초고속 Myrinet 통신망에서의 PVM 성능 개선

김 인 수† · 심 재 홍†† · 최 경 희††† · 정 기 현†††† · 문 경 덕† · 김 태 근†††††

요 약

본 논문에서는 초고속 통신망인 Myrinet을 기반으로 하는 워크스테이션 연동(network of workstations) 환경하에서 병렬 프로그래밍 모델을 지원하는 PVM (parallel virtual machine)의 통신 성능 개선 방안을 제시한다. PVM을 위해 새로이 제안하는 Myrinet 기반 통신 모델은 커널을 경유하는 기존의 UDP/IP 프로토콜을 이용하는 통신 모델과 Myrinet API를 직접 이용하는 통신 모델을 혼합한 복합 통신 모델이다. 제안된 복합 통신 모델은 사용자 영역(커널)에서 커널(사용자) 버퍼로의 메시지 복사 부하를 제거하고 커널내의 프로토콜 스택 처리를 위한 통신 지연 요인을 감소시키므로, Myrinet상에서 보다 빠른 PVM 태스크간의 데이터 전송 속도를 보장한다. 또한, UDP/IP를 사용하는 기존 PVM에 본 논문에서 제안된 Myrinet 기반 복합 통신 모델을 적용시킨 EPVM (Enhanced PVM)을 구현하고, 이의 성능을 측정하였다. 실험 결과 EPVM의 통신 성능이 기존 PVM 보다 평균 1.5배 정도 우수하다는 것을 확인하였다.

PVM Performance Enhancement over a High-Speed Myrinet

Insoo Kim† · Jae-Hong Shim†† · Kyung-Hee Choi††† · Gi-Hyun Jung†††† ·
Kyeong-Deok Moon† · Tae-Geun Kim†††††

ABSTRACT

PVM (parallel virtual machine) provides a programming environment that allows a collection of networked workstations to appear as a single parallel computational resource. The performance of parallel applications in this environment depends on the performance of data transfers between tasks. In this paper, we present a new Myrinet-based communication model of PVM that improves PVM communication performance over a high-speed Myrinet LAN. The proposed PVM communication model adopts a communication mechanism that allows any user-level process to directly access the network interface board without going through UDP/IP protocol stacks in the kernel. This mechanism provides faster data transfers between PVM tasks over the Myrinet since it avoids data copy overhead from kernel (user space) to user space (kernel) and reduces communication latency due to network protocol software layers. We implemented EPVM (Enhanced PVM), our updated version of the traditional PVM using UDP/IP, that is based on the proposed communication model over the Myrinet. Performance results show that EPVM achieves communication speed-up of one to two over the traditional PVM.

1. 서 론

수년간 Ethernet 수준에 머물러 있던 컴퓨터 통신은 최근 몇 년 동안 급속한 발전을 이루어, HIPPI, FDDI, Fast Ethernet, ATM, 그리고 Myrinet 등과 같은 초고속 통신망이 출현하게 되었다. 이러한 초고속 근거리 통신망의 출현과 마이크로 프로세서의 성능 향상은 고성능을 요구하는 계산-중심(computation-intensive)의 문제를 풀기위한 기반 시스템으로서 웍스테이션 연동(workstation clusters)을 가능하게 하였다. 메시지 전달을 기반으로 하는 소프트웨어 툴인 *PVM(parallel virtual machine)* [1, 2]은 네트워

† 정 회 원 : 한국전자통신연구원 연구원
†† 정 회 원 : 아주대학교 대학원 정보 및 컴퓨터공학부
††† 정 회 원 : 아주대학교 정보 및 컴퓨터공학부 교수
†††† 정 회 원 : 아주대학교 전기전자공학부 교수
††††† 정 회 원 : 한국전자통신연구원 책임연구원
논문접수 : 1998년 12월 3일, 심사완료 : 1999년 12월 19일

크에 연결된 많은 이기종 웍스테이션들이 단일 병렬 계산 자원(single parallel computation resource)으로 사용자에게 인식되게 하는 병렬 프로그래밍 환경을 제공한다. PVM은 TCP/IP 네트워크 프로토콜을 기반으로 하여 데이터 통신과 프로세스 제어 기능들을 제공한다. 널리 사용되는 TCP/IP를 이용하여 구현되었다는 범용성으로 인하여, PVM은 현재 다양한 고속 통신망으로 연결된 고성능 웍스테이션 연동 분산 컴퓨팅을 제공하기 위한 de facto 표준으로서 널리 사용되고 있다.

PVM상에서 구현된 응용프로그램들의 성능은 웍스테이션들간의 데이터 교환을 위한 통신 성능에 상당히 의존적이며, 이러한 응용프로그램들은 크게 태스크 병렬(task parallel) 응용프로그램과 데이터 병렬(data parallel) 응용프로그램등 두 가지 범주로 나눌 수 있다. 태스크 병렬 응용프로그램의 경우 동기화(synchronization) 메시지의 통신 지연(latency)은 전체 응용프로그램 성능에 절대적인 영향을 미친다. 반면, 데이터 병렬 응용프로그램의 경우 데이터는 PVM 환경하의 모든 호스트에 분할되어 분산 배치되며, 각 호스트에서 수행하는 동일한 PVM 프로세스는 분할된 분산 데이터에 동일한 명령어를 실행한다. 작업 도중 PVM 프로세스들은 수정된 데이터 값을 교환하기 위해 상호간 통신을 하게 되며, 이 경우 대량의 데이터 전송이 일어난다. 이러한 웍스테이션 상호간의 대량의 데이터 교환은 빈번하게 일어나며, 이는 곧 전체 시스템 성능의 병목현상으로 작용한다. 따라서 태스크 병렬 응용프로그램과 데이터 병렬 응용프로그램을 위한 데이터 전송상의 통신 부하를 줄이기 위해서는 특별히 설계된 고성능-저지연(high-throughput low-latency) 통신망과 통신 메커니즘이 필연적으로 요구되어 진다.

Myrinet [3]은 640Mbps의 전송 속도를 제공하는 초고속 근거리 통신망이다. Myrinet을 기반으로 연결된 웍스테이션 연동 환경하의 각 호스트에서 수행되는 프로세스들은 커널에서 제공하는 TCP/IP 프로토콜 스택을 이용하거나, 또는 사용자 수준의 Myrinet API를 이용하여 Myrinet 인터페이스 보드에 접근할 수 있다. Myrinet API는 어떠한 프로세스들도 커널을 통하지 않고 사용자 수준에서 Myrinet 인터페이스 보드를 직접적으로 접근 가능하게 함으로써, 빠른 전송 속도와 낮은 통신 지연을 제공한다. 본 연구에서는 웍스테이션들을 연동하기 위한 기반 네트워크로 고속의 통신 성능을 제공하는 Myrinet을 선택하였다. 이는 PVM의 성능이 정보교환을 위한 데이터 통신 성능에 상당히 의존적이며, 고성능의 웍스테이션 연동 환경을 구축하기 위해서는 초고속 통신망이 필연적으로 요구되기 때문이다.

웍스테이션 연동 환경하에서의 PVM은 일반적으로 TCP/IP를 기반으로 하는 UNIX 소켓을 사용하여 모든 통신이 이루어 진다. 범용성에 초점을 두고 설계된 TCP/IP 프로토콜 스택은 커널내의 프로토콜 소프트웨어 계층과 데이터 복사 부하로 인하여 데이터 통신상의 성능에 많은 제약을 받게 된다. 따라서 PVM이 상대적으로 통신 지연이 큰 TCP/IP를 기반으로 구현되었다는 것은 곧 Myrinet을 통한 PVM의 데이터 전송 성능은 전적으로 TCP/IP의 성능에 의존한다는 것을 의미한다. 따라서 PVM의 통신 메커니즘과 메시지 전송경로를 수정할 경우 상당한 통신상의 성능 향상이 기대된다 [19].

본 연구는 Myrinet을 기반으로 하는 워크스테이션 연동 환경하에서 전반적인 시스템 성능 향상을 위한 연구의 일환으로 PVM 시그널 데이터 전송 경로와 PVM 메시지 데이터 전송 경로를 구분함으로써, Myrinet을 통한 PVM 데몬과 데몬간의 데이터 통신 성능을 개선하기 위한 방법을 논의한다. 본 연구의 핵심적인 요점은 초고속 통신망인 Myrinet API를 기반으로 하는 빠른 사용자 수준의 새로운 통신 모델을 제안하고, 이를 바탕으로 하여 PVM 서비스의 주 데이터 전송 경로를 교체하는데 있다. 제안된 통신 모델을 사용한 EPVM(Enhanced PVM)의 성능 실험에서 PVM 데몬과 데몬간의 데이터 통신은 TCP/IP를 기반으로 하는 기존 PVM 보다 1배에서 2배의 성능 향상을 보였다.

