Switched Network로 연결된 Cluster의 MPICH에서 효율적인 MPI_Allgather Algorithm

김철환° 정유진
한국 외국어 대학교 컴퓨터 공학과
{redfoot°, chungyj,}@hufs.ac.kr

Effective MPI_Allgather Algorithm in MPICH for Clusters Connected by Switched Networks

Chul-hwan Kim° Yoojin Chung
Dept. of Computer Eng. Hankuk Univ. of Foreign Studies

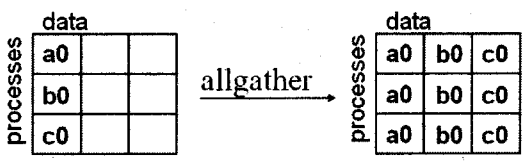
요 약

본 논문은 Linux Cluster의 MPICH에서 MPI_Allgather Algorithm의 성능을 개선하고 실험을 통해 최대 30%의 성능향상을 증명하였다. MPICH의 기존 버전이 메시지의 크기와 실행 프로세스 수에 따라 Recursive Doubling, Bruck Algorithm, Ring Algorithm을 차등 적용했던 것을, 앞의 Algorithm을 개선하여 Double Bruck Algorithm, Double Ring Algorithm을 제안, 구현하였다.

1. 서 론

많은 계산 시간을 요하는 문제를 해결하기 위하여 고속 프로세서가 다양하게 연결된 병렬 컴퓨터가 개발되어 왔고 병렬처리 기술이 널리 사용되고 있다. 병렬처리를 효과적으로 구현하기 위해서는 프로세서들 사이에 공유 메모리를 두어 서로 데이터를 공유하는 공유 메모리 방식을 쓰거나, 프로세서마다 지역 메모리를 두고 상호 연결된 통신망을 이용하여 데이터를 주고받는 메시지교환 방식을 사용한다. 이 두 방식은 서로 장단점을 가져 대비되는 방식으로 확장성이 우수한 메시지 교환방식에 의해 많은 병렬 컴퓨터가 제작되어 왔다. 메시지 교환방식을 구현하는데 필수적으로 메시지를 주고받는 하드웨어와 통신망, 사용자가 통신을 위해 호출하게 되는 소프트웨어가 지원되어야 한다. 이중 소프트웨어는 서로 다른 시스템 간에도 메시지를 주고받을 수 있게 하기 위하여 표준화가 이루어져 있으며, 그 중 널리 쓰이는 것이 MPI(Message Passing Interface)[1]와 PVM(Parallel Virtual Machine)[2]이다.

MPI는 메시지 교환 표준을 구현한 통신 라이브러리를 제공하며 Point-to-point communication, Collective communication 등의 수십여 가지 함수를 포함한다. 이 중 Collective communication은 다수의 프로세스가 통신에 참여하는 것으로 MPI_Allgather, MPI_Scatter, MPI_Gather, MPI_Bcast등의 함수가 있다. 그 중 MPI_Allgather는 high-level collective communication 함수로서 [그림 1]과 같이 동작한다. MPI_Allgather에 참여하는 프로세스는 데이터의 일부분을 분배한다. 그래서 MPI_Allgather 함수가 끝나게 되면 각 프로세스는 모든 다른 프로세스에게 데이터를 분배하게 되고 결과적으로 같은 데이터들을 갖게 된다. MPI_Allgather는 각 프로세스가 모든 다른 소구역의 결과에 따라 그 소구역의 계산에 책임이 있는 시뮬레이션이나 모델링 응용프로그램에 사용된다. 때문에, MPI_Allgather의 구현은 응용 프로그램의 성능에 많은 결과를 미치게 된다.



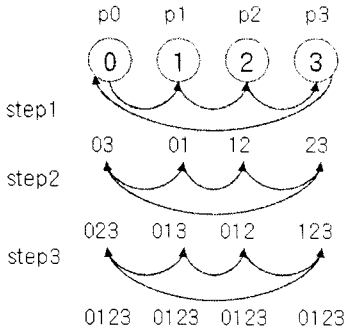
[그림 1] MPI_Allgather

본 논문에서는 FastEthernet과 같은 Switched Networks에서 널리 사용되는 병렬 라이브러리인 MPICH[3]에서의 MPI_Allgather 함수를 개선된 알고리즘을 이용하여 성능이 향상되도록 설계하고 병렬 컴퓨터에 직접 구현하여 원래의 MPI_Allgather 함수와 성능을 비교하였다.

