CXL 메모리 및 활용 소프트웨어 기술 동향

Technology Trends in CXL Memory and Utilization Software

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

ABSTRACT

Artificial intelligence relies on data-driven analysis, and the data processing performance strongly depends on factors such as memory capacity, bandwidth, and latency. Fast and large-capacity memory can be achieved by composing numerous high-performance memory units connected via high-performance interconnects, such as Compute Express Link (CXL). CXL is designed to enable efficient communication between central processing units, memory, accelerators, storage, and other computing resources. By adopting CXL, a composable computing architecture can be implemented, enabling flexible server resource configuration using a pool of computing resources. Thus, manufacturers are actively developing hardware and software solutions to support CXL. We present a survey of the latest software for CXL memory utilization and the most recent CXL memory emulation software. The former supports efficient use of CXL memory, and the latter offers a development environment that allows developers to optimize their software for the hardware architecture before commercial release of CXL memory devices. Furthermore, we review key technologies for improving the performance of both the CXL memory pool and CXL-based composable computing architecture along with various use cases.

KEYWORDS CXL, memory pool, resource disaggregation, SDK, software emulator

Ⅰ. 서론

최근 인공지능 분야에서는 데이터와 모델의 크기가 크게 증가하며 대규모 데이터 처리 성능 향상이 절실히 요구되고 있다. 예를 들어 널리 활용되고 있는 LLM 중 하나인 GPT-3 모델은 NVIDIA A100 가속기를 1,500여 개 활용하여 학습 시간을 23일까지 단축하였지만[1], 그보다 훨씬 큰 GPT-4 모델의 경우 동종 가속기 개수를 2배 늘려도 83일이라는 긴 학습 시간이 소요되고 있다[2]. 이러한 한계를 극복

* DOI: https://doi.org/10.22648/ETRI.2024.J.390106
* 본 논문은 한국전자통신연구원 내부연구개발사업 대규모 데이터 처리 응용 성능 향상을 위한 새로운 메모리 연결망(CXL) 기술 기반 MPI 분산병렬처리 구조 연구 및 성능분석(23YS1700)의 지원을 받아 수행된 연구임.

하기 위해 다양한 컴퓨팅 구조들이 연구 개발되고 있다.

우선, 기존 프로세서 중심 컴퓨팅 구조에서 데이터 집약적 응용을 처리할 때 CPU와 메모리 사이의 데이터 이동이 많아지면 병목현상이 심화되는 문제를 해결하기 위해 메모리 중심 컴퓨팅 구조가 연구되고 있다. 이 구조에서는 데이터를 메모리 근처 또는 메모리 내부에 위치하는 프로세서에서 처리하여 CPU와 주메모리 사이 데이터 이동량을 줄여 데이터 처리 성능을 향상시킬 수 있다.

또한, 대규모 데이터 처리를 위해 자원 분리 기술(Resource Disaggregation)[3,4]이 활발히 연구되고 있다. 자원 분리란 컴퓨팅 자원 풀(Resource Pool)을 구성하고 자원 풀과 호스트 CPU 사이의 빠른 통신을 구현하여 원격(Remote) 자원을 자신의 로컬(Local) 자원 수준으로 빠르게 이용하는 기술이다. 예를 들어, 대규모 스토리지 서버를 구성하고 호스트에서 해당 서버를 자신의 로컬 저장소와 비슷한 수준의 성능으로 활용하는 NAS는 잘 구현된 자원 분리의 예이다. 자원 분리 기술은 클라우드 환경에서 응용에 충분한 자원을 제공하면서도 개별 서버들이 필요 이상의 컴퓨팅 자원을 장착하는 오버프로비저닝(Over-provisioning) 문제와 컴퓨팅 시스템의 전력 낭비 문제를 해결하는 데 기여한다.

특히, 자원 풀을 실현하는 기술로서 최근에는 CXL(Compute eXpress Link) 고속 인터커넥트 기술이 인텔을 주도로 빠르게 세를 확장하며 메모리 자원도 스토리지처럼 풀 형태로 분리하는 것이 가능해지고 있다. CXL은 CPU와 메모리, 가속기, NIC, 스토리지 등 다양한 컴퓨팅 자원의 효율적 통신을 지원하여 자원들 사이의 빠르고 안정적인 통신 기능을 제공한다. 따라서 CXL 기반 메모리 분리(Memory Disaggregation) 기술로 구현된 CXL 메모리 풀은 응용들에 충분한 양의 메모리를 제공하면서도 유휴 메모리를 줄여 전체 시스템의 메모리 사용효율을 증가시킬 수 있다. PCIe 5.0을 사용하는 CXL 메모리 풀은 기존의 네트워크로 연결된 원격 메모리를 RDMA 기반으로 접근하는 것에 비해 10배 이상 빠른 접근이 가능하기 때문에 분산컴퓨팅 환경에서 메모리 부족 문제의 해결책이 될 수 있을 것으로 기대되고 있다.

그러나 CXL 메모리 풀을 응용 계층에서 쉽고 효과적으로 활용하기 위해서는 CXL 기반 소프트웨어 기술개발이 필수적으로 동반되어야 한다. 그러나 CXL 장치들은 아직 개발 초기이며 제조사들은 단일 장치와 시제품을 개발하는 단계에 있다. 이로 인해 소프트웨어 분야에서는 가용한 CXL 장치들이 부재하여 에뮬레이터와 같은 하드웨어를 모사한 소프트웨어 환경을 타겟으로 개발해야 한다. 이를 위해 CXL 장치 제조사와 CXL 컨소시엄은 CXL 메모리를 에뮬레이션하는 소프트웨어와 CXL 메모리 장치의 효과적인 사용을 위한 SDK를 개발 및 공개하기 시작하였다. 본고에서는 해당 분야의 최신 소프트웨어 기술 동향을 소개한다.

