MPI 집합통신 성능 향상 연구 동향

Research Trends for Improving MPI Collective Communication Performance

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

ABSTRACT

Message Passing Interface (MPI) collective communication has been applied to various science and engineering area such as physics, chemistry, biology, and astronomy. The parallel computing performance of the data-intensive workload in the above research fields depends on the collective communication performance. To overcome this limitation, MPI collective communication technology has been extensively researched over the last several decades to improve communication performance. In this paper, we provide a comprehensive survey of the state-of-the-art research performed on the MPI collective communication and examine the trends of recently developed technologies. We also discuss future research directions for providing high performance and scalability to large-scale MPI applications.

KEYWORDS MPI, 고성능컴퓨팅, 병렬프로그래밍, 슈퍼컴퓨팅, 집합통신

Ⅰ. 서론

일반적으로 고성능 컴퓨팅(High Performance Computing)이란 여러 컴퓨팅 노드들을 활용하여 크고 복잡한 계산을 고속으로 병렬처리하는 기술을 말한다[1].

슈퍼컴퓨터 및 대규모 컴퓨터 클러스터 환경에서의 고성능 컴퓨팅 기술은 과학 및 공학 분야의 다양한 문제를 해결하고 있다[2,3]. 고성능 컴퓨팅 기술의 대표적인 활용 예로는 인간의 단백질 구조를 유전체 수준까지 모델링하는 단백질 구조 분석[4], 분자 역학 시뮬레이션을 통해 수십억이 넘는 생체 분자들의 상호작용 예측을 통한 신약 개발[5], 지진, 태풍, 폭염 등 기상 예측[6], 위성영상 처리 및 분석[7], 핵융합 시뮬레이션[8] 등이 있다.

해당 분야에서는 멀티코어, 멀티프로세서 또는 다수의 컴퓨터로 구성된 클러스터 환경에서 병렬

* DOI: https://doi.org/10.22648/ETRI.2022.J.370605
* 이 논문은 대한민국 정부(과학기술정보통신부)의 재원으로 한국연구재단 슈퍼컴퓨터개발선도사업의 지원을 받아 수행된 연구임[과제번호: 2021M3H6A1017683].

컴퓨팅(Parallel Computing) 기술을 활용하여 크고 복잡한 문제를 작게 나눈 후 동시에 한꺼번에 처리한다. 병렬 컴퓨팅 기술을 활용하는 응용 프로그램은 OpenMP, MPI와 같은 병렬 프로그래밍 라이브러리들을 활용하여 구현할 수 있다.

MPI는 병렬 컴퓨팅을 수행하기 위해 프로세스 간 메시지를 전달하는 기법을 정의한 라이브러리의 표준이다. 점 대 점(Point-to-Point) 통신과 집합(Collective) 통신으로 구성되며, 프로세서와 네트워크 기술이 발전함에 따라 해당 기술들에 최적화된 형태로 꾸준히 개발되어왔다.

MPI 집합통신은 다수의 프로세스가 동시에 참여하는 프로세스 간 통신 방법으로 대규모 행렬의 곱셈, 선형대수 연산, 반복 알고리즘의 종료 조건 계산 등 다양한 목적으로 활용된다. 대표적인 예로, 최근에 활발히 연구되고 있는 인공지능 분야에서 대규모 딥러닝 모델을 분산 학습할 때 반복적으로 수행되는 파라미터 업데이트 작업을 들 수 있다.

그러나 집합통신은 통신에 참여하는 프로세스의 개수와 전송할 데이터의 양이 증가할수록 프로세스 간 통신 횟수가 많아지며 네트워크 트래픽(Network Traffic)이 발생하는 고비용 연산이다. 이로 인해 고성능 컴퓨팅 분야의 대규모 데이터 분석이나 기계학습과 같이 데이터 집약적인 워크로드의 처리 성능은 집합통신 성능에 의해 좌우된다. 이러한 이유로 집합통신 성능 향상을 위한 다양한 시도가 이루어지고 있으며, 중점적으로 연구되어야 하는 주제 중 하나로 그 중요성이 날로 커지고 있다.

본고에서는 집합통신 성능 향상을 위한 대표적인 연구 사례와 최신 연구들을 중심으로 기술 동향을 분석하였다. 본고의 구성은 다음과 같다. Ⅱ장에서는 메모리 구조에 따른 프로세스 간 통신 기법을 살펴보고, Ⅲ장에서는 MPI와 집합통신 기술을 설명한다. Ⅳ장에서는 집합통신 성능 향상 연구들을 소개한다. Ⅴ장에서는 향후 연구 방향을 점검하며 결론을 맺는다.

Ⅱ. 메모리 구조에 따른 프로세스 IPC

이 장에서는 MPI의 근간이 되는 기술인 프로세스 간 통신을 소개한다. 프로세스 간 통신 기술은 프로세스와 메모리의 연결 구조에 따라 세 가지로 나뉜다. 다음 절에서부터 공유 메모리, 분산 메모리, 분산 공유 메모리 구조에서의 프로세스 간 통신 기술을 설명한다.

