VIA 기반의 병렬 라이브러리 구현에 관한 연구

서울대학교 하순회* · 기양석 · 김선재

1. 서 론

하드웨어와 소프트웨어 기술이 발전함에 따라, 컴퓨팅 환경은 급속히 변화하고 있다. 현재 상업용 프로세서의 성능은 눈부신 발전을 이루어 누적 성장률이 1992년 이래로 154%에 이르고 있으며, 네트웍 분야에서도 데이터의 특성이 과거의 텍스트 위주에서 멀티미디어 등의 대용량으로 변함에 따라, 100Mbps 또는 Gbps 이상의 대역폭을 갖는 장비들이 등장하였다[1]. 또한, 분산 처리를 위한 다양한 소프트웨어 도구들이 개발되었고, 운영 체제도 분산 처리를 위한 기능을 지원해 가고 있는 추세이다. 이와 같은 기술의 발전은 병렬 처리 분야에서도 새로운 시도를 가능하게 하였는데, 이 중 하나가 클러스터링(clustering) 기법이다.

클러스터는 독자적인 컴퓨터를 상호 연결망으로 연결하여 함께 동작하게 함으로써 하나의 통합된 계산 자원으로 활용할 수 있게 하는 병렬 또는 분산 컴퓨터 시스템이다[1]. 클러스터는 기존의 SMP(Symmetric Multi-Processor)나 전통적인 분산 시스템과는 다른 개념이다. SMP가 여러 프로세서들이 메모리, I/O 등의 자원을 공유하여 사용하는 반면, 클러스터는 여러 컴퓨터들이 독자적인 메모리, I/O 등의 자원을 소유한다. 따라서, SMP는 클러스터에 비해 확장성, 가용성, 시스템 관리, 소프트웨어 사용 등의 측면에서 제약을 가지고 있다. 분산 시스템은 독자적인 컴퓨터를 비교적 느린 네트웍으로 연결한 약결합(loosely-coupled) 상태로 되어 있다. 반면, 클러스터는 각 노드의 익명성이 보장되어 시스템 전체적으로 하나의 이미지를 갖게 하며, 노드들은 고성능의 네트웍으로 강결합(tightly-coupled) 상태를 이룬다. U.C Berkeley의 워크스테이션 기반의 NOW(Network Of Workstation) 시스템[2]이나, NASA 산하 CESDIS(Center of Excellence in Space Data and Information Sciences)의 리눅스 기반의 PC를 활용한 Beowulf 프로젝트[3] 등이 대표적인 클러스터 시스템들이다.

비록, 하드웨어 기술의 발달로 물리적으로 클러스터를 구성하는 것은 가능해졌지만, 이 클러스터를 활용하기 위한 소프트웨어의 발전은 비교적 더디게 진행되었다. 특히 MPP(Massively Parallel Processor)에 비해 클러스터가 갖는 약점이면서, 진정한 클러스터 구현에 있어서 중요한 장애가 되었던 것이 큰 통신 오버헤드이다. 이 통신 오버헤드의 주요한 원인은 기존의 프로토콜을 소프트웨어적으로 구현함에 있어 생긴 비효율성에 근거한다고 한다[1,4,5]. TCP/IP와 같은 기존의 통신 프로토콜은 LAN/WAN 환경에서 다양한 네트웍에 연결되어 있는 컴퓨터들을 안정적으로 접근하고, 사용자에게 투명한 네트웍의 접근을 허용하기 위해 두터운 프로토콜 스택이 필요했다 과거에는 물리적 네트웍의 성능을 고려할 때, 소프트웨어 오버헤드는 크게 문제되지 않았지만, 물리적 네트웍의 성능이 향상됨에 따라 소프트웨어 오버헤드는 전체 통신 지연(latency)의 가장 중요한 원인이 되었다. 따라서, 큰 대역폭과 짧은 지연 시간을 요구하는 SAN

* 정회원

(System Area Network) 환경에 적합한 새로운 프로토콜의 필요성이 제기되었고, 이를 위한 다양한 연구가 진행되었다. 대표적인 연구로는 Berkeley NOW 시스템의 Active Message (AM) [2], Cornell 대학의 U-Net[6], Princeton 대학의 SHRIMP(Scalable High-performance Really Inexpensive Multi-Processor) 프로젝트에서 제안한 VMMC(Virtual Memory-Mapped Communication)[7], Illinois 대학의 Fast Message(FM)[8]과 Intel, Compaq, Microsoft가 주축이 되어 제안한 Virtual Interface Architecture(VIA)[5] 등이 있다. 이들 새로운 프로토콜의 주요한 관심사는 패킷 전송 중의 불필요한 복사를 피하는 것과 기존 프로토콜을 단순화시키는 것에 있다[9].

이러한 고성능 프로토콜을 응용 프로그램에서 사용하기 위해서는 이들 프로토콜을 기반으로 하여 구현된 표준 프로그래밍 라이브러리가 제공되어야 한다. 표준 프로그래밍 라이브러리로는 MPI[6], PVM[7], BSPlib[8] 등이 널리 알려져 있으며, 이들은 대개 기존의 소켓 프로토콜을 기반으로 하여 구현되어 있다. 그러나, 이러한 필요에도 불구하고 고성능 프로토콜을 기반으로 하여 표준 라이브러리를 구현하는 것은 그리 쉬운 일은 아니다. 왜냐하면, 고성능 통신 프로토콜에서는 통신 주체끼리 밀접하게 상호 작용을 해야 하는 경우가 있기 때문이다.