본고의 구성은 다음과 같다. Ⅱ장에서는 CXL 표준, 주요 CXL 장치, CXL 메모리 관련 기술개발 현황을 살펴보고, Ⅲ장에서는 CXL 메모리 에뮬레이터 소프트웨어들과 CXL 메모리 SDK 기술들을 소개한다. 끝으로, Ⅳ장에서는 결론 및 제언과 함께 본고를 마무리한다.

Ⅱ. CXL 소개

이 장에서는 CXL 컨소시엄에서 표준으로 정의하는 CXL 버전과 CXL 장치유형을 소개하고 제조사들이 개발 중인 주요 CXL 장치들과 CXL 메모리 관련 기술개발 현황을 살펴본다.

1. CXL 표준

가. CXL 버전(Version)

CXL 컨소시엄에서는 CXL 1.1('19년 6월), 2.0('20년 11월)을 거쳐 3.0('23년 5월)까지 표준을 제정하였다[5]. CXL 버전 1.1은 CXL.io, CXL.cache, CXL.mem의 세 가지 CXL 서브 프로토콜의 제정과 함께 단일 호스트와 단일 장치 간 P2P 통신에 대한 표준이다. CXL 버전 2.0은 CXL 스위치를 통해 연결된 다중 디바이스와 다중 호스트 간 자원의 풀링(Pooling) 기능을 지원하는 것에 대한 표준이다. CXL 버전 3.0은 리프(Leaf) 스위치들을 연결한 스파인(Spine) 스위치를 통해 더욱 넓고 다양한 토폴로지의 장치들을 연결하는 것과 하드웨어 레벨의 캐시 일관성을 보장하는 것에 대한 표준이다.

나. CXL 장치 유형(Type)

CXL 장치 유형은 CXL 서브 프로토콜의 조합에 따라 그림 1과 같이 세 가지로 분류된다. 그림 1(a)와 같이 CXL 유형 1 장치는 CXL.io, CXL.cahce를 사용하는 NIC과 호스트가 관리하는 메모리를 갖지 않는 가속기를 예로 들 수 있다. 그림 1(b)와 같이 CXL 유형 2 장치는 CXL.io, CXL.cache, CXL.mem 서브 프로토콜을 모두 사용하며 호스트 프로세서가 캐시 일관성을 유지하며 접근할 수 있는 자체 메모리를 내장한 GPU, FPGA와 같은 카드 형태의 가속기를 예로 들 수 있다. 그림 1(c)와 같이 CXL 유형 3 장치는 CXL.io, CXL.mem 서브 프로토콜을 사용하며 DRAM 또는 NVRAM으로 구성되어 호스트 프로세서가 load/store 명령으로 접근하는 메모리 확장장치를 예로 들 수 있다.

2. 주요 CXL 장치

이 절에서는 CXL 선도 업체들이 개발 중인 CXL 장치를 CPU, 메모리, 스위치로 나누어 소개한다.

가. CXL CPU

CXL CPU는 인텔의 Xeon과 AMD의 EPYC, ARM의 Neoverse 시리즈가 있다. '23년 1월 출시된 Xeon 시리즈의 Sapphire Rapids는 유형 1과 유형 2 장치를 공식적으로 지원한다. '22년 11월 출시된 AMD EPYC 시리즈의 Genoa는 현재 유형 3 장치만 지원하고 있으며, '24년 이후 유형 1과 유형 2 장치를 지원할 예정이다. 두 CPU는 현재 PCIe 5.0 환경에서 CXL 버전 1.1을 지원하고 있으며, '24년 이후로

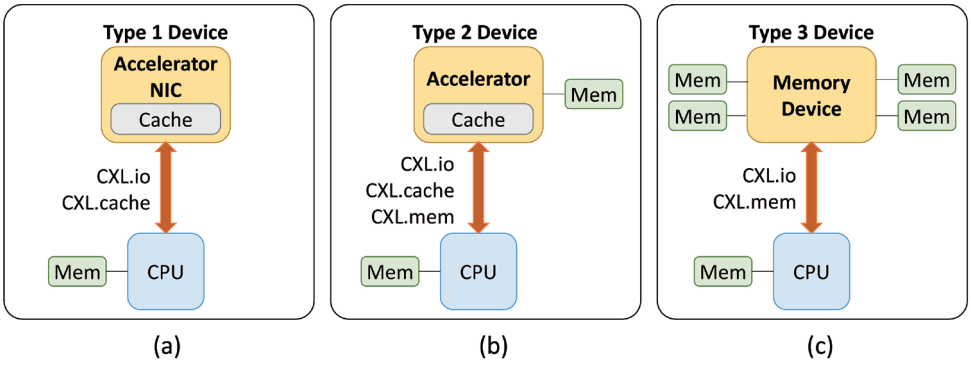

그림 1 (a) CXL 유형 1 장치, (b) CXL 유형 2 장치, (c) CXL 유형 3 장치

CXL 버전 2.0 이상을 지원할 예정이다. ARM은 고성능컴퓨팅용 Neoverse V시리즈와 데이터센터용 N시리즈를 '23년 CXL 버전 2.0과 '24년 CXL 버전 3.0을 지원하도록 개발 중이며, 지원할 장치 유형은 아직 공개되지 않고 있다.

나. CXL 메모리

CXL 메모리의 대표 예는 삼성전자의 CXL Memory Expander[6]와 SK 하이닉스의 CXL Memory[7]가 있다. 두 장치 모두 CXL 버전 2.0을 지원하는 유형 3 장치로 아직 상용화 전의 시제품이다. 두 메모리 장치를 응용에서 사용하기 위한 SDK가 함께 개발되고 있으며 해당 소프트웨어의 특징과 기능은 Ⅲ장의 CXL 메모리 SDK 절에서 보다 상세히 기술하고자 한다.

