ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

병렬 프로그램에서의 효율적인 대용량 파일 입출력 방식의 비교 연구☆

Research for Efficient Massive File I/O on Parallel Programs

황 규 현[1] 김 영 태[2*]
Gyuhyeon Hwang Youngtae Kim

요 약

분산 메모리형의 병렬 프로그램에서는 프로세서들이 독립적으로 입출력을 처리하기 때문에 여러 유형의 파일 입출력 방식이 사용된다. 본 논문에서는 분산 메모리형 병렬 프로그램에서의 대용량 파일에 대한 효율적인 입출력 방식을 알아보기 위하여 다양한 방식을 구현하고 비교 분석하였다. 구현된 방식으로는 (i) NFS를 활용한 병렬 입출력 방식, (ii) 호스트 프로세서에서의 순차 입출력과 도메인 분산 방식, 그리고 (iii) 메시지 전송 전용 입출력(MPI-IO) 방식 등이 있다. 성능 분석을 위해서 별도의 파일 서버를 사용하였으며 한 대 및 두 대의 계산 클라이언트에서 다중 프로세서를 사용하였다. 비교 분석 결과, 입력의 경우에는 NFS 병렬 입력 방식이, 출력의 경우에는 도메인 전송을 통한 순차 출력 방식이 가장 효율적으로 나타났으며, 예상과는 다르게 메시지 전송 전용 입출력 방식의 성능이 가장 낮게 나왔다.

☞ 주제어 : 병렬 입출력, 집합 입출력, 분산 메모리 컴퓨터, MPI-IO, NFS

ABSTRACT

Since processors are handling inputs and outputs independently on distributed memory computers, different file input/output methods are used. In this paper, we implemented and compared various file I/O methods to show their efficiency on distributed memory parallel computers. The implemented I/O systems are as following: (i) parallel I/O using NFS, (ii) sequential I/O on the host processor and domain decomposition, (iii) MPI-IO. For performance analysis, we used a separated file server and multiple processors on one or two computational servers. The results show the file I/O with NFS for inputs and sequential output with domain composition for outputs are best efficient respectively. The MPI-IO result shows unexpectedly the lowest performance.

☞ keyword : Parallel I/O, Collective I/O, Distributed memory computer, MPI-IO, NFS

1. 서 론

일반적으로 병렬 컴퓨터에서의 파일 입출력의 성능은 프로그램의 실행 시에 빈도수가 많지 않기 때문에 계산의 성능에 비교하여 덜 중요하게 여겨 왔다. 하지만 병렬 컴퓨터의 성능 향상으로 인하여 계산 시간이 획기적으로 단축됨에 따라 파일 입출력 성능이 전체 성능에 있어서 병목이 될 수 있기 때문에 파일 입출력의 성능 개선은 점차 중요하게 인식되고 있다[1,2]. 더욱이 근래에 들어서 다양한 분야에서 주목을 받고 있는 빅 데이터의 처리 및 실시간으로 대용량의 파일 입출력을 처리하는 고성능 컴퓨팅의 경우에는 입출력 성능의 중요성은 더욱 증대하게 되었다[3].

본 논문에서는 분산 메모리형 병렬 컴퓨터에서 효율적인 파일 입출력 방식을 알아보기 위하여 다양한 입출력 방식을 구현하여 비교 분석하였다. 분산 메모리형 병렬 컴퓨터는 가장 대표적인 병렬 컴퓨터의 형태로서 병렬로 계산하는 각 프로세서가 분산된 해당 메모리에만 접근이 가능하다. 따라서 병렬로 계산하기 이전에 계산하고자 하는 전체의 도메인을 분할하여 서브 도메인의 형태로서 각 프로세서의 메모리로 분산하며 필요에 따라서는 계산을 마친 후 분산되어진 서브 도메인을 다시 전체의 도메인으로 통합을 한다. 한편 네트워크로 연결된 프로세서들이 동일한 파일에 접근하기 위하여 NFS(Network File System) 등의 공유 파일 시스템을 사용하여 파일 입출력 시스템

1 IT Division, U2Bio Co., 68 Geomiro Songpa, Seoul, Korea
2 Department of Computer Engineering, Gangneung-Wonju National University, 150 Namwonno, Wonju Gangwon, 26403, Korea
* Corresponding Author (ykim@gwnu.ac.kr)
[Received 13 October 2016, Reviewed 17 October 2016, Accepted 8 January 2017]
☆ 본 논문은 강릉원주대학교 대학원 전일제장학금으로 지원된 논문입니다.

