기가비트 이더넷상에서의 M-VIA 구현
(M-VIA Implementation on a Gigabit Ethernet Card)

윤 인 수 † 정 상 화 ††
(In-Su Yoon) (Sang-Hwa Chung)

요 약 클러스터들을 연결시키는 통신 모델로 업계 표준인 VIA(Virtual Interface Architecture)가 있다. VIA의 소프트웨어적인 구현으로는 M-VIA를 대표적으로 들 수 있다. 본 논문에서는 TCP/IP를 지원하는 기존의 AceNIC 기가비트 이더넷 카드의 디바이스 드라이버에 수정을 가하여 M-VIA를 지원할 수 있도록 구현하였다. 그리고 M-VIA의 데이터 세그멘테이션 과정을 분석하여 기가비트 이더넷 카드가 1514 bytes이상의 MTU를 지원할 경우, 기존의 M-VIA 데이터 세그멘테이션 크기가 가지는 문제점을 보이며 이를 개선하기 위해 MTU와 M-VIA 데이터 세그멘테이션 크기를 다르게 해서 실험하였고 그 성능을 비교하였다.

키워드 : M-VIA, 기가비트 이더넷, PC 클러스터, 병렬처리

Abstract The Virtual Interface Architecture(VIA) is an industry standard for communication over system area networks(SANs). M-VIA is a software implementation of VIA technology on Linux. In this paper, we implemented the M-VIA on an AceNIC Gigabit Ethernet by developing a new AceNIC driver for the M-VIA. We analyzed the M-VIA data segmentation processes. When a Gigabit Ethernet MTU is larger than 1514 bytes, M-VIA data segmentation size leaves much room for improvement. So we experimented with various MTU and M-VIA data segmentation size and compared the performances.

Key words : M-VIA, Gigabit Ethernet, PC Cluster, Parallel Processing

1. 서 론

현재 저렴한 PC들 혹은 웍스테이션급의 컴퓨터들을 고성능의 네트웍으로 연결하는 클러스터 기술이 나날이 발전하고 있다. 이러한 네트웍을 구성하는 하드웨어들 중에서 기가비트 이더넷은 높은 대역폭을 제공하며, 서버급의 연결에 사용되는 다른 네트웍 기기들에 비해 가격이 저렴하다는 장점을 지니고 있다.

한편, 통신망 자체가 기가비트급으로 급격히 발전하더라도 통신에 사용되는 TCP/IP와 같은 소프트웨어가 많은 부하를 가지면 실제 사용자 프로그램에서 사용 가능한 성능은 통신망의 물리적 성능에 훨씬 미치지 못하게 된다. 이러한 현상의 주요한 원인은 데이타 송수신시 발생하는 커널로의 문맥 전환(context switch)과 TCP/IP 프로토콜 스택을 이동하면서 생기는 데이타 복사가 있다. 이러한 문제점들을 해결하기 위해 사용자 수준 통신(user-level communication) 모델들이 제시되었다. 사용자 수준 통신 모델은 통신을 커널이 아닌 사용자 수준에서 처리하게 하고 통신 계층을 단순화시킴으로써 커널 내부에서 소요되는 시간과 통신 중에 발생하는 데이타 복사 회수를 최소화시킨다. 사용자 수준 통신 모델의 대표적인 연구로는 U-Net[1], Fast Message[2], Active Message[3] 등이 있다. 이러한 방법론들의 표준으로 Compaq, Intel, Microsoft사 등이 참여하여 산업계 표준인 Virtual Interface Architecture(VIA)[4]가 등장하게 되었다. 그 이후 VIA를 소프트웨어적으로 혹은 하드웨어적으로 구현한 성과들이 많이 나왔는데 소프트웨어측으로는 M-VIA[5], Berkeley VIA[6] 등을 들 수 있겠고, 하드웨어측으로는 Emulex사의 cLan[7] 등이 있다. 이러한 VIA의 구현들 중에서, M-VIA는 기

* 본 연구는 정보통신부의 정보통신학술기초연구지원사업(과제번호 : 2001-005-3)으로 수행한 연구결과입니다
† 비 회 원 : 부산대학교 컴퓨터공학과
isyoon@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
shchung@pusan.ac.kr
논문접수 : 2002년 5월 23일
심사완료 : 2002년 10월 25일

가비트 혹은 Fast 이더넷을 하위 통신 기반으로 이용하고 있으며, Berkeley VIA는 Myrinet[8]을, cLAN은 독자적인 네트웍을 이용한다.

