ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 44, No. 2, pp. 124-131, 2017. 2
https://doi.org/10.5626/JOK.2017.44.2.124

MPI 노드 내 통신 성능 향상을 위한 매니코어 프로세서의 온-패키지 메모리 활용

(Using the On-Package Memory of Manycore Processor for Improving Performance of MPI Intra-Node Communication)

조 중 연 [†] 진 현 욱 [††] 남 덕 윤 [†††]
(Joong-Yeon Cho) (Hyun-Wook Jin) (Dukyun Nam)

요 약 고성능 컴퓨팅 환경을 위해서 최근 등장한 차세대 매니코어 프로세서는 전통적인 구조의 메모리와 함께 고대역 온-패키지 메모리를 장착하고 있다. Intel Xeon Phi Knights Landing(KNL) 프로세서의 온-패키지 메모리인 Multi-Channel DRAM(MCDRAM)은 기존의 DDR4 메모리보다 이론적으로 네 배 높은 대역폭을 제공한다. 본 논문에서는 MCDRAM을 이용하여 MPI 노드 내 통신 성능을 향상시키기 위한 방안을 제안한다. 실험 결과, 제안된 기법을 사용할 경우 DDR4를 사용하는 경우와 비교해서 MPI 노드 내 통신 성능을 최대 272% 향상시킬 수 있음을 보인다. 또한 MCDRAM 활용 방법에 따른 성능 영향뿐만 아니라 프로세스의 코어 친화도에 따른 성능 영향을 보인다.

키워드: Knights Landing, 온-패키지 메모리, MPI, 노드 내 통신, 고성능 컴퓨팅

Abstract The emerging next-generation manycore processors for high-performance computing are equipped with a high-bandwidth on-package memory along with the traditional host memory. The Multi-Channel DRAM (MCDRAM), for example, is the on-package memory of the Intel Xeon Phi Knights Landing (KNL) processor, and theoretically provides a four-times-higher bandwidth than the conventional DDR4 memory. In this paper, we suggest a mechanism to exploit MCDRAM for improving the performance of MPI intra-node communication. The experiment results show that the MPI intra-node communication performance can be improved by up to 272 % compared with the case where the DDR4 is utilized. Moreover, we analyze not only the performance impact of different MCDRAM-utilization mechanisms, but also that of core affinity for processes.

Keywords: knights landing, on-package memory, MPI, intra-node communication, high-performance computing

· 이 논문은 2016년도 한국과학기술정보연구원의 지원(No.K-16-L01-C03-S03, 매니코어 기반 슈퍼컴퓨터 작업 및 데이터 처리 기술 연구)과 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0101-16-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구)
· 이 논문은 2016 한국컴퓨터종합학술대회에서 '매니코어 프로세서의 온-패키지 메모리에 의한 MPI 노드 내 통신 성능 영향'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 건국대학교 컴퓨터공학부
jycho@konkuk.ac.kr
†† 종신회원 : 건국대학교 컴퓨터공학부 교수(Konkuk Univ.)
jinh@konkuk.ac.kr
(Corresponding author임)
††† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅센터 선임연구원
dynam@kisti.re.kr

논문접수 : 2016년 8월 18일
(Received 18 August 2016)
논문수정 : 2016년 11월 22일
(Revised 22 November 2016)
심사완료 : 2016년 11월 23일
(Accepted 23 November 2016)

정보과학회논문지 제44권 제2호(2017. 2)

1. 서 론

고성능 컴퓨팅 환경을 위한 차세대 매니코어 프로세서는 전통적인 구조의 메모리와 함께 고성능 온-패키지 메모리를 사용할 수 있다[1,2]. 온-패키지 메모리는 프로세서 패키지 외부에 위치하던 기존의 메모리와 다르게 프로세서 패키지 내부에 위치하여 메모리 접근에 집중적인 응용의 처리 속도를 획기적으로 향상시킬 것으로 기대된다[3].

Message Passing Interface(MPI)[4]는 클러스터 시스템을 위한 표준 병렬 프로그래밍 모델로서 고성능 컴퓨팅 시스템에서 널리 사용되고 있다. MPI 통신은 노드 간 통신과 노드 내 통신으로 구분될 수 있는데, 매니코어 시스템의 발전에 따라 노드 내 통신비율이 증가되고 있다. 따라서 노드 내 통신 성능을 향상시키기 위한 연구 들이 진행되었으며[5-9], 코어 수가 점차 증가하는 매니코어 시스템에서의 노드 내 통신 성능은 더욱 중요한 성능요소가 될 것이다. 노드 내 통신은 메모리 집중적인 동작으로 이루어져 있기 때문에 Intel Xeon Phi Knights Landing(KNL) 프로세서가 제공하는 Multi-Channel DRAM(MCDRAM)과 같은 고성능 온-패키지 메모리를 활용한다면 높은 성능향상을 기대할 수 있다.

