KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.3 pp.105~112 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.3.105

Design and Implementation of Initial OpenSHMEM Based on PCI Express

Young-Woong Joo† · Min Choi††

ABSTRACT

PCI Express is a bus technology that connects the processor and the peripheral I/O devices that widely used as an industry standard because it has the characteristics of high-speed, low power. In addition, PCI Express is system interconnect technology such as Ethernet and Infiniband used in high-performance computing and computer cluster. PGAS(partitioned global address space) programming model is often used to implement the one-sided RDMA(remote direct memory access) from multi-host systems, such as computer clusters. In this paper, we design and implement a OpenSHMEM API based on PCI Express maintaining the existing features of OpenSHMEM to implement RDMA based on PCI Express. We perform experiment with implemented OpenSHMEM API through a matrix multiplication example from system which PCs connected with NTB(non-transparent bridge) technology of PCI Express. The PCI Express interconnection network is currently very expensive and is not yet widely available to the general public. Nevertheless, we actually implemented and evaluated a PCI Express based interconnection network on the RDK evaluation board.. In addition, we have implemented the OpenSHMEM software stack, which is of great interest recently.

Keywords : PGAS, OpenSHMEM, PCI Express, NTB, RDMA, One-Sided Communication

PCI Express 기반 OpenSHMEM 초기 설계 및 구현

주 영 웅† · 최 민††

요 약

PCI Express는 고속, 저전력 등의 특성으로 인하여 프로세서와 주변 I/O 장치들을 연결하는 업계 표준의 버스 기술이다. PCI Express는 최근 고성능 컴퓨터나 클러스터/클라우드 컴퓨팅 등의 분야에서 시스템 인터커넥션 네트워크로서 그 활용가능성을 검증하고 있는 추세이다. PCI Express가 시스템 인터커넥션 네트워크로서 활용가능하게 된 계기는 PCI Express에 NTB(non-transparent bridge) 기술이 도입되면서부터이다. NTB 기술은 물리적으로 두 PCI Express subsystem을 연결가능하도록 하지만, 필요할 경우 논리적인 격리(isolation)를 제공하는 특징이 있다. 또한, PGAS(partitioned global address space)와 같은 공유 주소 공간(shared address space) 프로그래밍 모델은 최근 멀티코어 프로세서의 보편화로 인하여 병렬컴퓨팅 프레임워크로 각광받고 있다. 따라서, 본 논문에서는 차세대 병렬컴퓨팅 플랫폼을 위하여 PCI Express 환경에서 OpenSHMEM을 구현하기 위한 초기 OpenSHMEM API를 설계 및 구현하였다. 본 연구에서 구현한 15가지 OpenSHMEM API의 정확성을 검증하기 위해서 Github의 openshmem-example 벤치마크의 수행을 통하여 확인하였다. 현재 시중에서는 PCI Express 기반 인터커넥션 네트워크는 가격이 매우 비싸고 아직 일반인이 사용하기 용이하도록 NIC형태로 널리 보급되지 않은 실정이다. 이러한 기술개발 초기단계에서 본 연구는 PCI Express 기반 interconnection network를 RDK(evaluation board) 수준에서 실제로 동작하는 실험환경을 구축하고, 여기에 추가로 최근 각광받는 OpenSHMEM software stack를 자체적으로 구현하였다는 데 의의가 있다.

키워드 : PGAS, OpenSHMEM, PCI Express, NTB, RDMA, One-Sided Communication

1. 서 론

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 인터커넥션 네트워크 기술에 기반을 두고 있다. 대부분의 슈퍼 컴퓨터에

※ This Research has been jointly supported by a peroject No. R0101-16-0081, Research and Development of Dual Operating System Architecture with High-Reliable RTOS and High-Performance by ETRI and a project No. K-16-L01-C03 by KISTI.
† 준 회 원 : 충북대학교 정보통신공학부 석사과정
†† 종신회원 : 충북대학교 정보통신공학부 교수
Manuscript Received : September 5, 2016
First Revision : November 11, 2016
Accepted : December 13, 2016
* Corresponding Author : Min Choi(mchoi@cbnu.ac.kr)

이용된 인터커넥션 네트워크 기술은 인피니밴드(infiniband)와 이더넷(ethernet) 기술이 이용되고 있다[1, 2]. 이외의 인터커넥션 네트워크 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 인터커넥션 네트워크 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 인터커넥션 네트워크와 PGAS 프로그래밍에 대한 관련 연구를 설명하고, 3장과 4장에서는 각각 PCI Express와 OpenSHMEM의 개요와 특징들을 기술한다. 5장과 6장에서는 PCI Express 기반 OpenSHMEM의 설계 및 구현에 대해 기술하고 7장에서는 구현한 OpenSHMEM API를 간단한 매트릭스 곱셈 예제를 통해 실험하였다. 마지막으로 8장에서는 결론과 향후 연구에 대하여 기술한다.

