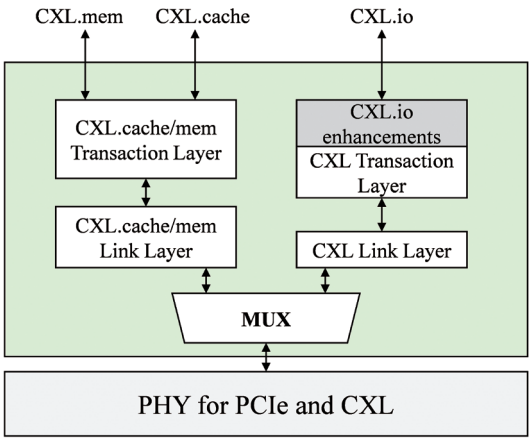

그림 2 CXL 프로토콜 구조

CXL 프로토콜 구조는 그림 2와 같다. CXL의 물리 계층은 PCIe의 PHY 계층에 CXL logical PHY를 추가한 형태이다. 물리 계층의 상단에는 CXL.io 프로토콜의 트랜잭션과 CXL.mem, CXL.cache 프로토콜의 트랜잭션을 구분하기 위한 MUX가 위치하며, MUX 상단에는 각 서브 프로토콜에 해당하는 링크 계층과 트랜잭션 계층이 위치한다.

2. CXL 장치 유형

CXL 장치는 장치가 사용하는 서브 프로토콜의 조합에 따라 그림 3과 같이 총 세 가지의 장치 유형

으로 구분할 수 있다.

우선, 유형 1 CXL 장치는 호스트 CPU의 캐시 메모리와 coherent한 캐시를 가지지만, 호스트가 관리하는 메모리를 가지지는 않는 장치 유형을 뜻한다. 유형 1 CXL 장치의 대표적인 예시로는 NIC(Network Interface Card) 및 호스트가 관리하는 메모리를 가지지 않는 가속기를 들 수 있다. 유형 1 CXL 장치에서 사용되는 CXL 서브 프로토콜은 CXL.io 및 CXL.cache의 D2H(Device to Host) 요청 및 H2D(Host to Device) snoop 트랜잭션 등이 있다.

유형 2 CXL 장치는 CXL.io, CXLcache, CXL.mem 세 가지 서브 프로토콜을 모두 사용하며, 호스트와 fully coherent한 캐시를 가지는 동시에 호스트가 관리하는 장치 메모리를 가지는 유형이다. 유형 2 CXL 장치의 예시로는 HBM(High-Bandwidth Memory) 같은 메모리를 내장한 GPU(Graphic Processing Unit) 카드 또는 메모리를 내장한 FPGA(Field Programmable Gate Array) 보드를 예로 들 수 있다. 유형 2 CXL 장치는 호스트의 메모리에 coherent하게 접근할 수 있으며, 호스트 프로세서 또한 장치 메모리에 coherent 또는 non-coherent하게 접근할 수 있다. 유형 2 CXL 장치는 가속기 유닛과 장치 메모리 간 넓은 대역폭을 이용하여, atomic 연산만 가능한 유형 1 CXL 장치 대비 더욱 복잡한 연산을 처리할 수 있다[2].

마지막으로, 유형 3 CXL 장치는 CXL.io와 CXL.mem 서브 프로토콜을 지원하며, 호스트가 관리하는 메모리로 구성된 장치 유형이다. 유형 3 CXL 장치의 대표적인 예시로는 DRAM(Dynamic Random Access Memory) 또는 비휘발성 메모리(예: Flash 메모리)로 구성된 메모리 확장장치를 들 수 있다. 유형 3 CXL 장치의 확장 메모리는 이기종 메모리들로 구성될 수 있으며(예: 휘발성 메모리 + 비휘발성 메모리), 호스트 프로세서는 해당 메모리에 load/store 명령을 통해 메모리 시맨틱하게 접근하여 이를 NUMA(Non Uniform Memory Access) 노드의 메모리로써 자신의 로컬 메모리처럼 사용할 수 있다.

3. CXL 버전별 특징

CXL 표준은 2019년 인텔에서 제안한 1.0 버전을 시작으로, 현재 1.1, 2.0, 그리고 3.0 버전까지 공개되었다.