다. CXL 스위치

CXL 스위치 제조사들은 CXL 버전 2.0 이상의 리소스 풀링을 지원하는 스위치를 개발 중이다. Enfabrica사는 고속 네트워크를 활용하여 원격 메모리 접근 시간을 크게 단축시키는 ACF(Accelerated Compute Fabric) CXL 스위치 시제품을 개발 중이다[8]. 해당 스위치는 CXL 기반 컴포저블 컴퓨팅 구조에서 계층(Tier) 1의 로컬 DRAM/ HBM과 계층 3의 RDMA 기반 원격 메모리 사이의 계층 2에 위치하여 메모리 집약형 응용에 원격 고성능 대용량 메모리를 캐시용도로 사용할 수 있도록 지원함으로써 응용의 성능 향상에 기여한다.

Xconn Technologies사[9]는 CXL 버전 2.0 스위치 SoC를 최초 개발하고 공개했다. 해당 스위치를 이용하면 하나의 CXL 스위치에 0.5TB의 CXL 버전 2.0 메모리 장치를 30개씩 장착해서 15TB를 구성하고 해당 스위치가 최대 4개 연결된 형태로 호스트 프로세서에 장착하여 총 60TB까지 메모리를 확장하여 대규모 메모리 풀을 구성할 수 있게 될 예정이다.

3. CXL 메모리 관련 기술개발 현황

이 절에서는 CXL 메모리 관련 최신 기술개발 현황을 살펴본다. 표 1은 조사한 연구들에서 활용한 CXL CPU, CXL 메모리, CXL 소프트웨어와 출판연도를 정리한 표이다. 특히, CXL 메모리는 개발 형

표 1 CXL메모리 분야 최신연구 및 개발환경

<table>
<tr><th rowspan="2">Refs.</th><th rowspan="2">CXL CPU</th><th colspan="3">CXL Memory</th><th colspan="2">CXL 소프트웨어</th><th rowspan="2">출판
연도</th></tr>
<tr><th>개발
형태</th><th>CXL
Type</th><th>CXL
Version</th><th>개발형태</th><th>오픈소스</th></tr>
<tr><td>[10]</td><td rowspan="5">Intel Xeon
Sapphire Rapids</td><td>FPGA Card with AMD
Xilinx CXL 2.0 IP</td><td>2</td><td>v.2.0</td><td rowspan="3">Proprietary</td><td>X</td><td>'22</td></tr>
<tr><td>[11]</td><td rowspan="2">FPGA Card with Intel
CXL IP</td><td rowspan="6">3</td><td rowspan="2">v.1.1/
v.2.0</td><td>O</td><td>'23</td></tr>
<tr><td>[12]</td><td>X</td><td>'23</td></tr>
<tr><td>[13]</td><td>v.1.1</td><td rowspan="2">Custom
Library</td><td>X</td><td>'23</td></tr>
<tr><td>[14]</td><td rowspan="2">Custom ASIC</td><td>v.2.0</td><td>O</td><td>'23</td></tr>
<tr><td>[15]</td><td>AMD EPYC
Genoa</td><td>v.2.0/
v.3.0</td><td rowspan="3">Proprietary</td><td>O</td><td>'23</td></tr>
<tr><td>[16]</td><td rowspan="2">CXL Host
processor using
RISC-V</td><td rowspan="2">FPGA Card using
Customized CXL IP</td><td>v.2.0</td><td>X</td><td>'23</td></tr>
<tr><td>[17]</td><td>2</td><td>v.3.0</td><td>X</td><td>'23</td></tr>
</table>

태, CXL 유형, CXL 버전을, CXL 소프트웨어는 개발 형태와 오픈소스 여부를 구체적으로 명시했다.

8개의 연구 중 5개가 인텔의 CXL CPU를 사용[10-14], 1개가 AMD의 CXL CPU를 사용[15], 2개는 RISC-V 기반으로 자체 개발한 CXL CPU를 사용[16,17]하고 있었다. AMD Xilinx CXL IP를 이용하여 개발한 메모리 장치 시제품 1개[10], Intel CXL IP를 이용한 메모리 장치 시제품 3개[11-13], ASIC 형태의 메모리 장치 시제품 2개[14,15], 자체 개발한 CXL IP와 FPGA 카드를 이용하여 개발한 메모리 장치 시제품 2개[16,17]로 구성되었다. 이 중 6개가 CXL 유형 3의 메모리 장치[11-16]이고, 2개는 가속 기능이 함께 제공되는 유형 2 메모리 장치[10,17]이다. 또한 CXL 버전 2.0을 지원하는 메모리가 3개[10,14,16], 버전 3.0을 지원하는 메모리가 1개[17], 버전 1.1과 2.0을 동시 지원하는 장치가 2개[11,12], 버전 2.0과 3.0을 지원하는 장치가 1개[15]였다.

CXL 소프트웨어는 자체 개발(Proprietary) 형태가 6개[10-12,15-17], 장치에 맞춤형(Customized)으로 개발된 소프트웨어 라이브러리가 2개[13,14]였다. CXL 소프트웨어 중 3개는 오픈소스[11,14,15]로 제공되었다. 그러나 앞의 오픈소스들은 해당 연구에서 자체 개발한 CXL 메모리 장치가 구동 중인 환경에서 이용할 수 있었다.

이상의 조사를 통해 CXL 메모리 실물 장치 없이는 CXL 메모리 관련 소프트웨어 개발환경을 갖추는 것이 불가능하다는 것을 알 수 있다. 이러한 이유로 CXL 메모리 장치를 모사하는 소프트웨어들이 개발되고 있으며 다음 장의 CXL 메모리 에뮬레이터 소프트웨어 절에서 자세히 기술한다.

Ⅲ. CXL 메모리 활용 소프트웨어 기술

이 장에서는 CXL 메모리 활용 소프트웨어 기술의 개요를 살펴보고 대표적인 소프트웨어 기반의 CXL 메모리 에뮬레이터 소프트웨어들과 CXL 메모리 SDK들을 소개한다.

1. CXL 메모리 활용 소프트웨어 개요

그림 2는 CXL 메모리 개발사들이 제공하는 소프트웨어를 계층별로 구분하여 정리한 그림이다.