2. 관련 연구

고성능 컴퓨터나 컴퓨터 클러스터를 위해 시스템 인터커넥션 네트워크 기술에 대해 많이 연구되어왔다. 고성능 컴퓨터나 컴퓨터 클러스터에 적용된 대부분의 시스템 인터커넥션 네트워크 기술로는 인피니밴드와 이더넷 기술이 있으며, 이외에도 PCI Express 기반 시스템 인터커넥션 네트워크도 많은 연구가 되고 있다[2-7].

또한, 인피니밴드와 이더넷의 시스템 인터커넥션 네트워크 기반의 단측 RDMA를 구현하기 위해 병렬 처리에 많이 쓰이는 MPI (message passing interface)를 이용하기도 한다[8, 9]. 인피니밴드를 이용한 컴퓨터 클러스터에서는 MPI 모델의 로드 밸런싱이나 성능적인 측면의 문제점을 보완하기 위해 PGAS 프로그래밍 모델을 이용하고 있다[5, 6].

PCI Express 기반 인터커넥션 네트워크 기술에 관련해서는 Dolphinics[13]사에서 PCI Express 기반으로 local network을 생성하는 솔루션을 제공한다. PCI Express의 high throughput과 low latency 장점을 활용하여 storage 파일 및 데이터의 빠른 데이터 전송을 가능하도록 하며, system offload를 실현한 것이 특징이다. Dolphinics에서는 PCI Express Switch인 IXS600 Gen3 switch와 PCI Express Adpater card인 PXH812 PCI Express Gen3 Host and Target Adapter를 판매하고 있다. IXS600 PCI Express switch는 powerful, flexible, Gen2 switching 솔루션을 제공하며, IDT사의 transparent and NTB (non-transparent bridging)기술을 활용하여 Dolphin의 소프트웨어 기술을 접목함으로써 I/O scaling과 inter-processor communication을 통한 clustering technology를 실현하였다. IXS600을 활용하면 다수의 PCI Express 장치들을 통해 고성능의 컴퓨팅 클러스터를 구축할 수 있다.

IDT사[14]는 PCI Express network (switch, bridge, signal integrity, timing 솔루션 등)을 구축하기 위한 extensive product portforlio를 보유하고 있다. 신호 무결성 제품(Signal Integrity Product), 최대 64Lane, 24 port 까지 지원하는 switch로서 자유로운 port configuration 가능하며 최대 8가지 NTB function을 바탕으로 한 multi-root application, PCIe to PCI/PCI-X Bridge, PCI-X to PCI-X Bridge, PCI to PCI Bridge, 타이밍(Timing) 관련 부품(clock synthesizer, spread spectrum clock generator, PLL zero-delay buffer, Jitter attenuators 등)을 제공한다.

3. PCIe Non-transparent Bridge(NTB) 기술

PCI Express Switch는 가용할 수 있는 범위 내의 PCI Express의 사양에 준수한 레인들과 포트들을 이용하여 하나의 Host에 여러 주변 I/O 장치들이나 여러 Host와도 연결함으로써 높은 I/O 확장성과 고성능, 고속의 특징을 지원하는 기술이다. PCI Express Switch 기술은 Multi-Host 시스템이나 Host Failover 시스템에 많이 이용된다[10]. Fig. 1은 PCI Express Switch를 적용한 Multi-Host 시스템의 계층 구조이다.