을 구성한다. 공유 파일 시스템을 통하여 연결하게 되면 모든 프로세서는 네트워크로 접근 가능한 파일을 지역 파일과 동일한 방식으로 접근할 수가 있다. 따라서 모든 프로세서들이 동일한 파일로부터 해당 영역만을 분산된 메모리로 입출력하게 되면 병렬 입출력이 가능하게 된다. 이에 본 논문에서는 효율적인 파일 입출력을 알아보기 위하여 NFS 환경에서 순차 및 병렬 입출력 프로그램과 메시지 전송 방식 라이브러리인 MPI(Message Passing Interface) 전용 입출력 함수들을 이용한 프로그램을 구현하고 비교 분석하였다.

본 논문에서의 구현 방식으로는 (i) NFS 상에서의 하나의 파일에 연결된 다중 프로세서들의 병렬 입출력 방식, (ii) 호스트 프로세서에서의 순차적인 입출력 후에 MPI 함수를 이용하여 다른 프로세서로 송수신하는 방식 그리고 (iii) MPI 라이브러리에서 제공하는 병렬 입출력 전용 함수의 사용 등이 있다. 프로그램에서 계산 도메인은 계산의 형태에 따라 1차원 혹은 2차원 이상의 다차원으로 구분될 수 있다. 본 논문에서는 병렬 컴퓨터에서 2차원 형태의 프로세서의 배열이 분산된 프로세서간의 통신 등에 있어서 가장 효율적이며 3차원 이상의 도메인은 2차원 프로세서의 배열에 투영하여 사용할 수 있기 때문에[4,5] 프로세서의 2차원 배열에 대해서만 설명한다.

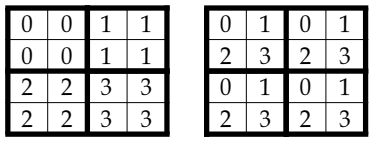
본 논문의 2장에서는 병렬 프로그램에서의 파일 입출력의 구현에 대해 설명하고, 3장에서는 성능의 비교 분석, 그리고 4장의 결론으로 구성된다.

2. 분산 메모리 병렬 프로그램에서의 파일 입출력 방식

이 장에서는 먼저 분산 메모리형의 병렬 프로그램에서 사용하는 도메인 분산 방식을 알아보고 이를 기본으로 하는 3가지의 병렬 입출력 방식에 대하여 설명한다.

2.1 병렬 프로그램에서의 도메인 분산

분산 메모리형 병렬 컴퓨터에서는 각각의 프로세서와 그에 대한 메모리가 분산되어 있다. 따라서 병렬로 계산하기 위해서는 계산하고자 하는 전체의 도메인을 나누어 각 프로세서의 메모리로 분산하여야 한다. 도메인 분산은 일반적으로 다차원의 형태로 이루어지며, 본 논문에서는 프로세서의 2차원 구조가 효율적으로 많이 사용되기 때문에 2차원 형태의 도메인 분산을 위주로 설명한다.



(a) local 분산 (b) scattered 분산

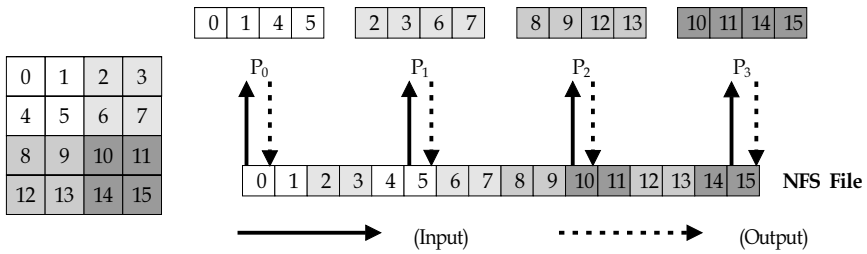
(그림 1) 병렬처리에서의 도메인 분산 방법
(Figure 1) Domain decomposition for parallel processing