1. 공유 메모리 구조에서의 IPC

프로세스들이 통신하는 방법은 프로세스와 메모리의 연결 구조에 따라 달라진다. 그림 1은 공유 메모리 구조를 나타낸다[9]. 하나의 메모리 풀을 공유하는 프로세스들이 물리적인 버스로 연결되어 있으며, 적은 수의 프로세스가 협업 시에 빠른 성능을 제공하는 장점이 있다. 다수의 프로세스가 하나의 메모리를 공유하기 때문에 공유 메모리의 동기화(Critical Section) 문제가 발생하지 않도록 뮤텍스(Mutex)와 세마포어(Semaphore)를 사용해야 한다. 따라서 프로세스 개수가 늘어남에 따라 공유 메모리 버스트래픽 병목 현상으로 성능이 저하된다. 또한, 프로세스

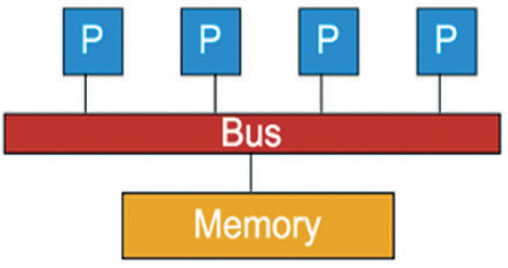

그림 1 공유 메모리 구조

들의 캐시일관성(Cache Coherence) 유지를 위한 스눕(Snoop) 오버헤드가 증가하는 단점이 있다.

공유 메모리 구조에서 프로세스들이 통신하는 방법은 프로세스들이 공유 메모리 영역을 만들고 해당 영역을 함께 읽고 쓰는 방식으로 이루어지며, 다음과 같이 세 단계로 통신을 수행한다. 1) OS 커널이 공유 메모리 영역을 생성한다. 2) 프로세스 A는 공유 메모리에 데이터를 쓴다. 3) 프로세스 B는 공유 메모리의 데이터를 읽는다.

공유 메모리 프로세스 간 통신 기능을 제공하는 대표 라이브러리는 Pthreads와 OpenMP가 있다.

2. 분산 메모리 구조에서의 IPC

그림 2는 분산 메모리 구조를 나타낸다[10]. 각 프로세스는 자신의 로컬 메모리를 소유하고 있고, 프로세스들이 네트워크로 연결되어 있다. 이 구조는 메모리 공유로 인한 동기화 문제와 캐시일관성 유지를 위한 오버헤드가 없으며 프로세서의 개수가 늘어나도 성능이 저하되지 않는 장점이 있다.

그러나 프로그램을 잘못 작성하는 경우, 교착상태(Deadlock)에 빠질 수 있고, OS 커널의 메시지 큐(Message Queue)에 프로세서들이 전달한 메시지가 쌓이는 오버헤드가 발생할 수 있다.

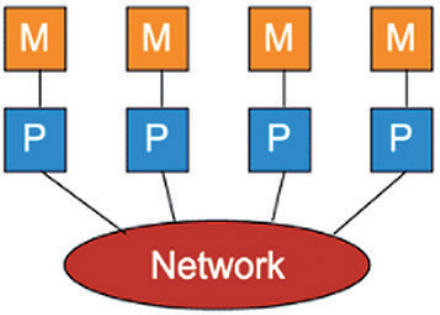

그림 2 분산 메모리 구조

분산 메모리 구조에서 프로세스들이 통신하는 방법은 다음과 같이 두 단계로 이루어진다. 1) 프로세스 A가 메시지 큐로 데이터를 전송한다. 2) OS 커널이 프로세스 B에게 메시지를 가져가도록 정보를 알려주면 프로세스 B가 직접 메시지 큐에서 데이터를 가져간다. 또는, OS가 프로세스 B에게 메시지를 직접 전송한다.

분산 메모리 프로세스 간 통신 기능을 제공하는 대표적인 라이브러리는 MPI가 있다.

3. 분산 공유 메모리 구조에서의 IPC

그림 3은 분산 공유 메모리 구조를 나타낸다[11]. 각 프로세서는 자신의 로컬 메모리와 메모리 관리자를 가진다. 메모리 관리자는 분산되어있는 물리적인 메모리들을 논리적으로 생성된 하나의 큰 공유 메모리 형태로 만들어 프로세서들에게 제공하는 기능을 수행한다. 분산 공유 메모리 구조에서 프로세서들은 Invocation/Response 기법을 사용하여 논리적인 공유 메모리를 활용하는 방식으로 통신한다.

현재 분산공유 메모리 프로세스 간 통신 기능을

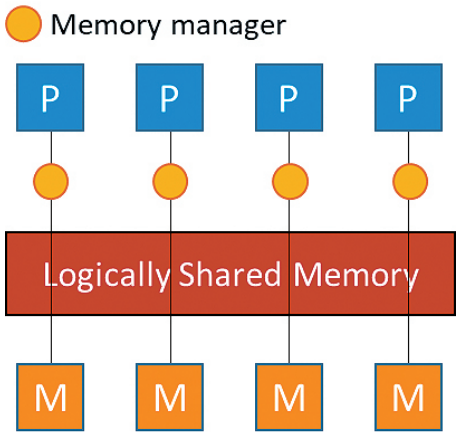

그림 3 분산공유 메모리 구조

제공하는 대표적인 라이브러리는 없으며, 필요에 따라 직접 구현해서 사용하는 것이 권장된다[12].