이 논문에서는 고성능 통신 프로토콜 기반의 표준 라이브러리 구현에 있어 고려해야 할 점과 최적화의 가능성에 대해 살펴보고자 한다. 우리는 고성능 프로토콜로써 VIA를, 표준 라이브러리로 MPI와 BSPlib을 채택하였다. MPI는 메시지 전달 방식 라이브러리의 대표적인 표준으로 PVM과 더불어 가장 널리 알려진 표준이고, BSPlib는 VIA를 가장 효과적으로 적용할 수 있는 구조를 가지고 있는 라이브러리이기 때문이다. 이를 통해, TCP/IP(UDP/IP) 기반의 라이브러리를 VIA로 대체했을 때 얻을 수 있는 성능 향상에 대해 살펴본다.

2. VIA(Virtual Interface Architecture)

컴팩, 인텔, 마이크로소프트 등 세 회사에 의해 사용자 수준의 통신 구조로써 Virtual Interface Architecture(VIA)[5]가 제안되었다. VIA는 네트웍에 연결된 두 단말 사이에 통신 인터페이스를 제공한다. VIA는 "보호된 사용자 수준의 무복사(zero-copy) 프로토콜"로써 규정지을 수 있다. LAN 환경에서의 네트웍 소통량에 관한 연구[13][14]에 의하면 네트웍 상의 패킷은 주로 두 가지 형태를 갖는 것으로 알려져 있다. 즉, 대부분의 패킷이 200-byte보다 작거나, 8-Kbyte보다 크다는 것이다. 이러한 관찰을 바탕으로 VIA는 이들 경우를 신속히 처리하도록 최적화가 이루어졌다

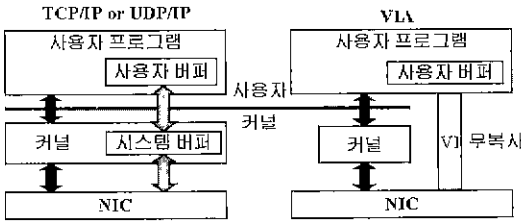


그림 1 TCP/IP(UDP/IP)와 VIA 비교

그림 1은 VIA와 TCP/IP(UDP/IP)의 차이를 도시하고 있다. 전통적인 TCP/IP(UDP/IP) 프로토콜에서는 단지 커널만이 네트웍 인터페이스에 접근할 수 있다. 따라서, 네트웍을 통한 통신은 사용자 버퍼 영역과 시스템 버퍼 영역 사이에서 데이터의 복사를 동반하게 된다. 그러나, VIA는 가상 인터페이스(VI)를 통해 사용자 프로세스가 독점적으로 네트웍 인터페이스를 소유한 것처럼 보이게 한다. 사용자 프로세스는 VI를 통하여 운영 체제의 간섭을 받지 않고 안전한 방법으로 네트웍 인터페이스에 직접 접근할 수 있다. 사용자 프로세스가 네트웍 인터페이스에 대한 직접 접근을 할 수 있게 함으로써 사용자 버퍼와 시스템 버퍼 사이의 데이터 복사를 피할 수 있다. TCP/IP에서는 커널만이 네트웍 디바이스에 접근할 수 있기 때문에 보호 기능이 커널 수준에서 제공되었다. 그러나, 이와 같이 사용자 프로세스가 디바이스를 접근하는 경우에는 가상 주소의 해석과 보호 기능을 VIA가 제공해야 한다.

2.1 VIA의 구조

VIA는 가상 인터페이스(VI), 완료 큐(CQ), 제공자(VI-Provider), 수요자(VI-Consumer) 등

크게 네 개의 구성 요소로 이루어져 있다(그림 2). 제공자는 물리적인 네트웍 카드와 디바이스 드라이버를 포함하는 커널 에이전트로 이루어지고, 자원 관리 및 통신에 관한 모든 작업을 관장한다. 수요자는 사용자 에이전트라고도 부르는 VIPL과 여러 가지 통신 라이브러리, 그리고 응용 프로그램으로 구성된다. VI는 송신 큐와 수신 큐의 쌍으로 구성된 작업 큐로 이루어져 있고, 통신 주체들은 이들 VI 사이에 채널을 설정함으로써 통신을 할 수 있다. 마지막으로 종료 큐(CQ)는 여러 VI의 연산 종료에 대한 단일한 검사를 가능하게 해 준다. 예를 들어, P개의 프로세스와 연결을 설정한 경우를 가정해 보자. 프로세스가 P개의 채널을 통해 들어오는 메시지를 검사하기 위해서는 모든 VI를 하나씩 검사해야 한다. 그러나, VI를 하나의 CQ에 수신에 대하여 연관시키면, 프로세스는 CQ만을 검사함으로써 연관된 임의의 VI에 수신 메시지가 있는지 알 수 있다.