2. 관련연구

MPICH에서 구현된 초기의 MPI_Allgather 알고리즘은 Ring Algorithm를 사용하여 구현하였다. Ring Algorithm은 프로세스 간에 가상의 원으로 연결되어 있다고 가정하고 데이터를 보내는 방식이다. 즉, 첫 번째 단계에서 프로세스 p1은 프로세스 p1+1에 데이터를 전송하고 다음 단계에선 p1프로세스가 p1-1에게 수신한 데이터를

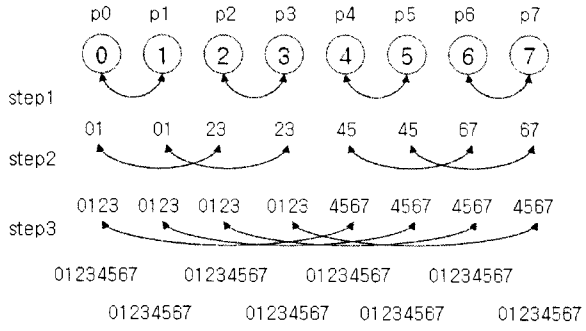
다시 p1+1에게 전송하는 형태이다[그림 2]. 그래서 Ring Algorithm의 전체 알고리즘은 p-1(p : 실행 프로세스의 개수)단계를 거치게 된다.



[그림 2] Ring Algorithm

2.1 Recursive Doubling

Recursive Doubling은 [그림 3]과 같이 동작한다.



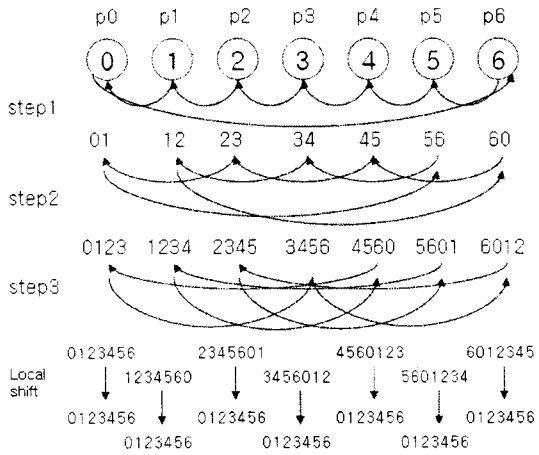
[그림 3] Recursive Doubling

첫 번째 단계에서 각 프로세스에서 거리 1의 프로세스에게 데이터를 전송하고, 두 번째 단계에서는 각 프로세스에서 거리 2의 프로세스에게, 세 번째 단계에서 각 프로세스에서 거리 4의 프로세스에게 자신이 가지고 있는 데이터를 전송하는 것을 반복함으로써 최종적으로 모든 프로세스가 각 프로세스의 모든 데이터를 가질 수 있도록 하는 알고리즘이다. 그 결과 총 lg p의 단계가 필요하게 된다.

하지만 Recursive Doubling Algorithm은 프로세스의 개수가 2의 배수가 아닐 경우 약간의 문제가 생기게 된다. 즉, 하나의 프로세스는 데이터 전송에 참가하지 못하게 되는데 MPICH-1.2.7p1에서는 이 문제를 Bruck Algorithm[4]을 통해 해결하고 있다.

2.2 Bruck Algorithm

Bruck algorithm은 프로세스의 개수가 2의 배수가 아닌 경우에도 ⌈lg p⌉의 단계에 처리가 가능하다. Bruck Algorithm의 동작은 [그림 4]와 같다.



[그림 4] Bruck Algorithm

각 단계에서는 Recursive Doubling처럼 거리 1, 2, 4의 식으로 데이터를 전달하지만 양방향이 아닌 하나의 방향으로만 전달한다는 차이가 있다. 첫 번째 단계에서는 거리 1의 p-1프로세스에게 데이터를 전달하고 두 번째 단계에서는 p-2, p-4의 식으로 프로세스가 모든 데이터를 수신할 때까지 진행된다. 이 과정은 ⌊lg p⌋ 단계까지 계속되지만 최종 단계에서 출력 버퍼에 데이터가 올바른 순서대로 저장되지 않기 때문에 각 프로세스 번호 별로 p 블록의 shift가 필요하게 된다. 그 결과 최종적은 ⌈lg p⌉의 단계를 거치게 된다.