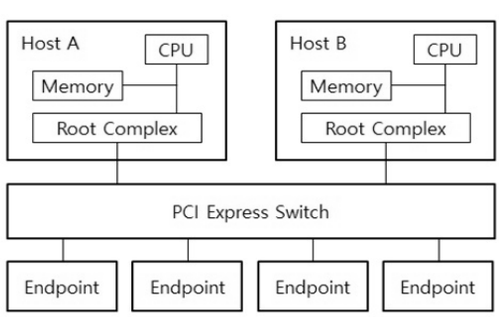

Fig. 1. The Hierarchy of Multi-host System

3.1 PCI Express

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. Table 1은 PCI Express의 세대/링크 폭에 따른 데이터 전송률을 보여주고 있다[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

3.2 Non-Transparent Bridge

PCI Express를 이용한 시스템 인터커넥션 네트워크를 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. Fig. 2는 PCI Express Switch의 NTB를 이용하여 두 개의 Host를 연결한 모습이다.

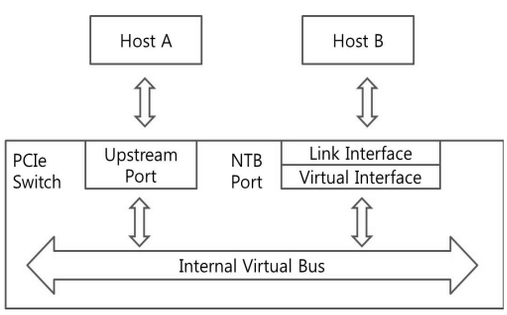

Fig. 2. System Interconnect with NTB

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.

4. OpenSHMEM : Open Shared Memory

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.

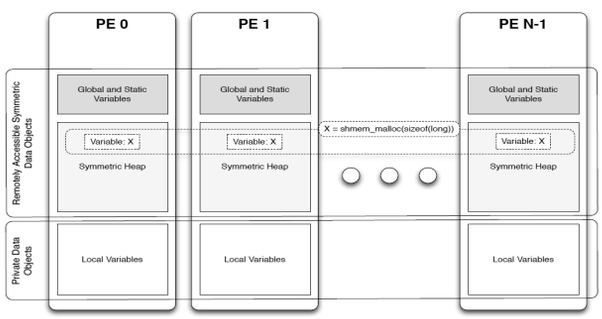

Fig. 3. Memory Model of OpenSHMEM

Fig. 3은 OpenSHMEM에서의 원격으로 접근 가능한 symmetric data object와 local data object를 이용한 PGAS 모델이다[12]. 원래 GPI는 과거 공유 주소 공간(shared address space) 모델을 기반으로 하는 것으로서, 최근 멀티코어 아키텍처가 각광받으면서 Fig. 3에서 각 노드는 전체 글로벌 메모리의 일부 파티션을 가지고 있으며,

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.

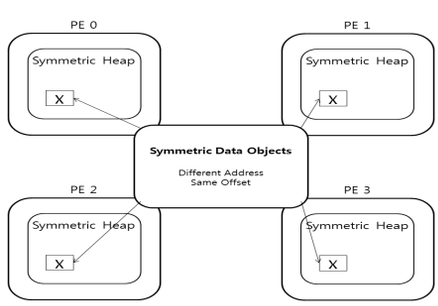

Fig. 4. Symmetric Data Object

symmetric data object는 shmem_malloc 함수의 호출을 통해 Fig. 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.

이와 같은 OpenSHMEM의 특징들을 바탕으로 5장과 6장에서는 PCI Express 기반의 RDMA를 구현하기 위해 OpenSHMEM의 shmem_malloc, shmem_double_put, shmem_barrier_all 함수의 설계 및 구현에 대해 기술한다.

5. PCI Express 기반 OpenSHMEM API 설계

5장에서는 OpenSHMEM의 shmem_malloc, shmem_double_put, shmem_barrier_all 함수의 설계에 대해 기술한다.

5.1 shmem_malloc 함수 설계

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

Table 2. Prototype and Example of Malloc Function

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

이 블록 공간은 symmetric heap 영역으로부터 할당된다. 따라서, 애플리케이션 내의 모든 PE들은 shmem_malloc 함수를 통해 대칭적이고 원격적으로 접근 가능한 메모리 블록들을 할당할 수 있고, 이 메모리 블록들은 shmem_put이나 shmem_get과 같은 OpenSHMEM 통신 루틴에 사용될 수 있다. 각 PE에서 할당된 대칭적인 메모리 블록들은 서로 메모리 주소가 다를지라도, 각 PE의 symmetric heap 영역 내에서 같은 offset을 가지고 있다.