도메인 분산 방식은 도메인을 각 프로세서로 분할하여 서브 도메인으로 분산하는 방식(local)과 도메인을 분할한 후에 서브 도메인을 전체 프로세서가 계산하는 방식(scattered)이 있다[6]. (그림 1)은 도메인 분산 방식을 설명한다. 그림에서는 도메인을 2×2로 분산하였으며 숫자는 프로세서의 개별 번호를 나타낸다. 도메인 분산 방식 중 scattered 분산 방식은 각 프로세서가 도메인의 일부분을 차지하기 때문에 계산이 진행되면서 도메인의 크기가 작아지는 LU 분해법[6,7] 등의 수치 함수 등에서는 많이 사용된다. 하지만 대량 계산을 하는 수치 모델 등에서는 도메인을 각 프로세서에 분할하는 local 분산 방식을 주로 사용하기 때문에 본 논문에서는 local 분산 방식만을 사용하였다.

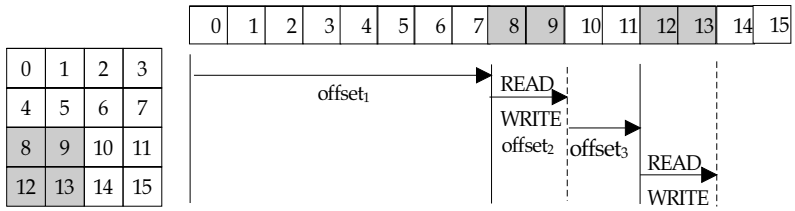
2.2 NFS 상에서의 병렬 입출력 방식

분산 메모리형 병렬 컴퓨터에서는 각 프로세서가 동일한 파일로부터 입출력을 실행하기 위하여 통신에 의존할 수밖에 없다. 이러한 통신을 쉽게 해결하기 위한 방법으로는 네트워크를 이용하여 하나의 파일을 공유하는 방법이 있다. 가장 많이 사용하는 방식으로는 NFS의 구성으로서 NFS를 통해 연결된 프로세서들은 네트워크를 통하여 하나의 가상 파일을 직접 연결된 지역 파일처럼 사용할 수 있기 때문에 각각의 프로세서가 동시에 논리적인 입출력을 실행할 수 있다[8].

(그림 2)는 NFS 상에서 병렬 입출력 방식을 나타낸다. 그림에서 P_i는 사용되는 프로세서를 나타내며 4개의 프로세서들이 NFS를 통하여 하나의 파일을 공유하며 지역 파일을 동시에 입출력하는 예의 형태를 보여준다. 각 프로세서들의 입출력 해당 부분은 다른 색으로 구분하였다. 왼쪽 그림은 파일에서 해당 데이터의 위치를 2차원으로 표현한 것이며 오른쪽은 1차원으로 다시 표현한 것이다.



(그림 2) NFS 상에서 프로세서들이 병렬로 입출력을 처리하는 방식
(Figure 2) Parallel I/O of processors on NFS



(그림 3) 병렬 파일 입출력을 위한 파일 포인터에 대한 예
(Figure 3) Example of file pointers for parallel I/O

이 방식에서는 각 프로세서가 파일에서의 해당 데이터에 접근하기 위하여 파일 포인터를 사용한다. (그림 3)은 (그림 2)에서의 P_2가 왼쪽 그림의 회색 데이터에 접근하기 위한 파일 포인터 계산의 예를 보여 준다. (그림 2)에서와 같이 파일은 1차원으로 저장이 되기 때문에 포인터 값의 계산을 위해서는 1차원의 데이터를 사용하였다. 그림에서 화살표는 파일 포인터의 이동 방향과 거리를 나타내며 $offset_i$은 각 이동 구간에 대한 값들이다.

파일 포인터의 이동을 위하여 fseek() 함수가 사용되며 feek() 함수 사용에 필요한 offset 값들은 아래의 식과 같이 계산된다. 아래의 식에서 도메인의 가로와 세로는 i와 j로 표현하였으며 크기는 max이다. 따라서 이 예에서는 imax와 jmax의 값은 각 4이다. 또한 각 프로세서에서의 분산된 서브 도메인의 지역 인덱스에 대해서는 시작을 begin으로 끝을 end로 표현하였다. 이 예에서 P_2에서의 ibegin과 iend는 0, 1 그리고 jbegin과 jend는 2, 3이 된다.