2.3 MPI_Allgather Operations in MPICH

초창기 MPICH의 MPI_Allgather 함수는 Ring Algorithm만을 사용하여 구현되었지만, 현재 버전 1.2.7의 MPI_Allgather는 복합적인 Algorithm을 적용하고 있다. Thakur[5]의 논문에 따르면 Bruck Algorithm은 메시지 크기가 작고 프로세스가 2의 배수가 아닌 경우 최상의 결과가 나온다는 것을 실험을 통해 증명하였다. 또한 Recursive Doubling은 메시지 크기가 작거나 중간일 경우, 프로세스는 2의 배수일 경우 최상의 결과를 보였으며 Ring Algorithm의 경우 어떤 수의 프로세스라도 큰 크기의 메시지 전송에는 가장 좋은 성능을 보였다.

위의 실험을 바탕으로 현재 MPICH는 메시지 크기가 80KB이하이며 프로세스의 개수가 2의 배수가 아닐 경우 Bruck Algorithm이 적용되었으며, 메시지 크기가 512KB 이하, 프로세스의 개수가 2의 배수일 경우에는 Recursive Doubling Algorithm이 적용된다. 마지막으로 프로세스의 수에 상관없이 512KB이상의 메시지일 경우에는 Ring Algorithm을 실행하도록 구현되어 있다.

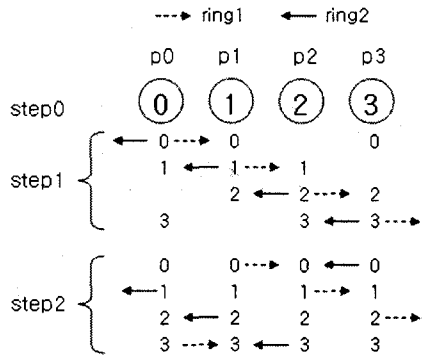
3. 효율적인 MPI_Allgather Agorithms

기본적으로 TCP/IP 프로토콜에서는 pair-wise exchange가 TCP/IP 트래픽을 최소화하는데 최상이라고 Gregory Benson은 밝히고 있다[6]. 때문에 이러한 관점

을 기반으로 응용된 Double Bruck Algorithm과 Double Ring Algorithm을 설계하고 구현하였다.

3.1 Double Ring Algorithm

Ring Algorithm은 Recursive Doubling이나 Bruck Algorithm에 비해 크기가 큰 메시지 전송에 유리하다. 이는 멀리 떨어진 프로세스 간 전송보다 가장 가까운 Neighbor Communication에 더욱 효율적이기 때문이다. 이러한 특징을 이용하여 Neighbor Exchange Algorithm을 고안, 어느 정도의 성능향상을 볼 수 있었다[7]. 본 논문에서는 기존의 Ring Algorithm이 갖고 있는 큰 크기의 메시지 전송의 유리함을 유지하기 위해 Ring Algorithm을 그대로 이용하되 이를 이중으로 활용하는 방법을 고안하였다. 즉, Ring Algorithm이 한 방향으로 이웃 프로세스에게 메시지를 전송하였다면, Double Ring Algorithm은 반대 방향에도 Ring Algorithm을 적용, 양 방향으로 메시지를 전송하도록 하였다[그림 5].



[그림 5] Double Ring Algorithm

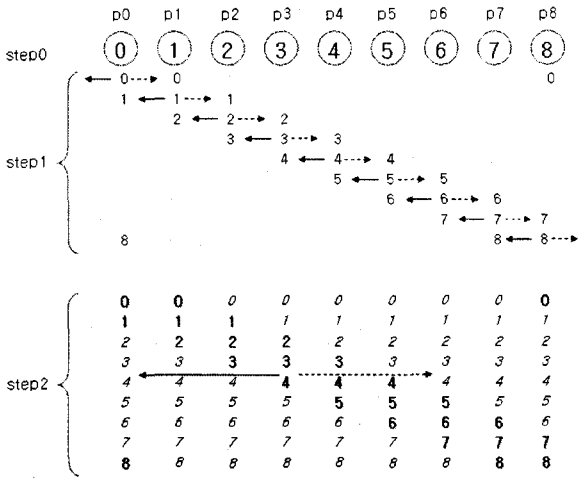
그래서 ring1과 ring2의 두 개의 Ring이 존재하게 되며, Ring Algorithm은 p-1 단계를 거친 반면 Double Ring Algorithm은 ⌊p/2⌋ 단계에서 해결이 가능하게 된다.

