MPI의 지속 통신 메커니즘을 이용한 병렬 유한차분시간영역 전산모사 프로그램의 통신 성능 향상에 관한 연구

김희운, 전경원, 김형규, 홍현표, 정영주†
광주과학기술원 정보통신공학과
e-mail:†ychung@gist.ac.kr

A Study on the Communication Performance Improvement of the Parallel Finite-Different Time-Domain Simulator by using the MPI Persistent Communication

Huioon Kim, Kyungwon Chun, Hyeong-gyu Kim, Hyunpyo Hong, and Youngjoo Chung
Dept. of Information and Communications, GIST

요 약

유한차분시간영역 방법은 전자기파 관련 분야의 전산모사에 많이 사용되는 수치해석기법이다. 이 방법을 이용하여 구현한 전산모사 프로그램은 많은 계산 자원 필요로 하기 때문에 병렬 계산 환경을 이용하게 되는 경우가 많다. 병렬 계산 환경에서 전산모사를 수행할 경우, 병렬로 수행되는 각 프로세스 간의 통신 속도와 네트워크의 지연 시간은 계산의 병목 현상을 초래하여 전체적인 성능을 저하시키는 원인이 된다. 따라서, 본 논문에서는 MPI의 지속 통신 메커니즘을 이용하여 병렬 프로세스 간 동기화 속도를 증가시킴으로써 유한차분시간영역 전산모사 프로그램에서의 MPI 통신 성능의 향상을 꾀하고, 그 결과를 그래프로 도시하였다. 또한 기존의 양방향 통신과 단방향 통신 메커니즘을 사용했을 때의 성능과 비교/분석하여, 병렬 유한차분시간영역 전산모사 프로그램에 있어서 지속 통신 메커니즘의 장/단점을 제시하고, 그 효용성에 관해 논의한다.

1. 서론

유한차분시간영역 (Finite-Different Time-Domain, FDTD) 방법은 수치해석기법의 하나로, 여러 가지 장점들로 인하여 전자기파 관련 분야의 전산모사 프로그램의 핵심 알고리즘으로 많이 사용된다[1]. 이 방법은 미분 방정식 형태의 맥스웰 방정식을 차분방정식 형태로 변환한 뒤 시공간 영역에서 해를 구하는 방법으로, 적용 분야에 있어서 거의 제한을 받지 않는다는 것이 장점이다. 반면, 이 방법은 높은 계산 능력과 많은 양의 메모리 공간을 요한다는 단점도 갖고 있는데, 이러한 단점을 극복하기 위한 방법으로 병렬 계산 환경의 이용을 들 수 있다. 1994년에 시작된 베오울프 (Beowulf) 프로젝트[2] 기반의 클러스터 컴퓨팅 시스템은 이와 같은 병렬 계산 환경의 하나로, 손쉽게 시중에서 구매할 수 있는 일반적인 컴퓨터 장비들을 이용하여 중·소규모의 병렬 계산 환경을 저렴하게 구축할 수 있다. 이러한 클러스터 컴퓨팅 시스템은 시중에서 판매되는 개인용 컴퓨터 급의 계산 노드가 이더넷과 같은 일반적인 네트워크 인터페이스를 통하여 상호 연결되어 있는 구성을 가지는데, 이와 같은 분산 메모리 환경에서는 메시지 전달 인터페이스 (Message-Passing Interface, MPI)[3] 방식의 병렬화 메커니즘이 잘 동작하는 것으로 알려져 있다.