우선, CXL 1.0 및 1.1 버전의 경우 이기종의 자원 간 유연한 연결 및 메모리 용량/대역폭 확대 수요를 충족시키기 위해 등장했다. 이들은 PCIe 5.0의 기능에 더해, 일관성 유지 메커니즘 및 메모리 시맨틱 접

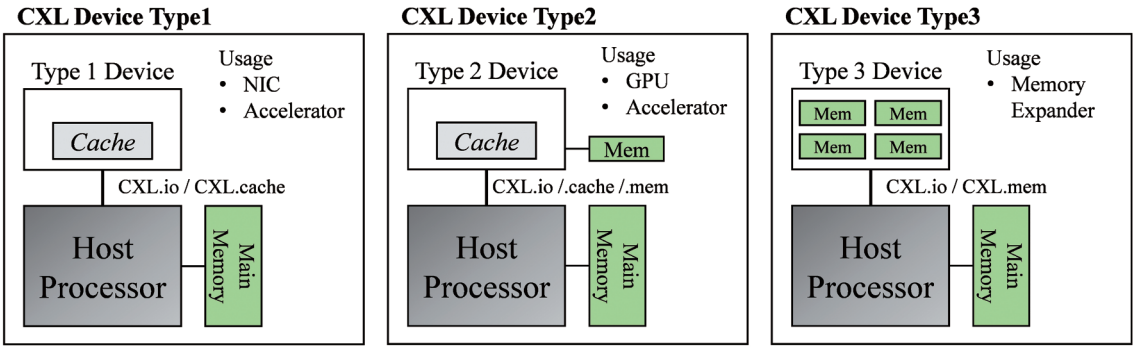

그림 3 CXL 장치 유형

근 메커니즘을 추가한 것을 가장 큰 특징으로 한다.

다음으로 CXL 2.0 버전의 경우 여러 컴퓨팅 노드들이 pooling 된 자원에 CXL 스위치를 통해 접근하는 메커니즘을 제공함으로써 메모리 또는 가속기를 pooling하여 사용할 수 있도록 하였다. 이를 통해 메모리 또는 가속기를 단일 컴퓨팅 노드가 과다하게 구성할 필요 없이 pooling 된 자원을 활용하여 일시적인 수요에 대응할 수 있다. 이는 메모리 오버프로비저닝과 같은 문제를 효과적으로 해결할 수 있는 수단으로, 데이터 센터와 같이 다수의 컴퓨팅 노드들을 운영하는 주체의 TCO(Total Cost of Ownership)를 크게 줄일 수 있다. 또한, CXL 2.0 버전의 pooling 기능은 기존의 단일 컴퓨팅 노드 수준에서의 메모리 시맨틱 접근 메커니즘을, 랙(Rack) 수준의 메모리 시맨틱 접근이 가능하도록 기능을 확장하였다.

마지막으로, CXL 3.0 버전은 PCIe 5.0을 기반으로 한 이전 버전들과 달리 PCIe 6.0을 기반으로 하며, PAM4[9] 신호 전송 메커니즘을 도입하여 기존 대비 통신 대역폭을 두 배로 증가시킨 동시에 같은 통신 지연 시간을 제공한다. 또한, CXL 3.0 버전은 CXL fabric 및 개선된 라우팅 메커니즘을 통해 최대 4,096개의 호스트 프로세서와 CXL 장치 간 연결을 가능하게 한다. 기존 CXL 2.0 버전에서는 단일 계층으로만 CXL 토폴러지를 구성할 수 있었지만, CXL 3.0 버전에서는 스위치 간 연결이 가능하여 다계층의 스위치 토폴러지 구성이 가능하다. 이를 통해 CXL 2.0 버전에서 지원하는 랙 수준의 연결 규모를 더욱 확장할 수 있다.

Ⅲ. CXL 연구개발 동향

이 장에서는 CXL을 지원하는 프로세서 및 장치들을 소개한 뒤, CXL 관련 최신 연구들을 1) 유형 3 CXL 장치 관련 연구, 2) NDP(Near Data Processing) 기능을 갖춘 CXL 장치 관련 연구, 3) CXL 관련 소프트웨어 연구로 나누어 소개하며 CXL 연구개발 동향을 살펴본다.