본 논문에서는 KNL 프로세서의 고성능 온-패키지 메모리를 활용하여 MPI 노드 내 통신 성능을 향상시키고, 고성능 온-패키지 메모리의 효율적인 활용 방안에 대해 논의하기 위해 통신 버퍼의 위치, 프로세스의 위치 등에 따른 성능 차이를 실험하고 분석한다. 본 논문은 새로운 구조를 제안하기 보다는 MPI 수준에서 KNL 프로세서의 MCDRAM을 활용하기 위한 방향성을 제시하고, 고성능 온-패키지 메모리가 MPI 노드 내 통신 성능에 미치는 영향에 대해 분석하여 해당 분야의 연구를 위한 초석을 마련한다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어 2장에서는 KNL 프로세서의 특성과 MPI 노드 내 통신에 대해서 설명하고, 3장에서는 MPI 노드 내 통신 성능 향상을 위해 고성능 온-패키지 메모리를 활용하는 방법에 대해 설명한다. 4장에서는 고성능 온-패키지 메모리를 활용한 MPI 노드 내 통신 성능에 대해 실험하고, 마지막으로 5장에서 본 논문의 결론 및 향후계획을 기술한다.

2. 연구 배경

본 장에서는 KNL 프로세서에 대해 기술하고, 기존 MPI 라이브러리가 노드 내 통신을 구현하는 방법에 대해서 설명한다.

2.1 Intel Xeon Phi Knights Landing(KNL) 프로세서

KNL 프로세서는 2D-mesh 형태로 연결된 32~36개

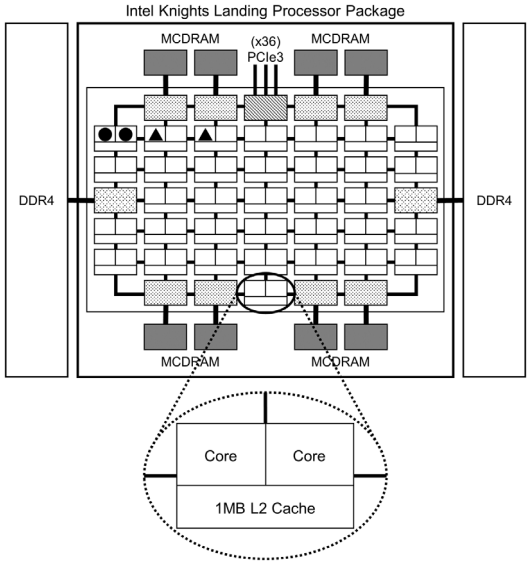

그림 1 KNL 프로세서 구조와 타일 구조
Fig. 1 Diagram of the KNL Processor and the Tile

의 타일로 구성되어 있으며, 각 타일은 L2 캐시를 공유하는 두 개의 코어로 구성되어 있다[1,2]. Intel Xeon Phi Knights Corner(KNC) 프로세서는 PCIe 슬롯에 연결되는 가속기 형태인 반면에, Knights Landing 프로세서는 독립적으로 운영체제를 동작할 수 있는 호스트 프로세서로 동작한다. 그림 1은 KNL 프로세서 구조와 타일 구조를 보여준다.

그림 1에서 볼 수 있듯이 KNL 프로세서는 두 가지 연결 방식의 메모리를 제공하고 있다. 프로세서 패키지 외부에 위치한 DDR4 메모리와 프로세서 패키지 내부에 위치한 MCDRAM이다. MCDRAM은 DDR4와 비교했을 때 이론적으로 네 배의 대역폭 차이를 보이는 고성능 메모리다. MCDRAM은 Cache, Flat, Hybrid 세 가지 모드 중 하나로 설정되어 사용할 수 있다. Cache 모드는 기존의 온-칩 캐시와 유사하게 동작하여 기존의 시스템에서 구현되고 사용되던 응용과 가장 좋은 호환성을 보인다. Flat 모드는 MCDRAM을 DDR4 메모리와 같이 직접 주소 할당할 수 있는 별도의 메모리 모듈로 동작하도록 한다. Hybrid 모드는 MCDRAM의 일부는 Cache 모드와 같이 온-칩 캐시 메모리로, 일부는 Flat 모드와 같이 직접 주소 할당이 가능한 메모리 노드로 동작하도록 한다. 본 논문에서는 MCDRAM을 Flat 모드로 설정하고 활용한다.

하나의 타일을 구성하는 두 개 코어가 공유하는 L2 캐시 역시 MPI 노드 내 통신 성능에 영향을 미칠 수 있다[10,11]. L2 캐시를 공유하는 두 개의 코어에서 통

신이 수행될 경우(그림 1의 원형 한 쌍) 공유된 캐시의 영향을 받지만, 서로 다른 타일에 위치한 두 개의 코어에서 통신이 수행되는 경우(그림 1의 삼각형 한 쌍) 캐시 공유에 의한 영향은 존재하지 않는다. 따라서 본 논문은 프로세스의 위치에 따른 MPI 노드 내 통신 성능을 함께 분석한다.

2.2 MPI 노드 내 통신 방법

MPI는 일반적으로 하나의 프로세서에 하나의 프로세스가 수행되는 환경을 가정하고 있기 때문에 단일 코어 시스템에서는 노드 내 통신에 대해 고려할 필요가 없다. 하지만 매니코어 시스템에서는 노드 내에서 수행되는 프로세스가 코어 수만큼 생성될 수 있기 때문에 노드 내 통신 성능은 중요한 성능 요소이다.