현재까지 M-VIA에서 공식적으로 지원하고 있는 기가비트 이더넷 카드는 총 4개이다. Packet Engines사의 GNIC-I, II, Intel사의 PRO/1000, Syskonnect사의 SK-98xx 계열 기가비트 이더넷 카드가 그것이다. 본 논문에서는 여기에 TCP/IP를 지원하는 기존의 AceNIC 기가비트 이더넷 카드의 디바이스 드라이버에 수정을 가하여 M-VIA를 지원할 수 있도록 구현, 추가하였다. 현재 AceNIC 네트웍 칩은 3Com, NetGear, Nortel등의 회사의 기가비트 이더넷 카드에 장착되어 널리 판매되고 있다. 본 논문에서 사용된 M-VIA 시스템은 M-VIA 버전 1.2 베타2[9] 에 준하며 AceNIC의 디바이스 드라이버는 acenic-0.84 버전[10]을 이용하였다.

AceNIC은 물리적인 MTU를 1514 bytes부터 9000 bytes(점보 프레임)까지 조정이 가능한 반면, M-VIA에서는 일반적인 이더넷 카드의 MTU(1514 bytes)에서 M-VIA 헤더(42 bytes)를 뺀 1472 bytes로 데이타 세그멘테이션 크기를 고정하고 있다. 이에 본 논문에서는 1514 bytes이상의 물리적인 MTU를 지원하는 기가비트 이더넷상에서 효율적인 M-VIA 데이타 세그멘테이션 크기를 찾기 위해 M-VIA의 데이타 세그멘테이션 과정을 분석하였고 이를 바탕으로 여러 가지 데이타 세그멘테이션 크기로 실험을 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 M-VIA 구조에 대한 개요를 보일 것이고 3장에서는 네트웍 카드의 MTU에 따른 효율적인 M-VIA의 데이타 세그멘테이션 방법을 보일 것이며, 4장에서는 AceNIC상에서 M-VIA를 구현한 시스템에 대한 실험과 성능을 분석하고 마지막 5장에서는 결론을 보인다.

2. M-VIA 구조 개요

VIA는 크게 VI Provider Library(VIPL), VI Kernel Agent, VI NIC으로 구성된다. VIPL은 데이타 전송과 노드들간의 연결설정, 큐 관리, 메모리 등록, 에러 처리 등을 담당하는 함수들로 구성되고, VI Kernel Agent는 VI 연결을 관리하고 메모리 등록에 필요한 커널 서비스를 제공한다. VI NIC은 실제적인 데이타 전송을 수행하는데 이러한 NIC은 하드웨어만으로 구성될 수도 있으며 하드웨어와 소프트웨어의 조합으로 구성될 수도 있다. 사용자는 우선 VIPL을 통해 통신하고자 하는 노드간의 연결(VI)을 VI Kernel Agent의 도움을 받아 설정한다. 설정된 VI는 송신 큐와 수신 큐의 쌍으로 이루어진 Work 큐로 구성된다. 그 이후의 송신 및 수신요청은 시스템 호출없이 바로 VI의 Work 큐에 송신과 수신에 필요한 정보를 담은 VI descriptor를 post함으로써 이루어진다. [그림 1]은 이러한 VIA 통신모델을 보인다.