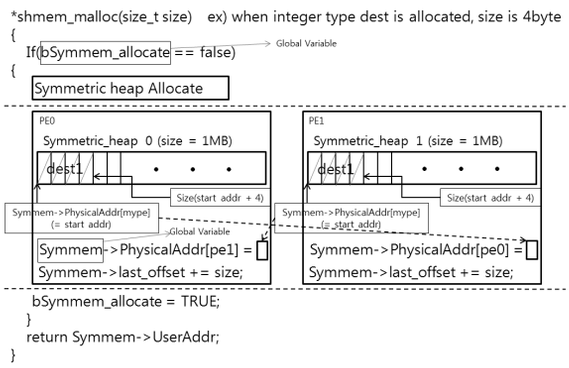

Fig. 5. Design of shmem_malloc function

symmetric data object들은 symmetric heap 영역으로부터 할당되기 때문에, PCI Express 환경의 shmem_malloc 함수는 먼저 symmetric heap 영역이 생성되었는지를 체크하도록 설계하였다. symmetric heap 영역이 생성되지 않았다면 Symmetric heap을 할당하여 할당된 Symmetric heap의 물리주소와 크기를 NT 포트의 scratchpad register를 통해 모든 PE가 주고 받는다. 이후, shmem_malloc 함수의 파라미터인 size만큼 Symmetric heap의 시작주소부터 Symmetric heap을 쪼개서 symmetric data object가 할당되도록 설계하였다.

예를 들어, Fig. 5에서의 shmem_malloc 함수는 int형 dest를 할당하는 예시로, 처음으로 shmem_malloc 함수가 호출된 상태이고, size가 4이기 때문에 last_offset에 4만큼 더하고, Symmetric Memory의 시작주소에 해당하는 가상주소를 반환하게 된다. last_offset은 다음 호출되는 shmem_malloc 함수를 통하여 symmetric 메모리 블록을 할당할 때 사용된다. 또한, shmem_malloc 함수를 통해 할당하고자 하는 메모리 블록의 크기만큼의 메모리가 symmetric heap 영역에 남아있지 않다면, 첫 shmem_malloc 함수 호출할 때와 같이 새로운 symmetric heap 영역을 생성하여 메모리 블록을 할당한다.

5.2 shmem_double_put 함수 설계

OpenSHMEM에서의 shmem_double_put 함수는 local PE 내의 매개변수 source (private이나 global) 메모리 블록으로부터 매개변수 dest로 데이터 전송이 시작된 이후에 리턴된다. shmem_double_put 함수의 dest는 put 함수를 통해 데이터를 받을 메모리 블록으로 매개변수 pe를 통해 local 메모리인지 remote 메모리인지를 판단한다. 즉, 매개변수 dest가 local 메모리 블록의 주소일지라도 매개변수 pe에 따라 pe 번호를 가지는 PE의 dest로 주소변환을 통하여 보내야 할 데이터의 크기만큼(매개변수 nelems) 데이터 전송을 하게 된다.

PCI Express 환경의 shmem_double_put 함수는 먼저 매개변수인 source가 local 메모리 블록인지 symmetric 메모리 블록인지 체크하도록 설계하였다. source가 symmetric 메모리 블록이라면 DMA 전송의 source address로 매개변

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);

수 source의 물리주소를 사용하면 되지만, 물리주소를 이용한 전송인 DMA의 특성상 가상주소를 가지는 local 메모리 블록의 경우에는 물리메모리를 할당하여 사용하여야 한다. 이후에, put 함수의 매개변수인 dest는 symmetric 메모리 블록이어야 하며, 매개변수 pe의 dest로 주소변환을 하여 DMA 전송의 destination address로 설정하여 DMA 전송을 시작하고 리턴하도록 설계하였다. 예를 들어, Fig. 6과 같이 PE 0에서는 PE 0의 src의 데이터를 PE 1의 dest2로 DMA 전송을 하고, PE 1에서는 PE 1의 src의 데이터를 PE 0의 dest 2로 DMA 전송을 하게 된다.