노드 내 통신을 지원하기 위한 방법으로는 대표적으로 세 가지 방법을 사용한다. 첫 번째 방법은 Loopback 인터페이스를 사용하는 방법이다. 가장 단순한 방법이지만 노드 내 통신임에도 불구하고 네트워크 프로토콜 스택을 거쳐야하기 때문에 불필요한 커널 수준의 프로세싱과 데이터 복사 오버헤드가 발생된다. 두 번째 방법은 MPI 수준에서 제공되는 공유 메모리를 사용하는 방법이다[5]. MPI 프로세스가 생성되는 과정에서 MPI 프로세스들이 사용할 수 있는 공유 메모리 영역을 할당해 노드 내 통신에 사용한다. 공유 메모리를 사용할 경우 네트워크 프로토콜 스택을 거치지 않고 노드 내 통신을 수행하지만 여전히 버퍼 복사를 위한 오버헤드가 존재한다. 세 번째 방법은 커널 수준의 메모리 매핑을 통해 데이터 송신 프로세스가 데이터 수신 프로세스의 수신 버퍼에 직접 데이터를 복사하는 방법이다[6]. 버퍼 복사를 위한 오버헤드를 효율적으로 제거할 수 있지만 커널 수준의 지원이 필요하기 때문에 커널 모듈 삽입 과정을 거쳐야하며 커널 인터페이스가 변경될 경우 그에 맞게 수정되어야 하는 단점이 있다.

본 논문에서는 메모리 특성에 따른 MPI 노드 내 통신 성능을 실험하기 위해 MPI 수준에서 제공하는 공유 메모리를 사용한다. Loopback 인터페이스를 사용하는 경우 네트워크 인터페이스 카드(Network Interface Card, NIC)와 드라이버 구현에 의존적이며, 커널 수준의 메모리 매핑 방법의 경우 특정 운영체제 또는 커널 버전에 의존적이기 때문에 MPI 수준에서 제공하는 공유 메모리를 사용하는 방법이 일반적으로 널리 적용될 수 있기 때문이다.

2.3 관련 연구

MPI 수준에서 매니코어 프로세서를 효율적으로 지원하기 위한 연구들은 활발하게 진행되어 왔다. KNC 프로세서의 구조적 특징을 고려한 MPI 구현[12-15]을 비롯해 MPI 노드 내 통신 성능 향상을 위한 연구[16,17]가 진행되었으며, KNC 프로세서가 장착된 클러스터 환경에서의 효율적인 Collective 통신을 위한 연구들[18,19] 역시 진행되었다. 하지만 기존의 연구들은 호스트 메모리와 KNC 프로세서에 장착된 메모리간의 효율적인 데이터 이동과 KNC 프로세서의 링 구조를 지원하기 위한 연구들로 KNL 프로세서의 새로운 구조에 적용하기 어렵다. 또한 KNL 프로세서의 가장 큰 특징인 고성능 온-패키지 메모리를 활용하기 위한 방법에 대해 고려하고 있지 않다.

본 논문에서는 KNL 프로세서의 고성능 온-패키지 메모리를 MPI 수준에서 활용하기 위한 방법을 제안하고 성능 분석을 통해 KNL 프로세서 기반의 시스템에서 MPI 병렬 프로그래밍 모델의 성능 요소에 대해 분석하고 활용 방안에 대해 논의한다.

3. MPI 노드 내 통신 성능 향상을 위한 고성능 온-패키지 메모리 활용

본 장에서는 MCDRAM을 활용하여 MPI 노드 내 통신 성능을 향상시키기 위한 MPI 수준의 접근 방법에 대해 설명한다. 본 논문에서는 MPI 수준에서 제공하는 공유 메모리 환경을 위해 MVAPICH [20]를 사용하였다.

MVAPICH는 MPI 수준의 공유 메모리를 제공하기 위해 MPI 프로세스 생성 과정에서 /dev/shm 디렉터리에 공유 메모리 영역을 위한 가상 파일을 생성한다. 리눅스가 제공하는 /dev/shm 디렉터리는 시스템 초기화 시점에 tmpfs 형태로 생성되기 때문에 DDR4 메모리를 사용한다. 따라서 MVAPICH가 제공하는 MPI 수준의 공유 메모리가 MCDRAM을 활용하도록 MCDRAM을 tmpfs 형태로 /dev/shm 디렉터리 하단에 새롭게 마운트 하고 MPI 프로세스 생성 과정에서 MCDRAM을 사용하는 디렉터리에 파일을 생성하도록 MVAPICH의 소스 코드를 수정하였다.

MPI 표준[4]은 사용자 수준의 메모리 할당을 위해 MPI_Alloc_Mem() 함수를 제정하고 있다. 사용자 수준의 통신 버퍼와 MPI 수준의 공유 메모리가 모두 MCDRAM을 사용하는 환경을 위해 MPI_Alloc_Mem() 함수가 사용자 수준의 통신 버퍼를 MCDRAM에 할당하도록 memkind[21] 라이브러리를 이용해 수정했다. 그림 2는 MCDRAM을 활용하기 위해 수정된 MVAPICH를 도식화한 그림이다.

그림 2에서 볼 수 있듯이 MPI 수준의 수정을 통해 MCDRAM을 활용한다면 응용 코드 수정 없이 응용 수준의 투명성을 유지하면서 MCDRAM의 성능 영향을 얻을 수 있을 것으로 기대된다.

