지구물리와 물리탐사 Geophysics and Geophysical Exploration
Vol. 19, No. 3, 2016, p. 131~135 http://dx.doi.org/10.7582/GGE.2016.19.3.131

ISSN 1229-1064 (Print)
ISSN 2384-051X (Online)

코어레이와 MPI를 이용한 병렬 파동 전파 모델링과 거꿀 참반사 보정 성능 비교

류동현 · 김아름 · 하완수*

부경대학교 에너지자원공학과

A Performance Comparison between Coarray and MPI for Parallel Wave Propagation Modeling and Reverse-time Migration

Donghyun Ryu, Ahreum Kim, and Wansoo Ha*

Department of Energy Resources Engineering, Pukyong National University

요 약: 코어레이는 포트란 2008 표준에 도입된 병렬 연산 기법이다. 코어레이를 이용하면 간단한 문법으로 분산 메모리 시스템에서 병렬 연산을 구현할 수 있다. 본 연구에서는 탄성파 자료 처리 프로그램에 코어레이와 MPI를 적용하여 병렬 처리 성능을 비교하고 이를 통해 코어레이의 적용 가능성을 살펴보았다. 파동 전파 모델링을 이용해 연산 성능을 비교하였고, 영역 분해 기법을 이용해 일대일 통신 성능을 비교하였다. 또한 거꿀 참 반사 보정 프로그램을 이용해 병렬 처리 성능을 비교하였다. 그 결과 연산 성능은 코어레이 프로그램과 MPI 프로그램에서 큰 차이가 없었지만 통신 성능은 MPI가 우수했다.

주요어: 코어레이, 포트란, 병렬 처리, 파동 전파 모델링, 거꿀 참 반사 보정

Abstract: Coarray is a parallel processing technique introduced in the Fortran 2008 standard. Coarray can implement parallel processing using simple syntax. In this research, we examined applicability of Coarray to seismic parallel processing by comparing performance of seismic data processing programs using Coarray and MPI. We compared calculation time using seismic wave propagation modeling and one to one communication time using domain decomposition technique. We also compared performance of parallel reverse-time migration programs using Coarray and MPI. Test results show that the computing speed of Coarray method is similar to that of MPI. On the other hand, MPI has superior communication speed to that of Coarray.

Keywords: Coarray, Fortran, Parallel processing, Wave propagation modeling, Reverse-time migration

서 론

탄성파 탐사 자료의 처리 분야는 대량의 계산 자원을 사용하는 수치 해석 분야로, 자료 처리 기술과 컴퓨터 기술 발달에 따라 이 분야에서 병렬 연산의 비중이 날로 커지고 있다. 대규모 병렬 연산을 사용하는 자료 처리 기술로는 참 반사 보정과 파형역산 등이 있다(Baysal *et al.*, 1983; Tarantola, 1984). 병렬 처리용 하드웨어를 제어하기 위한 대표적인 병렬 연산 기술로는 MPI (Message Passing Interface)와 OpenMP (Open Multi-Processing)를 들 수 있다(Pacheco, 2011). MPI는 프로세스(Processing element, 처리 소자)간에 메모리를 공유하지 않는 분산 메모리 시스템에서 병렬로 연산하며 정보를 주고 받기 위한 규약이고, OpenMP는 프로세스들이 동일한 메모리 공간에 접근할 수 있는 공유 메모리 시스템에서의 병렬 연산을 위한 규약이다. OpenMP는 순차적 프로그램의 반복문을 간편하게 병렬로 계산하는 방식으로 사용자가 쉽게 익힐 수 있는 반면에 공유 메모리 모델을 사용하므로 여러 개의 서버를 가진 클러스터 컴퓨터에서의 사용에 제한이 있다. MPI는 클러스터 컴퓨터에서 널리 사용되는 기술이지만 OpenMP에 비해 사용법이 복잡하다. 이러한 문제점들을 극복하고자 포트란 프로그래밍 언어에 코어레이(Coarray) 기능이 추가되었다. 본 연구에서는 파동 전파 모델링과 거꿀 참반사 보정 프로그램을 코어레이와 MPI로 구현하여 그 성능을 비교해보고 탄성파 병렬 처리에 코어레이를 사용하는 것이 유용한지 살펴본다.