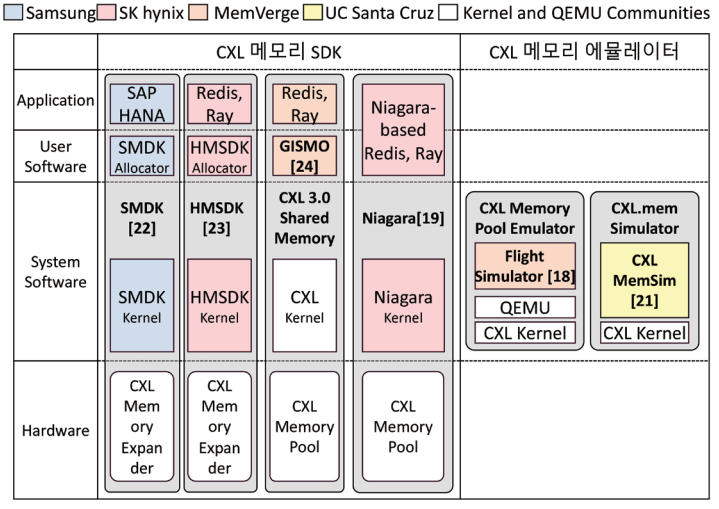

그림 2 CXL 메모리 개발사들이 제공하는 SW 종류

CXL 메모리 활용 소프트웨어는 크게 CXL 메모리 하드웨어 장치가 없는 경우 해당 장치를 모사하는 기능을 제공하는 CXL 메모리 에뮬레이터와 CXL 메모리 장치가 있는 사용자에게 해당 장치를 효율적으로 활용하는 기능을 지원하는 SDK 용도의 소프트웨어로 구분할 수 있다.

CXL 메모리 장치가 없는 환경에서 동작하는 대표적인 CXL 메모리 에뮬레이터는 Flight Simulator와 CXL MemSim이 있다. 해당 에뮬레이터들은 시스템 소프트웨어 계층에서 기능이 개발되어 있고 동작한다. CXL 메모리 장치가 있는 환경에서 장치를 쉽고 효율적으로 사용하는 기능을 제공하는 대표적인 SDK에는 SMDK, HMSDK, GISMO, Niagara이 있다. 해당 SDK 들은 시스템 소프트웨어 계층과 사용자 소프트웨어 계층에 걸쳐서 기능 개발이 되어있고 동작한다. 각 소프트웨어들에 대해 다음 절에서 더욱 자세히 설명한다.

2. CXL 메모리 에뮬레이터 소프트웨어

가. Flight Simulator

Flight Simulator는 MemVerge와 SK 하이닉스가 함께 '23년 8월 출시한 CXL 2.0 메모리 풀 에뮬레이터로 Niagara Pooled Memory 장치를 모델로 개발된 QEMU 기반의 오픈소스이다[18]. Niagara Pooled Memory는 DCS(Dynamic Capacity Service)를 탑재하여 다중 호스트들이 CXL 메모리 풀을 동적으로 할당(Allocate) 및 해제(Deallocate)하는 기능을 제공하는 멀티노드용 CXL 메모리 풀 시스템이다. 해당 메모리 시스템을 이용하는 DCS-enabled Kubernetes(k8s)는 호스트 메모리만 사용하는 기존 k8s보다 메모리 사용효율이 20% 증가한다고 발표되었다[19]. Flight Simulator는 순수 소프트웨어 에뮬레이터로 CXL 메모리 장치 없이 CXL 메모리 관련 소프트웨어를 개발하는 데 도움을 준다.

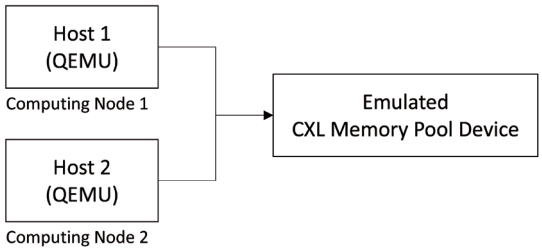

그림 3 Flight Simulator 형상

그림 3은 Flight Simulator의 형상이다. 본 에뮬레이터는 2개의 컴퓨팅 노드가 하나의 CXL 메모리를 공유하는 시스템을 모사한다. 각 노드에는 CXL을 지원하는 Fedora 운영체제(커널 버전: Fedora Build 6.3.7 + Additional Options)가 설치되어 있다. 아직 QEMU 메인라인에서는 CXL이 지원되고 있지 않으며 CXL을 지원하는 QEMU 워킹 브랜치를 설치 후 활용해야 한다[20].

그림 4는 서로 다른 컴퓨팅 노드에서 수행 중인 Writer와 Reader가 CXL 공유 메모리 풀에 데이터를 쓰고 읽으며 통신하는 소스코드와 결과화면이다.

두 소스코드는 공통적으로 CXL 메모리 풀의 경로와 용량을 정의한다(라인 5-6). 그림 4(a) Writer에서는 사용자로부터 문자열을 입력받고(라인 9), CXL 메모리 풀을 open하여 식별자(fd)를 반환하고(라인 14), mmap을 이용해 CXL 메모리 풀을 호스트의 가상 메모리 주소로 매핑한다(라인 17). 이후 사용자로부터 입력받은 문자열 길이 만큼의 메모리를 memset을 이용해서 초기화하고 입력 문자열을 memcpy를 이용해서 CXL 메모리 풀에 복사한 후 munmap과 close(fd)를 이용하여 메모리 사용을 마친다(라인 20-24).