다음 프로그램은 모든 프로세서에서의 파일 포인터를 사용한 병렬 입출력 함수이다. 프로그램을 그림 3에 적용하면 $offset_1$의 값은 8(=2×4), $offset_2$의 값은 0, $offset_3$의 값은 3(=4-1)이다.

```
offset1 = jbegin*imax;
offset2 = ibegin;
offset3 = imax-(iend+1);
fseek(offset1);
for (j=0; j<(jend-jbegin); j++) {
    fseek(offset2);
    for (i=0; i<(iend-ibegin); i++) {
        read(sub_domain(i,j));
    }
    fseek(offset3);
}
```

이 방식은 모든 프로세서가 별도의 병렬 함수를 사용하지 않는 장점이 있으며 입력시에는 모든 프로세서가 동시에 처리할 수 있기 때문에 성능 향상을 기대할 수 있다. 하지만 출력의 경우 하나의 파일 처리에 프로세서들의 충돌을 피하기 위하여 프로세서가 가상의 파일을 사용하기 때문에 동기화의 문제가 발생한다. 이를 위하여 프로세서들이 출력을 위한 순서를 정해야하기 때문에 성능이 비효율적일 수 있다.

2.3 순차 입출력과 MPI 전송 함수를 이용한 방식

이 방식은 호스트 프로세서만 입출력을 순차적으로 처리하며 병렬처리 라이브러리 MPI[9,10]의 전송 함수를 사

용하여 다른 프로세서로 분산된 서브 도메인을 송신하는 방식이다. 프로세서별로 분산되는 서브 도메인의 크기가 균일하지 못한 경우가 발생하기 때문에 p2p 데이터 송수신 함수인 MPI_Send()와 MPI_Recv()를 사용하였다. (그림 4)는 순차 입출력과 MPI 함수를 이용한 방식을 설명한다. 각 프로세서에서의 분산된 서브 도메인은 다른 색으로 표현하였다. 입력의 경우 호스트 프로세서가 먼저 전체 도메인을 읽은 후 각 프로세서로 분산된 서브 도메인을 전송한다. 출력의 경우에는 이와 반대로 호스트를 제외한 프로세서에서 서브 도메인을 호스트 프로세서로 보내고 호스트 프로는 전체 도메인으로 통합하여 출력한다.

이 방식은 분산 메모리형 병렬 컴퓨터에서 구현하기가 비교적 쉽지만 호스트 프로세서에서는 전체 도메인을 할당하게 되어 메모리의 낭비가 발생하며, 호스트 프로세서가 입출력을 실행하는 동안에 다른 프로세서는 대기하게 되는 경우가 발생하게 된다. 또한 도메인의 분산과 통합을 위하여 호스트 프로세서와 다른 프로세서들은 순서대로 통신을 진행하기 때문에 해당 프로세서가 통신을 실행하는 동안 나머지 프로세서는 대기하는 비효율성이 발생한다.

2.4 MPI 전용 입출력 함수를 이용한 방식

MPI 전용 입출력 함수를 이용한 병렬 입출력 방식은 2.2절의 NFS 방식과 같이 공유 파일을 사용하는 방식이지만 파일 포인트를 사용하는 대신에 MPI에서 제공하는 입출력 전용 함수를 사용한다[11]. 아래 프로그램은 MPI 함수를 사용하는 기본 방식을 보여 준다.

```
MPI_File_open(MPI_COMM_WORLD, ...);
MPI_File_seek(offset, MPI_SEEK_CUR, ...);
MPI_File_Read();
MPI_File_close();
```