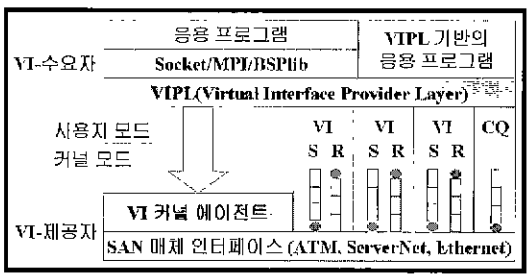


그림 2 VIA의 구조

2.2 VIA의 동작 원리

그림 3은 VI를 통해 두 단말이 통신하는 과정을 예시하고 있다. 통신을 하기 위해서는 먼저 두 단말이 VI를 생성하고, 이 VI 사이에 연결을 설정해야 한다. VIA의 무복사 원칙 하에서 성공적인 통신을 하기 위해서는 수신자가 송신이 이루어지기 전에 수신 데이터를 저장할 버퍼를 지정해야 한다. 이와 같은 송/수신에 관계된 정보는 디스크립터를 통해 커널 에이전트에 전달되는데, 이 디스크립터를 VI에 삽입하는 것을 포스팅이라고 부른다.

송신자는 수신자가 디스크립터를 포스팅했다는 전제에서 송신 큐에 디스크립터를 포스팅하여 송신을 시작한다. 디스크립터가 포스팅되면 네트웍 인터페이스 카드(NIC)는 순서대로 VI로부터 디스크립터를 꺼내 해당 정보에 따라 네트웍으로 데이터를 전송한다. 네트웍을 통해 전송된 데이터는 수신자의 해당 VI의 수신 큐에 있는 디스크립터가 지정한 버퍼에 저장된다. 연산이 완료되면 NIC은 디스크립터의 상태 변수에 연산이 완료되었음을 표시한다.

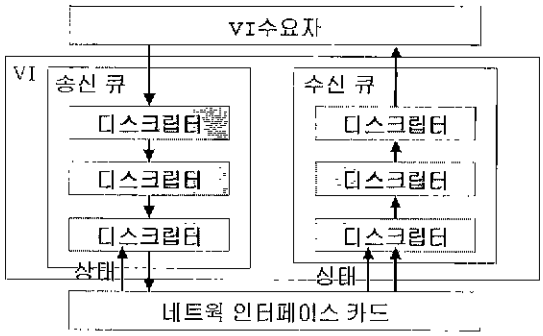


그림 3 VI를 통한 송/수신 과정

3. VIA 기반의 병렬 라이브러리

응용 개발자를 위해 VIA는 Virtual Interface Provider Layer(VIPL)이라 부르는 인터페이스를 제공한다. 비록 VIPL을 사용하여 직접 응용을 개발할 수 있지만, 프로그램의 이식성을 높이기 위해서는 PVM, MPI, BSPlib 등과 같은 널리 알려진 다양한 프로그래밍 라이브러리를 지원하는 것이 바람직하다. 따라서, 이번 장에서는 VIA를 이용하여 병렬 라이브러리를 구현할 때 고려해야 할 사항을 살펴본다.

3.1 VIA의 제약 조건

VIA를 기반으로 병렬 라이브러리를 구현할 때는 몇 가지 사항을 고려해야 한다. VIA에서 VI를 통해 임의의 프로세스에게 메시지를 전달하기 위해서는 수신자 측의 프로세스가 먼저 메시지를 수신하기 위한 기억 공간을 지정하고 수신 디스크립터를 포스팅하고 있어야 한다. 이 경우 문제는 수신 프로세스가 송신 프로세스가 메시지를 보낼 시점과 메시지의 크기를 알 수 없어 적절한 메모리 영역을 확보할 수 없다는 것이다.

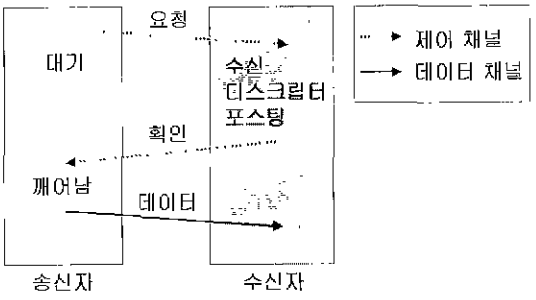


그림 4 3단계 핸드쉐이킹

이 문제는 3단계 핸드 쉐이킹(3-way handshaking)을 통해 해결할 수 있다(그림 4). 즉, 송신 프로세스는 수신 프로세스에게 요청 제어 메시지를 보냄으로써 송신 연산을 시작한다. 수신 프로세스가 그 제어 메시지를 받으면, 받은 정보에 따라 메모리를 할당하고 해당 수신 디스크립터를 포스팅한다. 이어 수신 프로세스는 제어 채널을 통해 송신자에게 확인 메시지를 보낸다. 마지막으로, 송신 프로세스가 확인 메시지를 받으면, 실제 데이터를 전송한다. 즉, 통신은 동기 모드로 동작한다. VI에 삽입된 디스크립터는 삽입 순서에 따라 처리되기 때문에, 제어 메시지와 데이터가 섞이지 않게 하기 위해서는 제어를 위한 채널과 데이터를 위한 채널을 별도로 두어야 한다.

3.2 제어 메시지 수신

비동기적으로 들어오는 제어 패킷을 처리하기 위해서는 폴링이나 인터럽트와 같은 비동기적 메커니즘이 필요하다. VIA는 통지 함수(notifying function)라고 부르는 함수를 통해 이러한 기능을 제공한다. 사용자는 통지 함수를 통해 특정 연산에 연관된 콜백(callback) 함수를 지정할 수 있다. 예를 들면, 특정 콜백 함수를 인자로 하여 VipRecvNotify 함수를 호출하면, 수신 연산이 종료되었을 때 해당 콜백 함수를 자동으로 호출하여 원하는 작업을 수행할 수 있다.