본 논문의 구성은 다음과 같다. 2장에서 PVM 및 Myrinet에 대해 간단히 논의하고, 3장에서는 Myrinet을 기반으로 하는 PVM의 새로운 통신 모델을 제안한다. 4장에서는 제안된 통신 모델을 채택한 수정된 PVM(EPVM)의 구조와 구현에 대해 기술한다. 5장에서는 구현된 EPVM의 성능 측정 결과에 대해 분석하고, 이와 유사한 PVM 관련 연구를 6장에서 논의한다. 마지막 7장에서는 본 연구의 결론과 향후 계획에 대해 기술한다.

2. PVM과 Myrinet Network의 개요

본 장에서는 병렬 프로그래밍 모델을 지원하는 PVM과 초고속 통신망인 Myrinet에 대해 간단히 알아본다.

먼저 PVM의 기본 구성요소와 태스크간의 메시지 전송 메커니즘에 대해 논의한 후 이의 문제점을 검토하고, 이후 Myrinet이 제공하는 두 가지 메시지 전송 경로에 대해 기술한다.

2.1 PVM (Parallel Virtual Machine)

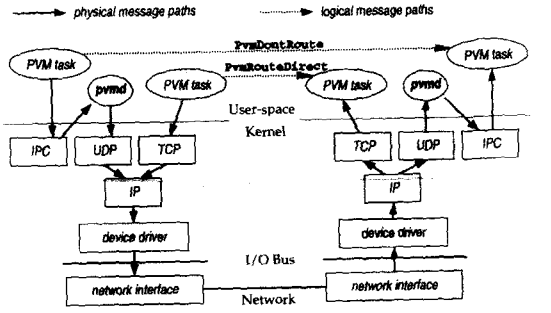
PVM은 메시지 전달 라이브러리(message passing library)를 기반으로 분산된 이기종간의 워크스테이션들을 *단일 가상 자원(single virtual resource)* 또는 단일 *병렬 시스템(single parallel machine)*으로 보이게 하여, 병렬 응용 프로그램의 수행과 개발을 손쉽게 하기 위한 소프트웨어 툴이다 [1, 2].

PVM은 크게 *PVM 데몬(PVM daemon, pvmd), PVM 태스크(PVM task),* 그리고 PVM 태스크를 위한 *PVM 라이브러리(PVM library)* 등으로 구성된다. C또는 Fortran으로 작성된 PVM 라이브러리는 점대점 데이터 전송, 메시지 브로드캐스팅, 상호 배제, 프로세스 제어, 프로세스 동시화(barrier synchronization) 등을 제공하여, 병렬 응용프로그램 개발자에게 손쉬운 프로그래밍 환경을 제공한다. Pvmd는 PVM 환경하에 있는 호스트들간의 메시지 경로 선택 및 메시지 제어 기능을 제공하고, 다양한 종류의 태스크를 여러 호스트에 투명하게 실행시켜 주며, 호스트의 추가 및 삭제를 제어하는 데몬 프로세스이다. PVM 라이브러리를 이용하여 작성된 하나의 PVM 응용프로그램은 여러 개의 PVM 태스크들로 구성되며, PVM 태스크들은 웍스테이션 연동에 참여하는 여러 호스트들에 투명하게 분산되어 동시에 병렬로 수행되는 프로세스들이다.

PVM은 확장성을 유지하기 위해 각 시스템에 하나씩 수행되는 데몬들 사이에는 UDP 소켓을 사용하여 통신하고, 한 시스템내의 지역 태스크들과 pvmd 사이의 통신을 위해서는 UNIX 도메인 스트림 소켓을 사용한다.

일반적으로 PVM은 태스크간 정보 교환을 위해 **PvmDontRoute**와 **PvmRouteDirect** 등 두 가지 메시지 전송 경로를 제공해 준다[1]. (그림 1)은 이들 두 가지 PVM 메시지 전송 경로를 도시화 한 것이다.

PvmDontRoute는 상호간 위치를 모르는 태스크간에 pvmd를 통한 투명한 원거리 통신을 지원해 주는 메시지 전송 경로이다. 이를 위해 pvmd는 각 태스크의 현재 위치를 정확히 파악하고 있어야 하며, 태스크간의



(그림 1) PVM의 두 가지 메시지 전송 경로

통신에 있어 모든 메시지는 pvmd를 통해서 전달된다. 통신을 원하는 지역 태스크(local task)는 위치를 모르는 원거리 태스크(remote task)에게 메시지를 전송하기 위해, 먼저 자신의 시스템에 상존하는 pvmd에게 목적지 태스크 정보와 메시지를 함께 전송한다. 메시지를 수신한 pvmd는 이를 해당 목적지 태스크가 존재하는 시스템의 pvmd에 전송하고, 목적지 시스템의 pvmd는 받은 메시지를 다시 해당 태스크에게 넘겨준다. 이 방법은 통신을 원하는 태스크 상호간에 투명성을 제공하고 이로 인한 태스크 이주(task migration)를 가능하게 하는 장점은 있으나, 모든 메시지가 반드시 pvmd를 통해서 전달되어야 하는 단점이 있다. 이는 곧 과다한 IPC로 인한 운영체제의 부하를 가중시키고 잦은 메시지 복사로 인한 통신지연을 초래한다.

반면, **PvmRouteDirect**는 **PvmDontRoute**와는 달리 pvmd를 경유하지 않고 태스크간에 직접적으로 TCP 소켓을 이용하여 통신하는 방법이다. 따라서 초기 태스크간의 연결 설정 시에만 pvmd가 개입하고, 나머진 pvmd의 협조 없이 태스크 상호간 독자적인 통신이 가능하다. 그러나 상호간 연결이 설정된 태스크는 연결 설정 당시의 시스템에서 다른 시스템으로 이주할 수 없으며, 계속해서 같은 시스템에 상주해야 한다. 이 경우 상대적으로 부하가 적은 새로운 유휴시스템이 생겨도 상호 연결된 태스크들은 유휴시스템으로의 이주를 할 수 없으며, 이는 곧 시스템 자원을 효율적으로 사용하지 못하는 결과를 초래할 수 있다. 또한, 하나의 PVM 태스크가 동시에 여러 PVM 태스크들과 메시지 교환을 원할 경우 통신하고자 하는 각 태스크마다 하나의 TCP 소켓을 필요로 한다. 이 경우 파일 테이블(file table)이나 파일 기술자(file descriptor) 등과 같은 시스템 자원 부족으로 인하여, 소켓 기술자

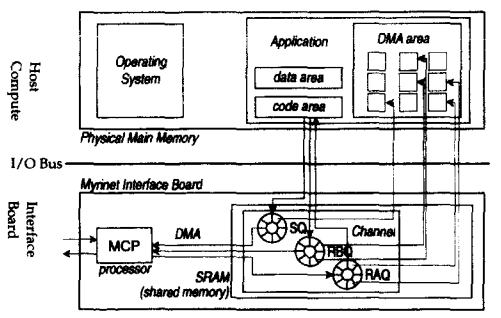
(socket descriptor) 할당에 제약이 따를 수도 있다.

그러나 이들 두 PVM 메시지 전송 경로는 반드시 커널내의 TCP/IP 프로토콜 스택을 경유해야 한다. 이는 커널내의 사용자 영역(커널)에서 커널(사용자 영역)로의 메시지 복사 부하를 유발하게 된다. 따라서 PVM이 상대적으로 통신 지연이 큰 TCP/IP에 기반을 두고 구현되는 한, PVM 내의 나머지 통신 지연 요소(slow list implementation, packing/unpacking)에서 발생하는 시간 손실은 통신 지연의 큰 제약 사항이 되지 않는다. 따라서 기반 네트워크가 초고속 통신망일 경우 메시지 복사 부하는 PVM의 전체적인 성능을 감소시키는 주된 요인이 되며, 이러한 메시지 전송 경로를 수정할 경우 상당한 통신상의 성능 향상이 기대된다[19].

2.2 High-speed Myrinet LAN

Myrinet은 Myricom사에 의해 공급되는 초고속 통신망이다[3]. 이는 저가의 비용으로 640 Mbps의 전송속도를 제공하며, Myrinet 인터페이스 보드와 네트워크 스위치로 구성된 스위치-기반(switch-based) 근거리 통신망(LAN)이다[4].

