PCIe NTB를 활용한 OpenSHMEM-Light의 설계 및 구현

주영웅, 최민
충북대학교 전자정보대학 정보통신공학부
e-mail: ywjoo@cbnu.ac.kr and mchoi@cbnu.ac.kr

Design and Implementation of OpenSHMEM-Light using PCIe NTB

Youngwoong Ju, Min Choi
Dept of Information and Communication Engineering,
Chungbuk National University

요 약

.PCI Express는 고속, 저전력 등의 특성으로 업계 표준으로서 많이 쓰이고 있는 프로세서와 주변 I/O 장치들을 연결하는 버스 기술이다. 또한, PCI Express는 인피니밴드와 이더넷과 더불어 고성능 컴퓨터나 컴퓨터 클러스터를 위한 시스템 인터커넥트 기술로 널리 쓰이고 있다. PGAS(partitioned global address space) 프로그래밍 모델은 컴퓨터 클러스터와 같은 다중 호스트 시스템에서 단측 RDMA(remote direct memory access)를 구현하는데 많이 이용된다. 본 논문에서는 PCI Express 기반 RDMA를 구현하기 위해 PGAS 프로그래밍 모델인 OpenSHMEM의 기존의 특징을 유지하여 PCI Express 기반 OpenSHMEM API를 설계 및 구현하였다. 구현한 OpenSHMEM API는 PCI Express의 NTB(non-transparent bridge) 기술로 2대의 PC를 연결한 시스템에서 매트릭스 곱셈 예제를 통하여 실험하였다.

1. 서론

오늘날 컴퓨팅 기술과 네트워크 기술이 발전함에 따라 프로세서가 처리해야 할 데이터의 양은 급증하고, 이에 따라 시스템에 있어 고성능의 컴퓨팅이 요구되고 있다. 최근 슈퍼 컴퓨터로 불리는 고성능 컴퓨팅 시스템들은 시스템 인터커넥트 기술에 기반을 두고 있다. 대부분의 슈퍼 컴퓨터에 이용된 인터커넥트 기술은 인피니밴드(infiniband)와 이더넷(ethernet) 기술이 이용되고 있다[1, 2]. 이외의 인터커넥트 기술로 PCI Express가 있으며, PCI Express를 활용한 인터프로세서 시스템이나 GPU 클러스터링 시스템이나 NTB(non-transparent bridge) 기법을 활용한 failover 시스템이 많이 연구되고 있다[3, 4]. 이러한 인터커넥트 기술로 구현된 클러스터링 시스템에서 노드 간의 통신에서 많은 양의 데이터를 고속으로 전송하고 CPU의 적은 부하를 위해 DMA(direct memory access) 기법이 많이 사용되고 있다.

응용 프로그램에서 보통 데이터를 보내는 송신 측과 데이터를 받는 수신 측의 양측(two-sided) 통신이 많이 이용되고 있다. 하지만 슈퍼 컴퓨터와 같이 노드의 개수가 많아질수록 송신 측과 수신 측의 일치가 이루어져야 하는 양측 통신은 시스템의 오버헤드를 야기할 수 있다. 이와 같은 문제점은 다른 노드의 메인 메모리에 직접적으로 데이터를 쓰거나 읽는 RDMA(remote direct memory access) 기법을 이용한 단측(one-sided) 통신을 통해 해결할 수 있다. 단측 통신은 한 노드에 의해 시작되는 RDMA 전송을 이용하기 때문에 양측 통신과 같이 송신 측과 수신 측의 일치가 필요하지 않다[5].

흔히 단측 통신을 구현하기 위해 PGAS(partitioned global address space) 프로그래밍 모델이 이용되고 있다. PGAS 프로그래밍 모델은 모든 노드들이 공유하며 접근할 수 있는 전역 주소 공간(global address space)을 분할하여 각 노드들의 메인 메모리에 할당하여 RDMA를 통한 단측 통신을 가능하게 한다[6].

본 논문에서는 PGAS 프로그래밍 API인 OpenSHMEM를 PCI Express 기반 다중 호스트 시스템에 적용할 수 있도록 설계 및 구현하였다. 2장에서는 클러스터링 컴퓨팅, 시스템 인터커넥트와 PGAS 프로그래밍에 대한 관련 연구를 설명하고, 3장과 PCI Express와 OpenSHMEM의 개요와 특징 그리고 설계 및 구현 방안에 대해 기술하고 4장에서는 구현한 OpenSHMEM API를 간단한 매트릭스 곱셈 예제의 실험결과를 제시한다.