통지 함수는 VipRecvNotify나 VipSendNotify 함수를 사용하여 VI와, VipCQNotify 함수를 사용하여 CQ와 연관시킬 수 있다. 그림 5에서 보는 바와 같이 CQ와 연관된 VI (ViHandle)에 원하는 연산이 종료되면(RecvQueue) 지정한 콜백 함수(Handler)를 호출할 수 있게 한다.

```
VipCQNotify(VIP_CQ_HANDLE CQHandle,
            VIP_PVOID Context,
            void(*Handler)(VIP_PVOID Context),
            VIP_NIC_HANDLE NicHandle,
            VIP_VI_HANDLE ViHandle,
            VIP_BOOLEAN RecvQueue))
```

그림 5 VipCQNotify 함수

M-VIA의 경우, 통지 함수는 스레드로 구현되어 있다. 개념적으로 통지 함수는 프로세스의 생존 기간동안 특정 연산들을 위해 할당된 스레드이다. 여러 프로세스로부터 수신된 메시지를 처리하기 위해 VipCQNotify를 사용하기 때문에 VI의 개수와 관계없이 프로세스 당 두 개의 스레드가 생성된다. 즉, 통신을 위한 스레드와 계산을 위한 스레드가 각각 하나 씩 생성된다.

3.3 준비 모드 통신

VIA 기반의 병렬 라이브러리 구현에서 메시지를 전달하기 위해서는 먼저 제어 채널을 통한 제어 정보 교환이 이루어져야 한다. 이는 메시지 전달 시 미리 정보 전달 단계를 반드시 거치도록 강요함으로써 적지 않은 오버헤드로 작용한다. 특히 크기가 작은 메시지를 전달할 경우에 실제 메시지 전달에 소요되는 시간에 비해 제어 정보 전달을 위해 소요되는 시간이 상대적으로 크기 때문에 전체적인 지연 시간을 증가시킨다. 이러한 문제점을 해결하기 위한 방법의 하나로 작은 크기의 메시지에 대해서 제어 채널을 통해 메시지를 제어 정보와 함께 보내도록 할 수 있다. 현재 8KB 이하의 메시지에 대해서는 송신 디스크립터에 제어 정보와 메시지를 함께 실어 포스팅하고, 수신자 측에서는 제어 정보와 8KB의 메시지를 받아들일 수 있는 버퍼를 포스팅하여 제어 채널을 통해 바로 수신할 수 있다. 이와 같은 방식은 MPI에서의 네 가지 통신 모드 중 준비 모드에 해당한다.

4. MPI(Message Passing Interface)

MPI(Message Passing Interface)[10]는 메시지 전달 프로그래밍 모델에서 프로세스 사이의 메시지 전달을 위한 표준 명세이다. MPI는 사용자에게 쉬운 프로그래밍 인터페이스를 제공하며, 가급적 불필요한 메모리 복사를 줄이고, 통신과 지역적인 계산의 중첩을 통한 효율적인 통신 방법을 제공한다. 또한 다양한 플랫폼에서 이기종 환경의 통신을 가능하게 하며, C, C-+, Fortran 77 등을 지원한다. MPI는 send와 receive로 이루어지는 일대일 통신을 위한 기본적인 함수 이외에도 집합적인 통신을 위한 함수와 프로세스의 그룹 지정 및 이들 간의 그룹 통신을 위한 함수, 위상 정의 등과 같은 다양한 기능을 제공한다.

MPI에서 데이터의 크기 및 송수신의 상태에 따라 효율적인 데이터 전달을 위해 일대일 통신을 위한 함수는 다음과 같은 네 가지의 통신 모드를 지원한다.

- 표준 모드 (Standard Mode)
- 버퍼 모드 (Buffered Mode)
- 동기 모드 (Synchronous Mode)
- 준비 모드 (Ready Mode)

버퍼 모드는 송신자와 수신자 사이에 비동기적인(asynchronous) 동작을 지원하기 위해 메시지를 임시로 버퍼에 저장하는 방식이며, 동기 모드는 수신자가 수신을 할 준비가 되어 있는 경우에만 송신자가 메시지를 전달하는 방식이다. 준비 모드는 수신자가 항상 수신을 할 준비가 되어있다는 가정에서 송신자가 무조건 메시지를 전달하는 방식이며, 마지막으로 표준 모드는 메시지의 크기와 송수신 상태에 따라 앞에서 언급한 모드들을 선택적으로 취하는 방식이다.

또한, 해당 연산의 완료 시점에 따라 블록화(blocking) 방식과 비블록화(nonblocking) 방식을 지원한다. 블록화 송신은 송신자의 버퍼의 내용이 의미적으로 수신자에게 전달되었다는 것을 보장하는 것으로, 이 연산으로부터 복귀한 후에 버퍼 내용에 대한 조작이 이전의 송신 연산에 영향을 주지 않는다. 블록화 수신은 수신 버퍼에 송신자의 메시지가 도착하여 수신 연산이 완료될 때까지 복귀하지 않는다. 이와는 달리 비블록화 연산의 경우, 메시지의 송수신 연산의 완료 여부에 관계없이 연산에 대한 요청만을 시스템에 전달하고 해당 연산으로부터 복귀한다. 따라서, 연산으로부터 복귀가 데이터 전송을 보장해 주지 않기 때문에, 연산 이후의 버퍼에 대한 조작은 이전 연산에 영향을 미칠 수 있다.