Myrinet 인터페이스 보드는 호스트를 네트워크에 연결시켜 주는 장치이며, 듀얼-포트(dual-port) 메모리를 가진 하나의 프로세서로 구성된다. (그림 2)는 인터페이스 보드의 구조와 호스트 컴퓨터와의 통신 메커니즘을 도시화 한 것이다. 듀얼-포트 온-보드(on-board) 메모리는 온-보드 프로세서와 호스트 컴퓨터 모두에 의해 접근(access)이 가능하다. 호스트 컴퓨터는 프로그램 가능한 I/O(programmable I/O, PIO)에 의해 온-보드 메모리를 읽거나 쓸 수 있다. 데이터는 호스트 인터페이스 보드에 의해 지원되는 DMA(direct memory access)를 통하여 온-보드 메모리와 호스트 주기억장치 사이를



(그림 2) Myrinet 인터페이스 보드와 채널

이동한다. 온-보드 프로세서에서 수행되는 Myrinet 제어 프로그램(Myrinet control program, MCP)은 PIO를 사용하여 호스트 컴퓨터로부터 인터페이스 보드로 로드될 수 있다. MCP는 호스트상에서 실행되는 여러 사용자 프로세스가 동시에 온-보드 메모리로 데이터를 전송할 수 있도록 하며, 각각의 프로세스들이 독립적으로 인터페이스 보드에 접근할 수 있게 한다.

MCP와 호스트에 의해 공유되는 큐(queue)들의 집합을 하나의 통신 채널(communication channel)이라 하는데, 일반적으로 Myrinet은 3개의 통신 채널을 제공한다. 하나의 통신 채널은 **Send_Queue(SQ), Receive_ Buffer_ Queue(RBQ), Receive_Ack_Queue(RAQ)** 등 3개의 큐를 제공하며, 각 큐의 항목(entities)들은 실제 패킷이 저장될 (읽을) 실기억장치(physical memory)의 주소를 가지고 있다. **Send_Queue**는 전송해야 할 패킷들의 실기억장치 주소를 저장하고 있는 큐이고, **Receive_ Buffer_Queue**는 향후 새로운 패킷들이 도착할 경우 이를 저장할 수 있는 주소를 가지고 있다. **Receive_ Ack_Queue**는 이미 수신되었지만 응용프로그램에 의해 아직 읽혀지지 않은 패킷들의 주소를 가지고 있다. 수신된 패킷들의 주소는 MCP에 의해 **Receive_Buffer_Queue**에서 **Receive_Ack_Queue**로 옮겨진다. Send_Queue나 Receive_Buffer_Queue에 새로운 패킷 주소의 추가나 **Receive_Ack_Queue**에서 읽은 패킷 주소의 제거는 Myrinet API를 이용한 응용프로그램에 의해 이루어진다.

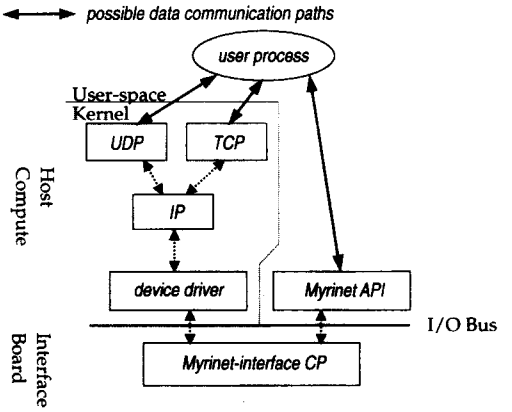
3. 통신 모델

본 장에서는 초고속 통신망인 Myrinet을 기반으로 하여 앞서 언급된 기존 PVM 시스템의 문제점을 개선하기 위한 다양한 데이터 전송 방법을 모색하고, 이를 바탕으로 PVM 환경에 가장 적합한 통신 모델을 제안한다. 또한 제안한 통신 모델의 타당성을 검증하기 위해 각 통신 모델의 성능을 실험하고, 실험결과를 분석한다.

3.1 기존 통신 모델

Myrinet은 사용자 수준에서 Myrinet API를 이용하여 커널을 통하지 않고 Myrinet 인터페이스 보드를 직접 호출하는 통신 모델과 기존의 소켓 라이브러리를 이용하여 커널 내의 TCP/IP 프로토콜 스택을 경유하는 통신 모델 등 2가지 데이터 전송 모델을 제공하고

있다. (그림 3)은 TCP/IP와 Myrinet API를 동시에 지원하는 Myrinet 통신 모델을 도시화 한 것이다.



(그림 3) Myrinet의 통신 모델

- Myrinet API를 이용한 통신 모델: 폴링(polling)을 이용한 통신 모델

Myrinet API를 이용한 통신은 커널을 통하지 않고 응용 프로그램에서 직접 Myrinet 인터페이스 보드에 접근할 수 있다. 따라서 커널 내의 프로토콜 스택을 처리하기 위한 통신 지연 요소를 감소할 수 있고, 사용자(커널) 영역의 데이터를 커널(사용자) 통신 버퍼로 복사하기 위한 복사 부하를 제거할 수 있다는 장점이 있다. 그러나 사용자 수준에서의 Myrinet API는 기본적으로 인터럽트를 이용한 event driven I/O (interrupted I/O) 방식을 지원하지 않고, 프로그램에서 직접 폴링(polling)해야 하는 programmed I/O 방식을 지원한다 (단, 커널내의 TCP/IP용 Myrinet 드라이버는 interrupt I/O방식 사용). 즉, **select()** 함수와 같은 블록킹 상태에서의 메시지 수신 여부를 확인하는 블록킹 I/O 메커니즘을 제공하지 않는다. 따라서 메시지 수신 여부를 확인하기 위해서는 지속적으로 네트워크 인터페이스 보드를 폴링해야만 한다. 그러나 이러한 폴링방법은 다중 프로세서 환경에서는 적합하지만, 단일 프로세서를 사용하는 대부분의 분산 환경에서는 CPU를 점령하는 심각한 요인으로 작용한다.

또한 기존의 응용 프로그램이 terminal, 소켓, file 등과 같은 다양한 정보원으로부터 **select()** 함수를 이용하여 블록킹 I/O를 수행 하였다면, 동일한 **select()** 함수를 Myrinet API에 적용시킬 수는 없을 것이다. 따라서 Myrinet API를 이용한 통신 모델은 커널의 부하를 줄이고 고속의 통신 성능을 제공하는 장점은 있으나, 메시지 수신여부를 확인하기 위한 폴링에 많은 시간을 허비해야 하는 단점이 있다.

- TCP/IP 프로토콜 스택을 이용한 소켓 통신 모델 : event driven I/O 통신 모델

Myrinet API를 이용한 통신 모델의 문제점을 해결하기 위한 한가지 방안으로 커널내의 TCP/IP 프로토콜을 이용한 기존의 소켓 통신 라이브러리를 그대로 활용하는 방법이다. Myrinet은 기존의 TCP/IP를 이용하는 다양한 응용 프로그램과의 호환성을 유지하기 위해 커널 내에 TCP/IP 전용 Myrinet 인터페이스 드라이버를 제공한다. Myrinet은 일반적으로 3개의 통신 채널을 제공하는데, 그 중 채널 번호 0은 Myrinet 드라이버가 사용하고 나머지 2개의 채널만을 일반 응용 프로그램이 사용할 수 있다. 사용자 수준에서 사용하는 Myrinet API는 폴링을 통한 programmed I/O 방식을 사용하지만, TCP/IP용 Myrinet 드라이버는 인터럽트를 이용한 I/O 방식을 사용한다. 따라서 MCP는 채널 번호 0으로 도착하는 패킷에 한해서만 인터럽트를 발생한다. 소켓 open 당시 주어진 IP 주소에 의해 해당 Myrinet 인터페이스 드라이버가 선택되며, 기존의 TCP/IP 프로토콜 스택은 그대로 활용된다. Myrinet의 MTU 크기는 8 KB이며, 이는 IP 프로토콜 계층에서 메시지 조각화(fragmentation)를 위해 사용된다.

TCP/IP를 이용하는 소켓 통신 모델은 기존의 많은 응용 프로그램을 수정 없이 그대로 사용할 수 있는 장점은 있으나, 많은 연구에서 언급된 바와 같이 사용자 영역의 데이터를 커널 내 통신 버퍼로의 복사 부하가 문제점으로 대두된다[11-16, 20].