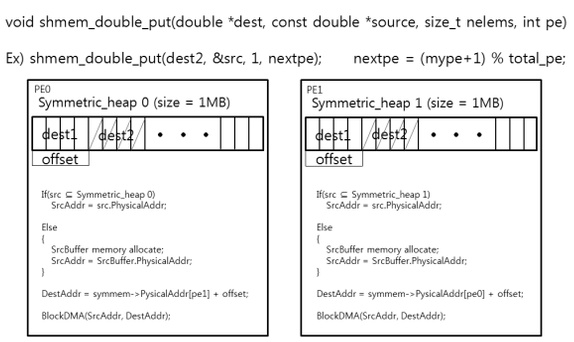

Fig. 6. Design of shmem_double_put function

5.3 shmem_barrier_all 함수 설계

Fig. 7은 PCI Express 환경의 shmem_barrier_all 함수의 설계이다. 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.

Table 4. Prototype and Example of barrier_all function

Prototype	void shmem_barrier_all();
Example	shmem_barrier_all();

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.

또한, NT 드라이버 레벨의 인터럽트 핸들러에서는 다른 PE들이 베리어에 진입했음을 알리는 도어벨 인터럽트가 발생하면 도어벨 인터럽트가 발생한 디바이스 단위로 인터럽트 개수를 증가시킨다. 이후에 발생한 인터럽트 개수와 자신을 제외한 전체 PE의 개수가 같은지를 확인한다.

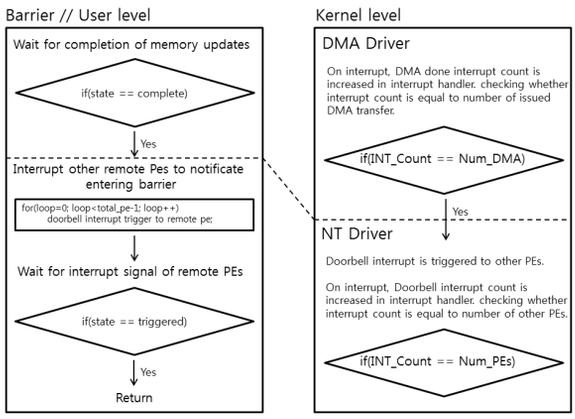

Fig. 7. Design of shmem_barrier_all function

6. PCI Express 기반 OpenSHMEM API 구현

6장에서는 5장에서 설계한 PCI Express 기반 OpenSHMEM의 shmem_malloc, shmem_double_put, shmem_barrier_all 함수의 구현에 대해 기술한다.

6.1 shmem_malloc 구현

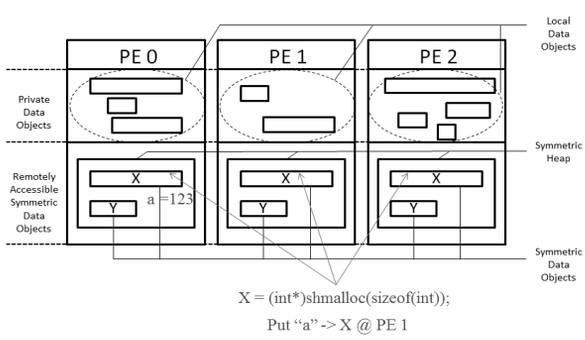

Fig. 8. Design and Implementation of shmem_malloc

Fig. 8은 5.1절에서 설계한 shmem_malloc 함수에 대한 구현으로, address lookup table로 사용될 symmem 구조체를 통해 할당받은 symmetric heap의 주소를 공유한다. PlxPci_PhysicalMemoryAllocate 함수와 PlxPci_PhysicalMemoryMap는 각각 물리 메모리를 할당하고, 할당한 메모리를 커널 가상 주소와 매핑한다. 할당한 메모리의 주소는 broadcast_symmem_buffer와 get_symmem_buffer 함수를 통하여 모든 PE들이 공유할 수 있다. OpenSHMEM의 symmetric memory를 구현하기 위해서 각 PE가 바라보는 symmetric heap의 각 data object들은 동일한 offset을 가진다. 따라서, 각 PE 노드는 다른 PE노드로 데이터를 전송하고자 할 때 목적지 노드의 메모리 주소를 알지 못하더라도 자신이 갖고 있는 data object의 offset을 바탕으로 데이터 전송이 가능한 장점이 있다.