1. CXL 지원 프로세서 및 장치

CXL 표준이 제정된 이후 산업계에서는 PoC(Proof of Concept) 수준의 제품에서부터 상용 제품에 이르기까지 CXL을 지원하는 다양한 제품이 출시되고 있다. 이 절에서는 CXL을 지원하는 하드웨어들을 소개하고자 한다.

우선, CXL을 지원하는 호스트 프로세서로는 인텔의 서버용 프로세서인 Xeon 시리즈와 AMD의 서버용 프로세서인 EPYC 시리즈가 있다. 표 1은 CXL

표 1 CXL 지원 호스트 프로세서 목록 및 관련 Specification

	Intel Xeon 시리즈 CXL 관련 Specification				AMD EPYC 시리즈 CXL 관련 Specification			
제품명	Sapphire Rapids	Emeralds Rapids	Granite Rapids	Diamond Rapids	GENOA	BERGAMO	TURIN	VENICE
세대	4th Gen Xeon	5th Gen Xeon	6th Gen Xeon	7th Gen Xeon	Zen 4	Zen 4C	Zen5	Zen 6?
출시 시기	2023.01	2023 (예정)	2024 (예정)	2025+ (예정)	2022.11	2023 (예정)	2024 (예정)	2025+ (예정)
메모리 지원	8x DDR5-4800	8x DDR5-5600	8x DDR5	8x DDR5	8x DDR5-4800	12x DDR5-4800	8x DDR5-6000?	8x DDR5-6000?
PCIe 지원	PCIe 5/4	PCIe 5.0	PCIe 5.0	PCIe 6.0	PCIe 5.0	PCIe 5.0	TBD	TBD
CXL 지원	CXL 1.1 (Type 1,2)	CXL 1.1	CXL 2.0	CXL 3.0	CXL 1.1+ (Type 3)	CXL 1.1+	TBD	TBD

을 지원하는 프로세서 목록을 정리한 표이다. 표 1의 Sapphire Rapids[10]와 Genoa[11]가 현재 상용으로 출시된 호스트 프로세서이며, 두 프로세서 모두 CXL 1.1을 지원한다.

Sapphire Rapids의 경우 유형 1 및 유형 2 CXL 장치를 지원하며, 메모리 확장 기능을 제공하는 유형 3 CXL 장치는 공식적으로는 지원하지 않는다. 하지만, 지금까지 개발된 유형 3 CXL 장치(예: CXL Memory Expander[12])들을 테스트한 환경 대다수가 Sapphire Rapids였음을 고려하면, 유형 3 CXL 장치를 지원하도록 설계되었지만, 안정성 문제 등으로 인해 공식적으로는 지원하지 않는다고 발표한 것으로 보인다. Genoa의 경우 유형 3 CXL 장치만을 지원하며, CXL 1.1의 메모리 확장 기능에 더해 2.0의 일부 기능(Tiered Memory 기능, 비휘발성 메모리에 대한 메모리 확장 기능)을 추가로 지원한다. CXL 2.0 및 3.0을 지원하는 호스트 프로세서에 대한 대략적인 계획은 표 1에서 확인할 수 있으며, 2024년 하반기에 이르러서야 CXL 2.0을 지원하는 호스트 프로세서가 출시될 것으로 보인다.

다음으로 CXL 장치들을 소개한다. 현재 가장 활발하게 개발되고 있는 CXL 장치 유형은 메모리 확장 기능을 제공하는 유형 3 CXL 장치이다. 유형 3 CXL 장치는 주로 메모리 벤더들에 의해 개발되고 있으며, 대표적으로 삼성전자의 CXL Memory Expander[12], SK 하이닉스의 CXL 2.0 Memory[13], 그리고 Micron의 CXL Memory Expansion Module[14]이 있다. 해당 제품들은 모두 CXL 2.0 버전을 지원하며, 제품에 따라 96GB에서 512GB의 확장 메모리 용량을 지원한다. 세 가지 제품 모두 대용량 스토리지에서 쓰이는 EDSFF(Enterprise & Data Center Standard Form Factor) 폼팩터를 사용하며, 이를 통해 컴퓨팅 노드의 PCIe 슬롯에 add-in 카드 형태로 제품을 장착하여 손쉽게 컴퓨팅 노드의 메모리 용량을 확장할 수 있다. 삼성전자와 SK 하이닉스에서는 자사 CXL 메모리 확장장치를 위한 SDK(Software Development Kit)를 개발하였다[2]. 현재 삼성전자의 SDK는 깃허브를 통해 공개되어 있으며[15], SK 하이닉스의 SDK 또한 가까운 시일 내에 공개될 것으로 보인다.