이상의 두 가지 통신 방법을 종합해 보면, Myrinet API는 커널을 경유하는 통신 부하를 없애고 사용자 수준에서 인터페이스 보드로 직접 데이터를 전송하는 메커니즘을 제공함으로써 상대적인 전송 속도 향상에 기여하지만, 응용 프로그램내에서 폴링을 통한 데이터 수신 여부를 직접 확인해야 하는 단점이 있다. 반면, 기존 TCP/IP 프로토콜 스택을 이용하는 소켓 통신은 호환성면에서는 이점이 있으나, 커널로의 데이터 복사 부하로 인한 통신지연의 단점이 있다.

3.2 UDP-Myrinet API 통신 모델

본 절에서는 앞에서 설명한 두 통신 모델의 장점을 결합한 새로운 통신 모델을 제안한다.

전송 프로세스는 하나의 메시지를 전송하기 위해 우선 UDP 소켓을 이용하여 해당 목적지 프로세스에게 시그널 데이터를 전송한다. 이는 수신 프로세스로 하여금 **select()** 함수를 이용한 블록킹 I/O(event-driven I/O)가 가능하게 하기 위함이다. 이후 Myrinet API를 이용하여 실제 데이터를 보낸다. 수신 프로세스는 메시지의 수신 여부를 확인하기 위해 기존의 **select()** 함수를 이용하여 다양한 정보원으로부터 메시지가 도착하기를 기다린다. Myrinet을 통한 UDP 소켓으로부터 시그널 데이터가 도착하면 수신 프로세스는 블록킹 상태에서 깨어나게 되며, 즉시 Myrinet API를 통하여 해당 채널로부터 원하는 메시지를 읽어낸다. 즉, 수신 프로세스를 깨우기 위한 제어용 데이터는 UDP를 통해 전송하고 메시지 데이터는 Myrinet API를 이용해 전송하는 두 가지 데이터 전송 경로를 혼용한 방식이라 할 수 있다. 이러한 통신 메커니즘을 *UDP-Myrinet API*라 지칭한다.

UDP-Myrinet API에서 전송 프로세스는 UDP를 통한 시그널 데이터가 정상적으로 수신되었는지에 대한 확인 여부와 통신 선로상의 장애로 인한 시그널 데이터와 메시지 데이터의 분실 여부는 상위 레벨의 응용 프로그램이 확인해야 한다.

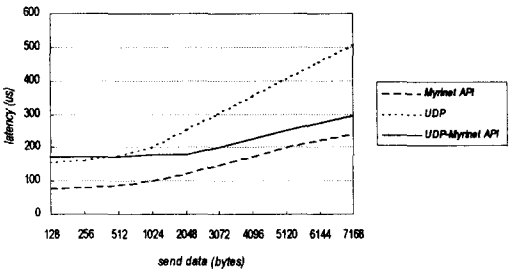
UDP-Myrinet API 통신 모델은 메시지 크기가 크면 클수록 UDP를 통한 시그널 데이터 전송에 소요되는 부하가 상대적으로 줄어들기 때문에 Myrinet API를 이용한 전송 속도와 유사한 성능을 발휘할 것으로 예측된다. 그러나 메시지의 크기가 작아 질수록 UDP 프로토콜을 사용한 소켓 통신 모델의 전송 속도와 비슷해지다가 특정 크기 이하일 경우 오히려 통신 부하가 UDP 소켓 통신 모델보다 더 커질 것으로 예측된다. 이는 하나의 메시지를 전송하기 위해 적어도 두개의 패킷(UDP를 통한 시그널 데이터 패킷과 Myrinet API를 통한 메시지 데이터 패킷)을 전송해야 하기 때문이다.

따라서 이 통신 모델은 두 가지 통신 경로를 사용함으로써, 블록킹 I/O 방법을 채택하고 있는 기존의 응용 프로그램과 일치하는 I/O 방식을 지원하는 이점과 긴 메시지의 경우 상대적인 성능상의 효율성을 기대할 수 있다. 그러나 짧은 메시지의 경우 두 통신 경로를 통과해야 하는 부하로 인해 오히려 성능이 더 떨어질 수 있는 단점이 있다.

3.3 복합 통신 모델

본 절에서는 앞에서 제시된 통신 모델들의 성능을 측정해 보고 이들의 특성을 고려한 새로운 복합 통신 모델을 제안하고자 한다.

(그림 4)는 Myrinet API, UDP, UDP-Myrinet API 통신 모델을 이용하여 각 데이터 크기별 전송 시간을 측정한 결과를 그래프로 표현한 것이다.



(그림 4) 각 통신 모델별 데이터 전송 지연

실험에 사용된 시스템은 PCI 버스상에서 200MHz Intel Pentium Pro 프로세서 및 Myrinet M2F-PCI32 통신 카드, 64MB 주기억 장치를 장착한 시스템을 사용하였고, 운영체제로는 Linux 2.0.30 [17]을 사용하였다. 실험에 이용된 2대의 호스트 시스템은 M2F-SW8 Myrinet 스위치를 통해 연결되었다. 한 시스템의 응용 프로그램 A가 다른 시스템 상에 존재하는 또 다른 프로그램 B에게 데이터를 전송하고, 전송된 데이터를 기다리던 프로그램 B는 데이터를 수신하는 즉시 이를 다시 프로그램 A에게 되돌려 보내는 전형적인 echo 방식으로 실험하였다. 실험 결과는 각 데이터를 크기별로 1000번씩 반복 전송하고, 이때 소요된 전송 시간(round-trip time)의 평균치를 2로 나눈 값이다.

이미 예상한 바와 같이, 응용 프로그램에서 커널을 통하지 않고 Myrinet API를 직접 이용하는 것이 가장 빠른 전송속도를 보여 주고 있으며, UDP/IP 프로토콜을 통한 통신속도가 가장 느리다는 것을 확인할 수 있다. 그러나 512 bytes 이하의 데이터 크기인 경우 UDP-Myrinet API가 가장 느린 전송속도를 보여 주고 있다. 이는 두 번에 걸친 데이터 전송(UDP를 통한 시그널 데이터 전송과 Myrinet API를 통한 실제적인 데이터 전송)보다는 UDP를 통한 한번의 전송(비록 커널로의 데이터 복사 부하가 있을지라도)이 더 빠르다는 것을 의미한다. 그러나 데이터 크기가 512 bytes 보다 클 경우, UDP 보다는 UDP-Myrinet API를 통한 데이터 전송이 더 효율적이다. 또한 데이터 크기가 증가할

수록 이에 비례하여 전송속도 격차도 커진다는 것을 알 수 있다. 그러나, Myrinet API와 UDP-Myrinet API는 데이터 크기가 512 bytes 보다 커질 경우 일정 격차를 유지하면서 더 이상의 성능 차이를 보이지 않는다. 이러한 일정 격차는 UDP를 통한 시그널 데이터 전송으로 인한 통신 지연인 것으로 해석된다.

이상의 실험에서 나타난 결과를 종합해 볼 때, 512 bytes 미만의 데이터는 UDP/IP 프로토콜을 사용하는 소켓통신이 유리하고, 512 bytes 이상의 데이터는 UDP-Myrinet API 통신 모델이 더 적합하다고 볼 수 있다. Myrinet API는 폴링으로 인한 CPU 점령 문제로 인해 고려대상에서 제외시켰다.

본 절에서는 이 결과를 토대로 하여 복합 통신 모델을 제안하고자 한다. 복합 통신 모델은 512 bytes 보다 큰 메시지일 경우 이미 제시한 UDP-Myrinet API 통신 모델을 사용하여 전송하고, 이 보다 작은 메시지일 경우 하나의 UDP 패킷에 시그널 데이터와 메시지 데이터를 함께 전송하는 모델이다. 이는 하부의 통신 프리미티브에서 데이터 크기별로 적절한 통신 라이브러리를 자동으로 선택할 경우, 최적의 통신 성능을 발휘할 수 있을 것으로 기대된다.

이러한 전략은 긴 메시지의 경우 기존의 UDP-Myrinet API 통신 모델의 장점을 그대로 살릴 수 있고, 짧은 메시지의 경우 별도의 통신 경로를 사용하지 않고 UDP로 단일화함으로써 두 번의 통신부하를 한 번의 통신부하로 줄일 수 있다. Myrinet의 MTU가 8KB인 것을 고려해 볼 때, 512 bytes보다 작은 메시지의 경우 하나의 UDP 패킷에 시그널과 메시지 데이터 모두를 실을 수 있으며, 이들의 크기가 상대적으로 작기 때문에 커널로의 복사 부하도 그만큼 줄어들 것으로 기대된다.