6.2 shmem_double_put 구현

shmem_double_put 구현은 RDMA 기능에 기반한다. 이를 위해서 우선적으로 주소변환(address translation) 과정이 필요한데, PCI Express 버스에서 주소 변환을 위해서 우선적으로 BDF (bus/device/function) scheme에 대한 주소변환이 선행되어야 한다. 이 때, function 정보는 특정한 디바이스에서 수행하는 기능에 대한 정보이기 때문에, 주소변환 전/후라도 차이가 없으므로 주소변환 대상에서 제외한다. BDF 중 bus/device에 대한 정보가 현재의 PCIe subsystem에 존재하지 않는 다른 PCIe subsystem의 정보를 활용하여야 하므로, 이 부분에 대해서 주소변환이 필요하며, Fig. 9와 같이 look-up table(LUT)을 통하여 변환 데이터(busNo/DevNo)에 대한 관리를 수행한다.

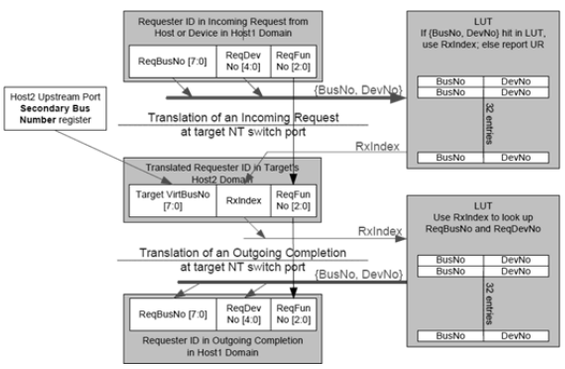

Fig. 9. Address Translation During shmem__double__put

매개변수 'source'의 주소가 로컬 메모리인지 symmetric heap의 메모리인지 확인한다. 로컬 메모리의 경우 가상 주소이기 때문에 물리 주소를 사용하는 DMA 전송을 위해 새로운 물리 메모리를 할당하여 'source'의 값을 카피한 후, 전송한다. 또한, 다른 PE로의 DMA 전송을 할 때 address_ translation 함수를 통해 알맞은 주소 변환을 하여 DMA의 'DestAddr'를 설정한다. 이후에 shmem_barrier_all 함수를 통해 DMA 인터럽트 개수를 확인하기 위해 put 함수의 리턴 전에 DMA 디바이스 오브젝트의 'DMA_Count'를 증가시킨다.

6.3 shmem_barrier_all 구현

shmem_barrier_all 함수는 배리어(barrier) 기능을 구현한 먼저 DMA 전송이 완료되었는지 PlxPci_BarrierDMANotificationWait 함수를 통해 확인한 후에 PlxPci_PlxMappedRegisterWrite 함수로 도어벨 IRQ 레지스터에 값을 써서 인터럽트 신호를 준다. PlxPci_BarrierDBNotificationWait 함수는 다른 PE들로부터 도어벨 인터럽트 개수를 확인한다. 본 연구에서 사용한 방법은 각 노드가 전체 노드 수 -1 만큼의 도어벨(doorbell) 인터럽트를 수신할 때 까지 대기하는 방법으로 barrier를 구현하였다.

PlxNotificationWait 함수는, 유저 레벨에서 PlxPci_Barrier DMANotificationWait 함수나 PlxPci_BarrierDBNotificationWait 함수를 호출했을 때, 각각 DMA 드라이버, NT 드라이버의 커널 레벨에서 인터럽트 개수를 확인하기 위해 호출된다. 인터럽트가 발생할 때마다 디바이스 오브젝트인 'pdx'의 'Interrupt_Count'가 증가하여 유저 레벨에서 전달받은 'Count' 값과 비교한다. PE가 베리어에 진입 후에 인터럽트를 기다리는 상황이면 wait_event_interruptible 함수를 통해 Waitqueue에 진입하여 인터럽트 핸들러에서 깨워주기를 기다린다.

7. 실 험

앞의 5장과 6장에서 설계 및 구현한 API를 기반으로 한 간단한 매트릭스 곱셈 예제에 대한 실험을 기술한다. 실험 환경은 PCI Express로 Host PC 2대를 연결한 시스템으로 Table 5와 같다.