또 다른 유형 3 CXL 장치로는 Astera Labs의 Aurora[16]가 있다. Aurora는 CXL 컨트롤러가 배치된 PCB 위에 DIMM 슬롯 4개가 있는 add-in 카드 형태로, CXL 2.0 버전을 지원하면서 최대 2TB의 확장 메모리를 지원할 수 있다. 메모리 벤더에서 공개한 메모리 유형 3 CXL 장치들의 경우 아직 상용화되지 않았지만, Aurora의 경우 상용 제품으로 현재 구매가 가능하다는 특장점이 있다.

CXL 2.0 및 3.0 버전의 주요 특징인 리소스 pooling을 위해서는 CXL 스위치가 필요하다. CXL 스위치를 구현한 제품으로는 XConn사에서 개발한 Apollo[17]가 있다. Apollo는 CXL 2.0과 PCIe 5.0을 지원하는 스위치로서 유형 2 및 유형 3 CXL 장치를 연결하기 위한 256개의 레인과 32개의 포트를 제공한다.

상기 소개한 CXL 장치들 외 PoC 수준에서 CXL의 기능을 검증하고 활용하기 위한 다양한 제품이 논문을 통해 공개되고 있으며, 이들에 대한 설명은 Ⅲ장 2절에서 후술한다.

2. CXL 관련 최신 연구 동향

가. 유형 3 CXL 장치 관련 연구

유형 3 CXL 장치를 통해 CXL 메모리에 접근하는 경우, 네트워크에 연결된 원격 메모리에 접근하는 경우 대비 매우 낮은 지연 시간을 보인다(200ns vs 4μs). 하지만, 여전히 메인 메모리에 접근할 때 발생하는 지연 시간(100ns)에 비하면 많은 시간이 소요된

다[18]. 이처럼 CXL 기반 확장 메모리를 컴퓨팅 노드의 메인 메모리와 함께 사용하는 경우, 상대적으로 느린 접근 시간으로 인해 성능 저하가 발생할 수 있다. 최근에는 이러한 성능 저하 수준을 확인하고 CXL 메모리를 효과적으로 활용하기 위한 다양한 연구들이 진행되고 있다.

선행연구[19]에서는 데이터 집약적 작업을 수행할 때 발생하는 메모리 부족 문제 및 이와 동시에 발생할 수 있는 메모리 오버프로비저닝 문제를 지적하면서, CXL 메모리 풀을 사용하는 환경에서 실시간 메모리 요구량에 따라 동적으로 CXL 메모리를 할당하는 DCS(Dynamic Capacity Service)를 제안했다. 이들은 CXL 메모리의 할당 상태 등을 기록하고 관리하기 위한 DCS 엔진을 FPGA를 통해 구현하였고, DCS를 사용하기 위한 SW 프레임워크 및 API(Application Programming Interface)를 구현하였다. CXL 메모리 풀에 연결된 컴퓨팅 노드들에 정적으로 CXL 메모리를 할당한 경우와 DCS를 통해 노드의 메모리 요구량에 따라 동적으로 CXL 메모리를 할당하는 경우 간 메모리 사용률을 비교했을 때, DCS를 사용하는 경우 20% 높은 메모리 사용률을 보였다.

선행연구[20]에서는 CXL.mem 프로토콜을 통해 호스트 프로세서와 CXL 메모리 풀을 연결하는 기술인 DirectCXL을 제안하였다. 이들은 RISC-V를 기반으로 CXL을 지원하는 호스트 프로세서, FPGA를 기반으로 한 CXL 컨트롤러, 그리고 CXL 스위치를 설계 및 구현하여 환경을 구성하였다. 이를 통해 호스트 프로세서가 CXL 메모리 풀에 접근하여 이를 필요한 만큼 자신의 로컬 메모리처럼 활용할 수 있도록 하였다. 딥러닝 및 인-메모리 데이터베이스 벤치마크 등을 활용한 DirectCXL과 RDMA(Remote Direct Memory Access) 기반 메모리 시스템 간 비교 결과에 따르면, DirectCXL은 RDMA 기반 시스템 대비 3배 높은 성능을 보였다. 이는 DirectCXL이 1) 네트워크 인터페이스(예: InfiniBand)와 PCIe 간 인터페이스 변환 작업이 필요한 RDMA 기반 시스템과 달리, 호스트 프로세서와 원격 메모리가 CXL을 통해 직접 연결되었고, 2) DMA를 통해 메모리에 읽기/쓰기 작업을 수행해야만 하는 RDMA 기반 시스템과 달리 CXL.mem을 통해 낮은 접근 지연 시간으로 원격 메모리에 접근하기 때문이라고 저자들은 설명하였다.

