제19회 한국정보처리학회 춘계학술발표대회 논문집 제10권 제1호 (2003.5)

PC 클러스터 상에서 압축알고리즘을 이용한 Collective Function의 성능측정

임동익, 이효종
전북대학교 전자정보공학부
e-mail:{dilim, hlee}@sel.chonbuk.ac.kr

Performance Measurement of Collective Functions Using Compression Algorithm on PC Cluster

Dong Ick Im and Hyo Jong Lee
Dept. of Electronics and Information Engineering
Chonbuk National University

요약

계산량이 매우 큰 작업의 경우는 단일 프로세서를 이용할 경우 많은 계산 시간이 소요된다. 이러한 문제점을 극복하고자 저비용 고효율의 PC Clustering 기법을 사용하면 비용적인 절감의 효과를 얻을 수 있다. 본 논문은 PC Clustering을 이용한 병렬처리를 수행함으로써 시간의 단축을 도모하되 표준 MPI 함수 중 Collective Communication을 취급하는 함수들의 성능을 향상시켜 개선하고 그 성능을 측정하는데 목적이 있다. 또한 표준 MPI 함수를 사용하는 MPICH와 표준 MPI 함수 중 Collective Communication을 사용하는 함수들의 데이터를 압축하여 전송하도록 MPI를 개선하였다. 실험은 윈도우 2000을 탑재한 20개의 노드를 가지는 시스템을 이용하였다. 본 실험의 결과로써 데이터의 양과 노드 수를 증가시킬수록 압축 MPI의 성능이 표준 MPI의 성능을 능가함을 확인할 수 있었다.

1. 서론n

병렬처리란 여러 개의 프로세서를 동시에 이용해서 연산능력을 극대화하여 성능의 한계를 극복하기 위한 방법인데, CPU를 여러 개 사용함으로써 그 성능의 몇 배에 달하는 효과를 낼 수 있다. 물론 중대형 컴퓨터를 이용하면 되겠지만 비용적인 측면에서 무리가 있으므로 여러 대의 일반 컴퓨터를 병렬로 연결하여 Clustering 함으로써 적은 비용으로 높은 효과를 낼 수 있다.

본 논문에서는 병렬프로그램의 도구인 표준 MPI (Message Passing Interface)의 collective 함수들의 자료를 압축하는 변형 MPI인 ZipMPI를 사용하였다. ZipMPI는 Multi-Threading 기법과 데이터 압축을 Collective Function에 접목시킴으로써 성능 향상을 추구하였다.

다중 CPU 시스템에서 프로세서 단위로 스케줄링이 이루어질 때는 아무리 많은 CPU가 있어도 프로세서는 한 CPU에 할당될 수밖에 없었다. 그러나 Threads를 사용할 경우 각 Thread 하나 하나가 각각의 CPU에 할당될 수 있기 때문에 성능 향상을 가져올 수 있다.

또한 데이터 압축을 함으로써 Point-to-Point 함수와 Collective 함수에서의 메시지 사이즈를 줄임으로써 메시지 전송에 필요한 시간을 줄이고 그에 따라서 작업을 수행하는데 걸리는 시간을 단축시킬 수 있다.

한편 이 논문에서는 병렬화된 Π값을 계산하는 알고리즘을 PC상에서 Clustering 기법을 이용하여 구현하였고 이를 바탕으로 표준 MPI와 ZipMPI를 비교하였고, 또한 WC(Word Count)를 이용해서 ZipMPI가 가진 압축 알고리즘의 최대 성능을 알아보고자 하였다. 본 논문에서는 Windows 2000이 설치된 Pentium

III 20대로 구성된 PC 클러스터 상에서 MPICH와 ZipMPI를 이용하여 실험하였다. 데이터의 크기에 따른 시간과 노드 수에 따른 시간을 측정함으로써 표준 MPI의 성능과 압축 MPI 성능을 비교하였다.

본 논문의 구성은 2절에서는 ZipMPI의 아키텍처에 대해서 기술하고, 3절에서는 성능 측정을 위한 벤치마크에 대한 설명을 하였다. 4절에서는 실험 결과 값을, 마지막으로 5절에서 결론을 기술하였다.