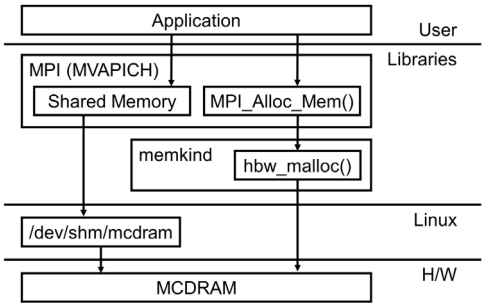

그림 2 MCDRAM 활용을 위한 MPI 수준의 접근 방법
Fig. 2 MPI-Level Approach for which the MCDRAM is used

4. 고성능 온-패키지 메모리의 MPI 노드 내 통신 성능 영향 분석

본 장에서는 실험을 통해 고성능 온-패키지 메모리의 MPI 노드 내 통신 성능 영향에 대해 분석한다. 실험에 사용된 서버는 16 GB의 MCDRAM 및 68개의 코어로 구성된 KNL 프로세서와 96 GB의 DDR4 메모리를 장착하였다. 그리고 리눅스(커널 버전 3.10.0)와 3장에 기술한 수정 사항이 적용된 MVAPICH(버전 2.2b)를 설치하였다. 성능 측정에는 MVAPICH와 함께 배포되는 OSU-Microbenchmarks를 사용하였으며, 캐시 영향을 최소화하기 위해 사용자 수준의 통신 버퍼를 재사용하지 않도록 코드를 수정하였다. 또한 MPI의 eager 프로토콜[5]을 사용하는 32 Kbyte 메시지 크기까지 실험을 진행 하였다.

4.1 메모리 요구량에 따른 MCDRAM의 성능 영향

본 절에서는 고성능 온-패키지 메모리의 MPI 노드 내 통신 성능 영향을 메모리 대역폭 관점에서 관찰하기 위해 통신 프로세스 수를 증가시키면서 실험한다. 하나의 MPI point-to-point 연결은 송신 프로세스와 수신 프로세스가 노드 내 통신을 수행하기 때문에 기본적으로 두 개의 프로세스가 생성된다. 응용의 메모리 요구량을 점진적으로 증가시키기 위해 2개의 MPI 프로세스가 노드 내 통신을 수행하는 환경과 34개의 MPI 프로세스가 노드 내 통신을 수행하는 환경, 68개의 MPI 프로세스가 노드 내 통신을 수행하는 환경을 차례대로 측정하였다. 실험에 사용된 KNL 프로세서는 68개의 물리적인 코어로 구성되어 있기 때문에 최대 34개의 MPI point-to-point 연결이 생성될 수 있기 때문이다. 하나의 MPI point-to-point 연결을 구성하는 두 개의 프로세스는 하나의 타일 내에 위치하도록 설정 하였으며, 다수의 MPI point-to-point 연결에 대한 통신 성능은 동시 생성된 n개의 MPI point-to-point 연결에 대한 통합 대역폭을 측정하였다. 그림 3은 프로세스 수에 따른 통신 성능을 보여준다.

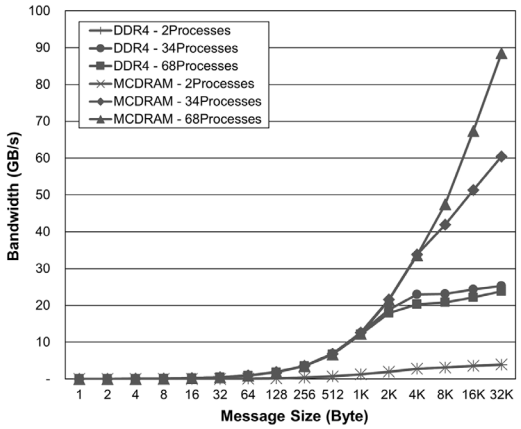

그림 3 프로세스 수에 따른 타일 내 통신 성능
Fig. 3 Performance of the Intra-Tile Communication for Different Numbers of Processes

그림 3에서 볼 수 있듯이 2개의 MPI 프로세스가 노드 내 통신을 수행하는 환경에서 DDR4를 사용한 경우와 MCDRAM을 사용한 경우의 성능 차이가 없는 것을 확인할 수 있다. 하지만 34개의 MPI 프로세스가 노드 내 통신을 수행하는 환경에서 MCDRAM을 사용할 경우 DDR4를 사용한 경우와 비교했을 때 메시지 크기가 32 KByte인 경우 139%의 성능향상을 확인할 수 있으며, 68개의 MPI 프로세스가 노드 내 통신을 수행하는 환경의 경우 메시지 크기가 32 KByte일 때 272%의 성능 향상을 확인할 수 있다. 실험 결과를 통해 적은 수의 MPI point-to-point 연결이 노드 내 통신을 수행할 경우 메모리 대역폭 요구량이 적기 때문에 DDR4를 사용한 경우와 차이가 없지만 다수의 MPI point-to-point 연결이 1 KByte 보다 큰 크기의 메시지를 이용해 통신할 경우 메모리 요구량이 증가함에 따라 MCDRAM의 성능 영향 역시 증가한다는 점을 확인할 수 있었다.