상기 연구들 외에도 클라우드 환경에서 CXL 메모리 풀을 활용하는 경우 발생하는 성능 저하 수준 및 비용을 분석하고, CXL 메모리 풀을 효과적으로 클라우드에 적용하기 위한 연구 또한 수행되고 있다. 선행연구[21]에서는 CXL 메모리 풀을 클라우드 환경에 적용할 때 메모리 풀에 연결된 호스트 CPU의 개수에 따라 1) 지연 시간, 2) CXL 메모리 풀 활용에 따른 DRAM 비용 절약 수준, 그리고 3) CXL 메모리 풀 활용에 따른 시스템 비용 절약 수준 측면에서 분석을 진행하였다. 해당 연구의 분석 결과를 요약하면 다음과 같다: 1) 심각한 성능 저하를 방지하기 위해, User VM에 할당된 메모리 중 최소한 25% 이상은 CXL 메모리가 아닌 로컬 메모리로 구성되어야 한다. 2) CXL 메모리 풀에 연결된 호스트의 수가 많을수록 switching latency와 같은 요인들로 인해 메모리 접근에 필요한 시간이 더욱 빠르게 증가한다. 3) CXL 메모리 풀에 연결된 호스트의 수가 증가할수록 시스템에 요구되는 DRAM의 총량이 감소하는 경향을 보이지만, 일정 규모 이상부터는 이러한 효과의 영향이 감소한다. 4) CXL 메모리 풀 도입을 통해 긍정적인 ROI(Return On Investment)를 얻기 위해서는 메모리 풀의 크기, 토폴러지, 수행하는 작업의 특성 등을 모두 고려하여야 한다.

선행연구[3]은 데이터 센터 하드웨어 비용 중

40~50%를 차지하는 DRAM이 런타임 중 최대 25%가 사용되지 않고 낭비되고 있음에 착안하여, CXL 메모리 풀을 통해 퍼블릭 클라우드 플랫폼의 하드웨어 비용을 줄이는 동시에 CXL 메모리가 NUMA 노드의 로컬 메모리와 비슷한 성능을 보이도록 하는 시스템 구조인 Pond를 제안하였다. Pond는 선행연구[21]에서 분석된 결과와 같이 CXL 메모리 풀에 연결된 호스트의 수를 8~16개로 제한하여 CXL 메모리 풀에 접근할 때 발생하는 지연 시간을 최소화하였다. 또한, VM이 실행할 애플리케이션이 지연 시간에 얼마나 영향을 받는지에 따라 VM이 할당받을 메모리 중 CXL 메모리와 로컬 메모리의 비율을 결정하는 기계학습 기반 예측 모델을 사용하였다. 해당 모델을 통해 지연 시간에 영향을 덜 받는 VM의 경우 CXL 메모리 풀에서 메모리를 더 많이 할당하여 메모리 사용 효율을 올릴 수 있다. Pond는 앞의 방법들을 통해 로컬 NUMA 노드의 메모리를 사용하는 경우 대비 1~5%의 성능 하락만을 동반하며 클라우드 플랫폼에서 사용되는 DRAM의 양을 7% 감소시켰다.

나. NDP 기능을 탑재한 CXL 장치 연구

NDP(Near Data Processing)는 연산에 필요한 데이터와 가까운 위치에서 연산을 수행하여 데이터 이동에 소모되는 비용을 줄여, 시스템의 성능 및 전력 효율을 올리는 설계 방법이다. 이 절에서는 이러한 NDP 개념을 채용하여 CXL 메모리와 가까운 위치에 특정 애플리케이션에 특화된 가속 연산 유닛을 배치하여 성능 및 전력 효율을 향상하고자 한 연구들을 소개한다.