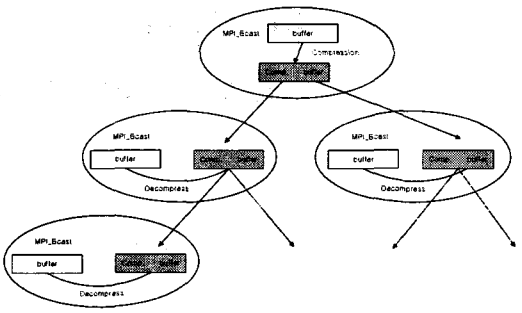
2. ZipMPI에서의 Multi-Threading과 자료 압축

ZipMPI의 설계 방식은 이식성, 효율, 그리고 기능성을 추구하는 면에서는 여타의 MPI와 다를 바 없다. 하지만 Collective Communication의 성능을 올리기 위해서 Thread와 데이터 압축을 사용하는 점에서는 ZipMPI와 여타의 MPI와의 차이점을 발견할 수 있고 그에 따른 성능 향상을 본 논문을 통해서 알아볼 수 있다.

ZipMPI의 구조는 크게 3개의 계층으로 구성되어 있다. Socket Layer라고 이름 붙여진 첫 번째 계층은 기본적인 서비스들을 제공하는 계층으로써 Unix나 Linux상에서의 socket interface나 Win32기반의 Winsocket interface를 사용하는 TCP/IP 프로토콜을 사용한다. XMP라 이름 붙여진 두 번째 계층은 프로세서 그룹들을 위해 간단한 통신 환경을 제공한다. 메시지를 교환하는데 사용되는 매우 간단한 인터페이스를 제공한다.

자료압축은 데이터의 사이즈를 줄이기 위해서 사용된다. 즉, 커다란 크기의 데이터나 통신량이 필요한 작업이 있을 때, 거기에 데이터 압축을 사용하게 되면 데이터의 크기의 감소로 인한 성능의 향상을 가져올 수 있다. 많은 압축 알고리즘이 존재하고 그 중 대부분의 압축 알고리즘은 파일에 대한 압축 기능을 제공한다. 이 논문에서는 메모리 압축에 최적화된 lzo 알고리즘을 사용하였다. 이 알고리즘은 실시간으로 데이터의 손 실없이 매우 빠른 압축 속도와 압축 해제 속도를 보여주기 때문에 시간대 압축비의 성능에서 매우 좋은 성능을 가지고 있다.

ZipMPI는 이러한 Thread와 자료압축을 MPI의 ColleMaster 노드와 Slave 노드 사이에서 전체 작업을 Master 노드가 조율하고 Slave 노드는 실제적인 계산 작업을 분산 처리함으로써 계산 시간의 단축을 꾀할 수 있었다.ctive 함수들에 결합시켰다. 예로써 (그림 1)에서와 같이 Master Process에서 다른 모든



(그림 1) 데이터 압축을 이용한 ZipMPI에서의 Broadcast의 동작 원리

Slave Process들로의 같은 데이터를 보내는 Broadcast 함수에 적용시켜 보았다. Master Process에서 Slave Process들로 데이터를 보내기 전에 데이터를 압축하고, 이렇게 압축함으로써 데이터의 크기도 줄어들기 되고, 그에 따라서 전송 시간이 줄게 되어서 Collective Function을 사용함에 있어서 성능 향상을 가져올 수 있었다.

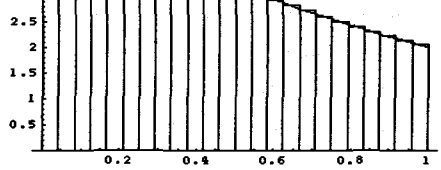
3. 성능 측정

본 논문에서는 표준 MPI와 ZipMPI와의 성능 비교를 위해서 계산 중심의 PI(Π) 값을 구하는 프로그램과 데이터 전송 중심의 파일 분석 프로그램을 이용하였다. 먼저 PI 값을 구하는 프로그램은 다음과 같은 Collective Function을 사용한다.

- MPI_Bcast : 그룹에 있는 모든 프로세스들에게 동일한 message를 전송한다.
- MPI_Reduce : 모든 프로세스들이 정해진 Operation을 수행한 후 연산 결과를 root 프로세스의 결과에 저장한다.

PI 값을 구하는 프로그램은 다음과 같은 값을 측정한다.