MPI는 병렬 프로그래밍 모델 중의 하나로, 가장 많이 사용되는 병렬화 메커니즘 중의 하나이다. MPI 자체는 메시지 전달을 위한 라이브러리의 명세로, 일종의 규약이라고 할 수 있으며, 이를 C/C++, Fortran 등의 언어로 구현한 구현체를 사용하여 병렬 프로그램을 작성하게 된다. 특히, MPI의 파이썬 (Python) 바인딩은 파이썬 언어로 작성된 프로그램에서 MPI 라이브러리를 사용할 수 있게 해주는 파이썬 모듈로, pyMPI, MYMPI, MPI for Python 등의 관련 프로젝트들이 있다. 파이썬 언어를 사용하여 전산모사 프로그램을 작성하게 되면 파이썬 특유의 간결함과 높은 생산성, 유지관리의 편의성 등의 많은 이점을 얻을 수 있기 때문에, 본 연구에서는 파이썬으로 전산모사 프로그램을 설계/구현하고, 파이썬 MPI 바인딩을 이용하여 병렬화 하였다.

MPI는 다양한 통신 메커니즘을 지원하는데, 이러한 통신 메커니즘은 실제 응용에 따라 그 성능과 적용 방법이 상이하다. 본 논문에서는 이와 같은 몇 가지 통신 메커니즘의 차이점을 보이고, 특히 유한차분시간영역 방법을 사

용한 전산모사 프로그램에서 지속 통신 메커니즘이 가져오는 성능상의 이점을 부각시켜 다른 통신 메커니즘과 비교하여 그 결과 및 효용성에 관해 논의한다.

2. 기반 연구

본 연구실에서 개발한 GMES (GIST Maxwell's Equestions Solver)[4]는 객체 지향적으로 구현된 FDTD 방법 전산모사 프로그램으로, 일부 코드를 제외하고는 파이썬 언어로 구현이 되어 있으며, 파이썬 MPI 바인딩과 스레드 프로그래밍 모델을 이용하여 병렬 계산을 구현하였다. 특기할만한 특징으로는, matplotlib[5] 모듈을 이용한 실시간 가시화, 1, 2, 3차원 직각좌표계 지원, CPML (Convolutional Perfectly Matched Layer) 흡수 경계 조건 지원, 포인트 소스 및 투과성 가우시안 빔 소스 지원, 등방성 분산 유전체 지원 등이 있다.

GMES에서 사용한 파이썬 MPI 바인딩은 MPI for Python으로, MPI-2의 C 구현을 기반으로 MPI-2의 C++ 구현과 유사한 구조로 구현한 파이썬 모듈이다. 이 모듈은 일반적인 MPI의 점 대 점 (point-to-point, p2p) 통신과 집합 (collective) 통신을 지원하는데, 다른 언어 구현과는 다르게 이 통신들에 있어서는 임의의 파이썬 객체를 전송할 수 있다는 것이 특징이다. 또한, 연속적인 (contiguous) 메모리 버퍼 인터페이스 (문자열, NumPy 배열 등)에 대해 블록킹/논블록킹/지속 (persistent) p2p 통신 및 집합 통신, 단방향 (one-sided) 통신을 지원하며, 그 밖에도 병렬 입출력과 그룹, 커뮤니케이터의 생성과 관리 및 사용자 정의 데이터 타입의 생성 역시 지원하며, 동적 프로세스 생성 및 관리 기능도 부분적으로 지원한다.

3. 병렬 통신 메커니즘

FDTD 계산 시에 연속적인 메모리의 불연속적인 특정 영역을 동기화할 필요성이 있는데, GMES에서는 블로킹 기반의 양방향 통신 인터페이스에 파이썬 객체를 직렬화 (Serialization)하여 전송하는 방식을 사용한다. 직렬화 시에 파이썬 내부 객체의 경우에는 marshal 모듈을 이용하며, 그 외의 일반적인 객체의 경우에는 cPickle 모듈을 이용한다. 이 방법을 사용하게 되면 구현이 간결해지지만 객체를 패킹/언패킹 하는데 추가적인 부하가 발생한다.