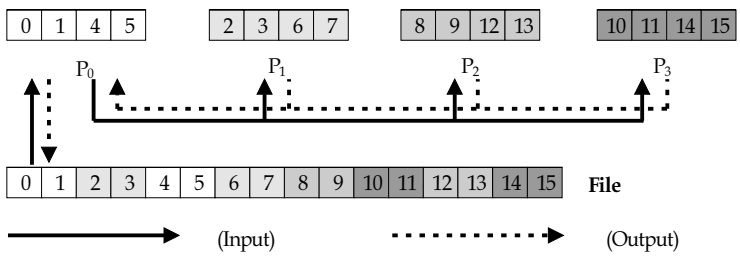
이 프로그램은 2.2절에서의 순차적인 파일 처리 방식과 유사하지만 하나의 공통 파일을 사용하기 위하여 파일을 열 때 모든 프로세서들의 참여를 의미하는 MPI_COMM_WORLD 상수를 사용하며, 파일 포인터 이동에 대한 상수로서 MPI_SEEK_CUR을 사용한다. offset 값의 계산 등은 2.2절에서의 방식과 동일하다. 이 방식으로 구현할 때도 마찬가지로 offset 값의 계산이 필요하지만 병렬 출력 시에 발생하는 동기화의 문제는 MPI 라이브러리에서 자동으로 해결이 된다[11]. 따라서 병렬 출력의 경우에는 좋은 성능을 기대 할 수 있다. 한편 출력이 이진(binary) 파일로만 가능하기 때문에 ASCII 파일로 변환하기 위해서는 별도의 프로그램을 사용해야 한다[11].

3. 성능 분석

이 장에서는 2장에서 설명한 입출력 방식들의 성능을 비교 분석하였다. 성능 분석은 입력과 출력으로 나누어서 비교하였다.

3.1 실험 환경

본 논문에서 성능 측정을 위한 계산 환경은 표 1과 같다. 병렬 컴퓨터는 파일 서버 1대와 계산 서버 2대의 클러스터로 구성되어 있다. 파일 서버는 6개 코어의 Intel Xeon E5-2602 2.0GHZ CPU 2개와 32GB의 메모리로 구성되며, 계산 서버는 각각 8개 코어의 Intel Xeon E5-2687W 3.4GHz CPU 2개와 32GB의 메모리로 구성되어 있다. 병렬 프로그램 라이브러리는 MPICH 3.1.3을 사용하였고 모든 컴퓨터는 기가비트(Gigabit) ethernet으로 연결되어 있다.



(그림 4) 순차 입출력과 MPI 전송 함수를 이용한 방식
(Figure 4) Sequential I/O and MPI functions

(표 1) 시스템 사양
(Table 1) System specification

File server	OS	Linux CentOS 6.6(kernel-2.6.32)
	CPU	Intel Xeon E5-2620 2.0GHz
	Cach Memory	L1 : 192KB, L2 : 1536KB, L3 : 15360KB
	#CPU	2(6 cores per CPU)
	Memory / Swap	32GB / 16GB
	Network	Ethernet(1GB/s)
Computa-tional server	OS	Linux CentOS 6.6(kernel-2.6.32)
	CPU	Intel Xeon E5-2687W 3.4GHz
	Cach Memory	L1 : 512KB, L2 : 2MB, L3 : 25MB
	#CPU	2(8 cores per CPU)
	Memory / Swap	32GB / 16GB
	Network	Ethernet(1GB/s)

입출력 비교를 위하여 동일한 8×10^7개의 32 비트의 실수 데이터를 사용하였다. 이 크기는 다중 프로세서로의 분산이 용이하며 여러 번의 비교 시에도 편차가 가장 적은 경우의 최대 크기이다.