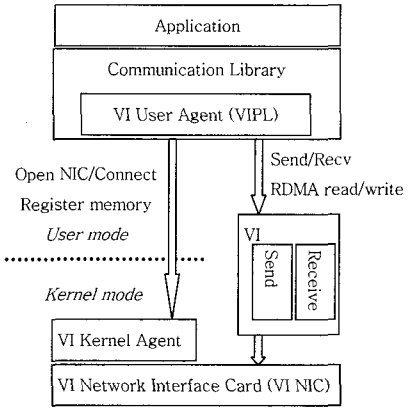

그림 1 VIA 통신모델

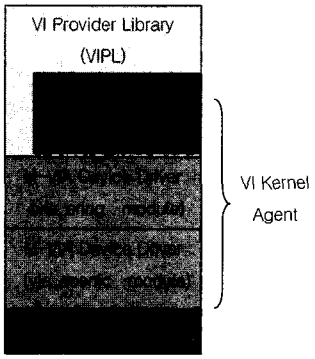

그림 2 AceNIC 사용시의 M-VIA 구조

M-VIA(Modular VIA)는 리눅스 운영체제 하에서 VIA를 소프트웨어적인 모듈들로 구현한 것이다. M-VIA는 사용자 수준의 라이브러리와 리눅스에 적재될 수 있는 적어도 두 가지의 커널 모듈로서 구현된다. 이 중 코아 모듈(M-VIA Kernel Agent)은 장치 독립적이며 VIA에 필요한 커널 서비스를 제공하는 역할을 한다. 그리고 디바이스 모듈(M-VIA Device Driver)은 장치 의존적인 모듈로서 장치 특유의 기능을 M-VIA에 맞게 구현하는 역할을 한다. 디바이스로 이더넷 카드를 택할 경우 M-VIA에서는 이더넷 계열 장치들에 공통된 부분들을 추상화한 via_ering 모듈과 각각의 이더넷 카드의 디바이스 드라이버에 M-VIA의 기능이 추가된 모듈, 이렇게 두 개의 디바이스 모듈로 구성된다. 본 논문에서는

TCP/IP를 지원하는 기존의 AceNIC 기가비트 이더넷 카드의 디바이스 드라이버에 수정을 가하여 M-VIA를 지원할 수 있도록 [그림 2]에서 보이는 via_acenic 모듈을 구현하였다. 또한 [그림 2]의 via_ering 모듈에서 데이타 세그멘테이션 관련 기능들을 담당하고 있는데 이것을 분석, 수정하여 물리적인 MTU가 1514 bytes이상 지원하는 기가비트 이더넷상에서 효율적인 M-VIA 데이타 새그멘테이션 크기를 찾기 위한 실험을 하였다.

3. 네트웍 카드의 MTU에 따른 효율적인 M-VIA 데이타 세그멘테이션

M-VIA는 Fast 이더넷, 기가비트 이더넷 둘 다를 지원할 수 있게끔 디자인 되었다. Fast 이더넷과 몇몇 기가비트 이더넷의 경우 최대 MTU는 1514 bytes로 고정되어 있다. 따라서 M-VIA에서는 모든 이더넷 카드가 지원할 수 있는 MTU인 1514 bytes에 맞게, 전송하고자 하는 데이타를 1472 bytes단위로 세그멘테이션하고 이렇게 잘려진 데이타들 앞에 42 bytes의 데이타 헤더를 붙여 이더넷 카드의 MTU 단위에 맞게 전송하게 된다. 3.1-2절에서는 9000 bytes까지 MTU를 조정할 수 있는 AceNIC 기가비트 이더넷카드상에서 효율적인 M-VIA 데이타 세그멘테이션 크기를 찾기 위해 M-VIA의 데이타 세그멘테이션 과정을 분석하였고 이를 바탕으로 여러 가지 데이타 세그멘테이션 크기와 물리적인 MTU간의 관계를 보인다.

3.1 M-VIA에서의 데이타 세그멘테이션

사용자는 데이타를 송신하고자 할 때 VI descriptor의 데이타 세그먼트 부분에 보낼 데이타의 가상 주소와 길이 정보를 담아 해당 VI의 Work 큐에 post한다. 하나의 VI descriptor는 보내고자 하는 데이타 영역이 많을 때, 그 영역들의 정보를 담고 있는 다수의 데이타 세그먼트를 descriptor안에 둘 수 있다. 그러면 M-VIA 디바이스 드라이버중 via_ering 모듈이 descriptor의 내용을 읽어 M-VIA의 세그멘테이션 단위로 데이타를 자른다. 이렇게 나누어진 데이타는 하위 NIC의 디바이스 드라이버에서 M-VIA와 관련된 송신함수들을 호출함으로써 최종적으로 전송이 된다. [그림 3]은 VIPL Post Send에 관련된 일련의 함수 호출 순서를 나타낸다.

[그림 3]의 via_ering 모듈에서 보이는 SEND_HDR, SEND_SEG, SEND_END 함수들은 송신에 관련된 것들로서, via_ering 모듈내에서는 이러한 함수들을 구현하고 있지 않은데, 그 이유는 각 이더넷 카드마다 그 하드웨어적인 특성이 다르기 때문이다. 따라서 기존의 이더넷 카드들이 M-VIA를 지원하기 위해서는 이러한 함

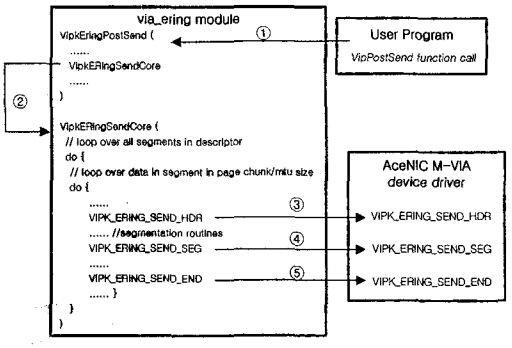

그림 3 VIPL PostSend에 관련된 일련의 함수호출 순서

수들을 자신의 하드웨어 특성에 맞도록 디바이스 드라이버내에 구현하는 것이 필요하다.