Ⅲ. MPI

1. MPI 개요

대표적인 오픈소스 라이브러리는 MPICH[13]와 Open MPI(Message Passing Interface)[14]가 있다. 슈퍼컴퓨터에서는 위 라이브러리들을 기반으로 하드웨어 구조와 소프트웨어의 특성에 최적화된 형태의 MPI 라이브러리를 제작하여 활용한다.

예를 들어, MVAPICH[15]는 고속 네트워크 인터페이스 기술인 인피니밴드에 최적화하여 만들어진 MPICH의 변형 라이브러리이다. 이외에도 IBM의 IBM Spectrum MPI[16], 마이크로소프트의 MS-MPI[17], 인텔의 Intel MPI[18]와 같이 슈퍼컴퓨터의 제조사들은 각자의 슈퍼컴퓨터 구조에 최적화된 형태의 MPI 라이브러리를 지속적으로 개발하고 있다.

2. MPI 집합통신

MPI 집합통신은 다수의 프로세스가 동시에 참여하는 통신이다. MPI 표준 1에서는 두 개의 프로세스 간의 점 대 점 통신만 가능했다. 그러나 점 대 점 통신만 사용하면 프로세스들의 개수가 많아질수록 메시지 전송 비용이 커지는 비효율이 발생한다. 집합통신은 다수의 프로세스가 적은 비용으로 빠르게 통신하기 위한 다양한 기능으로 구성되며 MPI-2부터 지원되기 시작하였다.

표준 집합통신은 크게 All-To-All, All-To-One, One-To-All과 같이 3가지 카테고리로 분류할 수 있다. All-To-All은 모든 프로세스가 결과에 기여하고, 모든 프로세스가 결과를 전송받는 통신 방식으로 all-gather, all-reduce, all-to-all, reduce-scatter, barrier가 있다. All-To-One은 모든 프로세스가 결과에 기여하고, 한 프로세스만 결과를 전송받는 통신 방식으로 gather와 reduce가 있다. One-To-All은 한 프로세스가 결과에 기여하고, 모든 프로세스가 결과를 전송받는 통신 방식으로 broadcast와 scatter가 있다.

Ⅳ. MPI 집합통신 성능 향상 연구

이 장에서는 MPI 집합통신 성능 향상을 위한 대표적인 기술들을 소프트웨어적 연구와 하드웨어적 연구로 나누어 살펴본다.

1. 소프트웨어적 성능 향상 연구

가. Network Topology 활용 기법

하나의 프로세스가 다수의 프로세스에게 메시지를 전송하는 가장 기본적인 구조는 마스터/슬레이브 방식이다. 이 방식에서는 마스터 프로세스가 슬레이브 프로세스들에게 순차적으로 메시지를 전송한다. 따라서 슬레이브 프로세스들의 개수가 많아질수록 마스터 프로세스에 병목이 발생하여 성능이 저하된다. 이 문제를 해결하기 위해 프로세스들이 병렬로 메시지를 전달하도록 다양한 형태의 통신 네트워크 토폴로지가 연구되었다.

참고문헌[19]는 바이노미얼 트리를 통신 네트워크 토폴로지로 사용하고 프로세스 간에 메시지를 패킷단위로 병렬 전송하여 집합통신 성능을 향상시킨 연구다. 기존에 완전이진 트리를 사용하는 경우 마스터/슬레이브 방식의 순차 전송 문제는 해결되지만, 자식 프로세스의 개수가 두 개로 제한되기 때문에 프로세스들의 개수가 많아질수록 성능이 저하되었다. 바이노미얼 트리는 자식 노드의 개수에 제

한이 없고 확장성 있게 트리가 커지는 특징을 가지며, 프로세스 개수가 증가해도 좋은 통신 성능이 유지되는 장점이 있다.

위 연구에서는 제안한 방법으로 프로세스 개수를 15개까지 늘려가며 실험을 수행한 결과가 마스터/슬레이브 방식에 비해 약 2배, 완전이진 트리에 비해 약 1.23배 성능이 향상됨을 보였다.

참고문헌[20]은 전역 집계연산 결과를 효율적으로 구하기 위해 butterfly 구조를 제안한 연구다. MPI all-reduce는 모든 프로세스의 메시지를 루트(Root) 프로세스에 모아 전역(Global) 집계 결과를 구하는 reduce 단계와 집계 결과를 루트 프로세스로부터 모든 자식 프로세스에게 전달하는 broadcast 단계로 구성된다. 이를 활용하면 모든 프로세스의 전역 합(Global Sum)을 계산하고 그 결과를 모든 프로세스에게 전송하는 과정을 한 번에 수행할 수 있다.

단순 트리 구조를 사용하는 경우에 reduce 단계에서는 두 프로세스의 값을 더하는 과정을 단계별로 진행하여 전역 합을 구한다. 이후, broadcast 단계에서 reduce 결과를 트리 구조에 따라 모든 프로세스에게 전송한다.

위 연구에서 제안한 butterfly 구조에서는 프로세스들이 데이터를 모아서 합산하는 대신, 데이터를 상호 교환한 후 합산하여 성능을 향상시킨다. 제안한 방법은 단순 트리 구조를 사용할 때보다 전송 횟수가 늘어나는 대신 전송 시간은 절반으로 감소하는 효과가 있다.