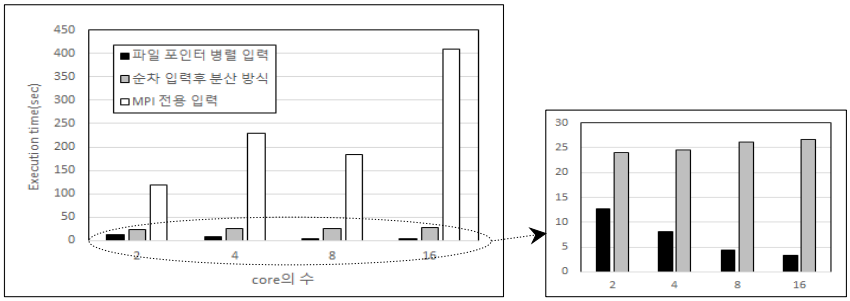
3.2 입력 방식의 성능 비교

(그림 5)와 (그림 6)은 구현한 방식들 중 입력에 대한 실행 시간을 보여준다. (그림 5)는 한 대의 계산 서버에서 총 16개의 코어(이하 프로세서로 표기)를 사용한 결과이다. 그림에서 모든 프로세서가 다른 파일 포인터를 사용하여 입력 함수를 동시에 실행한 경우가 가장 뛰어난 성능을 보였다(그림에서 오른쪽에서 별도 비교). 또한 이 경우에는 프로세스 수가 증가할수록 병렬성이 증가하기 때문에 실행 시간이 감소한 것을 볼 수 있다. 한편 하나의 호스트에서 순차적으로 입력을 한 후에 다른 프로세서로 서브도메인을 분산하는 방식도 비교적 효율적인 성능이 나왔는데 이는 통신 시간의 지연에 따른 부하가 파일 입력과 비교하여 비교적 적기 때문으로 분석된다. 오른쪽의 그림에서 프로세서의 수가 증가할수록 미세하게나마 시간이 더 걸리는 이유는 통신 데이터의 양은 적어지지만 통신 시간의 부하가 좀 더 커지기 때문으로 보인다. MPI 전용 입력 함수를 사용한 경우에는 포인터를 사용한 병렬 입력 방식에 비교하여 9배~180배 정도의 낮은 성능 차이를 보였다. 프로세서의 수가 8개일 경우가 4개의 경우보다 시간이 작은 부분도 발견되었는데 이는 MPI에서 하나의 공통 파일에 대한 입출력 관련 버퍼 크기가 상충되는 문제로 보인다[11].

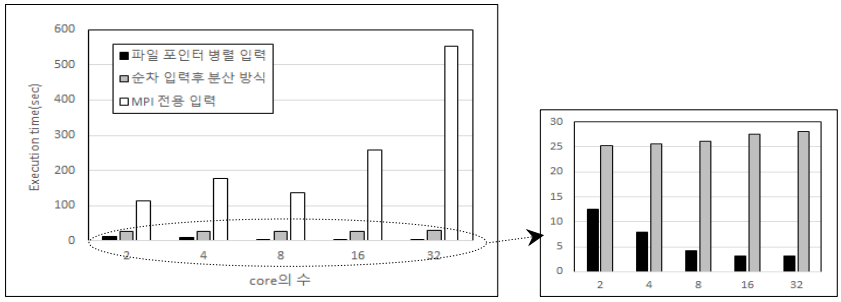
(그림 6)은 두 대의 계산 서버를 사용하여 측정한 결과를 보여 준다. 사용한 프로세서의 수는 두 대에 균등하게 배분을 하였다. 그림에서 성능은 동일한 수의 프로세서를 사용했을 경우 두 대의 서버를 사용한 경우가 한 대의 서버를 사용한 경우보다 비슷하거나 조금 느리게 나타났다. 그러나 MPI 전용 입력 함수를 사용한 경우에는 두 대의 서버를 사용한 경우가 한 대의 서버를 사용한 것 보다 평균 1.3배 정도 빠르게 나타났다. 이 경우에는 두 대의 서버를 사용할 경우에 통신 등에 있어서 MPI를 사용하는 것이 더 효율적이기 때문으로 보인다.

3.3 출력 방식의 성능 비교

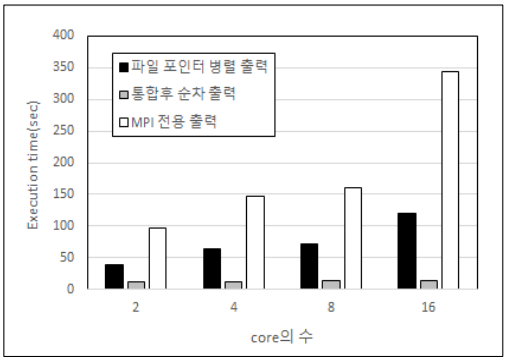
출력 방식은 입력 방식과 동일한 데이터를 사용하여 동



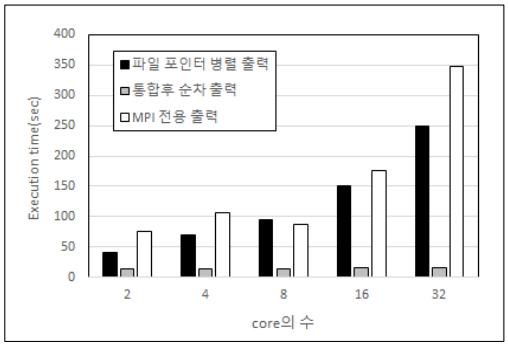
(그림 5) 한 대의 계산 서버를 사용한 파일 입력의 성능
(Figure 5) File input performance using 1 computational server