선행연구[22]는 유전체 분석 가속 모듈을 CXL 메모리 근처에 배치하여, NDP를 통해 유전체 분석 작업의 성능 및 에너지 효율 향상을 도모한 연구이다. 기존 DIMM 기반 NDP 가속기는 DIMM 모듈 간 낮은 대역폭 및 메모리 용량으로 인해 제한적인 성능을 보일 수밖에 없었으나, 해당 연구에서는 CXL의 넓은 대역폭 및 메모리 확장 기능을 통해 이러한 한계를 극복하고자 하였다. 해당 연구는 유전체 분석 작업(DNA seeding, K-mer counting 등)에 특화된 NDP 연산 모듈들을 CXL DIMM의 PCB 기판 위 또는 CXL 스위치에 배치하였다. 또한, CXL 메모리의 데이터 전송 단위와 유전체 분석 작업에 필요한 데이터 크기가 달라 발생하는 불필요한 데이터 전송을 없애기 위해 연산에 필요한 데이터를 모아서 전송하는 데이터 패킹 모듈을 배치하여 유전체 분석 작업에 사용되는 데이터 전송 효율을 높였다. 해당 연구의 시뮬레이션[23] 기반 실험 결과에 따르면, 1) CXL 기반 확장 메모리, 2) NDP 유닛, 그리고 3) 데이터 전달 메커니즘 최적화를 통해 기존 DIMM 기반 유전체 분석 NDP 가속기 대비 4배 이상의 유전체 분석 성능 향상을 이룰 수 있으며, 에너지 소모 또한 크게 줄일 수 있음을 밝혔다.

선행연구[24]는 CXL 메모리 카드 내부에 KNN(K-Nearest Neighbor) 연산에 특화된 NDP 모듈을 배치하여 연산 과정에서 소요되는 데이터 전송 오버헤드를 줄인 CMS(Computational CXL-Memory Solution)를 제안하였다. 해당 연구에서 저자는 메모리 집약적 작업을 처리하는 경우 여전히 부족한 CXL의 대역폭을 지적하며, 연산을 위한 데이터가 CXL을 거칠 필요 없이 호스트 프로세서가 NDP 모듈을 통해 계산된 결과만 바로 읽어갈 수 있도록 CMS의 구조를 설계하였다. 또한, CMS 내부 메모리 대역폭을 최대로 활용하기 위해 CXL 메모리의 채널을 효과적으로 인터리빙(Interleaving)하는 방법 및 성능 최적화된 MAC(Multiply and Accumulate) 구조, 그리고 CMS API를 제안하였다. Xilinx FPGA를 기반으로 한 CMS 프로토타입상에서 진행된 실험에 따르면, CMS는 CPU만 사용하는 컴퓨팅 환경 대비 3.2배

높은 KNN 벤치마크[25] 성능을 보였으며, 34% 낮은 에너지 소모를 보였다.

다. CXL 관련 소프트웨어 연구

CXL 장치를 사용자 수준에서 효과적으로 사용하기 위해서는 이를 지원하는 소프트웨어가 필수적이다. 선행연구[2]에서는 유형 3 CXL 장치의 확장 메모리를 효율적으로 활용하기 위한 SMDK (Scalable Memory Development Kit)를 제안하였다[15]. SMDK는 CXL 메모리와 로컬 메모리를 구분하여 관리하기 위한 리눅스 커널의 VMM(Virtual Memory Management) 기반 SMDK.kernel과 CXL 메모리와 로컬 메모리로 구성된 계층 메모리(Tiered Memory) 구조에서 애플리케이션에 대한 메모리 할당을 담당하는 SMDK.allocator로 구성되어 있다. 사용자는 SMDK.allocator에 포함된 User API를 통해 CXL 메모리와 로컬 메모리 중 할당받을 메모리의 우선도, 목표 대역폭 및 용량 등의 선호도 옵션(User Preference)을 설정할 수 있다. SMDK와 CXL 유형 3 장치를 사용한 시스템과 DRAM 메모리만을 사용한 시스템 간 딥러닝 및 인-메모리 데이터베이스 애플리케이션 기반 성능 비교를 수행한 결과, SMDK를 사용한 시스템이 1.5~1.99배 더 좋은 성능을 보였다.