그림 4(b) Reader에서는 CXL 메모리 풀을 open하여 fd를 반환하고(라인 9), mmap을 이용해 호스트

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>

#define FILENAME "/dev/dax0.0"
#define REGION_SIZE 4294967296

 int main(int argc, char *argv[]) {
 if (argc != 2) { printf("Usage: %s <string>\n", argv[0]); return 1; }

char *string_to_write = argv[1];
size_t size_to_write = strlen(string_to_write) + 1;

 int fd = open(FILENAME, O_RDWR);
 if (fd == -1) { perror("Error opening file"); return 1; }

 char *addr = mmap(NULL, REGION_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
 if (addr == MAP_FAILED) { perror("Error mapping file");  return 1; }

 memset(addr, 0, size_to_write);
 memcpy(addr, string_to_write, size_to_write);
 printf("Paragraph written to DAX device successfully.\n");
 munmap(addr, 4096);
 close(fd);
 return 0;
 }
```

(a)

```c
 #include <stdio.h>
 #include <fcntl.h>
 #include <sys/mman.h>

 #define FILENAME "/dev/dax0.0"
 #define REGION_SIZE 4294967296

 int main() {
 int fd = open(FILENAME, O_RDWR);
if (fd == -1) { perror("Error opening file"); return 1; }

char *addr = mmap(NULL, REGION_SIZE, PROT_READ, MAP_SHARED, fd, 0);
 if (addr == MAP_FAILED) { perror("Error mapping file"); return 1; }

 printf("Paragraph read from DAX device:\n%s\n", addr);

 munmap(addr, 4096);
 close(fd);
 return 0;
}
```

(b)

```
[fedora@localhost daxtest] sudo ./daxwriter
“Hello from Host1.”
Paragraph written to DAX device successfully.
```

(c)