(그림 6) 두 대의 계산 서버를 사용한 파일 입력의 성능
(Figure 6) File input performance using 2 computational servers



(그림 7) 한 대의 계산 서버를 사용한 파일 출력의 성능
(Figure 7) File output performance using 1 computational server



(그림 8) 두 대의 계산 서버를 사용한 파일 출력의 성능
(Figure 8) File output performance using 2 computational servers

일한 방식으로 비교 분석하였다. (그림 7)은 한 대의 계산 서버를 사용한 결과이다. 출력의 경우에는 입력과는 다르게 호스트에서의 분할 도메인의 호스트 전송 및 순차 출력 방식이 가장 효율적인 성능을 보였다. NFS 상에서의 병렬 출력의 경우에는 입력과는 다르게 여러 프로세서가 접근하는 것이 동시에 하나의 파일에 물리적으로 불가능하여 순차적으로 처리될 수밖에 없기 때문에 성능이 낮게 나온 것으로 분석된다. 한편 출력의 경우에도 MPI에서 제공하는 전용 병렬 출력 함수를 사용하는 방식이 가장 낮은 성능을 보였다. 이 방식은 가장 빠른 순차 출력과 비교하여 5배~22배 정도의 낮은 성능 차이를 보였다.

(그림 8)은 두 대의 계산 서버를 사용하여 측정한 결과를 보여 준다. 두 대의 서버를 사용한 경우 한 대의 서버를 사용한 경우와 유사한 성능을 보인다. 입력 방식과 마찬가지로 두 대의 서버를 사용한 경우가 대부분 조금 느린 실행시간을 보였고 MPI 전용 출력 함수를 사용한 방식만은 빠른 성능을 보였으며 입력 방식과 동일한 이유로 분석된다.

4. 결 론

본 논문에서는 분산 메모리형의 병렬 컴퓨터에서 효율적으로 실행이 될 수 있는 파일 입출력 방식을 사용하여 프로그램을 구현하고 비교하였다. 비교된 방식으로는 공유 파일 시스템에서 하나의 파일에 연결된 다중 프로세서들의 병렬 입출력 방식, 하나의 프로세서에서의 순차 입출력과 MPI 전송 함수를 이용하여 다른 프로세서로 송수신하는 방식, 그리고 MPI 라이브러리에서 제공하는 전용 병렬 입출력 함수를 사용하였다.

성능 비교 결과, 입력에 있어서는 공유 파일 시스템에서 모든 프로세서들이 파일 포인터를 사용하여 해당 영역만을 입력하는 병렬 입력 방식이 가장 효율적이었으며, 출력에 있어서는 각 프로세서로 분산된 서브도메인을 호스

http://dx.doi.org/10.7472/jksii.2014.00.00

트 프로세서로 통합하여 순차 출력하는 방식이 가장 효율적으로 나타났다. 입력과 출력의 방식의 효율성이 다르게 나타나는 이유는 파일에 병렬로 물리적으로 동시에 출력하는 것이 실제로는 불가능하기 때문이며, 이에 가시적으로 순차적인 출력이 더 효율적으로 나타났다. 한편 예상과는 다르게 MPI 라이브러리에서 제공하는 전용의 병렬 입출력 함수는 사용의 편리함에 비하여 성능의 효율성은 낮게 나타났다. 이는 다양한 계산 환경을 제공하는 범용의 함수이기 때문으로 보이지만 성능 개선의 필요성이 있다고 보여 진다.