이 방법은 메시지 전송에 매우 적은 시간이 소요되기 때문에 전송 횟수의 영향이 무시할만한 수준인 노드 내의 프로세스들이 통신하는 환경에서 특히 좋은 성능을 제공하며, MPICH-1.x부터 포함되어 현재까지 널리 활용되고 있다. 이외에도 집합통신을 위한 효율적인 통신 네트워크 토폴로지를 제안한 연구[21,22]가 있다.

효율적인 네트워크 토폴로지를 제안한 연구 분야와 달리 최적의 통신 네트워크 토폴로지를 선택하는 방법을 제안한 연구들도 있다[23-25]. 이 연구들은 집합통신이 수행될 컴퓨팅 노드의 개수 (n), 노드당 프로세스의 개수 (ppn), 메시지의 크기 (m)에 따라 휴리스틱을 활용하여 최적의 메시지 전송 네트워크 토폴로지를 선택하는 방법을 제안했다.

최근에는 MPI 집합통신 워크로드들의 벤치마크 수행 결과를 기계학습 모델로 학습하고 해당 모델을 활용하여 최적의 메시지 전송 알고리즘을 선택하는 연구가 이루어지고 있다[26].

위 연구에서 제안한 기계학습 기반 집합통신 알고리즘 선택 기법의 모델 학습 및 추론 과정은 다음과 같다. 예를 들어, 노드 개수 n이 16, 노드당 프로세스의 개수 ppn이 18, 메시지 크기 m이 128바이트로 주어졌을 때 all-gather를 알고리즘 아이디(ID) 1로 수행하여 84마이크로초(μs)가 소요되는 벤치마크 결과를 기계학습 기법으로 학습해 회귀(Regression) 모델을 생성해 놓는다. 추후에 실행할 MPI collective.c 코드와 함께 n, ppn, m으로 구성된 질의를 해당 모델에게 요청하여 MPI collective.c가 최적으로 수행되는 알고리즘의 아이디를 추론 결과로 받는다.

나. Shared Memory 활용 기법

기존의 집합통신은 크게 send/recv, shared memory, kernel-assisted 기법을 활용하여 구현되어 왔고 각각 다음과 같은 한계를 가진다.

Send/recv 기법은 집합통신이 만들어진 초창기에 공유 메모리를 사용하지 않고 점 대 점 통신만을 사용해서 집합통신하는 방식이다. 이 방식은 다수 프로세스 간에 작은 사이즈의 메시지를 전달할 때 메시지 처리 오버헤드로 인해 성능 저하가 발생하는 문제가 있다.

Shared memory 기법은 메모리의 특정 영역을 파일

로 대응시키는 OS의 mmap 기능을 사용한다. P1이 공유 메모리 영역에 데이터를 복사해 놓으면 P2가 해당 데이터를 복사해간다. 이 과정에서 데이터 복사가 두 번 발생하기 때문에 메시지 크기가 클수록 데이터 복사량이 증가하여 성능 저하가 발생한다.

Kernel-assisted 기법은 kernel address space를 활용한다. P1이 자신의 메모리 주소를 kernel address space에 매핑해 놓으면, P2가 해당 주소에서 데이터를 복사해간다. 이 과정에서 데이터 복사는 한 번 발생하므로 shared memory 방법보다 효율적이다. 그러나 프로세스들이 데이터를 읽어갈 때 kernel page-table에서 경쟁(Contention)이 발생하고, 이로 인한 page-table locks 오버헤드 때문에 프로세스의 개수가 증가할수록 성능이 저하되는 문제가 발생한다.

참고문헌[27]에서는 위의 한계를 극복하고자 리눅스 커널에 추가된 XPMEM[28]을 활용하여 많은 수의 프로세스들이 통신할 때 지연이 발생하지 않도록 하는 Shared-Address Space 집합통신을 제안하였다. XPMEM은 공유 메모리 영역에서 프로세스들이 경쟁 없이 메시지를 읽어갈 수 있게 해주는 shared memory 기술로 리눅스 커널 3.x 버전부터 지원되고 있다[29].

공유 메모리를 사용해서 집합통신 성능을 향상시키는 또 다른 연구로는 참고문헌[30]이 있다. 이 연구는 Shared Memory 기반 PMI(Process Management Interface)를 제안하여 집합통신 시간을 단축시키고 집합통신을 위해 필요한 메모리 사용량을 크게 줄였다. PMI[31]란 MPI 라이브러리와 프로세스관리자 간의 인터페이스를 말한다. 그림 4는 PMI의 구조를 나타낸다. PMI는 PMI API proper와 wire protocol로 구성된다. PMI API proper는 MPI 라이브러리에게 노출되는 클라이언트용 API이고, wire protocol은 프로세스 관리자와 프로세스들 사이의 통신 프로토콜이다. MPI 라이브러리는 PMI API proper를

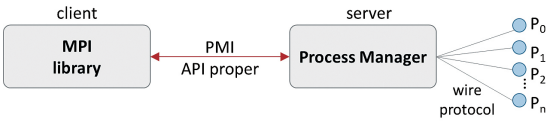

그림 4 PMI 구조

활용하여 프로세스 관리자에게 작업을 지시하고, 프로세스 관리자는 wire protocol을 통해 프로세스들에게 작업을 지시한다.