Received: 22 July 2016; Revised: 17 August 2016;
Accepted: 21 August 2016
*Corresponding author
E-mail: wansooha@pknu.ac.kr
Address: Department of Energy Resources Engineering, Pukyong National University, 45 Yongso-Ro, Nam-Gu, Busan, Republic of Korea (48513)

코어레이를 이용한 병렬 연산

코어레이는 포트란 2008 표준에 도입된 병렬 연산 방법이다 (Metcalf *et al.*, 2011). 분산형 메모리 모델을 이용하는 병렬 연산 방법으로, 클러스터 컴퓨터에서 사용 가능한 MPI와 비교할 수 있다. 그러나 메시지를 주고 받는 방식으로 통신을 수행하는 MPI와 달리 코어레이에서는 파티션 글로벌 주소 공간 개념을 사용한다. 파티션 글로벌 주소 공간 개념은 분산 메모리를 공유 메모리와 유사한 형태로 사용하되, 내부적으로 현재 프로세스의 메모리와 다른 프로세스의 메모리를 구분하여 효율적으로 사용할 수 있도록 하는 개념이다(Pacheco, 2011). 이로 인해 코어레이를 사용하면 직접적인 자료 전송 명령 없이 배열을 참조하는 것과 유사한 방식으로 타 프로세스의 메모리에서 자료를 읽어오거나 값을 쓸 수 있다. 그 예로, Fig. 1에 크기가 n인 실수 배열 data를 첫 번째 프로세스에서 두 번째 프로세스로 복사하는 포트란 코드를 제시하였다. 이 코드는 뒤에서 살펴볼 영역 분해 프로그램에서 영역간 통신 코드의 핵심 부분을 가져와 이해하기 쉽도록 변수명 등을 수정한 코드로, 동일한 기능을 각각 MPI와 코어레이를 이용하여 구현하였다. 코드에서 확인할 수 있는 것처럼, 사용자는 MPI에 비해 간단한 코드로 병렬 작업을 수행할 수 있다. 통신을 위해서는 변수나 배열을 코어레이로 선언해야 한다. 코어레이로 선언하지 않은 변수들은 모두 해당 프로세스에서만 사용할 수 있다. 코어레이에서 각각의 프로세스는 이미지(image)라 부르고 프로세스 번호가 0번부터 시작하는 MPI와 달리 이미지 번호는 1번부터 시작한다. 코어레이의 자세한 사용법은 Metcalf *et al.* (2011)에서 찾을 수 있다.

코어레이는 아직 MPI에 비해 기능이 부족하다. 예를 들면, MPI_Bcast, MPI_Reduce와 같은 집합 통신용 함수가 없다. 부족한 기능은 앞으로 나올 표준에서 추가될 것으로 보인다. 본 연구에서는 코어레이와 MPI의 기능을 비교하기보다는 동일한 작업을 수행하는 성능을 비교하였다.

```
! MPI version

if(ipe==0) call mpi_send(data, n, mpi_real, 1, tag, mpi_comm_world, ierr)

if(ipe==1) call mpi_recv(data, n, mpi_real, 0, tag, mpi_comm_world, ierr)



! Coarray Fortran version

if(image==1) data(1:n)[2]=data(1:n)
```

Fig. 1. Code snippets of MPI and Coarray programs. The programs copy real 'data' array with n elements from the first processing element to the second one.

병렬 프로그램의 성능은 연산 성능과 통신 성능으로 나누어 생각할 수 있다. 비교를 위해 세 가지 성능 시험을 수행하였다. 우선 통신이 없는 상태에서 파동 전파 모델링을 통해 연산 성능을 시험하였고, 동일한 연산에 대해 통신량을 변화시켜가며 일대일 통신 성능을 시험하였으며, 마지막으로 거꿀 참 반사 보정을 통해 병렬 처리 성능을 비교하였다. 코어레이와 MPI 프로그램은 인텔 컴파일러 16.0.2 버전을 이용하여 컴파일하였고, MPI 또한 인텔 MPI를 이용하였다. CPU (Central Processing Unit)는 인텔사의 Xeon E7-4850 2.30 GHz 모델을 사용하였고 병렬 프로그램에서는 한 CPU 코어당 하나의 프로세스 또는 이미지를 할당하여 계산하였다.