현재 CXL HW/SW 생태계가 안정화되지 않았기 때문에 선행연구[20]처럼 직접 모든 HW 및 SW를 구현하여 실험 환경을 구축하지 않는 이상 CXL 관련 연구를 진행하는 데 어려움이 있다. 이에 선행연구[26]에서는 유형 3 CXL 장치를 시뮬레이션할 수 있는 CXL 시뮬레이터[27]를 제안하였다. 제안된 시뮬레이터는 호스트 프로세서에서 동작하는 사용자 프로그램의 trace를 기반으로 하여, 해당 프로그램을 사용자가 설정한 CXL 메모리 토폴러지상에서 실행하는 경우를 시뮬레이션한다. 해당 CXL 시뮬레이터는 native 환경 대비 평균 41배 느린 애플리케이션 수행 시간을 보였지만, 시스템 레벨 시뮬레이터인 Gem5 기반 CXL 시뮬레이터[28] 대비 평균 73배 빠른 수행 시간을 보였다. CXL 관련 생태계가 안정되지 않은 현재 상태에서, 이러한 시뮬레이터를 기반으로 CXL 관련 연구를 수행하는 것이 가능할 것이다.

Ⅳ. 결론

최근 널리 사용되는 기계학습 및 대규모 데이터베이스와 같은 데이터 집약적 작업은 컴퓨팅 시스템에 대용량의 메모리를 요구하는 특성이 있다. CXL은 컴퓨팅 노드가 구조적 한계를 극복하며 대용량 메모리를 사용할 수 있도록 하며, 메모리 오버프로비저닝과 같은 대용량 메모리 사용에 따른 부작용 또한 완화할 수 있는 기술로 산업계 및 학계의 큰 관심을 받고 있다. 본고에서는 CXL에 대한 전체적인 내용을 소개하고, CXL 관련 최신 연구 동향을 살펴보았다.

CXL은 아직 기술개발 초기 단계에 있어 관련 하드웨어 및 소프트웨어 인프라 부족 등의 문제가 있지만, 다양한 하드웨어 자원들을 통합하여 사용할 수 있도록 하는 컴포저블 컴퓨팅을 가능하게 하는 등 그 가능성은 매우 무궁무진하다. 앞으로는 데이터 센터와 같은 대규모 컴퓨팅 환경에서부터 소규모의 서버 클러스터에 이르기까지 다양한 환경에서 기존 PCIe를 대체하여 CXL 기술이 적용될 것으로 보인다. 이에 따라 국내 산·학·연에서는 CXL 기술에 대해 적극적으로 기술 도입을 추진하고, 관련 연구를 수행하여 기술 경쟁력을 높여 나가야 할 것이다.

용어해설

메모리 오버프로비저닝(Memory Overprovisioning) 대부분의 애플리케이션은 메모리 사용량이 런타임 동안 계속 변하기 때문에, 최대 메모리 사용량을 기준으로 메모리를 장착한 경우 상당한 양의 메모리가 런타임 동안 사용되지 않는 현상

메모리 확장기술 컴퓨팅 노드가 자신의 로컬 메모리 외에 CXL 또는 PCIe와 같은 하드웨어 인터커넥트를 통해 연결된 외부의 확장 메모리에 접근하여 이를 사용 가능하도록 하는 기술

CXL(Compute eXpress Link) 인텔의 주도하에 2019년 개발된 CPU와 주변장치를 고속으로 연결하기 위한 하드웨어 인터커넥트

약어 정리

API	Application Programming Interface
CPU	Central Processing Unit
CXL	Compute eXpress Link
DRAM	Dynamic Random Access Memory
EDSFF	Enterprise & Data Center Standard Form Factor
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HBM	High Bandwidth Memory
MESI	Modified, Exclusive, Shared, Invalid
NDP	Near Data Processing
NIC	Network Interface Card
NUMA	Non-Uniform Memory Access
PCIe	Peripheral Component Interconnect express
PoC	Proof of Concept
RDMA	Remote Direct Memory Access
ROI	Return On Investment
SDK	Software Development Kit
TBD	To Be Defined
TCO	Total Cost of Ownership
VM	Virtual Machine
VMM	Virtual Memory Management

참고문헌

[1] T. Brown et al., "Language models are Few-shot learners," in Proc. NeurIPS 2020, (Vancouver, Canada), Dec. 2020, pp. 1877-1901.

