MPI-IO를 위한 로그 기반 특성 및 구조 분석

차광호
한국과학기술정보연구원 국가슈퍼컴퓨팅연구소
e-mail : khocha@kisti.re.kr

A Log-based Analysis on the Characteristics and Structure of MPI-IO

Kwangho Cha
National Institute of Supercomputing and Networking,
Korea Institute of Science and Technology Information

요 약

메시지 전달 방식의 병렬 프로그래밍에서 사용되는 MPI는 프로그램의 확장성 보장에 적합한 MPI-IO라는 파일 I/O 방법을 제공하고 있다. MPI-IO는 동시적인 병렬 I/O 수행으로 인한 성능 저하를 최소화하기 위하여 내부적으로 데이터 재정렬 후 I/O를 수행한다. 본 연구에서는 이와 같은 MPI-IO의 내부 처리 과정을 기록하기 위한 방안을 강구하여 실행 시간 로그를 기록하였고 이를 바탕으로 MPI-IO의 특성을 살펴보았다.

1. 서론

MPI-IO는 단일 파일 기반의 병렬 I/O을 지원하고 있어서 계산 노드의 수가 증가하고 있는 고성능 병렬 시스템에서 확장성을 보장하고자 하는 프로그램에 적합한 I/O 방법이라 할 수 있다. MPI-IO는 동시적인 병렬 I/O 수행으로 인한 성능 저하를 최소화하기 위하여 내부적으로 데이터 재정렬 후 I/O를 수행한다. 이 과정은 크게 데이터 교환 단계와 I/O 단계로 구분되며 기존의 MPI 함수를 이용하여 구현되어 있다[1].

본 연구에서는 이와 같은 MPI-IO의 내부 처리 과정을 로깅 가능하도록 MPI 라이브러리를 수정하여 I/O 수행 과정의 로그를 수집하였고 이를 바탕으로 집합 I/O의 수행 특성을 분석하였다. 로그를 분석한 결과, I/O 과정과 데이터 교환을 위하여 대부분의 시간이 사용됨을 확인할 수 있었다. 이는 집합 I/O의 성능 향상이 I/O뿐만 아니라 내부적인 데이터 교환 과정의 성능 개선을 통해서도 가능하다는 것을 의미한다고 할 수 있다.

2. MPI-IO

MPI-IO는 MPI-2에 정의된 I/O 기능으로 MPI 커뮤니케이터에 등록된 프로세스들이 대상이 되며 MPI 데이터형(Data Type)을 이용하여 각 프로세스가 접근하고자 하는 I/O 데이터의 위치를 기술하게 된다.

MPI가 메시지 전달에 의한 병렬 프로그래밍을 지향하는 만큼 다양한 통신 방법을 제공하고 있는데 MPI-IO 또한 메모리 공간의 데이터를 디스크 공간으로 이동시키는 일종의 확장된 통신 개념으로 접근하고 있다. 이러한 이유로 MPI 통신과 유사하게 MPI-IO도 독립(Independent) IO와 집합(Collective) IO, 블록(Block) IO와 비블록(Nonblock) IO, 연속(Contiguous) IO와 불연속(Noncontiguous) IO 등으로 구분되고 있다[2].

병렬 프로그램에서 파일 I/O 수행 시 확장성을 보장하기 위해서는 프로세스의 수와 파일 구조간의 의존성을 없애주어야 하고 단일 파일 기반의 병렬 I/O가 이를 지원한다. 또한 과학 기술용 병렬 프로그램의 경우, 풀고자 하는 전체 문제 영역을 각 프로세스가 쪼개어 처리하는 경우가 많은데, 이러한 경우 불연속적인 I/O를 파일 시스템에 요청하게 되어 I/O의 성능 저하를 초래한다. 이러한 불연속적인 요청들을 좀 더 연속적인 I/O로 바꾸어 처리하기 위하여 대부분의 MPI-IO 라이브러리는 I/O 요청을 데이터 교환 단계와 I/O 단계로 나누어 처리하는 2단계 I/O(two-phase I/O)방식을 취하고 있다[1].

3. 로그 생성을 고려한 MPICH 수정

앞서 설명한 바와 같이 MPI-IO는 내부적으로 여러 단계로 나뉘어 수행되고 있으며 이들 중 일부는 기존의 MPI통신 함수들로 구성되어 있다. 본 연구의 목적은 이와 같은 MPI-IO의 내부 루틴들이 어떤 분포로 실행 시간에 수행되는지를 확인하는데 있다. 이를 위하여 MPI-IO가 수행되는 과정을 기록할 필요가 있었고 본 장에서는 그 과정을 설명한다.