다중 송신원 파동 전파 모델링을 통한 연산 성능 시험

첫 번째 성능 시험에서는 다음의 시간 영역 2차원 음향파 파동 방정식을 이용하여 다중 송신원을 이용한 파동 전파 모델링을 수행하였다.

$$\frac{1}{v(x,z)}\frac{\partial^2 u(x,z,t)}{\partial t^2} = \frac{\partial^2 u(x,z,t)}{\partial x^2} + \frac{\partial^2 u(x,z,t)}{\partial z^2} + f(x,z,t) \quad (1)$$

위 식에서 v는 P파 전파 속도, u는 파동장, f는 송신원을 의미한다. 송신원에 따라 프로세스를 할당하도록 MPI와 코어레

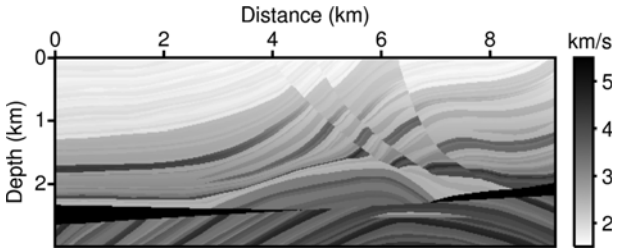

Fig. 2. Marmousi velocity model (Versteeg, 1994).

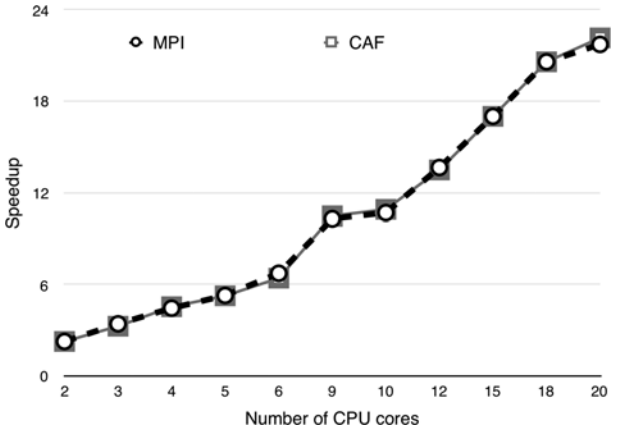

Fig. 3. Speedups of parallel modeling programs using MPI and Coarray Fortran (CAF) with respect to a serial program. No communication is included in this example.

이 병렬 프로그램을 구현하고 모델링 실행 시간을 비교하였다. 이 때 프로세스간 통신을 없애 계산 시간만 비교할 수 있도록 하였다. Marmousi 속도 모델(Versteeg, 1994)에서 총 180개의 송신원을 이용한 모델링을 수행하였다(Fig. 2). 격자 크기는 16 m, 격자수는 576 × 188개이며, 1 ms 샘플링 간격으로 총 4초간 기록하였다. 시간에 대해 2차, 공간에 대해 8차 유한 차분법을 사용하였다. 1개에서 20개까지 프로세스 수를 증가시켜가며 계산 시간을 비교하였다. 계산 시간은 프로그램을 번갈아가며 세 번 실행한 결과의 평균값을 이용하였다. Fig. 3에 순차적 실행 프로그램 대비 프로세스 수 증가에 따른 계산 속도 향상 정도를 나타내었다. 실험 결과 MPI와 코어레이 기법을 사용할 경우 프로세스 규모에 따른 연산 성능 차이는 크지 않은 것으로 나타났다.

영역 분해를 통한 일대일 통신 성능 시험

이번에는 MPI와 코어레이 기법의 통신 성능 비교를 위해 영역 분해를 이용한 파동 전파 모델링을 수행하였다. 유한 차분법을 이용한 2차원 파동 전파 모델링에서 매질을 프로세스 개수로 분해하고 각 프로세스에 하나의 영역을 할당하여 계산하였다. 영역 분해 기법은 보통 매질의 격자 수가 많을 때 계산 분배 또는 프로세스당 메모리 사용량 감소를 위해 적용한다(McCool *et al.*, 2012). 영역 분해 기법을 이용하면 매 시간 샘플 간격마다 영역 경계에서 프로세스간 통신이 필요하다. 이 때의 주요 통신 형태는 인접한 영역을 담당하고 있는 프로세스간 일대일 통신이다. 영역을 나눌 때에는 수직 방향으로 나누면 탄성파 모델들에서 일반적으로 통신량이 줄어서 유리하지만 본 연구에서는 통신 성능 비교가 목적이므로 가로 방향으로 영역을 분해하였다(Fig. 4).

