VIA기반의 통신 인터페이스 개발

이상기[0] 이윤영[00] 서대화[00]
경북대학교 정보통신학과[0] 경북대학교 전자공학과[00]
{leesky, yylee}palgong.knu.ac.kr, dwseo@ee.knu.ac.kr

Development of Communication Interface Based on VI Architecture

Sang-Ki Lee[0] Yoon-Young Lee[00] Dae-Wha Seo[00]
Dept. of Information and Communication[0], Kyungpook National University
Dept. of Electronics[00], Kyungpook National University

요약

하드웨어와 소프트웨어의 발전과 함께 컴퓨터에서 처리 해야 할 데이터량이 크게 증가하고 있다. 클러스터 내의 node들 사이에서 이런 대용량의 데이터들을 보다 빠르게 전송하기 위해서 Lightweight Messaging 기법이 등장하였으며, 대표적으로 AM, FM, U-Net, VIA등이 있다. 이 중에서 VIA는 커널 수준에서 구현된 TCP/IP를 대신해서 사용자 수준에서 커널을 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있게 하여 다양한 분야에서 사용되고 있으며, 새로운 프로토콜의 표준으로 자리를 잡아가고 있다. 그러나 이러한 장점에도 불구하고 프로그래밍의 난이성 때문에 제대로 숙지하기 까지는 많은 시간의 투자가 필요한 것이 사실이다. 이에 이 논문에서는 EVIL(Easy-to-use Virtual Interface Library)이라는, 개발자들이 좀더 쉽게 접근할 수 있는 라이브러리를 제안하였다. 그리고 EVIL, Native VIA, TCP/IP로 각각 같은 역할을 하는 프로그램을 작성하여 기존의 프로토콜들과 성능을 비교하였다.

1. 서론

하드웨어와 소프트웨어 기술이 발전함에 따라, 컴퓨터 시스템에서 처리해야 할 데이터의 크기 및 그 수도 늘어나게 되었고, 컴퓨터의 구조 및 성능도 하루가 다르게 변해서, 대용량의 데이터를 주고받는 일이 많아졌다. 특히 SAN(System Area Network)에서는 안정적인 데이터 통신이 가능해져 TCP/IP의 다양한 흐름 제어 메커니즘들은 오히려 오버 헤드로 작용하기도 한다[1][5].

이러한 점들 때문에 Berkeley NOW 시스템의 Active Message(AM) , Cornell 대학의 U-Net, Princeton 대학의 SHRIMP(Scalable High-performance Really Inexpensive Multi-Processor) 프로젝트에서 제안한 VMMC(Virtual Memory-Mapped Communication), Illinois 대학의 Fast Message(FM), Compaq, Intel, Microsoft의 VIA(Virtual Interface Architecture)등의 다양한 Lightweight Messaging 기법들이 등장하게 되었다[1][8].

이들 새로운 프로토콜들에 있어서 주요 관심사는 패킷 전송 중의 불필요한 복사를 줄여서 (zero-copy) 메시지의 전송 속도를 높이는 것과 기존 프로토콜을 단순화시키는 것에 있다[5][9]. Lightweight Messaging 기법을 도입함으로써 얻을 수 있는 장점은 범용 프로토콜에 비해 오버 헤드가 작아서 전송 능력이 높아지며 프로토콜 처리 루틴이 짧아지기 때문에 전송 지연이 줄어들고 짧은 메시지에 대한 전송 능력이 크게 개선된다. 이런 장점들로 인해 앞에서 열거한 새로운 프로토콜들은 점점 그 활용도가 높아지고 있다.

그 중 VIA는 TCP/IP등의 프로토콜을 대신해 CPU가 담당한 네트워크 통신 오버 헤드를 제거해 줄 수 있고,실제 기업 환경에 사용할 수 있는 시스템 수준의 고속 통신을 가능하게 하며, 응용 프로그램 수준에서 CPU를 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있도록 해줄 수 있다. 이런 점 때문에 현재 VIA가 널리 사용되고 있기는 하지만 프로그래밍의 난이성 때문에 제대로 숙달되기까지는 많은 시간의 투자가 필요한 것이 사실이다.

그래서 일반적인 개발자들이 쉽게 사용하면서도 성능을 최대한 발휘할 수 있는 인터페이스가 필요하게 되었고, 이에 이 논문에서는 EVIL(Easy-to-use Virtual Interface Library)이라는, 일반적인 개발자들이 좀더 쉽게 접근할 수 있는 라이브러리를 제안하여 위에서 말한 문제점들을 해결하고자 한다.

2. Virtual Interface Architecture

VIA는 Compaq, Intel, Microsoft의 3개의 기업이 클러스터 환경이나 SAN에서 사용할 고성능의 네트워크 기술을 위한 인터페이스 표준의 필요에 따라서 제안한 것이다[5][8].