```
[fedora@localhost daxtest] sudo ./daxreader
Paragraph read from DAX device:
Hello from Host 1.
```

(d)

그림 4 CXL 공유 메모리 풀 기반 통신 코드와 실행화면: (a) CXL 메모리 풀 writer 소스코드, (b) CXL 메모리 풀 reader 소스코드, (c) CXL 메모리 풀 writer 실행결과 (Host 1), (d) CXL 메모리 풀 reader 실행결과(Host 2)

의 가상 메모리 주소로 매핑한 후(라인 12) CXL 메모리 풀에 저장된 내용을 출력한다(라인 15). 이후 munmap과 close(fd)를 수행하여 메모리 사용을 마친다(라인 17-18).

그림 4(c)처럼 호스트 1에서 writer를 실행하여 사용자가 문자열을 입력한 후에 그림 4(d)처럼 호스트 2에서 reader를 실행하면 호스트 1의 사용자가 입력한 문자열인 “Hello from Host 1”이 호스트 2의 터미널에 출력된다. 이를 통해 두 노드가 공유하고 있는 CXL 메모리 풀을 이용한 프로세스 간 통신 과정을 확인할 수 있다.

나. CXLMemSim

CXLMemSim[21]은 다계층 메모리들이 다양한 토폴로지로 구성된 환경에서 CXL.mem 서브 프로토콜 기반 장치를 사용하는 응용들의 성능을 평가하는 기능을 오픈소스 형태로 제공한다. 해당 시뮬레이터는 Tracer, Timer, Timing Analyzer의 세 가지 컴포넌트로 구성되며 응용 프로그램의 실행을 여러 에폭(Epoch)으로 나누고 에폭마다 성능을 모니터링하여 이벤트 정보를 수집하고 트레이스 기반(Trace-Driven)으로 실행 시간을 측정한다. 이 시뮬레이터는 사용자가 메모리 계층 구조와 지연시간을 쉽게 조정할 수 있어 CXL 메모리 풀 기반 다계층 메모리 구조와 CXL 스위치에서 발생하는 메모리 읽기/쓰기 성능을 쉽게 예측 가능한 장점이 있다.

3. CXL 메모리 SDK

가. SMDK

Scalable Memory Development Kit(SMDK)[22]은 삼성전자에서 자사의 CXL 메모리 확장장치(CXL MXP: CXL Memory eXPander)를 사용자들이 쉽고 효과적으로 이용할 수 있도록 개발하고 있는 오픈소스 소프트웨어이다. SMDK는 커널 VMM(Virtual Memory Manager)을 확장하고 이기종메모리들 간의 인터페이스를 제공하여 SDM(Software Defined Memory)을 구축할 수 있게 한다. SMDK를 사용하는 응

용들은 주메모리에는 자주 접근되는 데이터를 상주시키고 CXL 메모리에는 자주 접근되지 않는 데이터를 상주시켜서 메모리 사용효율을 높일 수 있다.

SMDK는 사용자 영역(User Space)과 커널 영역(Kernel Space)에 걸쳐서 동작한다. 커널 영역 모듈들은 호스트의 주메모리와 CXL 메모리를 각각 EXMEM 노드와 NORMAL 노드로 관리하고 사용자 영역에 두 종류의 메모리를 가상주소로 매핑한다. 사용자 영역에서는 Allocator 모듈이 응용에 EXMEM 영역과 NORMAL 영역을 할당한다. 이때 Intelligent Allocation Engine은 응용의 메모리 사용 패턴에 따라 CXL 메모리와 주메모리로 구성된 메모리 풀을 용도에 맞게 할당한다. 또한, 응용이 사용하는 일반적인 API들은 Transparent API로 제공하고 특정 응용에 최적화된 Optimization API를 별도로 제공하여 소프트웨어 개발의 편의성과 응용의 성능 향상을 돕는다.

Transparent API에는 힙(Heap) 메모리를 관리하는 API인 malloc, realloc, mmap 등이 존재한다. Optimization API에는 특정 응용을 위해 설계된 s_malloc, s_calloc, s_relloc, s_posix_memalign, s_free, s_malloc_node, s_free_node와 같은 Allocation API와 s_get_memsize_total, s_get_memsize_used, s_get_memsize_available, s_get_memsize_node_avaliable 등의 Metadata API 가 존재한다. 또한 C, C++, Python, Java, Go와 같은 고수준 언어 바인딩을 제공한다.

나. HMSDK

Heterogeneous Memory Software Development Kit(HMSDK)은 SK 하이닉스가 자사의 CXL 메모리 풀을 효과적으로 사용하도록 개발 중인 소프트웨어이다[23]. 현재 1.1 버전이 오픈소스로 공개되어 있으며, 2.0 버전은 개발 중이다.

HMSDK 1.1은 CXL Memory Allocator와 Bandwidth-aware 인터리빙을 수행하는 CXL-aware Memory Placement 정책 모듈로 구성되어 응용이 사용하는 메모리 대역폭에 따라 인터리빙 기능을 수행한다. HMSDK 2.0에는 자주 접근되는(Hot) 페이지와 자주 접근되지 않는(Cold) 페이지를 구분하는 기능과 해당 페이지를 호스트 메모리에서 CXL 메모리로 이동시킬 때 발생하는 페이지 이동 오버헤드를 줄이는 기능이 포함될 예정이다. 이상의 기능들을 지원하기 위해 운영체제 계층에는 페이지 온도(Temperature) 프로파일러와 페이지 Promotion/Demotion 모듈이, 사용자 계층에는 페이지 이동을 결정하기 위한 Decision Maker가 개발되고 있다. 또한 C, C++, Java, Python과 같은 고수준 언어 바인딩이 지원될 예정이다.

다. GISMO

CXL 공유 메모리를 구현하기 위해서는 CXL 메모리 풀 영역의 캐시 일관성이 유지되어야 한다. 해당 기능은 CXL 버전 3.0의 하드웨어 장치 스펙 중 하나이지만 CXL 버전 2.0 장치를 이용하여 소프트웨어적으로 구현할 수도 있다. Global I/O-free Shared Memory Objects(GISMO) 는 후자의 형태로 MemVerge사에서 개발 중인 CXL 공유 메모리 풀 관리용 오픈소스 소프트웨어다[24].

그림 5는 GISMO 구조를 나타낸다. 서로 다른

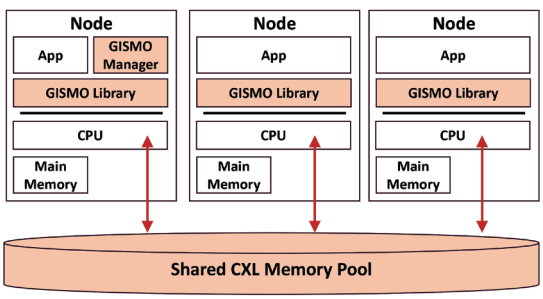

그림 5 GISMO 구조

노드에는 주메모리가 존재하고 각 노드의 CPU들은 CXL 메모리 풀에 연결되어 있다. 각 노드에서는 GISMO Library가 동작하고 마스터 노드에서는 GISMO Manager가 동작한다.

GISMO API는 다음과 같이 (1) 노드들을 GISMO Manager에 연결하는 connect, (2) 노드들과 GISMO Manager의 연결을 해제하는 disconnect, (3) CXL 공유 메모리 풀에 객체를 생성하는 create, (4) 메모리에 상주 중인 객체를 불변(Immutable)하게 하는 seal, (5) CXL 공유 메모리 풀의 데이터를 읽는 get, (6) 데이터 읽기가 끝난 후 공유 메모리 관리자에게 읽기가 끝났음을 알리는 release, (7) 메모리에서 객체를 삭제하는 delete, (8) CXL 메모리 풀 내 객체의 상태를 구독하여 알림 받는 subscribe로 구성된다.