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$



(그림 2) 중간점 공식을 이용한 Π값 계산

이 벤치마크에서는 원의 넓이 공식을 이용해서 위의 (그림 2)에서 보는 바와 같이 1/4 원을 n등분한 후에 각각의 면적의 합이 원의 면적의 1/4 과 같다는 공식을 이용해서 Π값을 구하게 된다. 이러한 과정에서 n이 크면 클수록 계산되어 나온 Π값의 오차는 줄어들게 되며, 그에 따른 계산 시간은 증가하게 된다. 한편 Master 프로세스는 각각의 프로세스에 계산해야할 작업을 MPI_Bcast를 이용해서 전송한다.

```
/* broadcast n from root process to everybody else
/

MPI_Bcast(&n,1,MPI_INT,ROOT_PROC,
MPI_COMM_WORLD);
```

각각의 Slave 프로세스들은 각각의 Slave 프로세스들이 계산한 넓이의 합을 MPI_Reduce를 이용해서 최종적인 넓이를 구하게 되고 그 구하게 된 결과 값을 Master 프로세스의 result에 저장하게 된다.

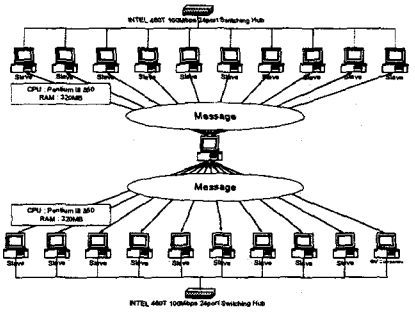
```
/* sum mypi across all processes, storing result as
pi on root process */

MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,
MPI_SUM,ROOT_PROC,
MPI_COMM_WORLD);
```

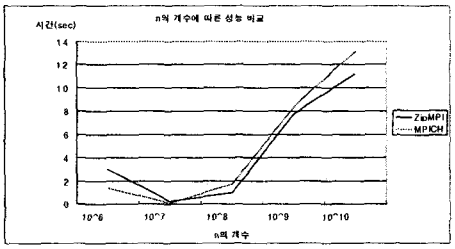
또한 두 번째로 사용한 벤치마크는 WC(Word Count)라는 프로그램을 압축 알고리즘의 벤치마크에 맞게 수정을 하여 사용하였다. WC라는 프로그램은 사용자가 입력한 문서 파일을 읽어 들여서 내부에 있는 문자수와 단어 수 그리고 문장 수를 계산하는 프로그램이다. 특히 이 논문에서는 압축 알고리즘의 성능을 극대화시키기 위해서 432,782,017 바이트의 크기를 갖는 문서파일을 테스트에 사용하였고 이 샘플 파일을 이용해서 WC 프로그램을 수행하였다.

4. 실험 결과

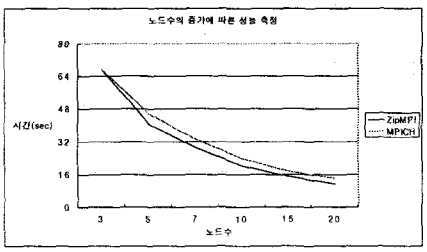
본 논문의 실험을 위해서 (그림 3)과 같이 20대의 PC를 100Mbps의 초고속 Ethernet으로 Clustering하여 이용하였고 각 PC의 사양은 Pentium-III 850MHz CPU, 320MB Memory이다. 또한 병렬 처리 관련 프로그램은 MPI를 사용하여 메시지를 이용한 통신을 하였다.



(그림 3) MPI 기반의 실험 환경



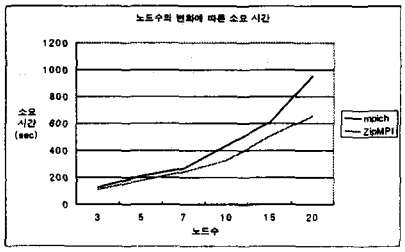
(그림 4) n의 개수에 따른 성능 비교



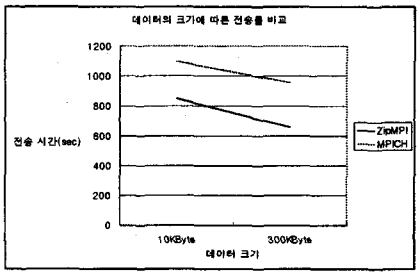
(그림 5) 노드수에 따른 성능 비교

(그림 4)는 PI계산에서 사각형의 개수(n)을 계속 증가시켜가면서 실험한 결과이다. 즉 n이 크면 클수록 Π값의 오차 값은 줄어들게 되지만 반면 연산 시간은 증가하게 된다. 사각형의 개수를 늘리면 늘릴수록 개선된 Collective Function을 사용하는 ZipMPI의 성능이 뛰어남을 알 수 있다.