병렬처리에서 파일 입출력은 전체 성능에서의 비중은 낮지만 Amdahl의 법칙 - 성능의 개선은 낮은 일부의 성능이 전체적인 성능에 영향을 미친다 - 에 따르면 그 중요성은 지대해진다. 따라서 본 연구에서 제안한 효율적인 병렬 입출력 방식을 사용하여 고성능 컴퓨팅 분야에서 계산 성능의 개선에 도움이 되기를 기대한다.

참고문헌(Reference)

[1] K. Cha and H. Cho, "An Analysis of Collective Buffer Effects on Collective I/O", J. of KIISE: Computing Practices and Letters, Vol. 19, No. 4, pp. 214-218, Apr. 2013. http://ksci.kisti.re.kr/browse/brow Detail.ksci?kojic=&browseBean.totalCnt=11&browseBean.atclMgntNo=JBGHIF_2013_v19n4_214&browseBean.curNo=9

[2] S. Park, "Benchmark for Performance Testing of MPI-IO on the General Parallel File System", J. of IPS, Vol. 8-A, No. 2, pp. 125-132, June. 2001. http://journals.kips.or.kr/digital-library/kipsa/6167

[3] Hashema. I., I. Yaqooba, N. Anuara, S. Mokhtara, A. Gania and S. Khanb, "ig data" on cloud computing: Review and open research issues", Information Systems: 47, pp. 98-115. ACM, 2009. http://www.sciencedirect.com/science/article/pii/S0306437914001288

[4] Y. Kim, "High Performance Computing Classes (HPCC) for Parallel Fortran Programs using Message Passing", J. of KIISE Vol. 38, No. 2, pp. 59-66, Apr. 2011. http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=JBGHG6_2011_v38n2_59

[5] Y. Kim, Y. Choi, and H. Park,"GP-GPU based Parallelization for Urban Terrain Atmospheric Model CFD_NIMR", J. of ICS, Vol. 12, No. 2, pp. 41-47, Apr. 2014. http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=OTJBCD_2014_v15n2_41

[6] G. C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, "Solving Problems On Concurrent Processors Volume I: General Techniques and Regular Problem". Englewood Cliffs: Prentice-Hall, 1998. http://dl.acm.org/citation.cfm?id=43389

[7] Y. Kim, "Performance Comparison of Two Parallel LU Decomposition Algorithms on MasPar Machines", Journal of IEEE Korea Council, Vol. 2, No. 2, pp. 247-255, 1999. http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=JGGJB@_1998_v2n2s3_247

[8] C. Lever and P. Honeyman, "Linux NFS Client Write Performance", Proceeding of USENIX 2002 Annual Technical Conference, pp. 29-40. 2002. http://www.citi.umich.edu/projects/nfs-perf/results/cel/write-throughput.html

[9] "MPI: A Message-Passing Interface Standard Version 3.1", Message Passing Interface Forum, June 2015. http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[10] W. Gropp, E. Lusk, and R. Thakur. "Using MPI-2 - Advanced Features of the Message Passing Interface". Massachusetts Institute of Technology, second edition, 1999. http://www.mcs.anl.gov/research/projects/mpi/ usingmpi2

[11] P. Dickens, and J. Logan, "Y-lib: a user level library to increase the performance of MPI-IO in a lustre file system environment", Proceedings of 2009 ACM International Symposium on High Performance Distributed Computing (HPDC'09), pp. 32-38. ACM, 2009. https://pdfs.semanticscholar.org/dc01/47fe0aa01f06dbc7df8b2046be23daf5e18d.pd

◐ 저 자 소 개 ◑

황 규 현

2014년 강릉원주대학교 컴퓨터공학과 졸업(학사)
2016년 강릉원주대학교 컴퓨터공학과 졸업(석사)
2016년~현재 ㈜유투바이오 기업부설연구소(IT)
관심분야 : 컴퓨터 시스템, 초고속컴퓨팅
E-mail : ghhwang@u2bio.co.kr

김 영 태

1986년 연세대학교 수학과 졸업(학사)
2001년 아이오와주립대학교 대학원 컴퓨터과학과 졸업(석사)
2006년 아이오와주립대학교 대학원 컴퓨터과학과 졸업(박사)
1989년~현재 강릉원주대학교 컴퓨터공학과 교수
관심분야 : 병렬처리, 초고속컴퓨팅
E-mail : ykim@gwnu.ac.kr