4.2 공유 캐시 및 통신 버퍼 위치에 따른 MPI 노드 내 통신 성능 영향 분석

본 절에서는 타일 내, 타일 간 통신으로 구분하여 통신 프로세스의 위치에 따른 통신 성능을 비교한다. 실험 방법은 4.1절의 실험 결과에 기반을 두어 동시 생성된 34개의 MPI point-to-point 연결에 대해서 통합 대역폭을 측정하였다. 타일 내, 타일 간 통신을 구분하기 위해서 리눅스 운영체제가 제공하는 정보와 sched_setaffinity() 시스템 호출을 이용하여 L2 캐시 공유 구조에 따라 프로세스를 배치하였다. 하나의 MPI point-to-point 연결을 구성하는 두 개의 프로세스가 같은 타일 내에

위치하는 경우 L2 캐시의 영향을 받을 수 있으며, 서로 다른 타일에 위치하는 경우 L2 캐시의 영향을 받을 수 없기 때문이다.

타일 내, 타일 간 통신 성능과 함께 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용하는 경우와 공유 메모리만 MCDRAM을 사용하는 경우를 함께 실험하였다. 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용하는 경우 가장 좋은 성능을 보일 것으로 기대되지만, 응용 구현 과정에서 MPI_ Alloc_Mem() 함수를 사용하지 않은 경우 코드를 수정해야 하는 오버헤드가 존재한다. 하지만 공유 메모리만 MCDRAM을 사용할 경우 응용 구현 방법과 무관하게 기존의 응용을 그대로 사용할 수 있을 것으로 기대되기 때문이다.

4.2.1 타일 내 통신 성능 분석

그림 4는 MPI 노드 내 통신을 수행하는 프로세스 쌍이 동일한 타일에 위치한 코어에서 수행되는 경우에 대한 실험 결과이다. 삼각형은 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용하는 경우를 나타내며, 마름모는 공유 메모리만 MCDRAM을 사용하는 경우를 나타낸다. 그림 4에서 볼 수 있듯이 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용하는 경우 사용자 수준의 통신 버퍼와 공유 메모리가 모두 DDR4를 사용하는 경우와 비교하여 최대 272%의 성능향상을 볼 수 있다.

4.1절의 실험 결과와 같이 메시지 크기가 작은 경우(1 Byte~1 KByte) 데이터 복사에 의한 오버헤드가 크지 않기 때문에 MCDRAM에 의한 성능 차이는 보이지 않는다. 하지만 메시지 크기가 증가할수록 성능 영향이 더 크게 나타나는 것을 확인할 수 있다.

공유 메모리만 MCDRAM을 사용하는 경우 사용자 수준의 통신 버퍼와 공유 메모리가 모두 DDR4를 사용하는 경우와 비교하여 최대 11%의 성능향상을 볼 수 있다. 실험 결과를 통해 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용했을 때 MCDRAM의 성능 영향이 더 큰 것을 확인할 수 있었다.

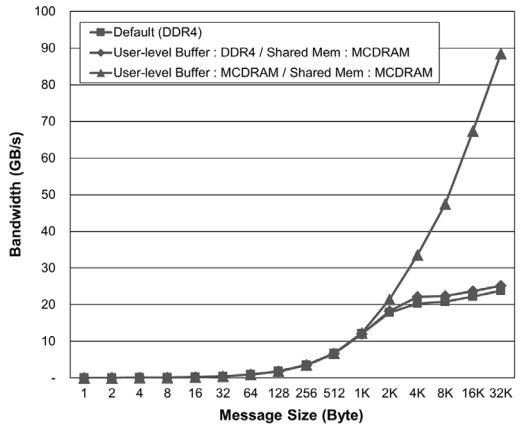

그림 4 통신 버퍼 위치에 따른 타일 내 통신 성능

Fig. 4 Performance of the Intra-Tile Communication for Different Locations of Communication Buffer

4.2.2 타일 간 통신 성능 분석

그림 5는 MPI 노드 내 통신을 수행하는 프로세스 쌍이 서로 다른 타일에 위치한 코어에서 수행되는 경우에 대한 실험 결과이다. 4.2.1항의 실험 결과와 동일하게 메모리 요구량이 증가할수록 MCDRAM에 의한 성능향상이 더 큰 것을 확인할 수 있다. 그 결과 사용자 수준의 통신 버퍼와 공유 메모리가 모두 DDR4를 사용하는 경우와 비교하여 최대 258%의 성능향상을 성취하였다.

공유 메모리만 MCDRAM을 사용하는 경우 역시 4.2.1항의 실험 결과와 동일하게 낮은 성능향상을 볼 수 있다. 사용자 수준의 통신 버퍼와 공유 메모리가 모두 DDR4를 사용하는 경우와 비교해 최대 39%의 성능향상이다.

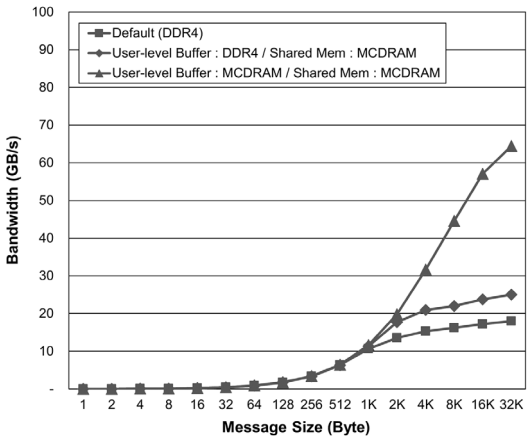

그림 5 통신 버퍼 위치에 따른 타일 간 통신 성능

Fig. 5 Performance of the Inter-Tile Communication for Different Locations of Communication Buffer

4.2.1항의 실험 결과와 본 항의 실험 결과를 통해 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용했을 때 MCDRAM의 성능 영향이 더 큰 것을 확인할 수 있었다. 따라서 3장에서 기술한 바와 같이 MPI_Alloc_mem() 함수가 사용자 수준의 통신 버퍼를 MCDRAM에 할당하도록 MPI 수준에서 지원하고 응용 코드에서 MPI_Alloc_mem() 함수를 사용한다면 응용 수준의 투명성을 유지하면서 MCDRAM에 의한 성능 향상을 함께 얻을 수 있을 것으로 확인된다.