VIA의 주목적은 실제 기업 환경에 사용할 수 있는 시스템 수준의 고속 통신을 가능하게 하기 위한 것이며 이는 응용 프로그램 수준에서 CPU를 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있도록 해 현재 범용으로 사용되는 TCP/IP 등의 프로토콜에서 CPU가 담당한 네트워크 통신 오버 헤드를 제거한다[7].

VIA의 특징은 다음과 같다.

- 각각의 VI는 통신상의 종점이며 이 한 쌍의 종점은 점 대 점 통신을 논리적으로 연결시켜 준다.
- 하나의 프로세스는 하나 이상의 VI를 가질 수 있다.
- 네트워크 어댑터는 통신상의 종점을 가상화하고, VI사이의 신뢰성 있는 데이터 전송을 담당한다.
- VI 구조는 기본적으로 가상 인터페이스, 완결 큐, VI 제공자, VI 소비자의 4개의 모듈로 구성된다[7].

그림 1.은 VIA와 TCP/IP(UDP/IP)의 차이를 보여 주고

있다. 전통적인 TCP/IP(UDP/IP)에서는 커널만이 네트워크 인터페이스에 접근할 수 있기 때문에 사용자가 네트워크를 통해서 통신을 하려면 사용자 버퍼 영역과 시스템 버퍼 영역 사이에서 데이터의 복사가 일어나게 된다.

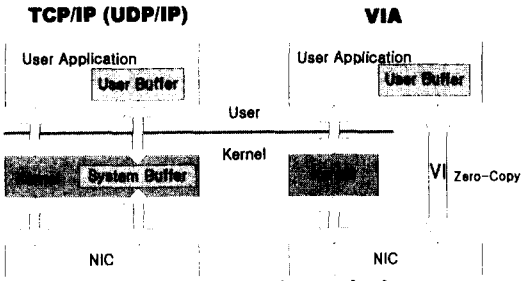


그림1. TCP/IP와 VIA의 비교

반면, VIA의 사용자 프로세스는 VI를 통하여 운영 체제의 개입이 없이 안전하게 네트워크 인터페이스에 직접 접근해서 사용자 버퍼와 시스템 버퍼 사이의 데이터의 복사 과정을 없앤다[5]. TCP/IP에서는 커널 만이 네트워크 디바이스에 접근할 수 있기 때문에 보호 기능이 커널 수준에서 제공되었으나, 이와 같이 사용자 프로세스가 직접 디바이스를 접근하는 경우에는 가상 주소의 해석과 보호 기능을 VIA가 제공하게 된다.

그러나 위의 장점에도 불구하고 VIA를 이용한 프로그래밍은 기존의 네트워크 프로그램 인터페이스 보다 어렵고 흐름 제어, 수신측의 디스크립터 준비와 같은 문제들을 프로그래머가 직접 준비해야만 한다[2].

이런 이유로 보다 쉽게 VIA프로그래밍을 할 수 있도록 하기 위해서는 프로그래머가 사용하기 쉬운 라이브러리가 필요하게 되었다.

3. 사용자 편의성을 고려한 VIA 라이브러리

3.1 EVIL의 특징

위에서 밝혔듯이 VIA가 제공하는 VIPL API (Virtual Interface Programming Library API)는 너무 복잡하여 프로그램을 작성하기 힘들고, VIA의 특성을 자세히 알지 못하는 상황에서 단순히 VIPL API[7]들만을 가지고 VIA를 이용하는 프로그램을 작성하는 것은 거의 불가능하다. 이에 본 논문에서는 사용이 용이하고, 직관적인 API들을 새로 정의하고 설계한 후 VIPL API들을 이용하여 이들을 구현하였다. 모든 응용에 최적화된 성능을 나타내는 API는 아니지만, 복잡한 여러 가지 단계를 줄여서 쉽게 SAN에서 VIA를 이용한 고성능의 응용 프로그램을 작성할 수 있는 API를 정의하고 이를 개발하였다.

3.2 EVIL API의 구성

EVIL API를 구성하는데 가장 중점을 둔 것은 사용의 편의성으로 VIA의 고성능 통신에 필요한 여러 단계들을 직관적인 단위로 묶어 단순한 함수 세트로 정의하였다.

그림 2.는 EVIL을 구성하는 함수들과 각 함수들의 역할을 간단히 나타낸 것이다.