위 연구에서는 위의 단점을 해결하기 위해 기존의 PMI의 한계로 다음과 같은 단점을 지적했다. 첫째, MPI 라이브러리와 프로세스 관리자가 클라이언트-서버 방식으로 통신하여 많은 개수의 동시 클라이언트가 PMI를 사용할 때 서버 측의 직렬화(Serialization) 문제로 인해 성능이 저하되었다. 왜냐하면, 이 구조에서 서버는 한 개 또는 수 개의 클라이언트 요청만 병렬로 처리할 수 있기 때문이다. 둘째, 삭제(Delete) 기능이 없어서 메모리 사용량이 매우 컸다. PMI는 MPI 작업(Job)과 프로세스들의 아이디를 key-value 형태로 관리하는데 삭제 기능의 부재로 MPI 작업이 종료될 때까지 전체 key-value 쌍을 모두 저장하고 있어야 해서 프로세스 개수가 증가함에 따라 메모리 사용량이 크게 증가했다.

또한, PMI의 통신 방식을 클라이언트-서버 기반에서 shared memory 기반으로 변경하였다. 이에 따라 MPI 라이브러리들이 shared memory 영역에 저장되는 key-value 쌍을 병렬로 찾을 수 있게 되어 프로세스 개수가 증가해도 집합통신 성능이 저하되지 않도록 개선되었다. 또한, PMI에 삭제 기능을 추가하여 key-value 쌍의 저장에 필요한 메모리 사용량을 줄였다. 기존의 PMI에서는 1백만 개의 프로세스가 동작할 때 약 1GB의 메모리 용량이 사용되었지만, shared memory 기반 PMI를 사용하는 경우 메모리 사용량이 기존의 약 4%인 40MB 정도로 줄었다.

2. 하드웨어적 성능 향상 연구

가. FGPA를 활용한 인-네트워크 컴퓨팅 기법

소프트웨어와 함께 하드웨어 레벨에서 집합통신을 최적화하면 더 높은 성능을 낼 수 있다. 참고문헌[32]는 MPI reduce 연산을 네트워크 스위치에 오프로딩시켜 집합통신 성능 및 확장성을 크게 향상시켰다. 구체적으로는 네트워크 레벨에서 reduce 연산을 수행하는 SHARP(Scalable Hierarchical Aggregation Protocol) 트리를 제안하고 해당 기술을 SwitchIB-2 ASIC에 탑재하여 Mellanox의 네트워크 스위치 제품으로 출시였다[33].

위 연구에서 제안한 SHARP 트리는 네트워크 스위치에서 비용이 큰 집계 연산을 수행하는 방식으로 노드 간 데이터 전송량을 감소시키며, 이를 통해 규모가 큰 시스템에도 높은 성능과 확장성을 제공할 수 있다.

MPI-FPGA[34]은 FPGA를 활용하여 MPI reduce 모듈을 설계하고 해당 모듈을 라우터에 탑재하였고, 그 결과 긴 메시지 전송의 성능은 기존 대비 2.5배 향상, 짧은 메시지 전송 성능은 기존 대비 10배 향상하였다. 위 연구의 저자들은 후속 연구인[35]에서 reduce, all-reduce, broadcast, scatter, gather, all-gather를 위한 여섯 개의 FPGA 모듈을 추가로 설계하고 스위치 주변에 배치하였다. 라우터로부터 패킷이 도착하면 Collective Control Logic(CCL)은 패킷헤더를 확인하여 집합통신 필요 여부를 판단하고, 집합통신용 패킷 중 reduction이 필요한 경우에 연산 종류에 맞게 패킷헤더에 명령어(Opcode)를 추가하여 스위치로 전달한다. Reduction Unit(RU)은 스위치로부터 패킷을 전달받아 부동소수점의 덧셈, 곱셈, 최댓값/최솟값을 하드웨어 로직 기반으로 연산한다. 이 연구에서는 제안한 기법이 기존 MPI 라이브러리와 함께 최적으로 동작하도록 MVAPICH 라이브러리에 새로운 하드웨어 모듈을 위한 소프트웨어 모듈을 추가했다.

위 연구에서 OSU 벤치마크[36]를 노드 개수를 증가시키며 수행하여 집합통신 기능들의 성능을 측정한 결과 여섯 개의 집합통신 기능들을 기존 기술 대비 평균 3.9배 향상시켰다.

128개 노드를 사용했을 때 all-reduce의 속도가 10.1배 향상되어 가장 큰 성능 향상을 보였다. 64개 노드를 사용했을 때 bcast의 속도가 6.69배 향상되어 두 번째로 큰 성능 향상을 보였다. Reduce와 gather는 평균 약 1.5배의 속도 향상이 이루어져 가장 적은 성능 향상을 보였다.

이외에도 네트워크 인터페이스 카드(NIC)와 네트워크 스위치에 집합통신을 오프로딩하여 집합통신용 전용 네트워크 장비를 개발한 연구들도 있다[37-39].

나. In-Memory Processing 활용 기법

프로세서와 메모리 간의 데이터 전송속도 차이에 따른 병목은 컴퓨터 시스템의 성능과 전력 효율 향상에 한계로 작용하며, 폰노이만 병목현상(Von-Neumann Bottleneck)으로 알려져 있다. 이는 자율주행 자동차나 엣지 컴퓨팅과 같이 클라우드 기반 어플리케이션이 등장함에 따라 더 큰 문제로 인식되며, 최근에는 이를 해결하기 위한 방법들이 더욱 활발히 연구되고 있다.