4.2.3 프로세스 위치에 따른 통신 성능 분석

그림 6은 MCDRAM을 활용했을 때 타일 내 통신 성

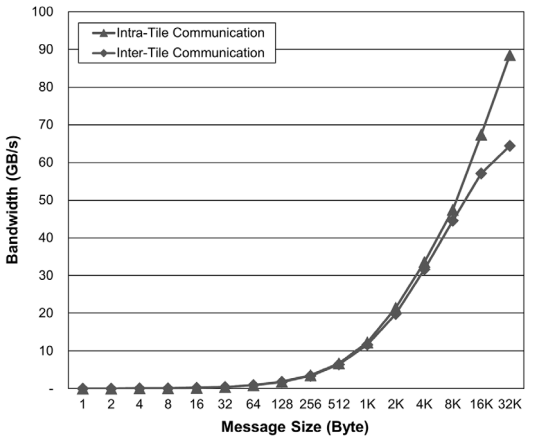

그림 6 타일 내 통신 성능과 타일 간 통신 성능 비교
Fig. 6 Comparison between the Performances of the Intra-Tile Communication and the Inter-Tile Communication

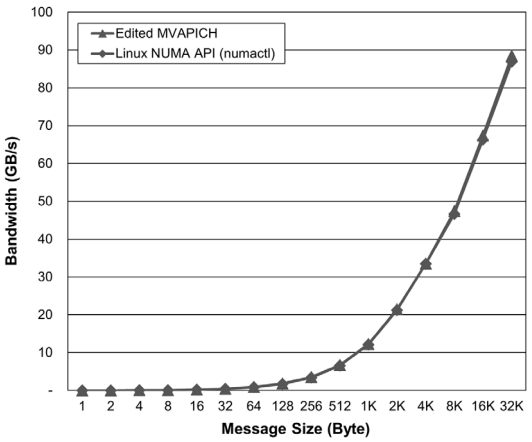

그림 7 MCDRAM 활용 방법에 따른 통신 성능
Fig. 7 Communication Performances of Different MCDRAM-Allocation Methods

능과 타일 간 통신 성능을 비교한 그래프다. 그림 6에서 볼 수 있듯이 메시지 크기가 32 KByte일 때 타일 내 통신은 타일 간 통신보다 37% 높은 대역폭을 보였다. 이것은 두 프로세스가 동일한 타일에서 수행될 경우, 노드 내 통신을 위해서 사용되는 MPI 수준의 공유 메모리가 해당 타일의 L2 캐시에 존재하게 되어, 수신 프로세스가 데이터 복사를 빠르게 할 수 있기 때문이다. 실험 결과를 통해 MCDRAM을 활용함과 동시에 송신 프로세스와 수신 프로세스의 위치를 함께 고려했을 때 가장 높은 성능향상을 성취할 수 있음을 확인할 수 있었다.

4.3 리눅스 NUMA API에 의한 MPI 노드 내 통신 성능 영향 분석

MCDRAM을 사용하기 위해서는 응용 또는 라이브러리의 소스 코드를 수정하거나 리눅스 NUMA API[22]가 제공하는 numactl 명령어를 활용할 수 있다. MCDRAM을 Flat 모드로 설정하고 활용할 경우 리눅스가 DDR4와 MCDRAM을 서로 다른 NUMA 노드로 인식하기 때문이다. 응용 또는 라이브러리의 소스 코드를 수정할 경우 성능 요소 분석을 거쳐 성능 영향이 높은 메모리 객체에 한해 MCDRAM에 적재되도록 수정하는 작업이 필수적이지만 제한적인 크기의 MCDRAM을 효율적으로 활용할 수 있다. 리눅스 NUMA API가 제공하는 numactl 명령어를 사용할 경우 소스 코드 수정 없이 기존의 응용을 손쉽게 사용할 수 있지만 응용 수행에 필요한 전체 메모리 이미지가 MCDRAM에 적재되기 때문에 MCDRAM의 낭비가 클 수 있다. 따라서 리눅스 NUMA API가 제공하는 numactl 명령어를 사용할 경우와 3장에서 기술한 방법을 이용해 MVAPICH의 소스 코드를 수정한 경우의 성능과 메모리 사용량을 분석하였다. numactl 명령어를 사용기 위해 소스 코드를 수정하지 않은 MVAPICH를 설치하고 numactl 명령어와 함께 실행하여 실험하였다. numactl 명령어가 제공하는 --membind 옵션은 응용이 실행되는 과정에서 사용할 NUMA 노드를 명시적으로 지정할 수 있다. numactl 명령어의 --hardware 옵션을 이용해 MCDRAM이 할당받은 NUMA 노드를 확인하고 응용 실행 단계에서 --membind 옵션의 인자 값으로 전달하여 실험하였다.