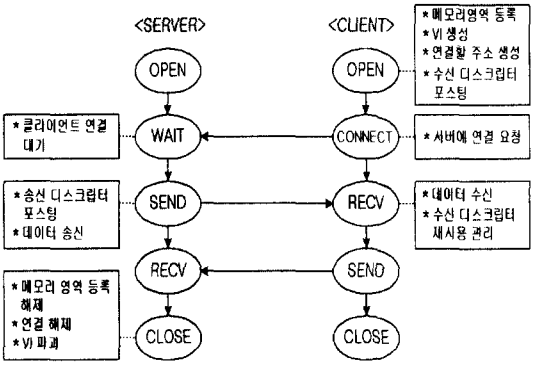


그림2. EVIL 함수의 역할

OPEN은 채널을 생성하는 역할을 하며 via_open() 함수로 구현되었다. VIA 채널을 생성하는 단계에서는 NIC(Network Interface Card)에 접근하여 사용 권한을 획득하고, VI를 생성하는데 필요한 여러 속성을 지정해 주며, 응용 프로그램에서 필요한 메모리 영역을 등록해 주며, 마지막으로 VI를 생성하도록 한다.

WAIT는 개설된 채널을 통해서 다른 노드로부터의 연결을 대기, 수락하며 VipConnectWait()와 VipConnectAccept() 함수를 한 함수 내에서 처리하도록 via_wait()함수로써 구현하였다.

CONNECT는 via_connect()함수로 구현하였는데 접속 대기 중인 상대방 호스트에 연결을 요청한다. 다른 호스트에 연결하고자 할 때 사용하는 함수로 내부적으로 VipConnect Request()함수를 호출하여, 접속 대기 중인 상대방 호스트에 연결을 요청한다.

SEND와 RECV는 각각 via_send()와 via_recv()로 구현하였고 VIA 채널을 통해서 데이터를 송수신하고 송수신 디스크립터를 관리한다.

CLOSE는 개설된 VIA채널을 닫는 역할을 하며 via_close() 함수를 사용하여 구현하였다. VIA채널을 닫기 위해서는 데이터의 전송이 끝난 후 VIPL을 통해 할당 받은 자원들을 시스템에 돌려주는 과정이 필요하다. 먼저 등록한 메모리를 반환 하여야 하며, 보호 테그(protection tag)도 반환하여야 한다. VIA 채널을 통해 연결도 끊어주어야 하며, VI를 파괴하고, NIC의 사용 권한도 반환하여야 한다. 이러한 많은 작업들을 동시에 하나의 함수를 통해서 수행하도록 하였으므로, 응용 프로그램의 작성이 한층 더 용이해 진다.

EVIL에서는 데이터 수신을 위해 다중 버퍼를 할당하고, 이를 라운드 로빈 방식으로 사용한다. via_send()는 응용 프로그램으로부터 송신을 요청 받은 데이터가 있는 메모리 공간을 등록한 후 이를 가리키는 디스크립터를 포스트 해 주도록 작성하였다. via_recv()에서는 대기 중인 여러 개의 디스크립터 중 데이터 수신이 끝나 종료되는 디스크립터가 가리키는 메모리 버퍼의 데이터를 응용 프로그램에 넘겨주도록 구현하였으며, 사용이 끝난 디스크립터는 다시 수신을 위해 대기하도록 포스트하였다.

채널의 정보를 가리키는 ViaChann 자료 구조는 채널의 이름(name)과 Virtual Interface의 핸들(viHand), NIC의 핸들(nicHand), 그리고 메모리 보호 태그(ptag)와 로컬 호스트와 리모트 호스트의 주소(localAddr, remoteAddr)가 있으며, 여러 개의 등록된 메모리 버퍼 영역(mem_buff)을 채널이 가지게 된다.

4. 실험 및 결과

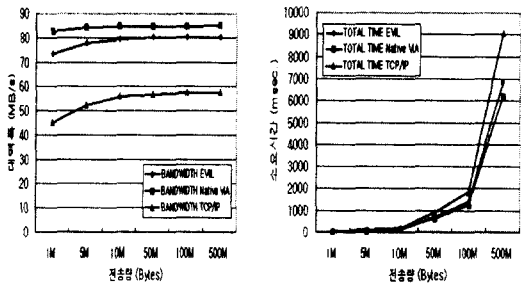
4.1 실험 환경

실험을 위하여 4대의 Intel Pentium Ⅲ-600Mhz 시스템을 사용하였다. 실험에 사용된 리눅스는 커널 2.2.14버전으로 VIA를 제공하는 환경으로는 Giganet 사의 cLAN을 사용하였다. Giganet 사의 cLAN은 cLAN5000 NIC상에서 동작하는 것으로 하드웨어적으로 VI를 지원하는 NIC이다.