4.1 VIA 기반의 MPI

이 장에서는 3장의 내용을 바탕으로 한 MPI 구현 내용을 살펴본다. VIA 기반의 MPI를 VMPI라고 부르기로 한다. VMPI는 병렬 프로그래밍을 위한 최소한의 루틴을 구현하는 데 목표를 두었다. 따라서, 초기화, 종료, 시스템 정보에 관련된 함수와 일대일 통신을 위한 블록화 함수와 비블록화 함수, 그리고 집합 통신으로 방송 함수를 구현했다.

송신 함수는 3.3에서 언급한 바와 같이 메시지의 크기에 따라 준비 모드와 동기 모드로 동작한다. 송신 함수에서 데이터를 전송하기 위해서는 우선 전송하고자 하는 데이터 영역을 VI 커널 에이전트에 등록해야 한다. MPI_Send 나 MPI_Isend 등에서 인자로 전달되는 버퍼는 임의로 달라질 수 있기 때문에 항상 내부적으로는 등록과 해제의 과정을 반복해야 하는 것이다.

수신 함수는 준비 모드에서는 수신 함수가 메시지가 도착하기 전에 수행되었는지 여부에 따라, 준비 모드를 위한 버퍼에서 사용자 버퍼로 복사를 하거나, 준비 모드를 위한 버퍼에서 임시 버퍼로 다시 임시 버퍼에서 사용자 버퍼로의 복사를 한다. 이러한 방식에 대한 정당성은 제어 메시지를 주고받는 오버헤드보다는 복사 오버헤드가 작다는 점이다. 동기 모드에서는 수신 함수가 메시지가 도착하기 전에 수행되었는지 여부에 따라, 임시 버퍼에서 사용자 버퍼로 복사하거나, 혹은 사용자 버퍼로 직접 데이터를 저장한다.

5. BSP(Bulk Synchronous Parallel)

처음 BSP 모델[15]은 하드웨어와 소프트웨어 사이의 교량 역할을 하기 위한 계산 모델로써 제안되었다. 그림 6과 같이 BSP 프로그램은 배리어 동기로 구분되는 일련의 슈퍼스텝으로 구성된다. 매 슈퍼스텝마다 각 프로세스는 지역 계산이나 메시지 교환을 한다. 이 때, 현 슈퍼스텝에서 교환한 메시지는 다음 슈퍼스텝에서 유효하다. BSP 모델은 1) 프로세서 개수, p, 2) 간격(gap)

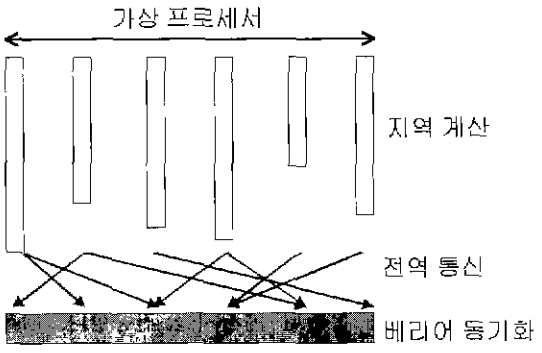


그림 6 슈퍼스텝

g, 3) 지연(latency) L 등 세 개의 인자로 이루어진다. 간격은 처리율과 관계된 인자로써 전체 교환(total exchange) 등과 같은 최악의 통신 환경 즉, 연속적인 소통 조건에서 단일 워드를 전송하는데 소요되는 비용을 나타내고, 지연은 배리어 동기를 하는데 소요되는 비용을 나타낸다. 슈퍼스텝 i의 수행 시간은 다음과 같이 정의된다. 단, w_i는 가장 오래 수행된 지역 계산의 양, h_i는 i번째 슈퍼스텝에서 한 프로세서에 의해 송/수신된 패킷의 최대 개수이다.

$$w_i + gh_i + L.$$

따라서, 한 프로그램의 총 수행 시간은 다음과 같다.

$$W + gH + LS.$$

PVM이나 MPI와 같은 다른 메시지 전달 라이브러리로 작성된 프로그램과 비교해서, BSP 프로그램은 이러한 비용 모델을 바탕으로 프로그램의 시간 복잡도를 정적으로 결정할 수 있다는 장점이 있다. 또한, 기능상의 유사점으로 인해, BSP 모델은 VIA 기반의 프로그래밍 모델로써 적합한 대상이다

BSP 모델이 제안된 이후, 몇 가지의 BSP 라이브러리가 제안되었다. 이들 라이브러리는 BSP 모델의 구현에 중점을 둔 것과 이 모델을 확장한 것으로 구분할 수 있다. 첫 번째 BSP 라이브러리는 Oxford BSP Library[16]이다 Oxford BSP 라이브러리는 Direct Remote Memory Access(RDMA)라고 부르는 공유 메모리 연산에 기반하고 있다. 이 라이브러리의 제약점은 단지 정적으로 할당된 공유 메모리만이 다른 프로세스에 의해 접근될 수 있다는 것이다. 대조적으로, Green BSP 라이브러리[17]는 Bulk Synchronous Message Passing(BSMP)라 부르는 메시지 전달 연산에 기초한다. Green BSP 라이브러리에는 명시적인 수신 연산이 없다. 대신, 수신자 측에 있는 큐에 메시지가 저장되고, 이 메시지는 다음 슈퍼스텝에서 유효하다. 뿐만 아니라, Green BSP는 프로세서 게수, 수신 패킷의 개수, 현재 슈퍼스텝 번호 등과 같은 BSP 기계에 대한 정보를 얻을 수 있는 함수도 함께 제공한다.