영역 분해 예시로 Fig. 4에서 전체 모델링 영역을 세 개의 영역으로 분해하였다. 유한 차분법 계산을 위해 모델링 영역 바깥에 층을 덧붙이게 되는데 전체 모델링 영역을 분해하게 되면 분해된 경계에도 층을 덧붙여야 한다. 이 때 새로 덧붙인 층은 원래 인접 영역의 내부에 해당하므로 모델링 과정에서 덧붙인 층의 값을 인접 영역의 파동장 값으로 갱신해야 한다. 이 과정에서 프로세스간 통신이 발생하게 되는데 경계가 길수록 코어간 통신량이 많아지게 된다. 경계의 길이가 일정할 경우에는 전체 영역을 더 많은 프로세스를 이용하여 더 작은 영역들로 분해할수록 전체적인 통신량이 증가한다. 본 연구에서는 경계의 길이를 고정하고 더 많은 영역들로 분해하는 경우와, 분해에 사용한 프로세스 수는 고정하고 경계의 길이를 늘리는 경우를 모두 살펴보았다. 개별 코어에서의 통신량은 다음과 같다.

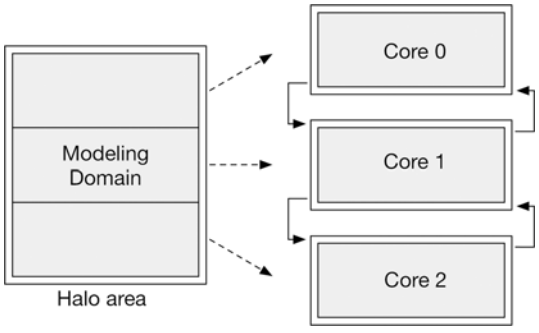

Fig. 4. A diagram explaining domain decomposition. Each core updates wavefields in its own modeling area. The halo areas between two cores are updated using the updated wavefields of adjacent cores.

$$N_e = \text{Wordsize} \times \frac{\text{Order}}{2} \times nx \times 4 \times nt \tag{2}$$

위 식에서 Wordsize는 단정도 실수를 사용할 경우 4 바이트, 배정도 실수를 사용할 경우 8 바이트가 된다. Order는 유한 차분법 차수로, 덧붙이는 층의 개수는 한쪽 경계에서 Order/2가 된다. nx는 가로 방향 격자 수이고, 상부 경계와 하부 경계에서 각각 데이터를 주고 받으므로 통신이 4회 발생한다. 이 과정을 모든 시간 격자에서 수행하므로 시간 격자수 nt를 곱하면 한 프로세스에서의 총 통신량이 된다. 단, 가장 위에 있는 층과 가장 아래 있는 층에서는 한쪽 경계에서만 통신이 발생하므로 첫 번째와 마지막 프로세스에서 총 통신량은 위 통신량의 절반이 된다. 따라서 모든 프로세스에서 발생하는 통신량의 총 합은 다음과 같다.

$$N_t = N_e(N_{pe} - 1) \tag{3}$$

위 식에서 N_{pe}는 프로세스의 수를 의미한다.

MPI에서 양방향 통신을 이용할 경우 동기식 통신과 비동기식 통신 방법이 존재한다. 동기식 통신은 통신이 끝날 때까지 다음 작업으로 넘어가지 않는 방법이고 비동기식 통신은 통신하라는 명령만 내리고 통신 여부와 상관 없이 다음 명령으로 넘어가는 방법이다. 통신 종료 여부는 별도의 함수를 이용하여 확인한다. 비동기식 통신을 사용하면 동기식 통신에 비해 CPU에 부하가 추가되지만 통신과 연산량이 많을 경우에는 통신과 연산 과정을 중복시킬 수 있어 유리하다. 영역 분해 성능 시험에서는 두 가지 방법 모두를 실행하여 결과를 비교하였다.