많이 알려진 프로파일링 도구로 TAU가 있다[3]. MPI도 지원하는 유용한 도구이나 기본적인 통신 함수를 대상으로 하고 있어서 MPI-IO와 같이 MPI 함수들이 복합적으로 구성된 경우에는 TAU의 사용에 제약이 따른다.

```
#ifdef ADIOI_MPE_LOGGING
  MPE_Log_event( ADIOI_MPE_WriteContig_a, 0, NULL );
#endif

  ADIO_WriteContig(fd, buf, count, datatype,
        ADIO_EXPLICIT_OFFSET, off, status, error_code);

#ifdef ADIOI_MPE_LOGGING
  MPE_Log_event( ADIOI_MPE_WriteContig_b, 0, NULL );
#endif
```

(그림 1) MPE 로그 사용을 위한 MPI-IO 소스 코드 수정 예

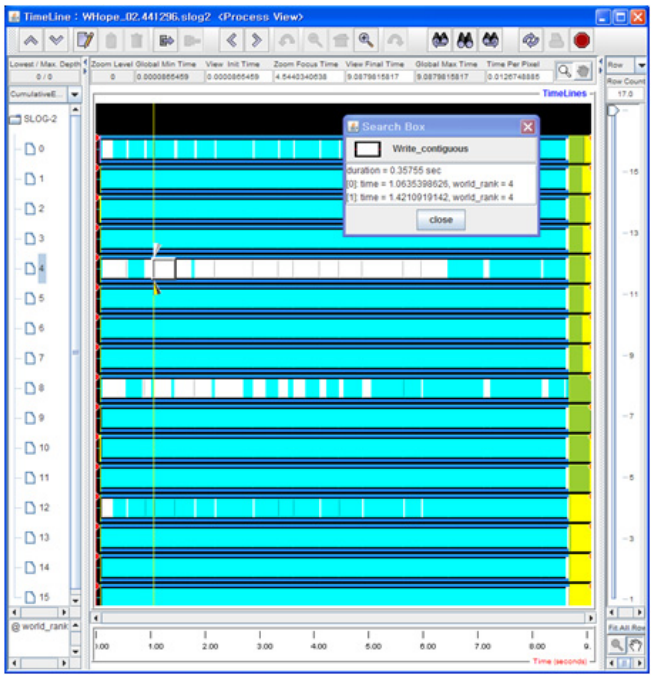
<표 1> 집합 I/O의 주요 내부 루틴

ADIOI_Calc_my_off_len	Collective I/O에 사용될 데이터의 파일에서의 Offset과 데이터 길이 계산	
ADIO_Calc_file_domains	IO Aggregator 노드들의 경우, 담당할 File Domain을 계산	
ADIO_Calc_my_req	각 프로세스들이 요청한 I/O 데이터가 어느 I/O Aggregator에 의해서 처리되고, I/O Aggregator는 어느 프로세스의 I/O를 처리해야 하는지 확인	
ADIO_Calc_others_req		
ADIO_Exch_and_write	ADIOI_W_Exchange_data	쓰고자 하는 데이터를 교환
	ADIO_WriteContig	쓰기 수행
ADIO_Read_and_exch	ADIO_ReadContig	읽기 수행
	ADIOI_R_Exchange_data	읽은 데이터를 교환

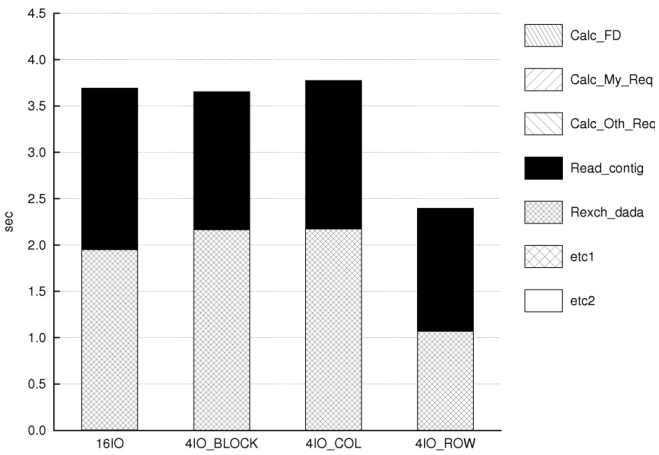
이러한 이유로 우리는 MPICH를 개발하는 과정의 문제 해결 목적으로 활용된 MPE라는 프로파일링 도구를 이용하였다[4]. MPI-IO의 주요 루틴 중에서 실행 시간을 측정하고자 하는 루틴의 소스 코드에 그림 1과 같이 MPE 로그 이벤트를 추가 한다. 본 연구에서는 표 1과 같이 MPI-IO중 집합 I/O의 주요 루틴을 대상으로 로그 이벤트를 추가하였다. 이외 몇 개의 헤더 파일에 사용된 이벤트를 정의하고 MPICH 라이브러리를 다시 컴파일한다. 이후 MPI-IO를 사용하는 MPI 어플리케이션을 이 MPICH 라이브러리를 이용하여 컴파일하여 사용하게 되는데 이때 MPE 기능을 활성화시켜야 한다.