[그림 3]에서 보이듯이 M-VIA에서 데이타를 보낼 때는 VI descriptor내에 있는 모든 데이타 세그먼트 부분에 대해 루프를 돌며, 각 세그멘트가 나타내고 있는 데이타에 대해서는 우선 페이지 크기(4Kbytes)로 나누고 이렇게 나누어진 부분을 다시 M-VIA의 데이타 세그멘테이션 단위로 나누는 작업을 반복하게 된다. 또한 이러한 작업들에는 앞서 언급한 SEND_HDR, SEND_SEG, SEND_END와 같은 함수들을 반복적으로 호출하게 된다. 각 함수들의 역할을 살펴보면 VIPK_ ERING_SEND_HDR 함수는 M-VIA 데이타 헤더에 대한 물리적인 주소와 길이를 AceNIC의 descriptor에 적고, VIPK_ERING_SEND_SEG는 실제로 전송하고자 하는 데이타의 물리적인 주소와 길이를 NIC의 descriptor에 적으며, VIPK_ERING_SEND_END는 NIC에게 보낼 준비가 되었다는걸 알리기 위해 AceNIC descriptor내의 플래그를 세팅한다. 이러한 함수들의 반복적인 호출은 통신에 소요되는 오버헤드를 증가시킨다. 따라서 데이타들이 세그멘테이션되는 횟수를 줄임으로써 이러한 반복적인 함수의 호출을 줄일 수 있다. 세그멘테이션 되는 횟수를 줄이는 방법으로 페이지 크기와 데이타 세그멘테이션 단위를 크게 하는 방법을 생각해 볼 수 있는데, 페이지 크기는 x86 아키텍쳐에서 4KB로 고정되어 있다고 볼 때, M-VIA 데이타 세그멘테이션 단위를 기존의 것보다 크게 하는 걸 생각해 볼 수 있다.

3.2 구체적인 세그멘테이션 과정

M-VIA의 세그멘테이션 과정을 좀 더 자세히 보면, 우선 데이타를 페이지 크기(4KB)로 나누고 이를 다시 M-VIA 세그멘테이션 단위인 1472 bytes로 자른다. 만약 12KB의 데이타를 보내게 된다면 [그림 4]처럼 잘라

보내게 된다. 즉 일반적인 네트웍 카드의 MTU인 1514 bytes 에서 M-VIA 데이타 헤더부분을 뺀 1472 bytes 단위로 데이타를 잘라 보낸다. 만약 페이지의 경계에 걸리게 되면 페이지의 끝까지 하나의 세그먼트(예. seg 3-1)를 만들어 보낸다. 그리고 네트웍 카드의 MTU보다 모자란 데이타는 세그먼트(예. seg 3-2)를 하나 더 만들어서 MTU를 채워 보내게 된다.

AceNIC에서 지원하는 최대 MTU인 9000 bytes에서 M-VIA의 데이타 헤더부분을 뺀 8958 bytes로 세그멘테이션 단위를 조정하여 12KB의 데이타를 보낼 경우, [그림 5]처럼 잘라 보내게 된다. 이 때는 1472 bytes 때보다는 세그멘테이션을 적게 하지만 역시 8958 bytes를 채우기 위해 페이지의 경계를 넘어선 부분에 대해서는 추가로 세그멘테이션하는 비용을 지니게 된다.

그러면 M-VIA의 세그멘테이션 단위를 4096 bytes로 하고 여기에 데이타 헤더 42 bytes를 더한 4138 bytes를 AceNIC의 MTU로 설정한다면 1 페이지당 1 세그먼트가 존재하게 된다. 이 경우 12KB 데이타의 세그멘테이션 예는 [그림 6]과 같게 된다.

이러한 세그멘테이션 단위의 변화에 따른 실험과 성능변화를 다음에서 보인다.