코어레이를 이용한 통신은 단방향 통신이지만 단방향 통신에도 두 가지 방법이 존재한다. 보내기 방식과 가져오기 방식으로, 보내기 방식은 한 이미지에서 다른 이미지의 코어레이에 값을 쓰는 방식이고 가져오기 방식은 한 이미지에서 다른 이미지의 코어레이에 저장된 값을 읽어오는 방식이다. 코어레이 역시 영역 분해 성능 시험에서 두 가지 방법 모두를 구현하여 실행하였다.

실험에 사용한 모델은 2차원 상속도 모델로, P파 전파 속도는 3 km/s, 격자 크기는 10 m, 깊이는 10 km로 공간에 대한

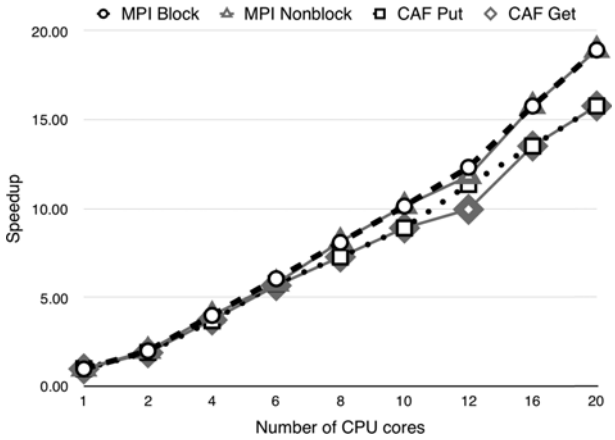

Fig. 5. Speedups of domain decomposition programs with respect to a serial program according to the number of CPU cores. (Block: MPI synchronous transfer, Nonblock: MPI asynchronous transfer, Put: Coarray's send method, Get: Coarray's receive method)

8차 유한차분식을 이용하여 계산하였다. 샘플링 간격은 1 ms로 총 2초간 기록하였다.

영역 분해의 첫 번째 실험에서는 격자 크기를 2000 × 1000으로 고정하고 프로세스 수를 증가시켜가며 모델링 시간을 비교하였다. 이 경우 프로세스 수 증가에 따라 전체 통신량은 증가하지만 전체 계산량은 일정하다. 개별 프로세스의 통신량은 일정하고 계산량은 감소한다. 이 예제에서 개별 프로세스의 통신량 N_e는 244 MB이다. 비교 결과 MPI 프로그램의 실행 속도가 코어레이 프로그램의 실행 속도보다 빨랐다(Fig. 5). 이전 실험에서 연산 성능 차이가 미미했으므로 이는 통신 성능 차이임을 알 수 있다. MPI의 경우 동기식과 비동기식 방식의 실행 시간에 거의 차이가 없는데 이는 하나의 시간 샘플에 해당하는 파동장 계산시 계산량과 통신량이 많지 않기 때문에 연산과 통신의 중첩에 따른 이득이 거의 없기 때문이다. 코어레이의 경우에도 보내기 방식과 가져오기 방식의 성능 차가 크지 않았다.

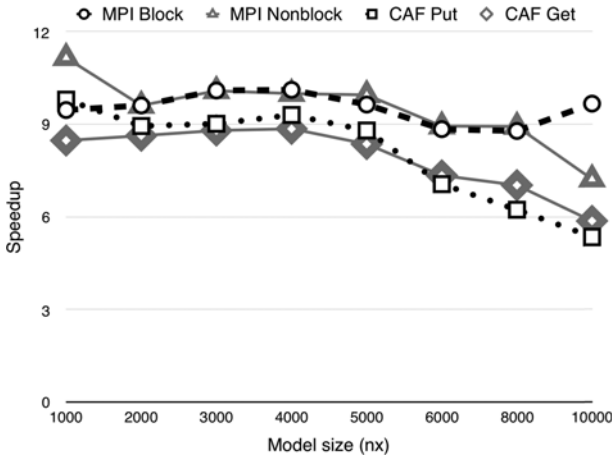

Fig. 6. Speedups of domain decomposition programs with respect to a serial program according to the model size. (Block: MPI synchronous transfer, Nonblock: MPI asynchronous transfer, Put: Coarray's send method, Get: Coarray's receive method)