4.2 성능 평가

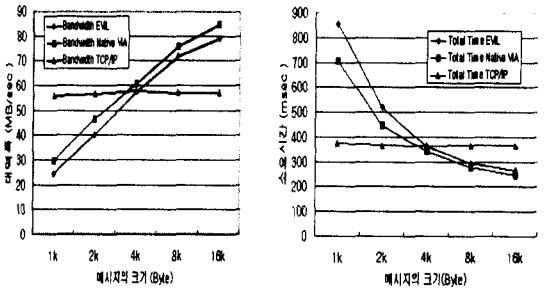
성능 평가를 위하여 EVIL과 VIPL, LANE(LAN Emulation)[3]을 이용하여 각각 동일한 크기의 메시지를 지정한 양만큼 반복하여 전송하여 그 성능을 측정하는 프로그램을 작성하였다.

그림 3.에 나타난 결과는 전송하는 데이터의 총량을 1MB에서 500MB로 늘여가면서 평균 전송 대역폭을 측정하였다. 이때 한번에 전송되는 메시지의 크기는 16KB로 고정하였다. EVIL을 사용하였을 때 Native VIA를 사용한 경우보다 약 5~10%의 대역폭 손실이 있지만 디바이스 드라이버 수준에서 IP를 에뮬레이트한 LANE를 이용한 경우보다는 39~62%의 성능 개선이 있음을 확인할 수 있었다. 그리고 전송하는 데이터의 양이 커지더라도 일정 수준의 성능을 나타내고 있었다.



대역폭(a) 총 소요 시간(b)

그림 3. EVIL, Native VIA, TCP/IP의 데이터의 크기 변화에 따른 성능 변화



대역폭(a) 총 소요 시간(b)

그림 4. EVIL, Native VIA, TCP/IP의 데이터 단위 변화에 따른 성능 변화

그림 4.는 20MB의 메시지를 전송하는 데, 한번에 전송하는 메시지의 크기를 변화시키면서 그 성능의 변화를 살펴보았다. 메시지의 크기가 클수록, 성능이 좋아지는 것을 확인할 수 있는데, 이것은 메시지의 크기가 작을수록 동일한 크기의 메시지를 보내기 위한 회수가 많아지기 때문이다. 한번에 전송하는 메시지의 크기를 변화시킨 결과 약 7~17%정도의 손실이 발생하였다.

5. 결론

본 논문에서는 SAN환경에서 Lightweight Messaging 기법의 일종인 VIA를 이용하는 응용 프로그램의 작성을 용이하게 해주는 라이브러리, 즉 EVIL을 구현하고 그 성능을 평가하였다.

EVIL을 사용하였을 때 Native VIA의 API 세트인 VIPL을 이용하여 작성한 프로그램의 약 1/3에 해당하는 코딩으로 같은 작업을 수행하는 프로그램을 작성할 수 있었으며, 그 성능 또한 크게 차이가 나지 않았다. 코딩의 양이 줄었다는 것은 그만큼 응용 프로그램의 개발 속도가 빨라짐을 의미하며 디버깅이 용이하여 졌음을 의미한다.

특히 EVIL에서는 매우 간단하면서도 직관적인 함수들을 이용하므로, VIA를 이용한 프로그램을 작성하는 것이 매우 간편해 졌다. 간편함을 추구하기 때문에 성능 저하가 우려되었지만, 실험을 통해 확인한 결과 Giganet 사에서 제공하는 예제 프로그램에 비해 성능이 크게 떨어지지 않았으며 따라서 Lightweight Message로써의 기능을 충분히 하고 있음을 확인할 수 있었다.

향후에는 파일 전송을 위한 보다 다양한 흐름 제어 방식들을 내부적으로 적용하여 작은 크기의 메시지의 전송에도 높은 성능을 발휘하여, 전체적으로 더 안정적이며, 높은 처리율을 나타내는 연구가 계속될 것이다.

참고 문헌

[1]"리눅스 상의 VIA 구현 비교", 김강호, 김진수, 김해진, 한국정보과학회 학술발표논문집, 2000년 10월(제27권 2호), Page(s): 627 -629.

[2]"VIA기반의 병렬 라이브러리에 관한 연구", 하순회, 기양석, 김선재, 정보과학회지, 2000년 3월, Page(s): 28 -39

[3]"cLan for Linux, Software User's Guide", Emulex, 2001

[4]"High Performance Cluster Computing", Rajkumar Buyya, Prentice Hal, Vol. 1, 1999.

[5]"The Virtual Interface Architecture", D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd. IEEE Micro. March/April 1998.

[6]"VI Architecture Software Developer's Guide", Emulex, 2001.

[7]"Virtual Interface Architecture Specification", draft revision 1.0, 1997.

[8]"Virtual Interface (VI) Architecture : The New Open Standard for Distributed Messaging Within a Cluster", Compaq, 1998.

[9]"xBSP: an efficient BSP implementation for cLAN", Yangsuk Kee, Soonhoi Ha, Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on, 2001. Page(s): 237 -244.