4. 실험 및 분석

4.1 실험 환경

본 논문에서는 성능 측정을 위하여 800MHz 펜티엄 III, 256MB의 100MHz SDRAM, 33MHz/32bit PCI 버스를 사용하였으며 네트웍 카드로는 AceNIC의 Tigon II 칩에 기반한 3Com사의 3C985B-SX 모델을 사용하였다. 운영체제로는 리눅스 2.4대의 커널에 기반한 RedHat 7.2 배포판을 사용하였다. 성능 측정은 M-VIA에서 제공하는 vnettest.c 프로그램을 이용하였다. 각 실험치는 100회 반복한 평균치이다. 지연시간은 왕복시간의 반으로 하였으며, 대역폭은 전송한 데이타의 총량을 지연시간으로 나누어 구한 것이다. 또한 M-VIA에서는 한 번의 PostSend 함수호출로 최대 전송 가능한 데이타의 크기가 32KB로 한정되어 있는데 이 부분을 수정하여 사용자가 한 번의 PostSend 함수호출로 최대 64KB까지 데이타를 전송할 수 있게끔 via_ering 모듈을 수정하였다.

4.2 AceNIC상에서 M-VIA 데이타 세그멘테이션 크기에 따른 성능 측정

[그림 7, 8]은 M-VIA의 데이타 세그멘테이션 크기를 각각 1472, 4096, 8958 bytes로 하고, 이에 M-VIA 데이타 헤더(42 bytes)를 더한 값인 1514, 4138, 9000 bytes로 AceNIC의 물리적인 MTU를 조정하여 실험했

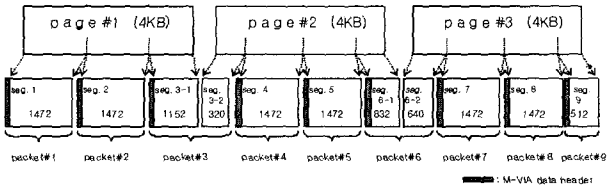

그림 4 1472 bytes 단위의 M-VIA 세그멘테이션

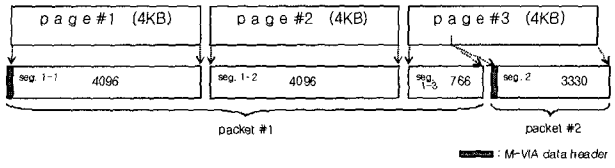

그림 5 8958 bytes 단위의 M-VIA 세그멘테이션

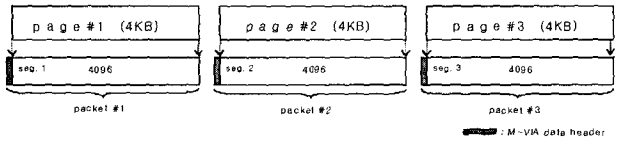

그림 6 4138 bytes 단위의 M-VIA 세그멘테이션

을 때의 지연시간과 대역폭 그래프를 나타낸다. 실험에 사용한 데이타는 최초 32 bytes부터 1024 bytes까지는 앞 데이타 크기의 배수(32, 64, 128, …, 1024)로 크기를 증가시켰으며, 1024 bytes이후부터는 1024 bytes씩 증가(1024, 2048, 3072, …, 65536)시켰다.