4. EPVM (Enhanced PVM)

본 장에서는 앞장에서 제안된 복합 통신 모델을 채택한 수정된 PVM(EPVM)의 구조를 제시하고, EPVM의 구현에 있어 중요한 설계 요소와 데이터 구조 및 주요 함수에 대해 논의한다.

4.1 EPVM의 구조

EPVM은 복합 통신 모델을 채택한 수정된 PVM시스템이다. EPVM은 고속의 데이터 전송을 위하여 Myrinet API를 이용하여 커널로의 데이터 복사 없이 바로 Myrinet 인터페이스 카드로 내려 보내는 구조를 사용한다. 그러나 순수하게 Myrinet API만을 사용할 경우 채널별 데이터 수신 여부를 폴링을 통해서 인식해야 하며, 이는 데이터 수신을 위한 pvmd의 부하를 가중시킬 뿐 아니라, 폴링으로 인한 pvmd와 지역 태스크간의 통신 지연을 초래할 수 있다. 따라서 Myrinet API는 단일 프로세서상에서 여러 개의 지역 태스크들과 pvmd가 공존하는 PVM 환경에서는 부적합 하다고 판단된다.

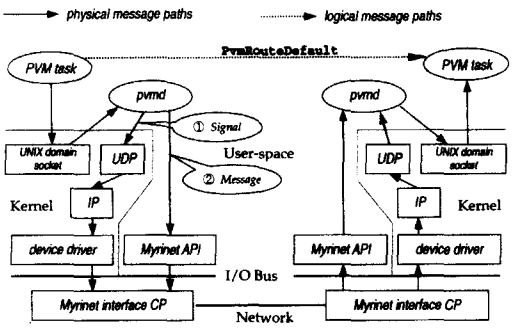
따라서 본 연구에서는 앞 절의 실험 결과를 바탕으로 복합 통신 모델을 채택한 그림 5와 같은 EPVM 구조를 설계하였다.

EPVM에서 지역 태스크간의 통신은 기존의 PVM에서와 같은 방식을 지원하며, 각 지역 태스크는 UNIX 도메인 스트림 소켓을 이용하여 pvmd와 접속하고 pvmd를 통해서 상호 transparent하게 통신한다. Pvmd를 통하지 않고 원거리 태스크간에 직접적으로 통신하는 PvmRouteDirect 방법 역시 기존 TCP/IP를 이용한 소켓통신을 그대로 활용하거나[8]에서 제안한 신뢰성 있는 ATP 프로토콜을 사용한다.

EPVM은 원거리 태스크간의 transparent한 통신을 제공하는 **PvmDontRoute**방법을 개선하였다. 기존 PVM의 **PvmDontRoute** 라우팅 방법의 경우 pvmd와 pvmd 사이의 통신은 소켓 라이브러리를 통해 우선 커널 버퍼로 메시지가 복사되고, 이는 곧 UPD, IP 프로토콜 스택을 거쳐 네트워크 인터페이스 보드에 도달하게 되며, 최종적으로 통신선로를 통해 목적지 시스템에 도착하게 된다. 목적지 시스템에 도착한 메시지는 전송과는 반대 과정을 거쳐 pvmd에 도착하게 된다. 이 과정에서 커널과 사용자 영역간에 두 번의 데이터 복사가 일어난다.

EPVM의 **PvmDontRoute** 라우팅 방법은 복합 통신 모델을 채택했기 때문에 기존 PVM 보다 더 복잡한 통신과정을 수행한다. 우선 메시지를 전송하고자 하는 pvmd는 UDP 소켓을 이용하여 목적지 pvmd에게 시그널 데이터를 전송하고, 이후 곧 바로 메시지 데이터를 Myrinet API를 통해 전송한다. 목적지 pvmd는 select() 함수를 이용하여 지역 태스크와 다른 원거리 태스크로부터의 메시지를 처리하다가 UDP 소켓으로부터 시그널 데이터가 도착하면 즉시 Myrinet API를 통하여 해당 채널로부터 원하는 메시지를 수신한다. 시그널 데이터 전송은 앞에서 언급한 기존 PVM에서와

동일한 메시지 전송 과정을 거치지만, 시그널 데이터 크기가 16 bytes이므로 이로 인한 복사부하는 크지 않다.



(그림 5) EPVM 구조

그러나 EPVM은 앞 절의 실험에서도 증명되었듯이, 메시지의 크기가 512 bytes보다 작을 경우 시그널 데이터와 메시지 데이터를 하나의 UDP 패킷으로 전송함으로써, 두 번의 데이터 전송 부하를 한번의 데이터 전송부하로 줄이는 방법을 채택했다.

복합 통신 모델은 메시지를 원격지에 전송 할 때 신뢰성 있게 전송할 수 없기 때문에 상위 수준에서 신뢰성 있는 통신 프로토콜을 지원해야 한다. 그러나 기존의 pvmd는 목적지 시스템의 pvmd와 신뢰성 없는 UDP 프로토콜을 사용하여 통신하기 때문에 pvmd 자체 내에 신뢰성 있는 메시지의 전송을 보장하기 위한 별도의 프로토콜을 가지고 있다. 따라서 본 연구에서는 PVM 데몬의 신뢰성 있는 통신 프로토콜을 그대로 활용하기로 결정하고, 이를 복합 통신 모델에 적용하는 방법을 택했다.

4.2 EPVM의 구현

PVM은 메시지의 전송 또는 수신을 위해 한 개의 메시지를 여러 개의 패킷으로 관리하며, 하나의 패킷은 다시 패킷 헤더와 데이터로 구성된다. 패킷은 **char** 포인터형인 **databuf**라는 필드를 가지며, **databuf**는 전송할 또는 수신된 실제 데이터를 저장하는 버퍼로서 송수신 시 동적으로 메모리를 할당 받는다. 따라서 EPVM의 구현을 위해 다음과 같은 디자인 요소를 채택했다.

- 패킷의 **databuf** 버퍼를 커널내의 DMA 가능한 데이터 블록으로 교체하여 Myrinet API를 통한 *zero-copy*가 가능하게 한다[13].
- **databuf**를 DMA가능한 블록으로 할당 받기 위한 동적 메모리 관리기법을 구현한다.
- Myrinet의 전송요소를 라이브러리화 하여 관리한다.
- 제안된 복합 통신 모델을 적용하기 위해 PVM의 전송 및 수신 프리미티브 함수들을 수정 또는 확장한다(복합 통신 모델).

● DMA 블록 할당 및 관리

사용자 영역에서 Myrinet API를 이용한 직접적인 데이터 송수신을 위해서는 우선 **pvmd**의 시스템 초기화 당시 **EPVM_Initialize()** 함수를 호출한다 [표 3참조]. 이 함수는 다시 **open_lannai_device_linux()** 함수를 호출하여 pvmd를 위한 Myrinet 채널을 할당하고, DMA 가능한 연속된 커널 메모리 블록을 할당 받기 위해 **ioctl(fd, MLANAI_GET_INFO, &BINFO[u_num])** 시스템 함수를 호출한다. 커널내의 Myrinet 드라이버는 연속된 커널 메모리 블록을 할당하고, 이를 드라이버내의 **mmap()** 함수를 이용하여 **pvmd**의 가상 기억공간으로 mapping한다. 마지막으로 **BINFO[u_num]**를 통해 블록의 실 기억공간의 주소와 가상 기억공간의 주소를 넘겨 준다. 이 때 할당되는 블록은 수신용 버퍼 20개와 송신용 버퍼 10개이며, 각 버퍼는 시스템 페이지 크기(일반적으로 4KB)와 동일하다. 메시지 송수신용 버퍼로 할당된 DMA 블록은 커널이 할당할 당시 사전에 해당 메모리를 locking (pin down) 시킴으로써, 가상 기억장치로 하여금 이 영역을 디스크로 swap(page) in-out 할 수 없게 한다. 따라서 **pvmd**가 블록킹된 후 디스크로 swap out 되어 있어도 메시지가 도착하는 즉시 Myrinet MCP는 등록된 수신 버퍼로 DMA access할 수 있다.