(그림 5)는 사각형의 개수를 10^{10}개로 고정시킨 상태에서 노드 수를 증가시켜가면서 그에 따른 측정 시간을 기록하였다. 이 벤치마크 역시 노드 수가 증가함에 따라 ZipMPI를 이용한 벤치마크의 시간이 적게 걸림을 알 수 있다. 20개의 노드를 모두 이용한 경우 속도 향상률은 표준 MPI의 경우에 비해 ZipMPI를 이용함으로써 16.9%의 성능 향상을 나타내었다.



(그림 6) 노드수에 따른 WC의 처리시간 변화



(그림 7) 데이터의 크기에 따른 전송 시간

(그림 6, 7)은 두 번째 벤치 마크 프로그램인 WC의 벤치 결과를 그래프로 나타낸 것이다. (그림 6)에서는 한번에 전송하는 데이터의 크기를 300KByte로 잘라서 전송했을 때의 노드 수에 따른 소요시간을 측정한 그림이며, 그림에서 보는바와 같이 노드수가 증가함에 따라서 압축률이 좋은 문서 파일의 특성상 압축 알고리즘의 성능이 강력하게 나타남을 확인할 수 있다. (그림 7)은 데이터의 크기에 따른 전송 시간을 측정한 것이다. 데이터의 크기를 10KByte와 300KByte로 달리하여 측정한 결과이며 데이터의 크기가 커짐에 따라 압축 알고리즘을 이용하는 ZipMPI의 성능이 좋아짐을 확인할 수 있다. 노드 20개를 이용해서 측정한 속도 향상율의 경우는 ZipMPI를 이용함으로써 45.5%라는 결과를 보여주었다. 이는 엄청나게 큰 사이즈의 텍스트 파일을 이용함으로써 압축 알고리즘의 강력한 성능을 보여주는 결과이다.

5. 결론

본 논문에서는 표준 병렬 처리 개발 도구인 MPI를 보다 더 개선시켜서 좀더 효율적인 계산을 해보고자 하였다. 본 논문에서는 Collective Communication을 향상시킨 ZipMPI를 계산량 위주의 프로그램인 PI와 데이터 전송량 위주의 프로그램인 WC에 적용을 시켜보았고, 표준 MPI에 비해서 각각 16.9%와 45.5%의 속도 향상률을 나타낼 수 있었다. 이는 데이터 전송 시간보다 계산을 위주로 하는 PI 보다는 데이터 전송 위주인 WC에 압축 알고리즘을 적용함으로써 데이터 전송 시간에 단축을 꾀함으로써 보다 높은 속도 향상률을 보여주었다.

참고문헌

[1] Hyo Jong Lee, Dong Ick Im, and Bum Hyun Lim, PC Cluster 상에서의 병렬 광선 추적 알고리즘의 성능 정보처리학회 춘계학술대회 vol.1, no 9, pp 389-392, April, 2002

[2] F. Garcia, A. Calderon, J. Carretero Evaluating MiMPI, a Multithread-Safe Implementation of MPI, on a Cluster of Workstations. International Conference of Software Engineering Applied to Networking and Parallel/Distributed Computing, Reims, May 2000.

[3] Chang-Geun Kwon, Hyo-Kyung sung, and Heung-Moon Choi. An implementation of a parallel raytracing algorithm on hybrid parall architecutre. IEEE Conference on Acoustics, Speech and Signal Processing, 3:1745 1748, August 1998.

[4] POV-Ray Team, Persistence of Vision Ray Tracer (Pov-Ray) Version 2.0 User's Documentation, Private Publication. 1993

****

동익,

1. 논문은 주간지 기사가 아니다. '엄청난' 또는 '놀라운' 등의 수식어는 삼가라.
2. 그림 7도 괜찮지만, 먼저 그림6의 경우 데이터 전송크기가 얼마인지를 명기하고, 그와 다른 크기의 데이터전송크기일때의 처리시간을 비교하는 것이 중요. 당연히 다른 결과가 나올 것이고, 그 차이는 그림7로 설명할 것이다.
3. 결론을 보강하여 최대한 자세하게 기술하라. 즉 실험 결과를 언급하여 계산중심의 경우의 성능향상과 데이터처리 중심의 경우의 성능향상이 어떻게 다른지, 왜 다른지를 기술할 것.
4. 졸업논문이 이와 같이 성의 없이 작성되면 더 이상 읽어보지 않을 것이다.