BSPlib[12]은 Oxford BSP 라이브러리와 Green BSP 라이브러리의 통합된 형태로 BSP 라이브러리의 표준이다. BSPlib의 주요 특징은 두 종류의 통신 모드에 있다. 하나는 일방적으로 원격 메모리에 대한 직접 접근이 가능한 연산이고, 다른 하나는 대량의 메시지를 동기화(bulk synchronization)하여 전달하는 연산이다. Oxford BSP 라이브러리가 정적인 공유 메모리 방식을 취한 반면, BSPlib은 등록 기법을 사용하여 동적인 공유 메모리를 지원한다. 패킷 전송 시간은 통신의 시작 오버헤드로 인해 패킷 크기에 대한 양수의 상수항을 갖는 일차 함수가 된다. 따라서, 작은 메시지들을 하나의 큰 메시지로 묶어 슈퍼스텝이 끝나는 시점에서 대용량의 동기(bulk synchronization)를 수행하는데 이를 복합 기법(combining)이라고 한다. 이 복합 기법과 더불어 BSP 라이브러리는 집중화로 인한 네트웍에서 충돌을 줄이기 위해 재순서(reordering) 기법을 채용하고 있다. 이들 기법들은 성능의 예측 가능성과 라이브러리가 BSP 모델에 근접하도록 하는데 기여한다.

반면, BSP 모델을 확장한 몇 가지 변종들이 있다. Parderborn 대학의 PUB 라이브러리[18]는 확장된 BSP 라이브러리 중의 하나다 이 라이브러리는 기본적인 BSPlib 루틴뿐만 아니라 집합 통신과 그룹 관리 기능을 제공한다. 특히, 제로-비용(zero-cost) 동기화와 비동기 통신 기법을 통해 더 좋은 성능을 제시했다. 비록 이러한 확장된 기능들이 BSP 모델의 분석적인 면에 좋지 않은 영향을 미치지만, PUB는 전형적인 메

시지 전달 라이브러리와 유사한 유연성을 제공한다.

5.1 VIA 기반의 BSP

이 장에서는 앞서 언급한 VIA기반의 병렬 라이브러리 기법과 더불어 BSPlib 구현 시 고려할 최적화 기법을 통해 실제 구현 내용을 살펴본다. VIA 기반의 BSP를 VBSP라고 부르기로 한다. 기본적으로 VIA는 메시지 전달 통신 모델에 기초한다. 전통적인 메시지 전달 통신에 더불어, VIA는 일방 메시지 전달 연산-원격 쓰기와 원격 읽기-을 제공한다 이러한 연산들은 BSPlib의 DRMA와 유사한 것으로 Remote Direct Memory Access(RDMA) 연산이라고 부른다. 비록 RDMA 루틴들이 DRMA 기능을 구현하기 위한 좋은 대상임에도 불구하고, M-VIA가 이들 루틴을 지원하지 않기 때문에, 우리는 모든 BSPlib 루틴들을 전통적인 메시지 전달 연산만을 사용하여 구현하였다.

그러나, 3단계 핸드 쉐이킹 기법은 데이터 전송 전에 제어 메시지를 교환해야 한다는 오버헤드가 있다. 특히, 메시지 크기가 작은 경우, 이 오버헤드는 두드러진다. 따라서, 우리는 묶음과 순서 재정의 기법 등 잘 알려진 기법을 사용한다. 또한, 준비 모드와 동기 모드 등 두 종류의 통신 모드를 통해 중첩 통신을 활용한다.

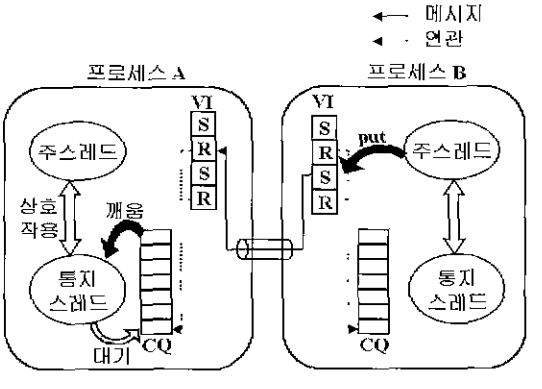


그림 7 VBSP 동작 메커니즘

첫 번째 최적화 기법은 복합 기법이다 복합 기법은 작은 메시지를 임시로 저장하여 큰 메시지를 만듦으로써 메시지 전달의 시작 비용을 분산시키는 기법이다. 이 기법은 네트웍 전송을 지역 메모리 쓰기 연산으로 대체한다. BSPlib의 경우, 한 슈퍼스텝에서의 모든 메시지를 임시 저장하여 동기화 단계에서 한꺼번에 전송하는 기법을 사용한다. 반면, VBSP는 단지 작은 메시지만을 임시 저장한다.