임의의 파이썬 객체를 전송하는 양방향 블록킹 통신 메커니즘 대신에 연속적인 메모리 공간을 사용하는 단방향 통신 메커니즘과 지속 통신 메커니즘을 사용하면 객체의 직렬화 부하를 없앨 수 있다. 이 경우에는 연속적인 메모리 영역을 사용해야 하므로, 적절한 데이터 타입의 정의가 필수적이다. 단방향 통신 메커니즘은 원격 메모리 접근 (Remote Memory Access, RMA)이라고도 불리는 메커니즘으로, 이 메커니즘을 FDTD에 적용했을 경우, 그 자체의 특성에 의해 프로세스 간 통신 속도가 향상될 뿐만 아니라, 직렬화 부하를 제거하여, 동기화 속도가 향상된다[6].

지속 통신 (Persistent Communication) 메커니즘은 병렬 구현 내부의 루프 내에서 동일한 인자 리스트를 사용하여 통신이 반복적으로 실행될 경우에 적용할 수 있는 통신 메커니즘으로, 동일한 인자 리스트가 바인딩 된 지속 통신 요청 객체를 단 한번만 생성하고, 그것을 반복적으로 사용함으로써, 통신 속도를 최적화할 수 있다[7]. 이는 프로세스와 통신 컨트롤러 사이의 부하를 제거하는 것으로, 직렬화 부하 및 프로세스-컨트롤러 간 부하의 제거로 통신 속도에 있어서 성능 향상을 기대할 수 있다. 병렬 FDTD 알고리즘은 메모리의 일정한 구역을 매 반복계산마다 동기화하므로, 지속 통신 메커니즘에 매우 적합한 구조라고 볼 수 있다.

지속 통신 메커니즘은 지속 통신 요청 객체를 생성 (*Send_init*, *Recv_init*, etc...)하고, 루프 내부에서 생성된 지속 통신 요청을 시작 (*Start*)하고 완료 (*Wait*)한 후, 객체를 해제 (*Free*)하는 순서로 이루어진다. MPI for Python은 지속 통신 요청 객체를 생성하는 *Send_init* 또는 *Recv_init* 등과 같은 메소드 뿐만 아니라, 통신을 시작하거나 완료하는 *Start*, *Wait* 메소드, 그리고 지속 통신 요청 객체를 해제하는 *Free* 메소드까지, 지속 통신 메커니즘을 사용하기 위한 모든 구성 요소를 지원한다.

4. 벤치마크

구현의 성능 평가는 GIST (광주과학기술원, Gwangju Institute of Science and Technology)의 Nerv 클러스터[8]에서 수행되었다. Nerv 클러스터는 1대의 마스터 노드와 총 76대의 계산 노드로 구성되어 있는, 베오울프 급 클러스터이다. 마스터 노드는 두 개의 듀얼 코어 CPU (인텔 제온 3.00Ghz 1MB 캐쉬)와 4GB의 메모리를 가지고 있으며 기가비트 이더넷 컨트롤러 (Broadcom BCM5721 PCIExpress)를 통해 백본에 연결되어 있다. 계산 노드는 3가지의 다른 성능의 컴퓨터로 구성되어 있다. 그 중 네트워크 인터페이스가 다른 두 가지 종류의 계산 노드로 계산 환경을 구성하여 성능을 측정하였다. 첫 번째 환경에 사용된 노드는 듀얼 코어 CPU (인텔 펜티엄D 3.00Ghz 2MB 캐쉬)와 4GB의 메모리 용량을 가지고 있고, 기가비트 이더넷 컨트롤러 (인텔 82573E)를 통해 허브 (3Com Baseline Switch 2824)에 연결되어 있다. 다른 환경은 Intel Pentium 4 CPU 2.80GHz, 512KB 캐쉬와 1GB의 메모리 용량을 갖는 노드와 3C17300 SuperStack*3 Switch 4224T 허브로 이루어져 있다. 클러스터의 모든 허브는 백본 스위치 (3Com Superstack 3 4900)를 통해서 서로 연결되어 있다.