2. PCI Non-Transparent Bridge (NTB)

PCI Express 기술은 Multi-Host 시스템이나 Host Failover 시스템에 많이 이용될 수 있다[10]. 다음 그림 1은 PCI Express를 적용한 Multi-Host 시스템

의 계층 구조이다.

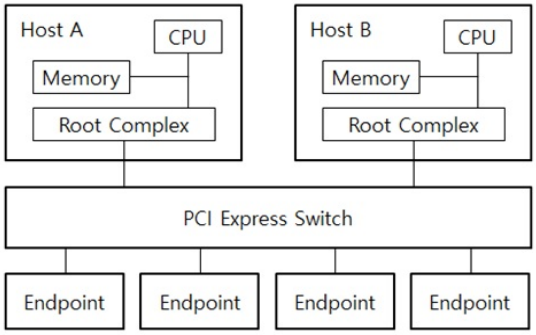

Fig 1. The hierarchy of multi-host system

PCI Express는 Intel 社가 개발한 고속 버스인 PCI에서 성능이 향상된 기술이다. PCI Express는 Full-Duplex 통신이며, 송신과 수신의 데이터 라인을 하나로 묶어 레인이라는 독립적인 링크를 통해 데이터를 송수신한다. PCI Express는 단일 레인(X1)부터 다수의 레인(X2, X4, X8, X16)을 사용하여 링크 폭을 조절할 수 있다. PCI Express는 Gen3 단일 레인 기준 8Gbps의 데이터 전송률의 성능을 가지고 있다. 다음 표 1은 PCI Express의 세대/링크 폭에 따른 데이터 전송률을 보여준다[2].

Table 1. Transfer Speed of PCI Express

Generation	Lane(width)				
	X1	X2	X4	X8	X16
Gen1 (Gbps)	2.5	5	10	20	30
Gen2 (Gbps)	5	10	20	40	60
Gen3 (Gbps)	8	16	32	64	96

PCI Express를 이용한 시스템 인터커넥트를 구현할 때 서로 다른 시스템을 상호 격리시키기 위해 사용되는 것이 Non-Transparent Bridge (NTB) 기술이다[11]. NTB는 하나의 포트 내부에 Endpoint와 같은 역할을 하는 두 가지의 Interface를 논리적으로 분리시켜 내장하고 있다. BIOS/OS Enumeration시 NTB 내부의 논리적으로 분리된 인터페이스로 인해 NTB 포트로 연결된 두 시스템은 서로 보여지지 않는다. 다음 그림 2는 PCI Express Switch의 NTB를 이용하여 두 개의 Host를 연결한 모습이다.

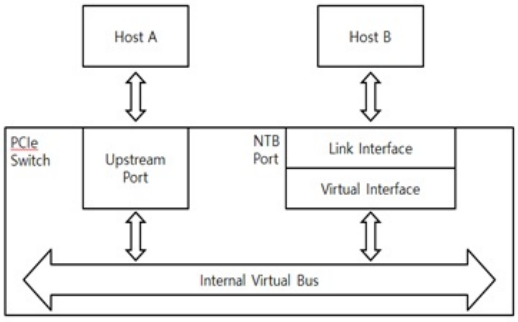

Fig 2. System interconnect with NTB

NTB로 인해 두 시스템은 상호 격리되어 일반적으로는 통신할 수 없기 때문에 Virtual Interface와 Link Interface 사이에서 두 Host의 메모리 주소 변환을 통해 통신할 수 있다. 주소 변환 시에 상대 Host의 메모리 주소를 알 필요가 있다. NTB 내부에는 각각의 Interface에서 접근할 수 있는 Scratchpad 레지스터가 존재한다. 두 Host는 Scratchpad 레지스터를 통하여 자신의 메모리 주소를 쓰고 다른 Host의 메모리 주소를 읽을 수 있다. 또한 각각의 Interface에 Doorbell 레지스터를 가지고 있어 상대 Host로 인터럽트를 알릴 수 있다.