[2] K.S. Kim et al., "SMT: Software-defined memory tiering for heterogeneous computing systems with cxl memory expander," IEEE Micro, vol. 43, no. 2, 2023, pp. 20-29.

[3] H. Li et al., "Pond: CXL-based memory pooling systems for cloud platforms," in Proc. ACM ASPLOS 2023, vol. 2, (Vancouver, Canada), Jan. 2023.

[4] 김선영 외, "CCIX 연결망과 메모리 확장기술 동향," 전자통신동향분석, 제37권 제1호, 2022, pp. 42-52.

[5] 오명훈 외, "메모리 중심 컴퓨팅 기술 동향," 융합연구리뷰, vol. 6, no. 3, 2020.

[6] CXL Consortium, CXL 3.0 Specification, https://www.computeexpresslink.org/download-the-specification

[7] Compute Express Link 3.0, https://www.computeexpresslink.org/_files/ugd/0c1418_a8713008916044ae9604405d10a7773b.pdf

[8] CCIX, About the CCIX Consortium, https://www.ccixconsortium.com/about/

[9] Tektronix, "PAM4 Signaling in High-Speed Serial Technology: Test, Analysis, and Debug," Available: https://download.tek.com/document/PAM4-Signaling-in-High-Speed-Serial-Technology_55W-60273.pdf

[10] A. Biswas, "Sapphire rapids," in Proc. IEEE HCS 2021, (Palo Alto, CA, USA), Aug. 2021, pp. 1-22.

[11] AMD EPYC™ 9004 Series Server Processors Launch, https://www.amd.com/en/events/epyc4

[12] S.J. Park et al., "Scaling of memory performance and capacity with CXL memory expander," in Proc. IEEE HCS 2022, (Palo Alto, CA, USA), Aug. 2022, pp. 1-27.

[13] SK hynix Newsroom, "SK hynix Develops DDR5 DRAM CXLTM Memory to Expand the CXL Memory Ecosystem," 2022, https://news.skhynix.com/sk-hynix-develops-ddr5-dram-cxltm-memory-to-expand-the-cxl-memory-ecosystem/

[14] CZ120 memory expansion module, https://www.micron.com/solutions/server/cxl

[15] Scalable Memory Development Kit, https://github.com/OpenMPDK/SMDK

[16] Astera Labs, "Leo Memory Connectivity Platform for CXL 1.1 and 2.0," 2022, https://www.asteralabs.com/products/cxl-memory-platform/leo-cxl-memory-connectivity-platform/

[17] XC50256 CXL2.0/PCIe5.0 switch, https://www.xconn-tech.com/product

[18] Enfabrica at ISC 2023, Scaling CXL Memory Using High Speed Networking, https://www.youtube.com/watch?v=YdJWhqeT5DM

[19] M.H. Ha et al., "Dynamic capacity service for improving

CXL pooled memory efficiency," IEEE Micro, vol. 43, no. 2, 2023, pp. 39–47.

[20] D.H. Gouk et al., "Direct access, high performance memory disaggregation with DirectCXL," in Proc. USENIX ATC 2022, (Carlsbad, CA, USA), Jul. 2022, pp. 287–294.

[21] D.S. Berger et al., "Design tradeoffs in CXL–based memory pools for public cloud platforms," IEEE Micro, vol. 43, no. 2, 2023, pp. 30–38.

[22] W. Huangfu et al., "BEACON: Scalable near data processing accelerators for genome analysis near memory pool with the CXL support," in Proc. IEEE/ACM MICRO 2022, (Chicago, IL, USA), Oct. 2022, pp. 727–743.

[23] Y.G. Kim et al., "Ramulator: A fast and extensible DRAM simulator," IEEE Comput. Archit. Lett., vol. 15, no. 1, 2016, pp. 45–49.

[24] J.S. Sim et al., "Computational CXL–memory solution for accelerating memory–intensive applications," IEEE Comput. Archit. Lett., vol. 22, no. 1, 2023, pp. 5–8.

[25] KNN benchmark, https://github.com/mnms/ltdb-knn-kernel-bench

[26] Y. Yang et al., "CXLMemSim: A pure software simulated CXL.mem for performance characterization," arXiv preprint, CoRR, 2023, arXiv: 2303.06153.

[27] CXLMemSim, https://github.com/SlugLab/CXLMemSim

[28] gem5-cxl, https://github.com/fadedzipper/gem5-cxl