GISMO를 이용한 CXL 공유 메모리 풀의 장점은 세 가지 사례를 통해 검증되었다. 첫째, GISMO와 CXL 공유 메모리 풀을 분산딥러닝 프레임워크 Ray[25]에 활용하면 모델 학습 시 파라미터 동기화에 필요한 셔플(Shuffle) 비용을 줄여 학습 시간을 단축할 수 있다. 이는 셔플 시 발생하는 노드 간 데이터 교환 과정이 공유 메모리 읽기/쓰기로 대체되어 기존 환경에서 셔플 시 발생하던 노드 간 데이터 교환에 필요한 네트워크 비용과 노드 간 데이터 복사 비용이 사라지기 때문이다. 해당 시스템에서 CXL 공유 메모리 풀은 노드 간 데이터양 불균형(Skew)으로 인해 노드들의 메모리가 낭비되는 문제를 해결하는 데도 도움을 준다.

둘째, GISMO와 CXL 공유 메모리 풀을 분산 데이터베이스의 캐시로 활용하면 데이터 처리 성능을 향상할 수 있다. 이는 CXL 공유 메모리 풀을 캐시로 이용하는 환경에서는 노드 간 캐시 일관성 유지 비용이 감소하고 나아가 캐시 적중률을 높일 수 있기 때문이다.

셋째, GISMO와 CXL 공유 메모리 풀을 인메모리 분산 스트림 데이터베이스의 체크포인트 기능에 활용하면 장애 복구 시간을 크게 단축할 수 있다[26]. 이는 장애 복구에 필요한 질의 데이터, 인덱스, 질의 상태 정보들을 CXL 공유 메모리 풀에 저장해 놓으면 장애 복구 시에 필요한 정보들을 네트워크 I/O와 노드 간 데이터 복사 없이 공유 메모리의 읽기/쓰기로 대체하여 장애 복구 시간을 크게 단축할 수 있기 때문이다.

라. Memory Machine

Memory Machine[27]은 MemVerge에서 CXL 기반 데이터센터의 효율적인 메모리 관리를 위해 개발 중인 CXL 메모리 풀 관리용 소프트웨어로 운영체제 하이퍼바이저와 VM(Virtual Machine)들의 사이에 위치하는 소프트웨어이다. 아직 오픈소스로 공개되지 않았으며 시제품 단계의 비공개 소프트웨어이다. 그림 6은 CXL 메모리 풀을 사용하는 시스템이 Memory Machine을 이용하여 VM들을 확장성 있게 운영하는 과정을 보인다. 그림 6(a)처럼 VM들이 필요로 하는 메모리를 주메모리 용량으로 충분히 제공할 수 있을 때는 CXL 메모리 풀을 사용하지 않고 호스트의 주메모리만 VM들에 할당하여 6개의 VM을 운영한다.

그러나 그림 6(b)처럼 VM의 개수가 증가하여 주메모리 용량이 부족해지면, Memory Machine은 추가로 필요한 메모리를 CXL 메모리 풀에서 추가 할당하며 하이퍼바이저들이 10개의 VM들을 운영할 수 있게 한다. 이후, 그림 6(c)처럼 VM 개수가 다시 6개로 줄어 필요 메모리양이 감소하면 Memory Machine은 CXL 메모리 풀에서 할당받았던 메모리를 해제하고 하이퍼바이저들은 다시 주메모리만 이용해서 VM들을 운영한다.

그림 7은 MemVerge사에서 제안하는 효율적인 CXL 메모리를 풀 관리를 위한 필수 소프트웨어 기

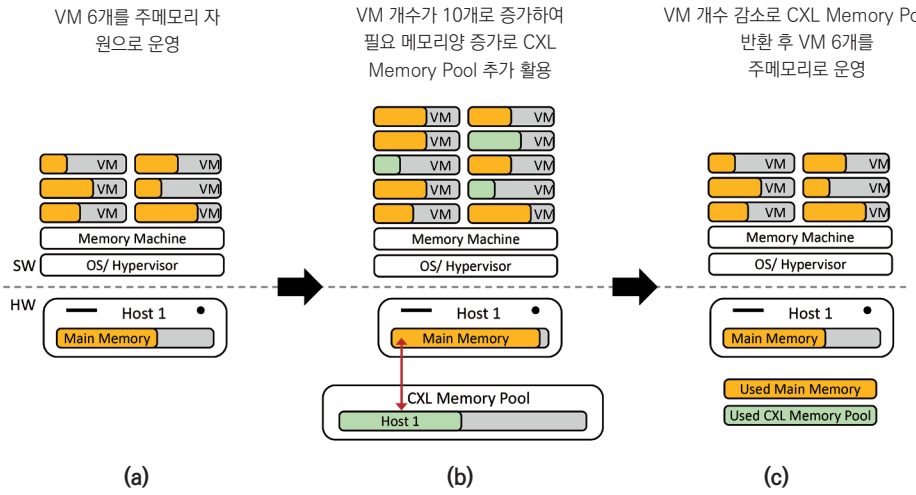

그림 6 Memory Machine을 이용한 확장성 있는 VM 운영

술들을 정리한 것이다. 하드웨어 레벨에서는 xPU들이 CXL 공유 메모리 풀에 연결되어 있다. 그 위 시스템소프트웨어 레벨에는 k8s의 메모리 자원 관리를 위한 Multi-Cell Control Plane과 하이퍼바이저들의 메모리 자원 관리를 위한 App-Centric Data Service가 위치한다. Multi-Cell Control Plane에서는 Global Memory Composer가 위치한다. 해당 모듈은 메모리 자원의 제공, 구성, 스케줄링 기능을 수행한다.

App-Centric Data Service에는 Memory Machine과 Memory Viewer가 위치한다. Memory Machine의 세부 기능은 다계층 메모리 관리 및 데이터 압축, 이기종 메모리 간 페이지 이동에 필요한 스냅샷 및 실시간 페이지 이동 관리 기능, 메모리 간 데이터 공유 기능으로 구성된다.

Memory Viewer는 응용을 프로파일링하고, 자원을 모니터링하는 기능을 수행한다. 이러한 시스템 소프트웨어 기술들의 지원을 통해 사용자 소프트웨어 영역의 응용들은 충분한 메모리를 이용해서 성능을 크게 향상시킬 수 있다.

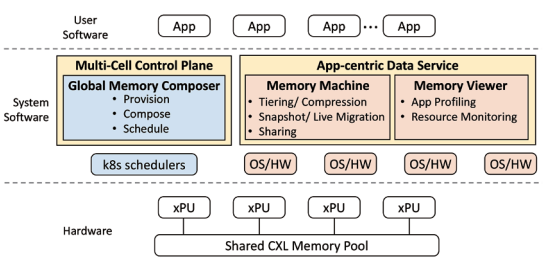

그림 7 CXL 메모리 풀 관리를 위한 필수 소프트웨어 기술

Ⅳ. 결론

본고에서는 CXL 표준과 주요 CXL 장치들을 살펴보고 CXL 메모리의 최신 관련 연구들을 소개하였다. 특히, CXL 메모리 활용 소프트웨어 기술에 해당하는 에뮬레이터들과 CXL 메모리 장치 및 시스템의 활용을 지원하는 CXL 메모리 SDK들을 살펴보았다.

CXL 메모리 에뮬레이터들은 CXL 메모리 장치의 출시에 앞서 해당 장치들에 최적화된 CXL 소프트웨어를 선행 개발할 수 있도록 개발 환경을 제공하는 중요한 역할을 하고 있었다. CXL 메모리 SDK들은 앞으로 출시될 CXL 메모리 장치 및 시스템의

쉽고 효과적인 활용을 지원하는 기능들로 개발되고 있었다. 메모리 사용량이 많은 데이터집약적 응용들은 CXL 메모리 장치 및 시스템과 해당 소프트웨어들을 함께 사용하여 데이터 처리 성능을 크게 향상할 수 있을 것으로 기대된다.

현재 CXL 시장은 CPU, 메모리, 스위치 분야의 선도 기업들이 단일 시제품 개발에 집중하고 있으며, 아직 상용화된 제품들이 부족한 상태지만 '29년까지 연평균 35% 성장할 것으로 예상되는 중요한 시장이다[28]. ETRI 슈퍼컴퓨팅기술연구센터에서는 대규모 데이터처리 응용의 성능 향상을 위한 CXL 기반 MPI 분산병렬처리 구조 및 성능 분석 연구를 수행 중이다. MPI는 단일 컴퓨팅 노드의 자원만으로는 처리가 불가능한 인공지능 분야 대규모 모델과 데이터를 다중 컴퓨팅 노드에서 분산 처리하는 데 필수적으로 사용되는 프로그래밍 모델이다. 해당 연구에서는 CXL 메모리 기술과 MPI 기술의 융합을 통해 고성능 컴퓨팅 시스템의 메모리 자원 이용 효율과 데이터 처리 성능을 대폭 향상시키는 것을 목표로 하고 있다. 이상의 연구에서 개발되는 기술은 컴퓨팅시스템의 전력을 절감하여 탄소중립 달성에도 기여할 것으로 기대된다.

용어해설

CXL(Compute eXpress Link) CPU와 메모리, 가속기, 스토리지 등 컴퓨팅 자원의 효율적 통신을 지원하여 시스템 확장성을 높이는 고속 연결망 기술

메모리 분리(Memory Disaggregation) 기술 메모리 자원을 풀(Pool)로 구성하고 호스트가 필요한 만큼의 메모리를 풀에서 할당/반환하며 메모리 자원 확장성과 이용 효율을 높이는 기술

공유 메모리(Shared Memory) 장치 다중 프로세스들이 프로세스 간 통신 IPC(Inter-Process Communication)를 수행할 때 통신 버퍼로 사용하도록 프로세스들에 공유된 형태로 제공되는 물리적인 메모리 장치

컴포저블(Composable) 컴퓨팅 구조 컴퓨팅 자원 풀을 구성하고 필요한 만큼 할당받아 동적으로 서버를 구성하여 자원 활용 효율을 높이는 구조

약어 정리

API	Application Programming Interface
CPU	Central Processing Unit
CXL	Compute eXpress Link
DRAM	Dynamic Random Access Memory
FPGA	Field-Programmable Gate Array
GPT	General Pre-trained Transformer
GPU	Graphic Processing Unit
LLM	Large Language Model
MPI	Message Passing Interface
NAS	Network Attached Storage