3.2 Double Bruck Algorithm

Double Bruck Algorithm은 [그림 6]과 같다.

Bruck Algorithm은 [그림 4]와 같이 첫 번째 단계에서는 이웃된 프로세스에게 메시지를 전송하지만 두 번째 단계에서 부터는 2, 4, 8... 의 거리의 프로세스에 메시지를 전달함으로써 각 단계마다 프로세스가 가질 수 있는 메시지의 개수는 2의 지수 증가를 보인다. 하지만 Double Bruck Algorithm은 첫 단계에서 인접한 두 개의 프로세스에게 메시지를 받아 3개의 데이터를 보유하게 되고 각 단계가 진행 될 때마다 3의 지수 증가로 데이터를 보유하게 된다.

그러나 Double Bruck Algorithm은 프로세스의 개수가 정확하게 3의 지수승이 아닐 경우 전송되는 총 바이트의 수가 $(m/p)*(3^{\lceil \log_3 p \rceil}-p)$ 만큼 증가하게 된다. 하지만 이는 많은 수의 프로세스 간 MPI_Allgather를 수행하게 될 경우 $\log_2 p$와 $\log_3 p$의 차이에 따른 총 실행 단계 단축에 따른 성능향상을 보일 것이다.



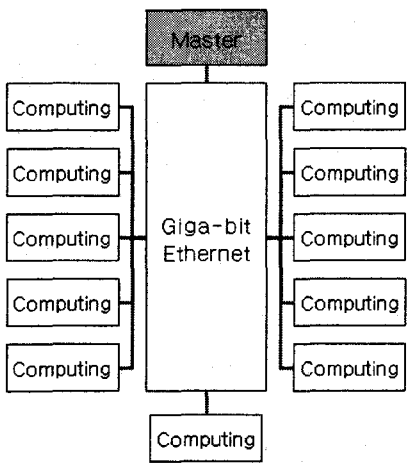
[그림 6] Double Bruck Algorithm

4. 실험 및 결과

4.1 실험환경

본 논문의 실험을 진행하기 위해 12 Node 병렬 컴퓨팅 환경을 구축하였다. 각 Node의 시스템은 PentiumIII 866Mhz의 CPU와 256MB의 RAM을 가진 컴퓨터로 이루어져 있으며, 각 Node는 Giga-bit Ethernet으로 연결되어 있다[그림 7].

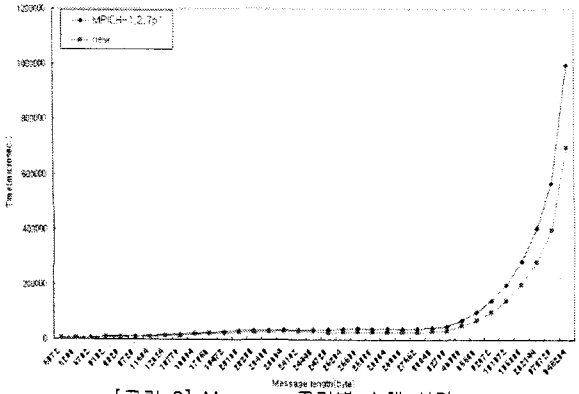
MPI 라이브러리는 버전 1.2.7의 MPICH-1.2.7p1[3]을 사용 하였으며, MPI_Allgather의 성능 측정을 위해 SKaMPI Benchmark[8]를 사용하였다. 그리고 각 Node 당 하나의 프로세스를 실행함으로써 총 12개의 프로세스까지 실험하였다.



[그림 7] 12 Node PC-cluster

4.2 결과

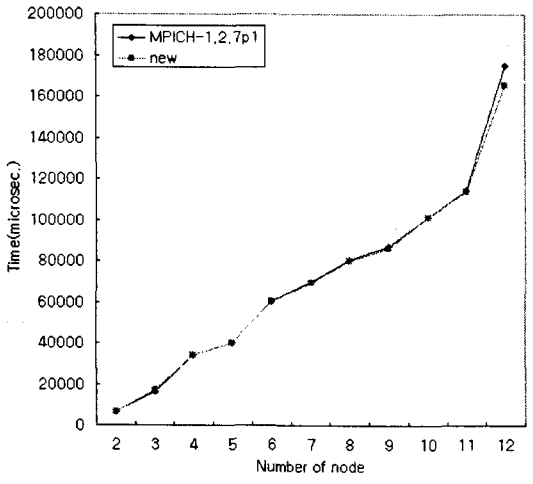
MPICH-1.2.7p1의 원래 MPI_Allgather Algorithm을 사용한 테스트 결과와 Double Ring Algorithm, Double Bruck Algorithm을 사용한 테스트(new) 결과를 SKaMPI Benchmark를 통해 산출하였으며 [그림 8], [그림 9]와 같다.