실험 방법은 4.2절의 실험 방법과 같이 동시 생성된 34개의 MPI point-to-point 연결에 대해서 통합 대역폭을 측정하였으며, 타일 내 통신 성능을 측정하였다. 성능 측정과 함께 numastat 명령어를 이용해 MCDRAM의 사용량을 수집하였다. 그림 7은 MVAPICH의 소스 코드를 수정한 경우와 소스 코드를 수정하지 않은 MVAPICH를 numactl 명령어와 함께 실행한 경우에 대한 실험 결과이며, 그림 8은 메모리 사용량을 보여준다.

그림 7에서 볼 수 있듯이 리눅스 NUMA API를 사용한 경우 역시 DDR4를 사용하는 경우와 비교했을 때 최대 264%의 성능향상을 볼 수 있다. MVAPICH를 수정한 경우와 유사한 성능향상 수준이다. 하지만 그림 8에서 볼 수 있듯이 리눅스 NUMA API를 사용한 경우와 MVAPICH를 수정한 경우의 메모리 사용량을 비교했을 때 메모리 사용량이 12% 증가한 것을 확인할 수 있다. 본 논문에서 실험에 사용한 응용 프로그램은 마이크로-벤치마크이기 때문에 통신 버퍼가 메모리 사용량의 대부분을 차지한다. 하지만 대규모 시뮬레이션을 수행하는 고성능 컴퓨팅 분야의 응용이 numactl 명령어를 사용해 MCDRAM을 활용할 경우 메모리 사용량의 격차는 더욱 증가할 것으로 예상된다. 두 가지 경우와 함

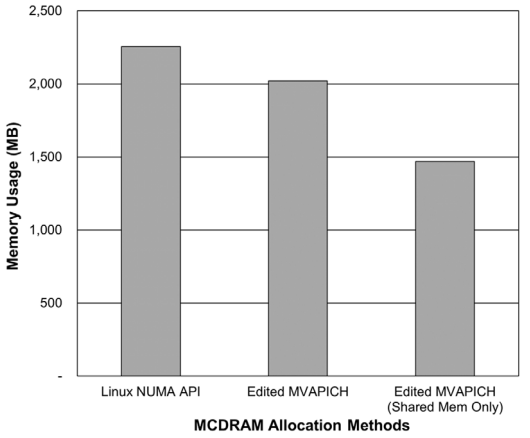

그림 8 MCDRAM 활용 방법에 따른 메모리 사용량
Fig. 8 Memory Usages of Different MCDRAM-Allocation Methods

께 공유 메모리만 MCDRAM을 사용하는 경우에 대해서도 메모리 사용량을 비교하였다. 공유 메모리만 MCDRAM을 사용하는 경우 사용자 수준의 통신 버퍼는 DDR4를 사용하기 때문에 가장 적은 메모리 사용량을 보이는 것을 확인할 수 있다. 하지만 4.2절의 실험 결과를 고려했을 때 사용자 수준의 통신 버퍼와 공유 메모리가 모두 MCDRAM을 사용하는 경우 가장 높은 성능향상을 성취할 수 있어 MCDRAM의 크기를 고려해 적합한 방법을 사용해야 한다. 실험 결과를 통해 리눅스 NUMA API를 사용할 경우 코드 수정 없이 응용 수준의 투명성을 유지하면서 MCDRAM의 성능 영향을 얻을 수 있지만 메모리 사용량 증가에 따른 불이익을 고려해야 한다는 것을 확인할 수 있었다.

5. 결론 및 향후 계획

본 논문에서는 KNL 프로세서의 MCDRAM을 활용하여 MPI 노드 내 통신 성능을 향상시킬 수 있음을 보였다. 성능 측정을 통해서 MCDRAM을 사용할 경우 DDR4를 사용하는 경우보다 노드 내 통신 성능을 최대 272% 향상시킬 수 있음을 보였으며, 하나의 MPI point-to-point 연결을 이루는 두 프로세스의 위치가 같은 타일에 위치했을 때 가장 높은 성능향상을 성취할 수 있음을 보였다. 또한 리눅스 NUMA API를 사용한 경우와 비교하여 제한적인 크기의 MCDRAM을 효율적으로 사용하기 위한 방법에 대해 논의하였다.

향후 계획으로 32KByte 이상의 메시지에 대해서 성능 분석을 수행할 예정이다. 큰 메시지에 대해서 MPI는 rendezvous 프로토콜을 사용하므로, 이와 관련된 버퍼를 MCDRAM에 할당하도록 해야 한다. 또한 point-to-point 통신뿐만 아니라 collective 통신에 대한 성능분석을 수행할 계획이다.