영역 분해의 두 번째 실험에서는 깊이 방향 격자 수를 1000개로 고정하고 가로 방향 격자 수(nx)를 1,000개에서 10,000개까지 1,000개 단위로 증가시켜가며 모델링 계산 시간을 비교하였다. 병렬화를 위한 프로세스 수는 10개로 고정하였다. 따라서 격자 수가 증가함에 따라 프로세스당 계산량과 통신량이 모두 증가한다. 개별 프로세스의 통신량은 nx가 1,000일 때 122 MB, nx가 10,000일 때 약 1.2 GB이다. 실험 결과 MPI 통신 성능이 코어레이 통신 성능보다 우수함을 알 수 있었다(Fig. 6).

거꿀 참 반사 보정을 이용한 성능 시험

거꿀 참 반사 보정은 첫 번째 실험에서 Marmousi 속도 모델로부터 얻은 파동장을 관측 자료로 하고 실제 Marmousi 속도 모델을 사용하여 수행하였다(Fig. 7). 거꿀 참 반사 보정은 시간 영역에서 모델링시 사용한 것과 동일한 송신원을 사용하여 실행하였다(Kim *et al.*, 2013). 프로그램은 송신원에 따라 프로세스를 할당하도록 병렬화하였고, 두 가지 집합 통신 기능을 추가하였다. 프로그램 시작시 속도 모델을 MPI_Bcast를 이용해 각 프로세스에 복사하였고, 각 송신원의 결과 영상을 더하기 위해 MPI_Reduce를 사용하였다. 코어레이에는 집합 통신을 위한 표준 기능이 없기 때문에 일대일 통신 기능을 이용해 병렬로 복사가 일어나도록 프로그램을 구현하였다(McCool *et al.*, 2012). Fig. 8에서 볼 수 있듯이 순차적인 반복문을 이

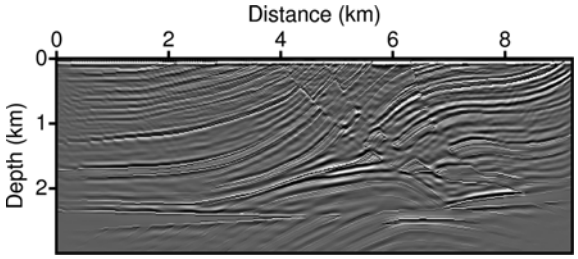

Fig. 7. A migration image obtained using Coarray Fortran with 10 CPU cores.

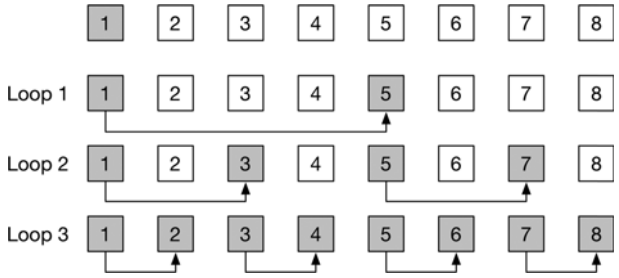

Fig. 8. A diagram explaining a parallel copy algorithm. Each box indicates an image of Coarray. Shaded boxes are images with the data.

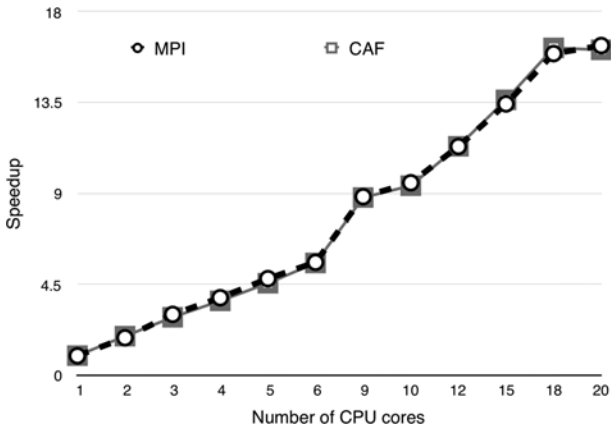

Fig. 9. Speedups of parallel reverse-time migration programs with respect to a serial program according to the number of CPU cores.