대표적으로 메모리 계층 구조에서 데이터 전송량을 줄이기 위한 방법으로 PIM, NDP, ISP를 주제로 많은 연구가 수행되고 있다. Compute Cache[40]는 캐시(Cache) 메모리에서 비트단위 연산을 수행하여 CPU와 메모리 사이의 데이터 전송량을 줄이는 연구다.

참고문헌[41]은 최근 MRAM 기반의 인-메모리 컴퓨팅을 지원하는 저전력 칩을 개발한 연구다. 이

는 대량의 정보를 CPU와 메모리 간에 이동 없이 메모리 내에서 병렬 연산하여 차세대 인공지능과 뉴로모픽 칩을 개발하는 데 초석이 될 것으로 기대되고 있다. 이외에도 DDR 메모리의 용량과 대역폭 한계를 극복하기 위하여 TSV, Die-Stacked Memory, HBM을 키워드로 다양한 연구들이 수행되고 있다.

Active-Routing[42]은 NVIDIA, Intel, Texas A&M 대학 연구진들이 PIM이 지원되는 메모리를 라우터에 활용하여 집합통신 성능을 향상시킨 연구다. 이 연구에서는 집합통신 수행 시 데이터 로컬리티를 기반으로, 데이터와 가까운 라우터의 메모리에서 직접 집계 연산을 수행하고 라우팅 경로를 따라 집계하며 결과를 도출한다. 그 결과, 집합통신 성능을 기존 기술 대비 평균 60% 향상시켰고, 전력 사용량을 80% 절감했다.

다. In-Storage Processing 활용 기법

참고문헌[43]은 저장 장치에서 데이터 처리를 함께 제공하는 Catalina SSD 보드를 제안한 연구다. 위 연구에서는 최대 24TB의 용량을 지원하는 Catalina SSD 보드를 제작하고, 해당 SSD 16장으로 384TB 스토리지 서버를 구성하고 이를 호스트 서버와 PCIe로 연결해 활용하였다.

각각의 Catalina SSD 보드는 MPI 기반 유사 이미지 검색을 수행할 때 MPI 슬레이브로 동작하며 자신의 낸드플래시 메모리에 저장되어 있는 이미지들 중 질의 이미지와 유사한 이미지들을 찾아내어 로컬 결과를 도출하고 지역(Local) 집계 결과를 호스트 서버의 MPI 마스터에게 전송한다. 이미지 유사도 계산은 유클리디언 거리로 측정한다. 각 이미지는 128차원의 고차원 부동소수점으로 표현되며, 유클리디언 거리 계산 시 고차원 벡터의 내적이 수행된다. 이때 가속기를 활용해 고차원 벡터의 내적을 빠르게 병렬 처리한다. 최종적으로 MPI 마스터는 로컬 결과들을 집계하여 가장 유사한 이미지들로 전역 결과를 도출한다.

이 연구에서는 성능 검증을 위해 페이스북의 유사 이미지 검색 라이브러리인 Faiss[44]와 대규모 이미지 데이터셋인 ANN_SHIFT[45]를 이용하였다. 1백만 장의 이미지 데이터셋과 1만 장의 질의 이미지를 사용하여 각 질의 이미지당 100장씩의 유사 이미지 검색을 수행한 결과 16개의 Catalina SSD 보드를 사용하면, 사용하지 않을 때에 비해 처리속도가 약 11배 향상되고, 질의당 전력 사용량이 약 7배 감소했다.

이 연구에서 제안한 방법은 다음과 같은 특징으로 해당 연구 이전의 ISP 저장 장치인 BlueDBM[46], Biscuit[47], Scan&Join[48]에 비해 효율적이고 유연한 장점을 제공한다.

첫째, 스토리지 전용 프로세서인 쿼드코어 ARM A53 프로세서와 FPGA 가속기를 가진다[49]. 이를 통해 스토리지 내에서 데이터를 저전력으로 빠르게 병렬처리한다. 둘째, C++, Python과 같은 고수준 언어를 지원하므로 기존의 RTL 코드나 장치에 특화된 API를 활용한 방식에 비해 프로그래밍이 쉽다. 셋째, 스토리지의 전용 프로세서에서 OS가 동작하므로 호스트(Host)의 도움 없이 스토리지 레벨에서 분산 파일시스템이 지원된다. 넷째, MPI와 Hadoop MapReduce[50]와 같은 분산처리 라이브러리들을 지원한다.

Ⅴ. 결론

MPI 집합통신 성능 향상은 소프트웨어와 하드웨어 분야에서 모두 활발히 연구되고 있었다. 공유 메모리 기술을 활용하는 분야에서는 OS 커널의 공유 메모리 관리 기술이 발전함에 따라 이를 활용하여

노드 내 프로세스들의 통신 성능을 향상시키는 방향으로 연구가 진행되었다.

네트워크 토폴로지를 활용하는 분야에서는 효율적인 네트워크 토폴로지 구조에 대한 연구와 최적의 네트워크 토폴로지를 선택하는 연구로 MPI 집합통신 기술의 등장 시점부터 최근까지 꾸준히 연구되었다.

FPGA를 활용하여 인-네트워크 컴퓨팅을 실현하는 분야에서는 집합통신 연산을 네트워크 장비로 오프로딩하는 연구들이 다수였다.