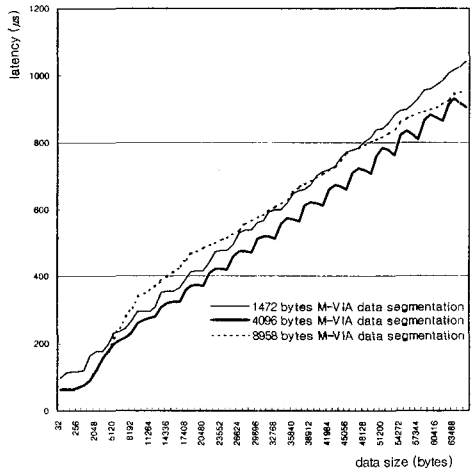

그림 7 M-VIA 데이타 세그멘테이션 크기에 따른 지연시간(latency) 비교

우선 [그림 7]의 지연시간 그래프를 보면 4096 bytes로 M-VIA 데이터 세그멘테이션 크기를 설정하고 4138 bytes로 AceNIC의 MTU를 설정한 것이 1472 bytes 세그멘테이션 크기와 1514 bytes MTU의 조합과 8958 bytes 세그멘테이션 크기와 9000 bytes MTU의 조합에 비해 낮은 지연시간으로 좋은 특성을 보인다는 것을 알 수 있다. 그리고 4096 bytes로 M-VIA 데이터 세그멘테이션 크기를 설정했을 경우, 그래프에서 지그재그 형태를 뚜렷이 나타낸다. 그래프의 반복되는 가파른 지연시간 상승부분은 보내는 데이터의 크기가 4096 bytes의 배수를 1024 bytes만큼 초과된 부분이다. 이 때 via_ering 모듈에서는 세크멘테이션을 추가로 한 번 더 하게 되고 그에 따른 이더넷 패킷을 한 번 더 전송하게 됨으로써 생기는 지연시간 상승이다. 이렇게 가파른 지연시간의 상승을 보이고 나서는, 보내고자 하는 데이터의 크기가 증가하여 다시 4096 bytes의 배수가 되는 때까지는 지연시간의 상승을 보이지 않는다. 8958 bytes M-VIA 데이터 세그멘테이션 크기와 9000 bytes MTU의 조합의 경우, 보내는 데이터의 크기가 26000 bytes까지는 다른 것과 비교해 지연시간이 높게 나오고, 48000 bytes 데이터 크기까지는 1472 bytes로 M-VIA 데이터 세그멘테이션 크기를 설정한 것과 지연시간이 거의 비슷하며, 그 이상의 데이터에 대해서는 지연시간이 점점 낮아지며 64KB의 데이터를 보내는 부근에서는 4096 bytes로 M-VIA 데이터 세그멘테이션 크기를 설정한 것과도 지연시간이 비슷해진다. 이는 8958 bytes M-VIA 데이터 세그멘테이션 크기와 9000 bytes MTU의 조합의 경우, 자신의 9000 bytes MTU를 다 채울 때까지는 패킷을 보내지 않고 대기하고 있는데 반해 나머지의 경우는 MTU가 이에 비해 작기 때문에 보내고자 하는 데이터를 더 작은 단위로 세그멘테이션하여 좀 더 빨리 보낼 수 있기 때문이다. 그러나 보내고자 하는 데이터가 커질 경우, 점점 이러한 대기하는 시간에 의해 생기는 지연시간 증가보다 다른 경우와 비교하여 더 큰 데이터를 하나의 패킷에 실어 보낼 수 있어 수신측의 패킷처리 비용을 줄여 지연시간을 감소시키는 이득이 더 크게 되기 때문에 [그림 7]의 48000 bytes 데이터 크기 이상에서 이러한 결과가 나타난다.

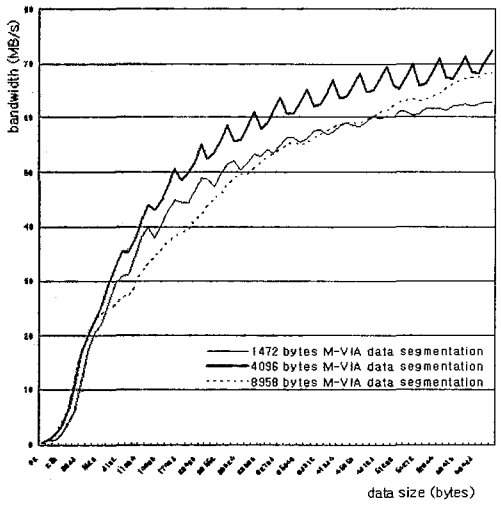

그림 8 M-VIA 데이타 세그멘테이션 크기에 따른 대역폭(bandwidth) 비교

[그림 8]의 대역폭 그래프는 전송한 데이타의 총량을 지연시간으로 나누어 구한 것이기 때문에 [그림 7]의 결과를 그대로 반영하고 있다. 8958 bytes M-VIA 데이타 세그멘테이션 크기와 9000 bytes MTU의 조합의 경우, 보내고자 하는 데이타가 커질 경우 점점 더 좋은 대역폭을 보이고 있으나 M-VIA에서 한 번의 Post Send 함수 호출로 보낼 수 있는 데이타의 크기를 정해

놓았기 때문에 64KB 이상의 데이타에 대해서는 실험을 할 수 없었다. 실험을 행한 구간에서 가장 좋은 대역폭을 내는 조합은 4096 bytes로 M-VIA 데이타 세그멘테이션 크기를 설정하고 4138 bytes로 AceNIC의 MTU를 설정한 것이다. 이 조합의 경우 [그림 7]의 지연시간 그래프와 같이 지그재그 형태의 그래프를 보여주는데 지그재그 형태의 상단과 하단의 대역폭 차이가 3 MB/s 정도로 나타난다. AceNIC 기가비트 이더넷을 사용하여 M-VIA를 기반으로 하는 응용 프로그램을 작성할 때는 보내고자 하는 데이타를 4096 bytes의 배수로 맞추는 것이 안정적인 대역폭을 낼 수 있다.