Nerv 클러스터 시스템의 운영체제는 데비안 GNU/리눅스 4.0r1 (Etch)이며, 리눅스 커널 버전은 2.6.18이다. MPI의 파이썬 바인딩으로 사용한 MPI for Python은 서브버전 저장소로부터 리비전 180을 다운로드 받아 사용하였으며, 파이썬 2.4.4 버전과 GCC 4.1.2 버전으로 컴파일 되었다. 사용된 MPI 구현은 MPICH2 1.0.8 버전이며, 각

프로세스 간 통신에 사용된 수치 배열은 NumPy 1.0.1-1 버전을 이용하였다.

성능 평가에 사용한 GMES는 네트워크 통신 부분의 성능 평가를 위하여 반복적으로 수행되는 FDTD의 실제 전자기장 계산 부분은 제외하고 사용하였다. 즉, FDTD 알고리즘의 의해 반복되는 매 시간 단계마다 MPI 통신만을 수행하며 실행 시간을 측정하였다. 또한, 원래 전자기장의 값은 초기화 단계에서 최초에 0으로 설정되지만, 이와 같은 일률적인 값은 직렬화 알고리즘에 이점을 줄 수 있기 때문에, 사실적인 측정을 위하여 0에서 1사이의 무작위 실수 값으로 초기화하여 사용하였다.

표 1. 직렬화 통신 메커니즘의 경우에 FDTD에서 사용하는 배열 크기

	배열 크기		
	ez (double)	hx (double)	hy (double)
Set 1	111,74,1	110,74,2	111,73,2
Set 2	111,166,1	110,166,2	111,165,2
Set 3	221,148,1	220,148,2	221,147,2
Set 4	276,184,1	275,184,2	276,183,2

표 2. RMA와 지속 통신 메커니즘의 경우에 FDTD에서 사용하는 배열 크기

	배열 크기		
	ez (double)	hx (double)	hy (double)
Set 1	112,75,1	111,75,2	112,74,2
Set 2	112,167,1	111,167,2	112,166,2
Set 3	222,148,1	221,148,2	222,147,2
Set 4	277,185,1	276,185,2	277,184,2

직렬화를 사용하는 블록킹 양방향 통신 메커니즘은 일반적인 파이썬 객체를 통신할 수 있다는 유연성 때문에 FDTD 계산에 최적화된 메모리 영역을 잡을 수 있다. 이는 각 계산 프로세스 마다 다른 메모리 크기를 가질 수 있다는 의미로, 이로 인해 다른 구현 보다 약간 작은 메모리를 사용하게 된다. 단방향 통신 (RMA) 메커니즘, 그리고 지속 통신 메커니즘의 경우에는 모든 노드가 통신에 동일한 데이터 타입을 사용해야 하므로 계산 메모리가 약간 커지게 된다. 계산에서 사용하는 데이터 배열의 모양 및 크기가 각각 표 1과 표 2에 나타나 있다. 성능은 통신할 데이터 크기에 따라 네 가지로 구분해서 측정하였는데, Set 1, 2, 3, 4는 이런 구분을 표시한 것이다. 즉, Set 1의 배열 크기가 가장 작고, Set 4의 배열 크기가 가장 크므로, 이는 Set 1에서 Set 4로 갈수록 통신 데이터 크기가 커진다는 의미를 내포한다.

그림 1은 1Gbps 이더넷으로 연결된 6개의 노드에서 성능을 측정한 결과이고, 그림 2는 100Mbps 이더넷으로 연결된 6개의 노드에서 측정한 결과이다. Serialization은 기존에 직렬화를 사용했을 때의 통신 성능을 나타내며, RMA Equiv. datatypes는 송수신 측이 동일한 데이터 타입을 사용한 RMA 통신 구현으로 *Put* 메소드와 *Get* 메소드 두 가지 메소드를 혼재하여 사용했을 때의 통신 성능이다. RMA Put only는 *Put* 메소드만을 사용해서 구현한 것이며, RMA Get only는 *Get* 메소드 만을 사용한 것이다. Persistent Standard는 지속 통신 메커니즘에서 send 오퍼레이션을 standard 모드로 수행한 것이며, Persistent Buffered, Ready, Sync. 등은 각각 buffered 모드, ready 모드, synchronous 모드로 수행한 결과이다. 그래프의 가로 축은 통신 데이터 크기를 나타내는데, 단일 노드, 단일 시간 단계에서 이루어지는 통신의 총 데이터 크기를 의미한다.