Table 5. Experiment Environment

Host PC hardware environment	
M/B	GIGABYTE GA-H61M-DS2V
Processor	Intel® Core™ i5-3470 CPU @ 3.20GHz X 4
Memory	Samsung DDR3 1333MHz 2GB
O/S	Centos Linux 7 64bit Kernel : 3.10.0-327.el7
Interconnect hardware environment	
PCI Express Switch	PLX PEX8749 RDK 48 Lane, 18 Port, PCI Express Gen 3 Switch
NIC	PLX PEX8732 Cable Adapter X 4
Driver	PLX SDK Device Driver

실행환경은 Fig. 10과 같이 공유파일시스템(Network File System) 으로 파일시스템을 공유하는 환경에서 각 PE 노드가 동일한 바이너리 파일을 볼 수 있도록 설정된 환경에서 gcc 등의 컴파일러를 통해 라이브러리(-lopenshmem)를 포함하여 링크하도록 컴파일한 후 실행하도록 하였다.

Fig. 10. rotget.c Execution Result

예제에서 설정된 ROWS와 COLUMNS는 모두 20으로, 각 PE에서는 매트릭스 a, b, c가 20×20의 크기로 shmem_malloc 함수를 통해 symmetric data로 할당된다. 각 PE들은 자신이 할당받은 매트릭스들을 초기화한 후, 매트릭스 c를 나누어 자신들이 계산해야할 범위인 'C_matrix_start'번째 열부터 'C_matrix_end'번째 열까지 'blocksize'만큼의 매트릭스 a와 매트릭스 b의 곱셈을 실행한다. 계산된 매트릭스 c의 결과는 다른 PE들에게 shmem_double_put/shmem_double_get을 통하여 RDMA 전송을 한다. Fig. 11은 shmem_double_get을 통한 RDMA 전송 이전의 PE 0과 PE 1의 데이터 전송이

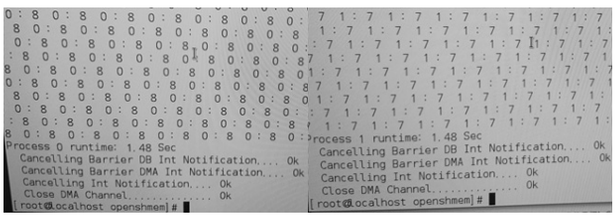

Fig. 11. rotget.c Execution Result

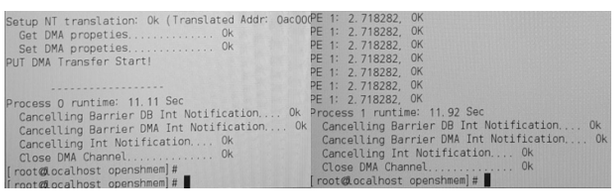

Fig. 12. dip.c Execution Result

올바르게 동작함을 나타낸다.

Fig. 12는 dip.c 예제의 실행인데, 130000개의 메모리 element에 대해서 데이터를 shmem_double_p 함수를 활용하여 전송한 것이다. shmem_double_p와 shmem_double_put 함수는 shmem_double_p 함수는 non-contiguous data를 전송하는 데 사용되며, shmem_double_put 함수는 contiguous data를 전송한다는 점에서 차이가 있다. Fig. 14는 randput.c 실행 결과로써, random하게 데이터를 shmem_double_put연산을 수행한 결과를 보여준다.

Fig. 13은 매트릭스 a와 매트릭스 b의 matrix multiplication 연산을 실행한 것이다. 계산된 매트릭스 c의 결과는 다른 PE들에게 shmem_double_put을 통하여 RDMA 전송되며 매트릭스 A와 매트릭스 B를 곱한 매트릭스 C를 구하는 예제이다.

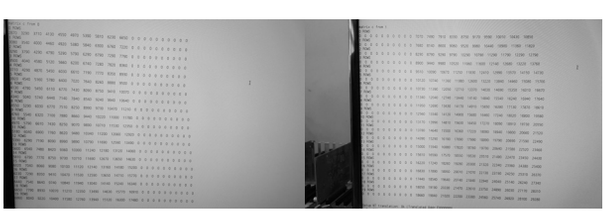

Fig. 13. Matrix c of PE 1 before RDMA Transfer

본 예제에서는 PE 0은 매트릭스 c의 0번째 열부터 9번째 열까지, PE 1은 매트릭스 c의 10번째 열부터 9번째 열까지 계산하고, 각각 PE 0과 PE 1이 PE 1과 PE 0에게 계산한 결과를 RDMA 전송한 후의 매트릭스 c의 결과를 보인 것이다.