4.3 AceNIC상에서 TCP/IP와 M-VIA의 성능비교

[그림 9, 10]은 M-VIA의 데이타 세그멘테이션 크기를 각각 1472, 4096 bytes로 한 것과 TCP/IP와의 성능 비교를 한 것이다. AceNIC의 MTU는 M-VIA의 경우 각각 1514, 4138 bytes로 설정을 했으며, TCP/IP는 이더넷 카드에서 가장 널리 쓰이는 MTU인 1514 bytes로 설정을 했다. TCP/IP의 성능측정에 사용된 응용 프로그램은 소켓 라이브러리를 이용하여 작성하였다. 실험에 사용한 데이타는 4096 bytes부터 4096 bytes씩 증가시켰다.

[그림 9]의 지연시간 그래프를 보면 TCP/IP와 M-VIA의 기가비트 이더넷 MTU를 1514 bytes로 같이 설정했을 때, M-VIA가 TCP/IP보다 더 낮은 지연시간을 보인다. 최초 32 bytes의 데이타를 보낼 때 TCP/IP가 119μs, M-VIA가 87μs로 32μs 정도의 지연시간 차이를 보이고, 데이타의 크기가 커짐에 따라 약 45~74μs 정도의 지연시간 차이를 보인다. 이는 M-VIA가 TCP/IP에 비해 데이타 송수신시 발생하는 커널로의 문맥 전환(context switch)비용을 최소화하고 TCP/IP 프로토콜 스택을 이동하면서 생기는 데이타 복사를 제거하였기 때문이다. 그래프의 전체구간에 대해 평균을 구하면 M-VIA가 TCP/IP에 비해 약 11%의 지연시간 이득을 보인다. 4096 bytes로 M-VIA 데이타 세그멘테이션 크기를 설정하고 4138 bytes로 AceNIC의 MTU를 설정한 경우와 TCP/IP를 비교하면 지연시간의 차이는 평균적으로 약 133μs, 지연시간 이득은 약 23% 정도를 얻을 수 있었다. 기존의 M-VIA가 일반적인 이더넷 카드의 MTU인 1514 bytes를 지원하도록 1472 bytes로 세그멘테이션 크기를 고정시킨데 비해, 이더넷 카드가 1514 bytes이상의 MTU를 지원할 수 있고 그에 따라 M-VIA의 세그멘테이션 크기를 조정했을 때 얻어진 결과이다.

[그림 10]의 대역폭 그래프에서 64KB의 데이타를 보낼 시 TCP/IP가 58.5 MB/s, 1472 bytes M-VIA 데이타 세그멘테이션 크기를 사용할 시는 62.7 MB/s, 4096 bytes 시는 72.4 MB/s의 최대 대역폭을 보인다. TCP/IP와 M-VIA 데이타 세그멘테이션 크기가 1472 bytes인 경우는 평균 4.8 MB/s, 4096 bytes인 경우는 평균 12.9 MB/s 정도의 대역폭 차이가 난다.