그림 1 1Gbps 이더넷 환경에서 각 구현의 계산 사이즈 변화에 따른 통신 시간 측정 결과

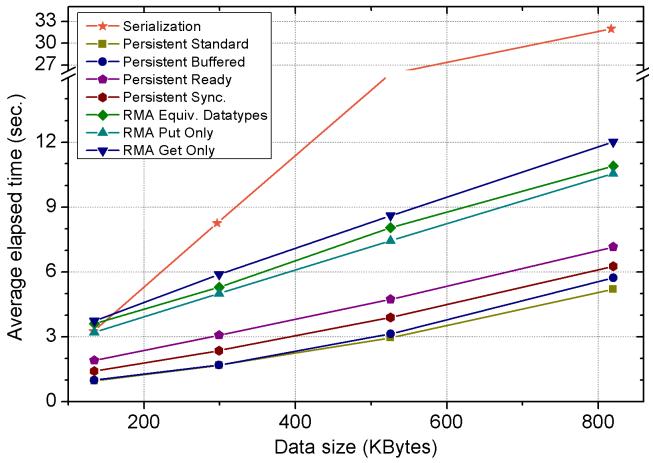
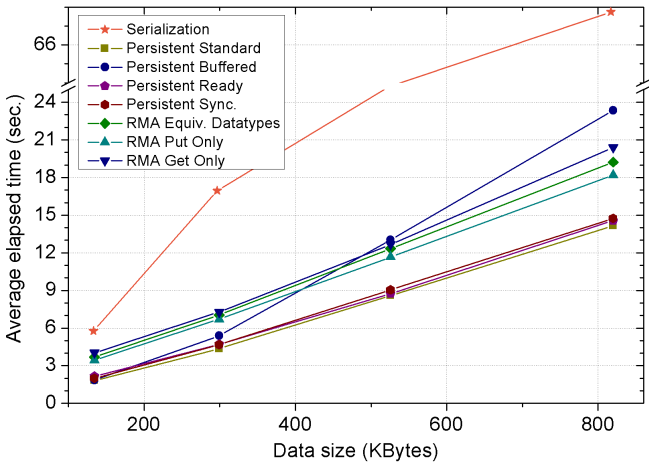


그림 2 100Mbps 이더넷 환경에서 각 구현의 계산 사이즈 변화에 따른 통신 시간 측정 결과



그래프에서 알 수 있다시피 기존의 블로킹 양방향 및 단방향 통신 메커니즘을 사용했을 때보다 지속 통신 메커니즘을 사용했을 때가 일반적으로 나은 성능을 보여준다. 블로킹 양방향 통신 메커니즘은 직렬화(pickling)를 사용

하므로, 통신 데이터 크기가 증가할수록 직렬화로 인한 오버 헤드가 급증하는 것을 볼 수 있는데, 이는 FDTD에서 계산 영역의 크기가 증가하거나 정밀도(resolution)가 증가하여 주고받을 데이터 크기가 증가될수록 동기화 성능을 크게 저하시킨다. 반면, RMA나 지속 통신은 데이터 크기의 증가에도 비교적 일정한 성능을 보여준다. RMA의 경우 *Put/Get* 메소드를 혼재해서 사용했을 때와 어느 한 쪽의 메소드만 써서 구현했을 때 성능의 차이가 발생하는데, 이는 원격 노드의 메모리에 쓰는 것과 읽는 것의 차이에서 기인하는 것으로 생각된다. 물론, 이 결과는 본원에서 보유한 클러스터 시스템으로 측정한 것으로, RMA에 최적화된 하드웨어에서 측정할 경우 결과가 다소 상이할 수도 있다.