DMA 가능한 송수신용 블록을 관리하기 위해 다음과 같은 **EPVM_Buffer**라고 하는 구조체를 사용한다. 할당된 각각의 송수신용 블록은 하나의 **EPVM_Buffer** 인스턴트를 통하여 관리된다.

```
struct EPVM_Buffer {
   myriApiDmaAddr_t d; /* 커널 포인터 */
   char *u             /* 사용자 포인터 */
   int used;           /* 버퍼의 현재상태 */
};
```

구성 필드 중 **char *u** 는 **pvmd**가 실제 **databuf**로 사용하는 가상 기억공간의 주소값을 저장하며, 이에 상응하는 실 기억공간의 주소는 **myriApiDmaAddr_t d** 에 저장된

다. 이는 상위 수준에서는 가상 기억공간을 사용하고, 하위 수준(Myrinet 인터페이스 보드내의 MCP)에서는 실 기억 주소공간을 사용하기 때문이다. 따라서 Myrinet 전송 라이브러리는 메시지의 송수신뿐 아니라, 두 주소공간사이의 주소 변환 기능도 함께 처리하여야 한다. 각 버퍼는 현재의 사용 여부에 따라 상태가 변하게 되며, 이는 used 필드를 통하여 표2와 같은 용도로 사용한다.

〈표 2〉 EPVM_Buffer의 used 필드 값의 종류와 의미

EPVM_Buffer의 used 필드	값	기 능
EPVM_MYRI_FREE	0	버퍼가 사용되지 않으므로 사용가능.
EPVM_MYRI_RECEIVE	1	수신 큐에 현재 버퍼가 할당 되어 있다.
EPVM_MYRI_USE	2	메시지가 수신되어 버퍼를 현재 사용하고 있다.
EPVM_MYRI_SEND	3	버퍼가 현재 송신용으로 사용되고 있다.

〈표 3〉 EPVM의 메시지 송수신을 위한 주요 함수

함수명	기능
int EPVM_Initialize (int,int)	Myrinet 디바이스를 초기화하고 DMA 가능한 송수신용 커널 버퍼를 할당 받는다.
unsigned int EPVM_Receive (char **)	메시지의 수신여부를 얻어 온다. 수신되었을 때는 EPVM_Buffer버퍼 하나를 사용하게 되며, 수신된 버퍼의 주소를 리턴한다.
int EPVM_Send (char *,int)	송신 버퍼를 외부로 전송한다.

● Myrinet 전송 라이브러리

이러한 버퍼 관리 기법을 바탕으로 표 3에서 보인 함수를 이용하여 실제 메시지를 송수신한다. **EPVM_ Initialize()**에서 통해 할당된 수신용 버퍼는 **myriApiAddReceiveBuffer()** 함수를 통해사전에 MCP의 수신용 큐에 등록되며, 이때 버퍼의 실 기억공간의 주소를 매개 변수로 넘겨 주어야 한다.

EPVM_Receive(char **)는 myriApiGetNumReceiveAcksPending() 를 이용하여 현재 수신된 버퍼가 몇 개인지 확인하고, **myriApiGetReceiverBuffer()**를 호출하여 수신 큐에서 해당 버퍼를 빼낸다. **EPVM_Receive(char **)**은 수신된 버퍼를 이에 상응하는 가상기억공간 주소로 변환하고 이를 호출자에게 넘겨준다. 상위 수준에서 각 패킷의 사용이 끝나면 패킷의 버퍼인 **databuf**는 **da_free()**를 통해 free된다. 이때 이 버퍼가 일반 **malloc()**을 통해 할당 받은 버퍼가 아닌 Myrinet 송수신용 버퍼일 경우, 이를 **myriApiAddReceiveBuffer()**를 통하여 MCP의 수신용 큐에 재등록 함으로써 다음 데이터 수신에 재사용할 수 있게 한다.

패킷의 송신은 **EPVM_Send()**를 통하여 이루어지며, 이는 버퍼의 주소를 실 기억주소공간의 주소로 변환한 다음 **myriApiSend()**를 호출하여 해당 버퍼를 MCP의 송신용 큐에 등록한다. 이후 **EPVM_Send()**는 **myriApiGetNumSendsPending()**를 이용하여 이전 **EPVM_Send()**에 의해 MCP의 송신용 큐에 등록된 버퍼들 중에서 MCP에 의해 이미 송신이 끝난 버퍼를 찾아 상태를 **EPVM_MYRI_FREE**로 설정함으로써 이 버퍼를 재사용 가능하게 한다.

● 복합 통신 모델

EPVM은 메시지 전송을 위하여 복합 통신 모델을 사용한다. 즉 수신 프로세스를 깨우기 위한 시그널 데이터를 UDP로 먼저 전송한 후 실제 데이터를 나중에 전송하는 방법을 사용하고 있다. 복합 통신 모델은 전송할 데이터량이 많아질 경우에 월등히 좋은 성능이 나올 수 있지만, 앞서의 실험에서와 같이 적은 양의 데이터 일 경우에는 UDP만을 이용하여 전송한다. 따라서 복합 통신 모델은 전송할 메시지가 512 bytes 이상일 경우에는 Myrinet API를 이용한 사용자 영역에서 메시지 전송을 하고, 512 bytes 미만일 경우에는 소켓을 이용한 커널 레벨의 UDP로 전송한다. 실제 전송의 경우 512 bytes 미만일 경우에는 **malloc()**을 이용하여 일반적인 메모리를 할당 받아 사용하며 512 bytes 이상일 경우에 4 KB의 **EPVM_Buffer**를 할당 받아 사용한다.

EPVM 데몬은 다음과 같은 데이터 구조를 이용하여 시그널 데이터를 전송한다.

```
struct EPVM_sigdata {
    char src[8]        /* 보내는 쪽의 Myrinet 주소*/
    char SignalID[8]   /* signal임을 인증하는 데이터
                          값 M_SIGNAL */
}
```

전송할 때 **src**필드에 자신의 Myrinet 주소를 채운 후, 보낼 메시지 크기가 512 bytes 이상이면 **SignalID** 필드를 **M_SIGNAL**로 설정한 후, UDP 소켓을 통해 **EPVM_sigdata**를 전송하고 **EPVM_Send()** 함수를 호출하여 Myrinet으로 실제 메시지를 전송한다. 512 bytes 미만이면 **SignalID**필드를 **M_DATA**로 채운

후, UDP 소켓을 통해 **EPVM_sigdata**와 메시지 데이터를 하나의 패킷에 담아 전송한다.

이를 수신한 측에서는 네트워크로부터의 입력이 있을 경우, **netinput()** 함수부에 있는 **recvfrom()** 함수를 통해서 UDP로 전송 받은 시그널 데이터의 크기가 16 bytes임을 확인하고, **SignalID** 필드의 값이 **M_SIGNAL**일 경우에 **EPVM_Receive()**를 호출하여 메시지 데이터를 수신하고, **M_DATA**인 경우 시그널 데이터와 함께 도착한 메시지 데이터를 그대로 이용하는 방식으로 구현하였다. 메시지 데이터의 분실로 인해 일정 시간 동안 폴링해도 메시지 데이터가 수신되지 않을 경우, 수신측의 EPVM 데몬은 UDP 시그널 데이터가 도착하기 이전의 상태로 모든 메시지 상태를 복귀 시킨다. 이 경우 전송측의 EPVM 데몬은 **ACK**를 받지 못하기 때문에 일정시간 후에 다시 재 전송을 하도록 구현하였다.

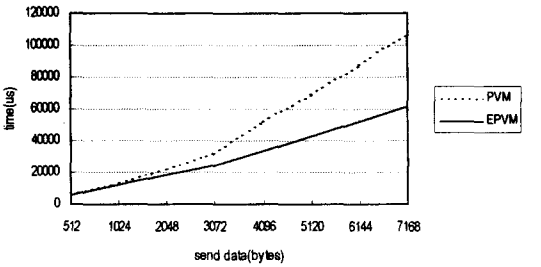
5. 실험 및 분석

EPVM의 성능 향상을 확인하기 위해 각 데이터 크기별 전송시간을 측정하였다. 실험에 사용된 시스템은 PCI 버스상에서 200MHz Intel Pentium Pro 프로세스 및 Myrinet 통신 카드, 64MB 주기억 장치를 장착한 시스템을 사용하였고, 운영체제로는 Linux 2.0.30 [17]을 사용하였다. 실험에 이용된 2대의 시스템은 M2F-SW8 Myrinet 스위치를 통해 연결되었다.

<표 4>와 (그림 5)는 데이터를 보내는 크기 별로 1000번씩 반복하여 측정하여 얻어낸 turn-around 시간의 평균값을 반으로 나눈 결과이다.