[그림 8] Message 크기별 수행 시간

[그림 8]은 12 Node에서 각각 Message 크기별로 MPI_Allgather가 수행되는데 걸리는 시간을 나타낸 것이다. 그래프에서 작은 크기의 메시지인 6072byte이하는 성능 차이가 미비해서 생략하였으며 메시지 크기가 커짐에 따라 MPI_Allgather 함수의 효율이 좋아짐을 알 수 있다. 마지막 645264byte의 메시지의 경우 MPI-1.2.7p1의 경우 996772.8micro sec인 반면 제안한 알고리즘은 697741.3micro sec이어서 최대 30% 정도의 수행 시간 단축 효과를 볼 수 있었다.

[그림 9]에서는 실행 Node(프로세스)의 수에 따른 수행 시간 결과를 보여주고 있는데, Double Bruck Algorithm이 실행되는 프로세스가 3의 지수승 증가를 보일 경우 전송 시간도 함께 증가하게 되지만 기존의 Bruck Algorithm에 비해 좀 더 나은 결과를 낼 수 있기 때문에, 실행되는 프로세스의 수가 더욱 증가될수록 성능향상 효과가 커질 수 있음을 볼 수 있는 대목이다.



[그림 9] 실행 Node의 수에 따른 수행 시간

5. 결론 및 향후 과제

본 논문에서는 여러 프로세스가 동시에 통신에 참여하는 MPI_Allgather 함수를 기존의 Ring Algorithm, Bruck Algorithm을 수정하여 구현하였으며 SKaMPI Benchmark를 통해 최대 30%정도의 성능향상 효과를 볼 수 있었다. 하지만 제한된 실행 프로세스의 개수로 인해 16, 32, 64개로 증가하는 다수의 프로세스에서 성능향상을 확인할 수 없었다. 이는 추후 실험을 통해 검증해야 할 것이며, 추가로 MPI 통신 라이브러리의 다른 collective communication 함수의 성능 개선을 통해 전체 collective communication의 성능 향상을 실행 할 것이다.

References

[1] W. Gropp, E. Lusk and A. Skjellum, Using MPI, MIT Press, 1994

[2] A. Geist et al, PVM : Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994

[3] MPICH(Message Passing Interface) http://www-unix.mcs.anl.gov/mpi/

[4] Bruck, J., Ho, C-T., Kipnis, S., Upfal, E., and Weathersby, D. 1997. "Efficient algorithms for all-to-all communications in multiport message-passing systems". IEEE Transactions on Parallel and Distributed Systems 8(11):1143-1156

[5] R. Thakur, Rolf Rabenseifner, William Gropp. 2005. "Optimization Of Collective Communication Operations In MPICH". The international Journal of High Performance Computing Applications, Vol 19, No. 1, Spring 2005, pp. 49-66.

[6] Gregory D. Benson, Cho-Wai Chu, Qing Huang, and Sadik G. Caglar, "A comparison of MPICH allgather algorithms on switched networks", In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI Users' Group Meeting, pages 335-343. Lecture Notes in Computer Science 2840, Springer, September 2003.

[7] Jing Chen, Yunquan Zhang, Linbo Zhang, Wei Yuan, "Performance Evaluation of Allgather Algorithms On Terascale Linux Cluster with Fast Ethernet", Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region, IEEE 2005.

[8] Worsch, T., Reussner, R., and Augustin, W. 2002. On benchmarking collective MPI operations. "Recent Advances in Parallel Virtual Machine and Message Passing Interface", 9th European PVM/MPI Users' Group Meeting, Lecture Notes in Computer Science Vol. 2474, D. Kranzlmuller, P. Kacsuk, J. Dongarra, and J. Volkert, editors, Springer Verlag, Berlin, pp.. 271-279