8. 결 론

본 논문에서는 고성능 컴퓨팅이나 클러스터 컴퓨팅에 많이 사용되는 시스템 인터커넥션 네트워크 기술 중 PCI Express 기반 RDMA를 구현하기 위해 PGAS 프로그래밍인 OpenSHMEM API를 초기 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 시스템에서 을 실험하였고, host들이 나누어 계산한 Github의 openshmem 벤치마크 예제 수행 결과를통해 확인할 수 있었다.

본 연구는 PCI Express 기반 인터커넥션 네트워크 환경이 채 보급되지도 않은 현 시점에서 PCI Express 기반 interconnection network를 RDK(evaluation board) 수준에서 실제로 동작하는 실험환경을 구축하고, 여기에 추가로 최근 각광받는 OpenSHMEM software stack를 자체적으로 구현하였다는 데 의의가 있다.

향후 shmem_malloc 함수를 통해 할당한 symmetric heap에서 메모리 공간을 나누는 방법 외에 즉시 메모리 공간을 나누어 공유하거나 shmem_put 함수를 통한 DMA 전송의 불필요한 오버헤드를 줄이고 QoS를 향상시켜 PCI Express 기반 OpenSHMEM API를 정형화한다면 OpenSHMEM의 플랫폼의 다양성에 도움이 될 것으로 판단된다.

References

[1] Interconnect family statistics of supercomputer top 500 [Internet], https://www.top500.org/statistics/list. [Accessed: August, 28, 2016].

[2] Y. W. Kim, Y. Ren, and W. Choi, “Design and Implementation of an Alternate System Interconnect based on PCI Express,” *Journal of the Institute of Electronics and Information Engineers*, Vol.52, No.8, pp.74-85, Aug., 2015.

[3] V. Krishnan, “Towards an Integrated IO and Clustering Solution using PCI Express,” *Cluster Computing, 2007 IEEE International Conference on*, pp.259-266, Sept., 2007.

[4] Jong Min Lee, Jung Hwa Lee, and Seong Woo Kim, “Implementation of a GPU Cluster System using Inexpensive Graphics Devices,” *Journal of Korea Multimedia Society*, Vol.14, Issue 11, pp. 1458-1466, Nov., 2011.

[5] J. Dinan, P. Balaji, Jeff R. Hammond, S. Krishnamoorthy, and V. Tipparaju, “Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication,” *Parallel & Distributed Processing Symposium, 2012 IEEE 26th International*, pp.739-750, May, 2012.

[6] J. Breitbart, M. Schmidtobreick, and V. Heuveline, “Evaluation of the Global Address Space Programming interface(GASPI),” *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp.717-726, May, 2014.

[7] Hao Wang, Sreeram Potluri, Devendar Bureddy, Carlos Rosales, and Dhabaleswar K. Panda, “GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation,” *IEEE Transactions on Parallel and Distributed Systems*, pp.2595-2605, Oct., 2014.

[8] Ryan E. Grant, Mohammad J. Rashti, Ahmad Afsahi, and Pavan Balaji, “RDMA Capable iWARP over Datagrams,” *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp.628-639, May, 2011.

[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, Apr., 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 [Internet], http://www.avagotech.com/support/download-search. [Accessed: August, 29, 2016].

[12] OpenSHMEM Specification document [Internet], http://openshmem.org/site/Specification. [Accessed: August, 29, 2016].

[13] IXS600 PCI Express Gen 2 Switch, http://www.dolphinics.com.

[14] Non-transparent Bridging with IDT 89HPES32NT24G2 PCI Express NTB Switch, Application Note AN-724, IDT.

주 영 웅

e-mail : jwy3675@naver.com

2015년 충북대학교 정보통신공학부(학사)

2010년~현 재 충북대학교 정보통신공학부 석사과정

관심분야 : Parallel Computing, Interconnection Network

최 민

e-mail : mchoi@cbnu.ac.kr

2003년 KAIST 전산학과(석사)

2009년 KAIST 전자전산학과(박사)

2008년~2010년 삼성전자 반도체총괄 책임연구원

2010년~2011년 원광대학교 컴퓨터공학과 교수

2011년~현 재 충북대학교 정보통신공학부 교수

관심분야 : Parallel Computing, Interconnection Network, Computer Architecture, Embedded System