BSPlib과 VBSP의 차이점은 VBSP는 배리어 동기 전에 메시지 전송이 일어날 수 있다는 것이다. 저장된 메시지의 크기가 일정 수준을 넘으면, VBPS에서는 복합된 메시지를 전송한다. 메시지 전송을 동기화 시점까지 늦추는 것은 메시지 전송을 직렬화시킬 수 있다. 그러나, 계산과 통신을 중첩시킴으로써 이 지연은 줄일 수 있다.

두 번째 최적화 기법은 재순서 기법이다. 재순서 기법은 네트웍 충돌을 줄이거나 회피하기 위해 전송 순서를 섞는 기법이다. [11]에 의하면, 재순서 기법은 전형적인 네트웍에서 절대 전송시간과 전송 시간의 분산을 줄일 수 있는 것으로 알려져 있다

세 번째 최적화 기법은 계산과 통신을 중첩하는 것이다. 그러나, 단순한 계산-통신의 중첩은 3단계 핸드 쉐이킹 오버헤드로 인해 오히려 지연을 가중시킬 수 있다. 3단계 핸드 쉐이킹은 네트웍의 소통량을 증가시키고, 제어 메시지 교환이 완료될 때까지 데이터 전송을 지연시킨다 따라서, 우리는 핸드 쉐이킹의 부담을 갖지 않는 중첩된 통신을 추구한다. 이를 위해, VBSP는 준비 모드를 지원한다 VBSP 프로그램이 시작하면, 각 제어 채널에 대하여 임시 버퍼를 갖는 수신 연산을 몇 개 미리 시작한다 수행 시간 중에는 미리 시작된 수신 연산이 수행되자마자 새로운 수신 연산을 포스팅한다 즉, 항상 일정 수의 수신 연산이 미리 포스팅되어 있음을 보장한다. 이 준비 모드에 대한 주요한 정당성은 작은 메시지의 복사 오버헤드는 전송 지연 시간보다 작다는 것이다.

그림 7은 VBSP의 bsp_put() 수행에 대한 예시를 보이고 있다. 단, 수신측에서는 들어오는 제어 메시지를 위해 CQ에 통지 스레드가 기다리고 있다고 가정한다. 먼저, 송신 프로세스는 VI를 통해 송신 연산의 시작을 NIC에 알린다 그러면, 데이터는 네트웍을 통해 수신 프로세스로 전송되고 수신 프로세스에 의해 지정된 버퍼에 저장된

다. 모든 데이터가 도착하면 통지 스레드가 깨어나 콜백 함수를 호출하여 적절한 일을 수행한다.

6. 실험 결과

MPI의 성능 측정을 위해서 64MB의 메모리를 가진 Pentium Celeron 433Mz PC를, BSP의 성능 측정을 위해서는 64MB의 메모리를 갖는 Pentium III 450Mhz PC를 Fast Ethernet 스위치에 연결하였다. 네트워 인터페이스 카드는

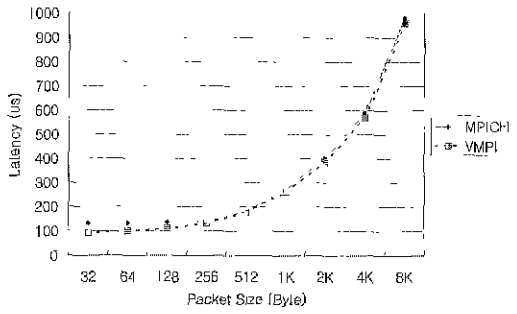


그림 8 MPI 지연 시간 비교

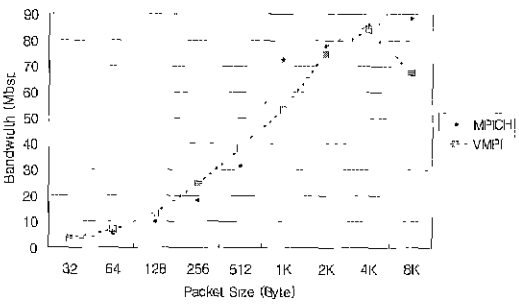


그림 9 MPI 대역폭 비교

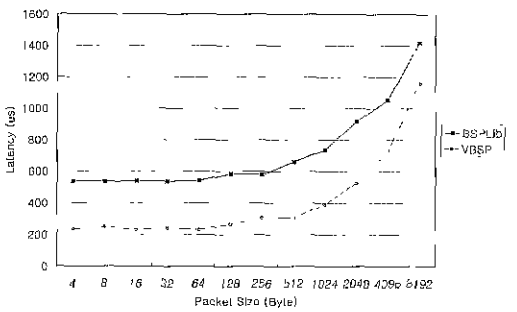


그림 10 BSP 지연 시간 비교

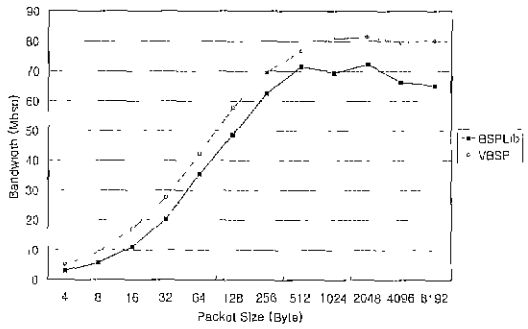


그림 11 BSP 대역폭 비교

M-VIA에서 지원하는 네트워 카드가 한정된 관계로 DEC의 Tulip 칩셋을 탑재한 삼성 SmartEther 랜 카드를 사용하였다. 각각의 컴퓨터에는 RedHat 6.0을 설치하였으며, M-VIA 1.0[19]을 설치하여 클러스터 환경을 구성하였다