<표 4>는 데이터 크기별 PVM과 EPVM상의 데이터 전송 시간을 μs 단위로 보여주고 있다. 데이터의 크기가 클수록 PVM과 EPVM의 데이터 전송 성능에 격차가 점점 더 커짐을 알 수 있으며, 1배에서 약 2배까지의 성능 격차가 생기는 것을 알 수 있다.

Myrinet의 MTU가 8 KB인 관계로 8 KB 데이터 크기까지만 성능 테스트를 하였다. 커널내의 UDP/IP 프로토콜에서도 이 MTU 크기를 기반으로 fragmentation을 시킨다. 따라서 Myrinet의 MTU 보다 더 큰 크기의 메시지를 전송한다면 커널내의 *fragmentation* 부하로 인해 더 큰 성능 격차를 보일 것으로 추정된다.



(그림 5) 통신 지연 비교

8KB 이하인 각 데이터는 Myrinet상에서는 하나의 패킷으로 전송된다는 사실을 고려해 볼 때, PVM과 EPVM의 성능 격차는 pvmd로부터 커널내의 전송 버퍼로의 데이터 복사로 인한 부하 때문인 것으로 쉽게 추정할 수 있으며, 따라서 데이터 크기가 크면 클수록 복사 부하는 더더욱 커질 것으로 예측된다.

EPVM의 경우 각각의 데이터를 전송하기 전에 UDP/IP 프로토콜을 사용하여 데이터가 전송되었음을 알려 주는 시그널 데이터를 먼저 전송하고, 이후 Myrinet API를 통해 실제 데이터를 전송한다. 따라서 위의 표는 EPVM에서 UDP를 통한 시그널 데이터 전송 부하를 고려한 측정치이므로, 실제 PVM에서 커널을 통한 데이터 복사 부하는 위의 표에서 보이는 격차 이상의 값을 가진다는 것을 알 수 있다.

(그림 5)는 <표 4>를 그래프로 표현한 것으로 데이터의 크기가 클수록 격차가 더 커짐을 한 눈에 파악할 수 있다. 이는 UDP/IP를 사용하는 기존 PVM의 경우 커널 내로 데이터 복사 부하, 그리고 이를 위한 메모리 할당(한 페이지의 크기가 4KB임) 및 가상 메모리 관리에 따른 부하 등으로 인한 것으로 추정된다.

〈표 4〉 통신 지연 비교

(단위 : μs)

Systems \ Data Size(bytes)	512	1024	2048	3072	4096	5120	6144	7168
PVM	6470	13528	21882	31840	52035	69322	87909	107473
EPVM	6025	12207	18237	24273	33463	42700	52028	61306

6. 관련 연구

최근의 응용 프로그램간 통신은 ATM, Myrinet, Fast Ethernet등 초고속 통신을 지원하는 H/W기술을 적극 활용하고, 고성능 프로세서의 성능을 최대한 수용하기 위한 방향으로 연구가 이루어지고 있다. 이를 위해 기존 커널 영역에서 관리하던 프로토콜 스택을 사용자 영역에서 관리하게 하고 커널의 간섭을 가능한 배제시키며 메시지 버퍼 복사를 없애는 zero-copy 기법들이 제안되고 있다 [11-16, 20].

따라서 PVM에서 이러한 초고속 통신망을 활용하고 직접 네트워크 인터페이스에 접근하는 것이 고속의 응답과 광대역 통신을 제공할 수 있고, MPP와 유사한 성능을 발휘할 수 있다는 것이 여러 연구에서 발표되고 있다[5-10].

Zhou와 Geist는 [5]에서 PVM내에 **PvmRouteAtm**라는 새로운 원거리 태스크간의 통신 방식을 제안하였다. **PvmRouteAtm**은 PVM 3.3에서부터 제공된 **PvmRouteDirect**의 장점을 살리면서 이를 ATM API를 사용하여 구현하였다. 이는 pvmd를 경유하지 않고 태스크간 직접 데이터를 전송하는 매커니즘으로 TCP 대신 ATM API를 사용하였다. 기존의 **PvmDontRoute**는 ATM상에서 커널 내에 구현된 TCP/IP 에뮬레이션을 사용하였다. 여기서 저자들은 작은 크기의 메시지를 전송하는 경우 **PvmDontRoute**를 사용하는 것이 더 빠른 성능을 보이고, 대량의 데이터를 보내는 경우 **PvmRouteAtm**을 사용하는 것이 더 빠른 성능을 보인다는 것을 실험을 통해 확인하였다.

PSPVM [6]은 ParaPC/ParaStation [7]이라는 클러스터링 시스템에 이식된 PVM 버전이다. ParaPC/ParaStation은 네트워크 하드웨어에서 라우팅을 직접 제공함과 동시에 신뢰성 있는 데이터 전송을 보장한다. 뿐만 아니라, 가변적인 패킷 길이를 지원함으로써 패킷의 조각화 및 재조립(segmentation & reassemble) 하는 부하를 줄였다. 또한 ParaPC/ParaStation은 네트워크 H/W를 사용자 수준에서 제어할 수 있는 라이브러리를 제공하며, 이를 이용하여 구현한 것이 PSPVM이다. ParaPC/ParaStation은 275 MHz Alpha 칩을 장착한 8대 클러스터링 시스템에서 1 GFlops의 고성능을 달성했으며, 이를 기존 PVM과 비교할 때 최대 8배 나은 응답시간(latency)을 가진다고 저자들은 주장한다. 그러나 특별한 하드웨어를 사용함으로써 이식성이 결여된다는 단점을 가지고 있다.

Xu와 Fisher [8]는 Myrinet을 이용하여 사용자 수준 통신을 제공하고 메시지 경로(data path)와 제어 메시지 경로(control message path)를 구분한 PVM을 구현하였다. Xu와 Fisher는 Myrinet API를 이용한 사용자 수준의 신뢰성 있는 통신 프로토콜인 ATP를 구현하였으며, 이는 커널 내에 구현된 TCP/IP 보다 2배 빠른 응답시간을 제공했다. 또한 512 KB 보다 큰 메시지 전송에서는 Myrinet API 성능의 94%까지 도달하는 성능 향상을 달성하였다.

PVM/ATM [9]은 대역폭이 높은 통신망을 채택하여 H/W상의 부하요소와 S/W상에서의 부하요소를 줄일 수 있는 방안을 제시하였다. PVM/ATM은 기존 프로토콜 스택을 사용하는 통신방법을 배제하고 Fore System의 ATM API를 이용하여 하드웨어에 직접 접근하는 방식을 사용함으로써 통신상의 부하 요소를 줄였다. PVM/ATM은 메시지의 안전한 전송을 위하여 선택적 재전송 방법(selective retransmission mechanism)으로 양단간 유통 제어 스킴(end-to-end flow control scheme)을 구현하였다. 또한, ATM 스위치에서 제공되는 멀티캐스팅 기능을 PVM의 멀티캐스팅에서 직접 사용하여 PVM/TCP/ATM으로 전송할 때의 대역폭(1MB 전송 기준)이 20 Mbps가 나오던 것을 PVM-ATM AAL3로 약 27Mbps정도의 대역폭을 제공했다.

PVM/LLA [10]는 HIPPI (high-speed local area network) LAN기반에서 소켓을 이용한 통신상의 프로토콜 스택으로 인한 전송 부하를 저 수준의 Hewlett Packard LLA(Link Level Access) HIPPI API만을 사용하여 전송함으로써, PVM의 성능을 향상시키는 방안을 제시하였다. 이들은 PVM 데몬간 연결에 사용되는 UDP와 **PvmRouteDirect** 라우팅에서 사용되는 TCP 대신에 LLA API를 이용하여 구현함으로써, 일반적인 PVM의 성능과 비교해 38%의 round-trip latency 향상을 보였다.

7. 결 론

본 논문에서는 웍스테이션 연동 환경에서의 병렬 프로그램 모델을 지원하는 PVM의 문제점중의 하나인 커널내의 통신 부하를 최소화하고 원거리 응용 태스크간의 통신 효율성을 증진시켜, 전체적인 시스템 성능을 향상시키기 위한 연구의 일환으로 추진되었다. 본 연구는 초고속 통신망인 Myrinet을 이용하는 EPVM을 설계하고, 이를 기반으로 메시지의 크기에 따라 응용

프로그램에서 커널을 통하지 않고 통신 인터페이스를 직접적으로 이용하거나 기존 UDP를 혼용하여 사용하는 복합 통신 모델을 제시하였다. 또한 제안된 모델의 사용 가능성과 타당성 및 효율성을 검증하기 위해 이를 실험적으로 구현하였다.