PIM과 ISP 기술을 활용하여 메모리 및 스토리지 내에서 집합통신을 처리하는 기술들은 비교적 최근에 활발한 연구가 진행되고 있다.

고성능 컴퓨팅 기술의 발전과 함께 인공지능 전용 프로세서, 차세대 메모리 반도체, 새로운 소프트웨어 기술이 등장함에 따라 이들의 특성을 최적으로 활용하는 집합통신 기술에 대한 연구는 앞으로도 계속될 것으로 전망된다.

ETRI는 슈퍼컴퓨터개발선도사업을 통해 충분한 메모리와 함께 대규모 데이터를 병렬처리하는 가속기 기능이 함께 제공되는 메모리 확장 장치를 개발 중이다[51]. 이와 더불어 ETRI 슈퍼컴퓨팅기술연구센터에서는 위 기술을 사용하여 메모리 레벨에서 집합통신을 수행하여 성능을 향상시키는 연구를 수행 중이다. 향후 이 기술은 병렬 처리가 필요한 대규모 응용프로그램들에게 높은 성능과 확장성을 제공할 것으로 기대된다.

용어해설

MPI 집합통신(MPI Collective Communication) 다수의 프로세스가 동시에 통신에 참여하여 서로의 메시지를 주고받는 프로세스 간 통신 기법

약어 정리

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CPU	Central Processing Unit
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
FPGA	Field Programmable Gate Array
HBM	High-Bandwidth Memory
IPC	Inter Process Communication
ISP	In-Storage Processing
MPI	Message Passing Interface
MPSoC	Multiprocessor System on a Chip
MRAM	Magnetoresistive Random Access Memory
NDP	Near Data Processing
NIC	Network Interface Card
OpenMP	Open Multi-Processing
OS	Operating System
PIM	Processing In Memory
PMI	Process Management Interface
PThreads	POSIX Threads
RTL	Register Transfer Level
SSD	Solid-State Drive
TSV	Through Silicon Via

참고문헌

[1] D.K. Panda et al., "The MVAPICH project: Transforming research into high-performance MPI library for HPC community," J. Comput. Sci., vol. 52, 2021, article no. 101208.

[2] K.S. Jin, S.M. Lee, and Y.C. Kim, "Adaptive and optimized agent placement scheme for parallel agent-based simulation," ETRI J., vol. 44, no. 2, 2021.

[3] B. Andjelković et al., "Grid-enabled parallel simulation based on parallel equation formulation," ETRI J., vol. 32, no. 4, 2010, pp. 555-565.

[4] M. GAO et al., "Proteome-scale deployment of protein structure prediction workflows on the summit supercomputer," arXiv preprint, CoRR, 2022, arXiv: 2201.10024.

[5] A. Acharya et al., "Supercomputer-based ensemble docking drug discovery pipeline with application to COVID-19," J. Chem. Inf. Model., vol. 60, no. 12, 2020, pp. 5832-5852.

[6] M. Tolstykh et al., "SL-AV model: Numerical weather prediction at extra-massively parallel supercomputer," in Russian Supercomputing Days, Springer, Cham, Switzerland, 2018, pp. 379-387.

[7] V. Khryashchev et al., "Comparison of different convolutional neural network architectures for satellite image segmentation," in Proc. Conf. Open Innov. Assoc. (FRUCT), (Bologna, Italy), Nov. 2018, pp. 172-179.

[8] M.P. KATZ et al., "Preparing nuclear astrophysics for exascale," in Proc. SC20: Int. Conf. High Perform. Comput., Netw., Storage Analysis (Atlanta, GA, USA), Nov. 2020, pp. 1-12.

[9] Wikipedia, Shared Memory, https://en.wikipedia.org/wiki/Shared_memory

[10] Wikipedia, Distributed Memory, https://en.wikipedia.org/wiki/Distributed_memory

[11] Wikipedia, Distributed Shared Memory, https://en.wikipedia.org/wiki/Distributed_shared_memory

[12] Z. Huang et al., "VODCA: View-oriented, distributed, cluster-based approach to parallel computing," in Proc. IEEE Int. Symp. Cluster Comput. the Grid (CCGRID'06), (Singapore, Singapore), May 2006.

[13] Argonne, MPICH: A High-Performance, Portable Implementation of MPI, https://www.anl.gov/mcs/mpich-a-highperformance-portable-implementation-of-mpi

[14] The Open MPI Project, Open MPI: Open Source High Performance Computing, https://www.open-mpi.org/

[15] The Ohio State University, MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE, http://mvapich.cse.ohio-state.edu/

[16] IBM, IBM Spectrum MPI, https://www.ibm.com/products/spectrum-mpi?utm_content=SRCWW&p1=Search&p4=43700067987454012&p5=p&gclid=Cj0KCQiAr5iQBhCsARIsAPcwROMfviu3UCQI4w4tdjcY6gF9AzywHVCsqODz2ZBdV-RxaIcASCobBfMaAhPMEALw_wcB&gclsrc=aw.ds

[17] Microsoft, Microsoft MPI, https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi

[18] Intel, Intel MPI Library, https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html#gs.pcbhj9

[19] S.H. Cho and Y.H. Kim, "A fast transmission of mobile agents using binomial trees," The KIPS Trans.: Part A, vol. 9A, no. 3, 2002, pp. 341-350.