지속 통신 메커니즘을 사용한 구현은 앞서의 두 가지 통신 메커니즘에 비해 일반적으로 향상된 성능을 보여준다. RMA와 마찬가지로, 지속 통신 메커니즘 또한 통신하는 데이터 크기 증가에 크게 영향을 받지 않고 일정한 성능을 보여준다. 다만, buffered 모드의 경우엔 컴퓨터의 데이터 처리 성능에 따라서 데이터를 버퍼로 복사하는 부하가 통신 부하보다 더 큰 영향을 줄 수도 있다. ready 모드와 synchronous 모드의 성능이 낮은 것을 알 수 있는데, 이는 동기화 부하에 의한 결과이다. ready 모드의 경우, FDTD 구현에서 각 프로세스들의 통신 시점을 맞춰주기 위해서 *MPI.Barrier* 메소드로 동기화하는 것이 필요기 때문이다. standard 모드와 buffered 모드는 성능이 오차 범위 내로 서로 비슷한데, 이는 FDTD 알고리즘 자체의 특성에 의한 것으로, send 오퍼레이션 이후에 receive 오퍼레이션이 즉각 수행되므로 버퍼의 효용성이 그다지 크지 않기 때문이다. 즉, send 오퍼레이션이 receive 오퍼레이션의 상태와 무관하게 동작하는, 비동기성이 강하게 요구되는 병렬 프로그램에서는 standard 모드보다 buffered 모드의 성능이 더 나을 것으로 생각된다.

5. 결론

지속 통신 메커니즘은 연속적인 메모리 영역을 사용하여 비연속적인 임의의 파이썬 객체를 직렬화 하는 데에 따르는 추가 오버 헤드를 제거하고, 동일한 인자를 가지고 반복적으로 생성되는 통신 요청 객체를 단 한 번만 생성하여 프로세스와 네트워크 컨트롤러 간의 통신 속도를 최적화함으로써, 전체적인 MPI 통신 성능을 향상시킨다. 이 방식은 RMA가 구현되어 있지 않은 MPI 구현체에서도 사용 가능하므로, 적용의 유연성이 뛰어나다. 또한 기존의 FDTD 프로그램의 설계를 훼손하지 않으면서 손쉽고 간결하게 적용 가능하다는 장점 또한 지닌다. 다양한 통신 메커니즘을 시험해 본 결과, 예상과 같이 지속 통신 메커니즘이 FDTD에 가장 적합한 방식이라고 결론지을 수 있다.

6. 감사의 글

본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원 사업 (IITA-2009-C1090-0902-0014) 및 교육과학기술부와 한국과학재단의 NCRC 사업의 지원을 받아 수행된 연구임(R15-2008-006-03001-0).

참고문헌

[1] A. Taflove and S. C. Hagness, “Computational Electrodynamics: The Finite-Difference Time-Domain Method,” USA: Artech House Inc., 3rd ed., 2005.

[2] Beowulf.org, http://www.beowulf.org/

[3] The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/

[4] K. Chun, H. Kim, H. P. Hong, and Y. Chung, “Object-Oriented Implementation of the Finite-Difference Time-Domain Method in Parallel Computing Environment,” Korea e-Science AHM 2008, paper AO2-1, p. 80, 2008.

[5] matplotlib, http://matplotlib.sourceforge.net/

[6] 전경원, 김희운, 홍현표, 김형규, 정영주, “MPI의 단방향 통신을 이용한 병렬 유한차분시간영역 프로그램의 실행 속도 향상,” 한국 정보과학회 HPC 연구회 동계 학술발표대회, p. 107-110, 2009.

[7] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard (Ver. 2.1),” 2008.

[8] K.Chun, Nerv cluster at GIST, <http://ontl.gist.ac.kr/about_lab/equipment/folder.2007-01-11.9687899850/index_html>, 2003-2009.