3. OpenSHMEM Light 설계 및 구현

OpenSHMEM는 PGAS 프로그래밍 모델에 사용되는 커뮤니케이션 라이브러리이다. OpenSHMEM의 주된 특징으로는 one-sided 방식의 point-to-point 통신과 애플리케이션 내의 모든 프로세서끼리 서로 접근할 수 있는 공유 메모리, 프로그램 내의 전역적으로 공유 가능한 대칭적인 변수들을 통한 원자적 연산(atomic operation) 등이 있다. 이와 같은 대칭적인 변수들은 런타임동안 각 프로세스마다 OpenSHMEM에서 symmetric heap이라고 불리는 heap 영역에 할당된다. 이러한 특징들은 인피니밴드와 같은 interconnect network에서 지원하는 RDMA(Remote Direct Memory Access)의 사용을 가능하게 한다.

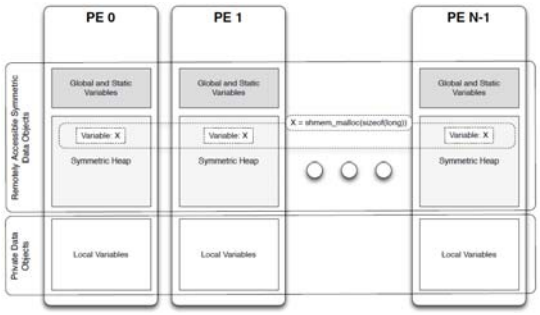

Fig 3. Memory model of OpenSHMEM

그림 3은 OpenSHMEM에서의 원격으로 접근 가능한 symmetric data object와 local data object를 이용한 PGAS 모델이다[12]. OpenSHMEM 프로그램은 각 PE(Process Element)만이 접근할 수 있는 local data object와 모든 PE들이 원격으로 접근 가능한 global data object로 이루어져있다. local data object는 각 PE의 local memory에 저장되며, 오직 자신만이 접근 가능하다. 반대로, global data object는 symmetric data object라고 불리며, OpenSHMEM 루틴들을 통하여 다른 PE들이 접근할 수 있다. 이 symmetric data object는 모든 PE들 상에 같은 이름, 타입, 크기로 할당된다.

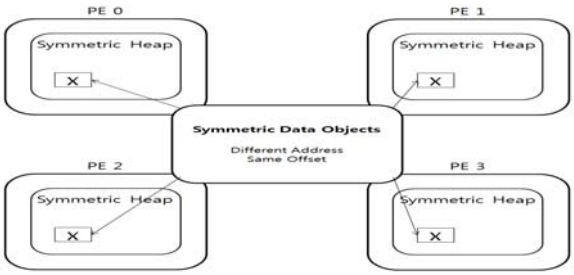

Fig 4. Symmetric data object

symmetric data object는 shmem_malloc 함수의 호출을 통해 그림 4와 같이 symmetric heap이라는 특별한 메모리 영역에 할당되며, symmetric heap 영역은 모든 PE들 상에 각기 다른 메모리 영역에 있다. 각 PE들의 symmetric heap 영역에 각각 할당되는 symmetric data object는 메모리 주소가 다를지라도, 각 symmetric heap 영역 안에서의 같은 offset을 가지게 된다.

OpenSHMEM에서 단측 RDMA를 위해 shmem_put이나 shmem_get과 같은 라이브러리를 사용하여 데이터를 전송하게 된다. OpenSHMEM에서의 put 함수는 단측 통신을 위해 source로부터 데이터가 전송되기 시작할 때 리턴된다. destination으로 데이터 전송이 완료되었는지를 확인할 필요가 없고, 데이터 전송이 완료될 때가 아닌 전송 시작할 때 리턴되어 DMA 전송이 언제 완료될지 모르기 때문에 오직 shmem_barrier_all과 같은 동기화 이후에서만 데이터 전송 완료가 보장된다.

OpenSHMEM에서는 shmem_barrier_all 함수를 통해 모든 PE들의 동기화와 put 함수의 호출로 인한 모든 RDMA 전송이 완료됨을 확인한다. shmem_barrier_all 함수는 PE가 베리어에 도착했음을 등록하고 모든 PE들이 shmem_barrier_all 함수를 호출할 때까지 프로그램 실행을 중지하고 대기한다.