NIC	Network Interface Card
NVRAM	Non-Volatile Random Access Memory
QEMU	Quick EMUlator
RDMA	Remote Direct Memory Access
SDK	Software Development Kit
SoC	System on Chip
VM	Virtual Machine

참고문헌

[1] J. Hoffmann et al., "Training compute-optimal large language models," arXiv preprint, CoRR, 2022, arXiv: 2203.15556.

[2] J.A. Baktash and M. Dawodi, "Gpt-4: A review on advancements and opportunities in natural language processing," arXiv preprint, CoRR, 2023, arXiv: 2305.03195.

[3] C. Guo et al., "Exploring the benefits of resource disaggregation for service reliability in data centers," IEEE Trans. on Cloud Comput., vol. 11, no. 2, 2022, pp. 1651-1666.

[4] I.H. Chung, B. Abali, and P. Crumley, "Towards a composable computer system," in Proc. HPC Asia 2018, (Tokyo Japan), Jan. 2018, pp. 137-147.

[5] MemVerge: The Road to Endless Memory, https://www.youtube.com/watch?v=rO4PdTAwLTY&t=622s

[6] [CES 2023 Innovation Award] Publishing the Boundaries of Memory, https://semiconductor.samsung.com/emea/news-events/tech-blog/pushing-the-boundaries-of-memory-samsung-goes-beyond-hardware-to-become-total-solution-provider-with-cxl-memory-expander/

[7] SK hynix CXL Memory, https://news.skhynix.com/sk-

hynix-develops-ddr5-dram-cxltm-memory-to-expand-the-cxl-memory-ecosystem/

[8] Enfabrica: Scaling CXL Memory Using High Speed Networking, https://www.youtube.com/watch?v=YdJWhqeT5DM&t=919s

[9] Xconn CXL Switches: Enablers of More Advanced HPC and AI/ML Cloud Computing, https://www.youtube.com/watch?v=VvKEHq3xjUw&list=PLsf8NUp2sz_iF3X4s_qazc8c_vYnlAVXv&index=4

[10] J. Sim et al., "Computational CXL-memory solution for accelerating memory-intensive applications," IEEE Comput. Archit. Lett., vol. 22, no. 1, 2022, pp. 5-8.

[11] Y. Fridman et al., "CXL Memory as Persistent Memory for disaggregated HPC: A practical approach," arXiv preprint, CoRR, 2023, arXiv: 2308.10714.

[12] D. Lee et al., "Elastic use of far memory for In-memory database management systems," in Proc. DaMoN 2023, (Seattle, WA, USA), June 2023, pp. 35-43.

[13] Y. Sun et al., "Demystifying CXL memory with genuine CXL-ready systems and devices," in Proc. MICRO 2023, (Toronto, Canada), Oct. 2023, pp. 105-121.

[14] K. 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.

[15] H. Li et al., "Pond: CXL-based memory pooling systems for cloud platforms," in Proc. ASPLOS 2023, vol. 2, (Vancouver, Canada), Mar. 2023, pp. 574-587.

[16] D. Gouk et al., "Memory pooling with cxl," IEEE Micro, vol. 43, no. 2, 2023, pp. 48-57.

[17] M. Kwon et al., "Failure tolerant training with persistent memory disaggregation over CXL," IEEE Micro, vol. 43, no. 2, 2023, pp. 66-75.

[18] Flight Simulator, https://memverge.com/memverge-and-sk-hynix-accelerate-memory-pooling-and-sharing-software-development-with-cxl-flight-simulator/

[19] M. Ha et al., "Dynamic capacity service for improving CXL pooled memory efficiency," IEEE Micro, vol. 43, no. 2, 2023, pp. 39-47.

[20] Jonathan Cameron's QEMU Working Fork, https://gitlab.com/jic23/qemu

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

[22] Scalable Memory Development Kit(SMDK) v1.5, https://github.com/OpenMPDK/SMDK

[23] HMSDK, https://github.com/skhynix/hmsdk

[24] MemVerge Project Gismo: Global IO-free Shared Memory Objects, https://www.youtube.com/watch?v=D66W7eqFbhc

[25] Ray, https://www.ray.io/ray-sgd

[26] Timeplus, https://www.timeplus.com/

[27] MemVerge Memory Viewer CXL and Memory Machine CXL, https://www.youtube.com/watch?v=cwZCORGjCsM

[28] GII Korea, "세계의 CXL 컨트롤러 IP 시장 분석," 2023, https://www.giikorea.co.kr/report/qyr1215847-global-cxl-controller-ip-market-research-report.html