용해 자료를 한 프로세스씩 복사하는 대신 병렬로 복사를 실행하면 통신 시간을 줄일 수 있다. 각 송신원의 결과 영상을 더할 때에는 그림의 과정을 거꾸로 실행하게 된다. 그 결과, 집합 통신 과정이 존재하지만 거꿀 참 반사 보정 연산량에 비하면 통신량이 많지 않기 때문에 MPI 프로그램과 코어레이 프로그램의 실행 시간에 차이가 거의 없는 것으로 나타났다(Fig. 9).

토 의

지금까지의 실험을 통해 MPI와 코어레이를 사용할 경우 연산 성능에는 큰 차이가 없지만 통신 속도는 MPI가 코어레이보다 빠르다는 것을 알 수 있었다. 실제 본 연구에 사용한 컴파일러에서 내부적으로 코어레이를 구현할 때 MPI를 기반으로 한다(Intel Developer Zone, 2016). 이 경우 코어레이는 MPI를 쉽게 사용할 수 있도록 해주는 추가적인 사용자 환경으로 생각할 수 있고, 코어레이 프로그램의 성능은 MPI 프로그램의 성능보다 좋을 수 없다. 따라서 영역 분해와 같이 통신량이 많은 작업에서는 MPI를 사용하는 것이 유리하고, 영역을 분해하지 않고 수행하는 거꿀 참 반사 보정과 같이 연산량에 비해 통신량이 많지 않은 작업에서는 코어레이를 사용해도 MPI와 유사한 성능을 얻을 수 있다. 기존에 MPI로 작성한 프로그램이 있다면 코어레이로 바꿀 필요는 없다. 그러나 통신량이 많지 않은 수치해석 프로그램을 새로 작성한다면 MPI 대신 상대적으로 사용법이 간단한 코어레이를 사용할 수 있을 것이다.

본 연구에서는 장비의 제한으로 인해 단일 서버 노드 내에서 최대 20개의 프로세스를 사용하여 코어레이와 MPI 병렬 연산 성능을 비교하였다. 대규모 병렬화를 위해서는 대규모 클러스터 서버 시스템에서의 성능 시험이 필요하다. 이 경우 서버 노드간 통신 성능이 통신 장비의 성능에 크게 영향을 받게 된다.

결 론

코어레이는 포트란 컴파일러에서 자체적으로 지원하는 병렬 연산 기능으로 별도의 라이브러리를 설치할 필요가 없고 MPI에 비해 사용법이 간단하다는 장점이 있다. 반면에 집합 통신을 위한 내장 함수가 없는 등 MPI 대비 기능이 부족하다는 단점이 있다. 파동 전파 모델링과 거꿀 참 반사 보정 예제를 이용한 성능 시험 결과 코어레이의 계산 성능은 MPI와 유사했지만 통신 성능은 MPI에 비해 떨어졌다. 아직은 코어레이가 표준에 도입된 초기이고 코어레이 기능을 완벽하게 지원하는 컴파일러도 많지 않다. 앞으로 후속 표준에서 미흡한 기능이 보완되고 컴파일러 최적화를 통해 통신 성능도 향상될 것으로 기대된다.

감사의 글

이 논문은 부경대학교 자율창의학술연구비(2015년)에 의하여 연구되었음.

References

Baysal, E., Kosloff, D., and Sherwood, J., 1983, Reverse time migration, *Geophysics*, **48**(11), 1514-1524.

Intel Developer Zone, 2016.7.20., https://software.intel.com/en-us/articles/distributed-memory-coarray-fortran-with-the-intel-fortran-compiler-for-linux-essential.

McCool, M., Robison, A., and Reinders, J., 2012, Structured parallel programming, Morgan Kaufmann, 100-101.

Metcalf, M., Reid, J., and Cohen, M., 2011, Modern Fortran explained, Oxford, 333-352.

Pacheco, P., 2011, An introduction to parallel programming, Morgan Kaufmann, 15-81.

Kim, Y., Cho, Y., and Shin, C., 2013, Estimated source wavelet-incorporated reverse-time migration with a virtual source imaging condition, *Geophysical Prospecting*, **61**, 317-333.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation, *Geophysics*, **49**(8), 1259-1266.

Versteeg, R., 1994, “The Marmousi experience: velocity model determination on a synthetic complex data set”, *Leading Edge*, **13**, 927-936.