[20] H. Zhao and J. Canny, "Butterfly mixing: Accelerating incremental-update algorithms on clusters," in Proceedings of the 2013 SIAM International Conference on Data Mining, SIAM, Philadelphia, PA, USA, 2013, pp. 785-793.

[21] 이정희, 한동수, "패킷단위 병렬데이터 전송을 통한 MPICH-G2 집합통신 성능 향상," 한국정보과학회, 가을 학술발표논문집, 제30권 제2호, 2003.

[22] R. Thakur et al., "Optimization of collective communication operations in MPICH," Int. J. High Perform. Comput. Appl., vol. 19, no. 1, 2005, pp. 49-66.

[23] M. Chaarawi et al., "A tool for optimizing runtime parameters of Open MPI," in Recent Advances in Parallel Virtual Machine and Message Passing Interface, vol. 5205, Springer, Berlin, Heidelberg, Germany, 2008, pp. 210-217.

[24] E. Nuriyev and A. Lastovetsky, "Accurate runtime selection of optimal MPI collective algorithms using analytical performance modelling," arXiv preprint, 2020, CoRR, arXiv: 2004.11062.

[25] J. Pješivac-Grbović et al., "MPI collective algorithm selection and quadtree encoding," Parallel Comput. vol. 33, no. 9, 2007, pp. 613-623.

[26] S. Hunold et al., "Predicting MPI collective communication performance using machine learning," in Proc. IEEE Int. Conf. Clust. Comput. (CLUSTER), (Kobe, Japan), Sept. 2020.

[27] J.M. Hashmi et al., "Design and characterization of shared address space MPI collectives on modern architectures," in Proc. IEEE/ACM Int. Symp. Clust., Cloud Grid Comput. (CCGRID), (Larnaca, Cyprus), May 2019.

[28] Google, Xpmem: Cross-process Memory Mapping, 2011, https://code.google.com/archive/p/xpmem/

[29] Google, Google Code Archive XPMEM, https://code.google.com/archive/p/xpmem/

[30] S. Chakraborty et al., "SHMEMPMI--Shared memory based PMI for improved performance and scalability," in Proc. IEEE/ACM Int. Symp. Clust., Cloud Grid Comput. (CCGrid), (Cartagena, Colombia), May 2016.

[31] P. Balaji et al., "PMI: A scalable parallel process-management interface for extreme-scale systems," in European MPI Users' Group Meeting, Springer, Berlin, Heidelberg, Germany, 2010, pp. 31-41.

[32] R.L. Graham et al., "Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction," in Proc. Int. Workshop Commun. Optim. HPC (COMHPC), (Salt Lake City, UT, USA), Nov. 2016.

[33] NVIDIA, NVIDIA Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP), https://docs.nvidia.com/networking/display/sharpv214

[34] J. Stern et al., "Accelerating MPI_Reduce with FPGAs in the Network," Proc Workshop on Exascale MPI. 2017.

[35] P. Haghi et al., "FPGAs in the network and novel communicator support accelerate MPI collectives," in Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC), (Waltham, MA, USA), Sept. 2020.

[36] Mvapich, OSU Collective MPI Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/

[37] S. Kumar et al., "Optimization of MPI collective operations on the IBM Blue Gene/Q supercomputer," Int. J. High Perform. Comput. Appl., vol. 28, no. 4, 2014, pp. 450-464.

[38] J. Liu, A.R. Mamidala, and D.K. Panda, "Fast and scalable MPI-level broadcast using InfiniBand's hardware multicast support," in Proc. Int. Parallel Distrib. Process. Symp., (Santa Fe, NM, USA), Apr. 2004.

[39] T. Hoefler, C. Siebert, and W. Rehm, "A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast," in Proc. Int. Symp. Parallel Distrib. Process. Symp., (Long Beach, CA, USA), Mar. 2007, pp. 1-8.

[40] S. Aga et al., "Compute caches," in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), (Austin, TX, USA), Feb. 2017.

[41] S. Jung et al., "A crossbar array of magnetoresistive memory devices for in-memory computing," Nature, vol. 601, 2022, pp. 211-216.

[42] J. Huang et al., "Active-routing: Compute on the way for near-data processing," in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), (Washington, DC, USA), Feb. 2019.

[43] M. Torabzadehkashi et al., "Catalina: In-storage processing acceleration for scalable big data analytics," in Proc. Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP), (Pavia, Italy), Feb. 2019.

[44] Github, Faiss, https://github.com/facebookresearch/faiss

[45] Texmex, Datasets for approximate nearest neighbor search, http://corpus-texmex.irisa.fr

[46] S.W. Jun et al., "Bluedbm: An appliance for big data analytics," in Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA), (Portland, OR, USA), June 2015.

[47] B. GU et al., "Biscuit: A framework for near-data processing of big data workloads," ACM SIGARCH Comput. Archit. News, 2016, vol. 44, no. 3, pp. 153-165.

[48] S.C. Kim et al., "In-storage processing of database scans and joins," Inf. Sci., vol. 327, 2016, pp. 183-200.

[49] SoCs, MPSoCs & RFSoCs, https://www.xilinx.com/products/silicon-devices/soc.html

[50] Hadoop, Hadoop MapReduce, https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

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