5. 결 론

본 논문에서는 TCP/IP를 지원하는 기존의 AceNIC 기가비트 이더넷 카드의 디바이스 드라이버에 수정을

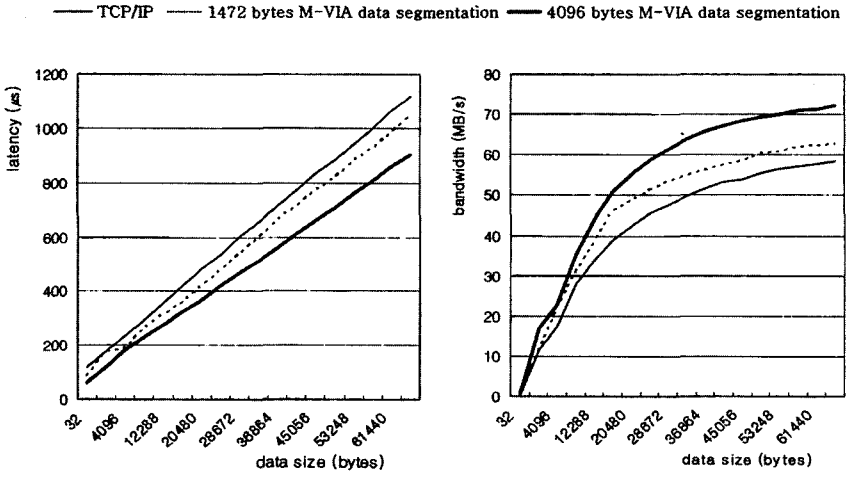

그림 9 TCP/IP와 MVIA의 지연시간 비교

그림 10 TCP/IP와 MVIA의 대역폭 비교

가하여 M-VIA를 지원할 수 있도록 구현하였고 M-VIA의 데이타 세그멘테이션 과정을 분석하여 이더넷 카드가 1514 bytes이상의 MTU를 지원할 때 그에 맞는 M-VIA 데이타 세그멘테이션 크기를 제안하여 실험하였다.

1514 bytes의 일반적인 이더넷 MTU를 쓰는 TCP/IP와 비교하여 본 논문에서 제안한 4096 bytes로 M-VIA 데이타 세그멘테이션 크기를 설정하고 4138 bytes로 AceNIC의 MTU를 설정한 경우, 대역폭 면에서는 평균 12.9 MB/s 정도의 이득을 얻을 수 있었고 지연시간 면에서는 평균 23% 정도의 이득을 얻을 수 있었다. M-VIA에서는 한 번의 PostSend 함수호출로 최대 전송 가능한 데이타의 크기가 32KB로 제한되어 있는데, 더 큰 데이타를 보낼 수 있게끔 이 부분이 개선된다면 8958 bytes의 M-VIA 데이타 세그멘테이션 크기와 9000 bytes MTU 조합의 성능이 64KB 이상의 데이타를 보낼 때 우수한 성능을 보일 것으로 예상된다.

참 고 문 헌

[1] T. Von Eicken, A. Basu, V. Buch, W. Vogels, "U-NET: A User Level Network Interface for Parallel and Distributed Computing", Proc. of the 15th SOSP

[2] S. Pakin, M. Lauria, A. Chien, "High Performance Messaging on Workstation: Illinois Fast Message (FM) for Myrinet", Proc. of Supercomputing'95

[3] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, Klaus Erik Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation", 19th International Symposium on Computer Architecture

[4] Intel, Compaq and Microsoft Corporations, Virtual Interface Architecture specification version 1.0, December 1997, http://developer.intel.com/design/servers/vi/

[5] Patrick Bozeman, Bill Saphir, "A Modular High Performance Implementation of the Virtual Interface Architecture", Proc. of the 2nd Extreme Linux Workshop, June 99

[6] P. Buonadonna, A. Geweke, D.E. Culler, "An Implementation and Analysis of the Virtual Interface Architecture.", Proc. of SC '98, Orlando, FL, Nov. 7-13, 1998

[7] Emulex Corporation, hardware-based (ASIC) implementation of the Virtual Interface (VI) standard, http://www.giganet.com/products/vi/clan1000.html

[8] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, et al., "Myrinet-A Gigabit Per Second Local Area Network", IEEE Micro, Feb. 1995

[9] M-VIA, http://www.nersc.gov/research/FTG/via/

[10] AceNIC Gigabit Ethernet for Linux, http://jes.home.cern.ch/jes/gige/acenic.html

[11] Mark Baker, Paul A. Farrell, Hong Ong, Stephen L .Scott, "VIA Communication Performance on a Gigabit Ethernet Cluster", Proc. of 7th International Euro-Par Conference, Manchester, United Kingdom, August 2001

[12] Hong Ong, Paul A. Farrell, "Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network", Proc. of the 4th Annual Linux Showcase & Conference, Atlanta, Georgia, USA, October 2000

[13] Piyush Shivam, Pete Wyckoff, Dhabaleswar Panda, "Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing", Proc. of SC2001, Denver, Colorado, USA, November 2001

윤 인 수

2001년 부산대학교 전자컴퓨터공학부 학사. 2001년 ~ 현재 부산대학교 컴퓨터공학과 석박사통합과정 재학중. 관심분야는 클러스터시스템, 병렬처리, VIA

정 상 화

정보과학회논문지 : 시스템 및 이론 제 29 권 제 10 호 참조