4. MPI-IO 로그 분석

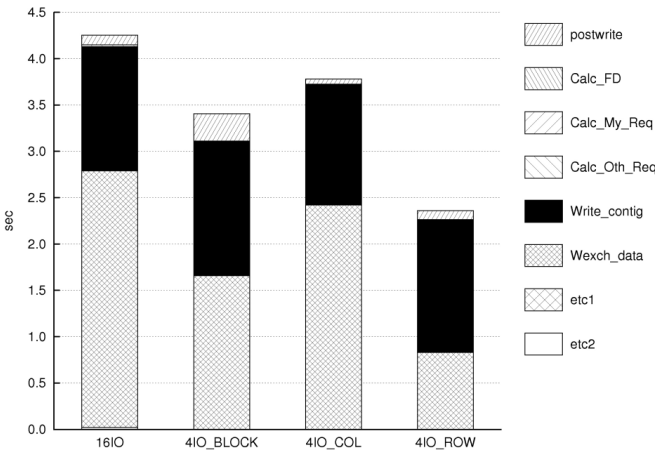
앞 장에서 기술된 방법으로 MPICH 라이브러리를 재생성하고 이를 이용하여 MPI-IO를 사용하는 프로그램을 컴파일하고 실행하면 각 루틴의 실행 시간이 기록된 이벤트 로그를 확보할 수 있다. 이를 가시화하기 위한 툴로 JAVA 언어로 작성된 Jumpshot이라는 프로그램을 활용할 수 있다[4]. 그림 2는 16개의 프로세스가 4개의 I/O Aggregator를 사용하여 4개의 OST를 사용하는 파일 시스템에 1GB 파일을 기록한 경우의 로그를 Jumpshot을 이용하여 확인한 것이다.



(그림 2) Jumpshot을 이용한 집합 I/O 로그 출력



(a) 1GB 파일 읽기 (4 OST 사용)



(b) 1GB 파일 쓰기 (4 OST 사용)

(그림 3) 집합 I/O 수행시 내부 루틴별 수행시간 분포

Jumpshot이 그래픽을 이용하여 MPI-IO의 수행 과정을 추적하기에는 용이한 환경을 제공하고 있으나 각 루틴의 누적 실행 시간과 같은 통계 값을 확인하기에는 제약이 있었다. 이에 생성된 이벤트 로그 파일을 가공하여 이벤트 별 통계 데이터를 구하는 작업을 추가로 진행하였다. 그림 3은 그림 2와 같은 조건의 쓰기와 읽기의 경우를 이상의 모든 과정을 거쳐 도출한 주요 루틴의 실행 시간 통계이다.

위의 결과와 같이 집합 I/O의 실행에는 여러 루틴들이 관여하고 있었으나 데이터 교환(W_Exch_data와 R_Exch_data)부분과 I/O 루틴(Write_contig와 Read_contig)부분이 집합 I/O의 시간을 결정하는 주요한 부분임을 확인 할 수 있었다.

5. 결론

본 연구에서는 단일 파일 기반의 병렬 I/O를 수행하는 MPI-IO의 실행 특성을 파악하기 위하여 MPI-IO 라이브러리를 수정하여 로그를 생성하였다. 과학 기술용 응용 프로그램의 특성상 발생되는 불연속적인 I/O 처리를 연속된 I/O로 바꾸기 위하여 MPI-IO 내부에 여러 루틴들이 존재하지만 프로파일링 결과는 데이터 교환과 I/O 단계가 실행 시간을 결정하는 주요 루틴들임을 보여 주었다. 이는 I/O 단계뿐만 아니라 데이터 교환 시간을 최적화하여 MPI-IO의 성능을 개선시킬 수도 있다는 것으로 해석되며, 향후 데이터 교환 시간을 단축시켜 MPI-IO의 성능을 향상시키는 연구가 필요할 것으로 예상된다.

참고문헌

[1] Kwangho Cha, “An Efficient I/O Aggregator Assignment Scheme for Multi-core Cluster Systems,” in IEICE Transactions on Information and Systems, vol. E96-D, no. 2, pp. 259~269, Feb. 2013.

[2] William Gropp, Ewing Lusk, and Rajeev Thakur, “Using MPI-2: Advanced Features of the Message Passing Interface,” MIT Press, 1999

[3] TAU-Tuning and Analysis Utilities, http://www.cs.uoregon.edu/Research/tau/home.php, Accessed 22 Feb. 2013.

[4] The MPICH wiki, http://wiki.mpich.org/mpich/index.php/Main_Page, Accessed 21 Feb. 2013.