References

[1] A. Sodani, "Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor," *Presented at Hot Chips 2015*, Aug. 2015.
[2] Intel. (2016, June 22). Intel Xeon Phi Processor Product Brief [Online]. Available: http://www.intel.co.kr/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-processor-product-brief.pdf (downloaded 2016, Aug. 14)
[3] A. Heinecke , A. Breuer, M. Bader, and P. Dubey, "High Order Seismic Simulations on the Intel Xeon Phi Processor (Knights Landing)," *High Performance Computing*, Vol. 9697, pp. 343-362, Jun. 2016.
[4] Message Passing Interface, https://www.mpi-forum.org/.
[5] L. Chai, P. Lai, H.-W. Jin, and D. K. Panda, "Designing an efficient kernel-level and user-level hybrid approach for MPI intra-node communication on multi-core systems," *Proc. of International Conference on Parallel Processing*, pp. 222-229, Sep. 2008.
[6] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda, "Lightweight kernel-level primitives for high-performance MPI intra-node communication over multi-core systems," *Proc. of IEEE International Cluster Conference*, pp. 446-451, Sep. 2007.
[7] D. Buntinas, B. Goglin, D. Goodell, G. Mercier, and S. Moreaud, "Cache-efficient, intranode, large-message MPI communication with MPICH2-Nemesis," *Proc. of International Conference on Parallel Processing*, pp. 462-469, Sep. 2009.
[8] B. Goglin, M. Stephanie, "KNEM: a generic and scalable kernel-assisted intra-node MPI communication framework," *Journal of Parallel and Distributed Computing*, Vol. 73, No. 2, pp. 176-188, 2013.
[9] L. Chai, A. Hartono, and D. K. Panda, "Designing high performance and scalable MPI intra-node communication support for clusters," *Proc. of the IEEE International Conference on Cluster Computing*, pp. 1-10, 2006.
[10] X. Wu, V. Taylor, C. Lively, and S. Sharkawi, "Performance analysis and optimization of parallel scientific applications on CMP cluster systems," *Proc. of International Conference on Parallel Processing-Workshops*, pp. 188-195, 2008.
[11] C. Zhang, X. Yuan, and A. Srinivasan, "Processor affinity and MPI performance on SMP-CMP clusters," *Proc. of the IEEE International Symposium on Parallel & Distributed Processing*, Workshops and Phd Forum, pp. 1-8, 2010.
[12] M. Si, Y. Ishikawa, and M. Tatagi, "Direct MPI Library for Intel Xeon Phi Co-Processors," *Proc. of the IEEE 27th International Parallel and Distributed*

Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 816-824, 2013.
[13] S. Potluri, K. Hamidouche, D. Bureddy, and D. K. Panda, "MVAPICH2-MIC: A High Performance MPI Library for Xeon Phi Clusters with InfiniBand," *Proc. of the Extreme Scaling Workshop*, pp. 25-32, 2013.
[14] M. Noack, F. Wende, T. Steinke, and F. Cordes, "A Unified Programming Model for Intra- and Inter-Node Offloading on Xeon Phi Clusters," *Proc. of the SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 203-214, 2014.
[15] S. Neuwirth, D. Frey, and U. Bruening, "Communication Models for Distributed Intel Xeon Phi Coprocessors," *Proc. of the IEEE 21st International Conference onParallel and Distributed Systems*, pp. 499-506, 2015.
[16] S. Potluri, A. Venkatesh, D. Bureddy, K. Kandalla, and D. K. Panda, "Efficient Intra-node Communication on Intel-MIC Clusters," *Proc. of the 13th IEEE/ACM International Symposium on Cluster*, Cloud and Grid Computing, pp. 128-135, 2013.
[17] A. Shimada, A. Hori, and Y. Ishikawa, "Eliminating Costs for Crossing Process Boundary from MPI Intra-node Communication," *Proc. of the 21st European MPI Users' Group Meeting*, pp. 119-120, 2014.
[18] K. Kandalla, A. Venkatesh, K. Hamidouche, S. Potluri, D. Bureddy, and D. K. Panda, "Designing Optimized MPI Broadcast and Allreduce for Many Integrated Core (MIC) InfiniBand Clusters," *Proc. of the IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 63-70, 2013.
[19] A. Venkatesh, S. Potluri, R. Rajachandrasekar, M. Luo, K. Hamidouche, and D. K. Panda, "High Performance Alltoall and Allgather Designs for InfiniBand MIC Clusters," *Proc. of IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 637-646, 2014.
[20] MVAPICH, [Online]. Available: http://mvapich.cse.ohio-state.edu/
[21] memkind library, [Online]. Available: http://memkind.github.io/memkind/
[22] A. Kleen, "A numa api for linux," Novel Inc, 2005.

조 중 연

2012년 건국대학교 컴퓨터공학 학사. 2014년 건국대학교 컴퓨터공학 석사. 2014년~현재 건국대학교 컴퓨터공학 박사과정. 관심분야는 운영체제, 클라우드 컴퓨팅, 고성능 컴퓨팅

진 현 욱

1997년 고려대학교 전산학 학사. 1999년 고려대학교 전산학 석사. 2003년 고려대학교 통신시스템공학 박사. 2003년~2006년 미국 오하이오 주립대학교 연구원. 2006년~2008년 건국대학교 컴퓨터공학부 조교수. 2008년~2015년 건국대학교 컴퓨터공학부 부교수. 2015년~현재 건국대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 고성능 컴퓨팅, 임베디드 컴퓨팅

남 덕 윤

1999년 포항공과대학교 컴퓨터공학과 학사. 2001년 한국정보통신대학교 공학부 석사. 2006년 한국정보통신대학교 공학부 박사. 2004년~현재 한국과학기술정보연구원(KISTI) 슈퍼컴퓨팅본부 선임연구원. 2016년~현재 KISTI 슈퍼컴퓨터SW연구실 실장. 관심분야는 분산시스템, 슈퍼컴퓨팅, 저전력컴퓨팅