Table 2. Prototype and example of malloc function

Prototype	void *shmem_malloc(size_t size);
Example	double *dest = (double *)shmem_malloc(sizeof(*dest));

Table 3. Prototype and example of double_put function

Prototype	void shmem_double_put(double *dest, const double *source, size_t nelems, int pe);
Example	shmem_double_put(dest, &src, 1, 1);

Table 4. Prototype and example of barrier_all function

Prototype	void shmem_barrier_all();
Example	shmem_barrier_all();

OpenSHMEM에서 shmem_malloc 함수 호출을 통해 각 PE의 symmetric heap 영역으로부터 모든 PE들이 접근할 수 있는 symmetric data object를 할당한다. shmem_malloc 함수는 표 2와 같이 매개변수로 받는 size byte만큼의 블록을 가리키는 포인터를 반환한다.

OpenSHMEM에서 shmem_double_put 함수는 표3과 같이 local PE 내의 매개변수 source(private이나 global) 메모리 블록으로부터 매개변수 dest로 데이터 전송이 시작된 이후에 리턴된다.

OpenSHMEM에서 shmem_barrier_all 함수는 표4와 같이 먼저 shmem_barrier_all 함수가 호출되면 이전에 요청된 DMA 전송들이 완료되었는지를 확인한다. DMA 전송이 완료되었다면 DMA 엔진은 자신의 호스트에게 DMA 전송이 완료되었다는 것을 알리기 위해 DMA done interrupt를 발생시킨다.

DMA 드라이버 레벨의 인터럽트 핸들러에서는 DMA done interrupt를 확인하여 인터럽트가 발생한 디바이스 단위로 인터럽트 개수를 증가시킨다. 이후에 발생한 인터럽트 개수와 요청된 DMA 전송 개수를 체크하여 같다면 자신이 베리어에 진입했음을 알리기 위해 다른 PE들에게 도어벨 인터럽트를 발생시킨다.

4. 결 론

본 논문에서는 고성능 컴퓨팅이나 클러스터 컴퓨팅에 많이 사용되는 시스템 인터커넥트 기술 중 PCI Express 기반 RDMA를 구현하기 위해 PGAS 프로그래밍인 OpenSHMEM API를 설계하고 구현하였다. OpenSHMEM는 흔히 인피니밴드 기반 컴퓨터 클러스터 환경에서 쓰이며, shmem_malloc 함수로 할당한 대칭적인 글로벌 데이터 오브젝트를 shmem_put 등의 함수로 단측 RDMA 전송하는 커뮤니케이션 라이브러리이다. 구현한 OpenSHMEM API는 간단한 매트릭스 곱셈 예제를 통해 클러스터링에 많이 쓰이는 PCI Express의 non-transparent bridge를 이용하여 2대의 PC를 연결한 시스템에서을 실험하였고, host들이 나누어 계산한 매트릭스 곱셈 연산 결과를 단측 RDMA로 통해 확인할 수 있었다.

Acknowledgments

This work is supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (NRF-2010-0025748).

참고문헌

[1] Ravi Budruk, PCI Express Basics, PCI-SIG
[2] Wikipedia, https://en.wikipedia.org/wiki/PCI_Express
[3] http://www.tldp.org/LDP/tlk/dd/pci.html
[4] Mark J. Sullivan, Intel Xeon Processor C5500/C3500 Series Non-transparent Bridge, Intel white paper
[5] Jack Regula, Using Non-transparent Bridging in PCI Express Systems, PLX Technology, Inc.
[6] Non-transparent Bridging with IDT 89HPES32NT24G2 PCI Express NTB Switch, Application Note AN-724, IDT
[7] Kwok Kong and Ale Chang, PCI express System Interconnect Software Architecture for x86-based Systems, Application Note AN-571, IDT
[8] Kwok Kong, Enabling Multi-peer Support with a Standard-Based PCI Express Multi-ported Switch, white paper, IDT
[9] Weihang Jiang, Jiuxing Liu, hyun-Wook Jin, D.K. Panda, W. Gropp and R. Thakur, "High performance MPI-2 one-sided communication over InfiniBand", Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on, pp. 531-538, April. 2004.
[10] Yong-Hwan Lee, Do-Suk Kim and Sang Yoon Oh, "QoS and Flow Control Support on PCI Express Interface Architecture", Korea Institute of Information Technology Magazine, pp. 45-52, Dec. 2009.
[11] NTB white papers, AVAGO TECHNOLOGIES, http://www.avagotech.com/support/download-search. [Accessed: August. 29, 2016]
[12] OpenSHMEM Specification document, http://openshmem.org/site/Specification. [Accessed: August. 29, 2016]