EPVM은 원거리 태스크간의 통신 메커니즘으로 큰 메시지의 경우 커널의 UDP 프로토콜을 사용하지 않고, 사용자 레벨에서 Myrinet API를 통한 통신 인터페이스를 직접 호출하였다. 따라서 데이터 복사로 인한 커널내의 통신 부하를 줄이고 UDP/IP 프로토콜 스택을 통과하는데 소요되는 통신지연을 감소시켰다. UDP 프로토콜은 단지 메시지 도착 시그널 기능과 작은 크기의 메시지를 전송하는 경우에만 활용하였다.

실험 결과 EPVM의 통신 성능이 커널내의 UDP 프로토콜을 사용하는 기존 PVM의 통신 성능보다 1배에서 2배 정도 우수하다는 것을 확인하였다.

현재 구현된 EPVM은 원거리 태스크간의 통신 효율성을 증가 시키기 위해 각 시스템간의 메시지 전달을 위한 다리 역할을 하는 pvmd와 pvmd의 통신에서 발생하는 통신부하 요소를 제거 했을 뿐, 지역 태스크간의 통신에는 여전히 UNIX 도메인 스트림 소켓을 사용하고 있다. 따라서 본 연구팀은 지역 태스크간의 통신 메커니즘으로서 공유 메모리를 이용하는 방법론을 연구하고 있다[18]. 또한 지역 태스크간의 통신을 위해 사용되는 공유 메모리와 원거리 태스크간 통신을 위해 사용되는 메모리간의 연계 문제도 함께 연구하고 있다. 지역 태스크간의 통신을 위해서는 데이터 복사가 되지 않지만, 지역 태스크와 원거리 태스크간의 정보 교환을 위해선 한번의 데이터 복사 과정이 필요하다. 이는 공유 메모리로부터 pvmd내의 Myrinet 인터페이스와 공유하는 DMA 가능한 메모리로의 복사가 필요하기 때문이다. 향후 연구에서는 별도로 운영되는 공유 메모리 관리를 통합하여, 하나의 공유 메모리를 지역 태스크와 pvmd, 그리고 Myrinet 인터페이스 카드 등이 동시에 사용 가능하게 하여 통합적인 메모리 사용이 가능하게 할 계획이다.

현재 PVM은 각 시스템의 부하 상태를 확인하여 부하가 가장 적은 시스템에 새로운 태스크를 할당 시켜주기위한 자원관리 인터페이스를 제공하고 있다. 따라서 본 연구팀은 이러한 인터페이스를 활용한 전역 스케쥴러 및 부하 감시 시스템을 개발하고 있으며, 이를 활용할 경우 전체적인 시스템의 성능 향상을 도모할 뿐 아니라, 유휴 시스템을 최대한 활용할 수 있는 터전을 마련할 것으로 기대된다.

참 고 문 헌

[1] A. Geist, A. Beguelin, J.Dongarra, W.Jiang,R. Manchek, and V. Sunderamm, "PVM 3 User's Guide and Reference manual," Oak Ridge National Laboratory, Oak Ridge, Tennesee 37831, Sept. 1994.

[2] R. J. Manchek, "Design and Implementation of PVM Version 3," Master Thesis, University of Tennessee, Knoxville, May 1994.

[3] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. K. Su, "Myrinet : A Gigabit-per-Second Local Area Network," *IEEE Micro*, Vol.15, No.1, pp.29-36, Feb. 1995.

[4] Myricom, "Myrinet Performance Measurements," http://www.myri.com/myrinet/performance/index.html

[5] H. Zhou and A. Geist, "Faster Message Passing in PVM," 9th *International Parallel Processing Symposium : Workshop on High Speed Network Computing*, Santa Barbara, April 1995.

[6] J. M. Blum, T. M. Warschko, and W. F. Tichy, "PSPVM : Implementing PVM on a high-speed Interconnect for Workstation Clusters," *Technical Report 17/96*, University of Karlsruhe, Dept. of Informatics Postfach, Karlsruhe, Germany, 1996.

[7] T. M. Warschko, J. M. Nlum, and W. F. Tichy, "The ParaPC/ParaStation Project : Efficient Parallel Computing by Clustering Workstations," *Technical Report 13/96*, University of Karlsruhe, Dept. of Informatics Postfach, Karlsruhe, Germany, 1996.

[8] H. Xu and T. Fisher, "Improving PVM Performance Using the ATOMIC User-Level Protocol," *The High-Speed Network Computing Workshop '95*, 1995.

[9] S. L. Chang, David H.C. Du, J. Hsieh, M. Lin, and R. P. Tsang, "Enhanced PVM Communications over a High-Speed Local Area Network," *IEEE Parallel*

& Distributed Technology, pp.20-32, Fall 1995.

[10] J. Hsieh, David H. C. Du, N. J. Troullier and M. Lin, "Enhanced PVM Communications over a HIPPI Local Area Network," *Proceedings of the 2nd International Workshop on High-Speed Network Computing* (HiNet '96), Honolulu, April 1996.

[11] A. Basu, M. Welsh, and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces," Department of Computer Science, Cornell University, Submitted for publication, Nov. 1996.

[12] M. Welsh, A. Basu, and T. von Eicken, "Low-Latency Communication over Fast Ethernet," *Proceedings of Euro-Par '96*, Lyon, France, Aug. 1996.

[13] A. Basu, V. Buch, W. Vogels, and T. von Eicken, "U-Net : A User-Level Network Interface for Parallel and Distributed Computing," *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)*, Copper Moutain, Colorado, Dec. 1995.

[14] S. Pakin, V. Karamcheti, and A. Chien, "Fast Messages : Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors," *IEEE Concurrency*, 1997.

[15] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa, "Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication," *Technical Report*, Tsukuba Research Center, Real World Computing Partnership, 1997.

[16] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active Messages : a Mechanism for Integrated Communication and Computation," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[17] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner, "Linux Kernel Internals," Addison-Wesley, 1996.

[18] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, "The Design and Implementation of the 4.4 BSD Operating System," Addison-Wesley, pp. 137-146, 1996.

[19] T. E. Anderson, D. E. Culler, and D. A. Patterson, "A Case for NOW (Network of Workstations)," *IEEE Micro*, Vol.15, No.1, pp.54-64, Feb. 1995.

[20] H. Tezuka, A. Hori, and Y. Ishikawa, "PM : A High-Performance Communication Library for Multi-user Parallel Environments," *Technical Report TR-96-015*, Tsukuba Research Center, Real World Computing Partnership, Japan, Nov. 1996.

김 인 수

e-mail : insoo@etri.re.kr

1981년 경기대학교 관광경영학과 졸업 (경영학사)

1983년 중앙대학교 컴퓨터공학과 졸업 (이학석사)

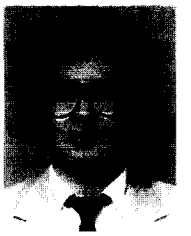
1999년 아주대학교 컴퓨터공학과 졸업 (공학박사)

1983년~현재 한국전자통신연구원 연구원

관심 분야 : 분산시스템, 멀티미디어, 실시간 시스템, 소프트웨어공학

심 재 홍

현재 아주대학교 정보 및 컴퓨터 공학부 박사 과정



최 경 희

e-mail : khchoi@madang.ajou.ac.kr

1982년 Paul Sabatier 대학(프) 정보공학과 졸업

1982년~현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 운영체제, 분산시스템, 멀티미디어, 실시간 시스템

정 기 현

e-mail : khchung@madang.ajou.ac.kr

1990년 Purdue 대학(미) 전기공학과 졸업

1992년~현재 아주대학교 전기전자공학부 교수

관심분야 : 컴퓨터 구조, VLSI 설계, 멀티미디어, 실시간 시스템

문 경 덕

현재 한국전자통신연구원 컴퓨터 및 소프트웨어기술연구소 연구원

김 태 근

e-mail : tkim@etri.re.kr

1987년 한양대학교 수학과 졸업(학사)

1990년 뉴욕주립대학교 전산수학 졸업(석사)

1993년 뉴욕주립대학교 전산수학 졸업(박사)

1992년~현재 ETRI 네트워크컴퓨터연구부 분산컴퓨팅 연구팀 팀장

1997년~현재 ETRI 네트워크컴퓨팅연구부 자바센터 센터장

현재 한국전자통신연구원 컴퓨터 및 소프트웨어기술연구소 책임연구원

관심분야 : 병렬 컴파일러, 고성능 컴퓨터시스템, 병렬/분산 시스템