인텔의 연구 결과에 의하면 소프트웨어 에뮬레이션만으로도 효과적으로TCP/IP를 대체할 수 있다고 한다[5]. 따라서, 우리는 VIA에 대해 M-VIA라는 소프트웨어 에뮬레이션을 채택하였다. 새로운 MPI라이브러리는 VMPI, BSPlib은 VBSP라고 부른다.

이 클러스터 환경에 M-VIA를 기반으로 한 VMPI와 VBSP를 구현하고, 이들을 널리 알려진 MPI 라이브러리인 MPICH 1.1.2 버전[20], BSPlib 1.41 버전과 비교한다. MPICH는 TCP/IP 기반 위에서 동작하며, BSPlib은 리눅스 시스템에서 TCP/IP 버전이 동작하지 않는 관계로 UDP/IP 버전을 설치하였다.

라이브러리들의 성능을 측정하는 기본 벤치마크로서 두 노드 사이의 지연 시간과 대역폭을 측정한다. 그림 8은 VMPI와 MPICH의 지연 시간을, 그림 9는 대역폭을 비교한 것이다. 지연 시간 측정을 위한 테스트 프로그램은 블록화 모드의 표준 일 대 일 통신 함수인 MPI_Send와 MPI_Recv를 사용하여 메시지를 주고받는 과정을 100회 되풀이 한 후, 평균 소요 시간을 반분함으로 값을 얻었다. 반면, 대역폭을 위한 테스트 프로그램은 지정된 크기의 메시지를 한쪽 노드에서 연속적으로 100번을 보내고 다른 한쪽 노드에서는 이를 모두 받은 뒤, 이에 대한 ACK 메시지를 보냄으로써 소요된 총 시간 동안 보낸 전체

메시지의 크기를 비트 단위로 환산하였다.

그림 8에서 보는 바와 같이 지연 시간의 경우 VMPI와 MPICH 사이에 작은 메시지를 제외하고는 거의 차이가 없는 결과를 보이고 있다. 이는 MPI가 복사를 줄이고 다양한 방법으로 통신의 효율을 높인 데에도 기인하지만, M-VIA가 가지고 있는 비효율성에도 기인한다. 즉, VMPI는 하나의 메시지를 보내기 위해서는 메모리 등록과 등록 해제, 메모리 복사 등의 오버헤드를 감수해야 한다. 따라서, MPI와 비교했을 때, 송신자 측에서의 복사를 줄였지만, 메모리 등록에 의한 오버헤드로 이 장점이 크게 부각되지 못하는 것으로 판단된다. 또한, 대역폭의 경우에는 흐름 제어에 의한 연속적인 송신 연산의 제한과 스레드 전환 오버헤드로 인해 큰 메시지에 대해서는 성능 떨어지는 현상을 보이고 있다

반면, BSP 라이브러리의 성능은 통신 성능에 대한 척도로써 일대일 통신에서의 대역폭과 지연 시간을 측정한다. 대역폭을 측정하기 위한 테스트 프로그램은 한 슈퍼스텝 내에서 고정된 크기의 메세지에 대하여 put 연산을 여러 번 반복한다 반면, 지연 시간을 측정하기 위한 테스트 프로그램은 고정 크기 메시지에 대한 put 연산과 베리어 동기화 연산을 여러 번 반복한다. 결과는 그림 10과 그림 11에 나타나 있다.

VBSP는 지연 시간의 관점에서 특히 작은 메시지 통신에 대하여 BSPlib보다 우수한 성능을 보였다 주요한 원인으로는 VIA가 UPD/IP보다 작은 프로토콜 오버헤드를 갖는다는 점과 VBSP의 동기화 오버헤드가 작다는 점을 들 수 있다. VBSP는 대역폭에서도 좋은 성능을 보이고 있다. VBSP의 대역폭은 대략 80Mbsp 정도이고 BSPlib의 대역폭은 대략 70Mbsp 정도를 보이고 있다. BSPlib의 경우 대역폭은 2KB보다 큰 경우에는 오히려 줄어들고 있다는 것에 주목할 필요가 있다. 이러한 결과는 VI와 UDP 비교를 한 인텔 연구진의 결과에 일치한다.

표 1과 표 2는 BSPlib과 VBSP의 비용 모델의 값을 나타내고 있다. 이 때, s는 각 프로세서의 명령어 수행 능력으로 여기서는 행렬 곱셈과 내적 계산의 평균 수행시간으로 결정한다. g와 L은 앞서 언급한 바와 같다. 비용 인자에 대한 자세한 내용은 [15]에 자세히 언급되어 있다. VBSP는 베리어 동기화 연산에서 BSPlib보다 좋은 결과를 보이고 있다. 비록 VBSP가 소규모 시스템에서는 좀 나쁜 g값을 보이고 있지만, 프로세서의 개수가 늘어남에 따라 BSPlib은 늘어나지만 VBSP는 줄어드는 추세를 보이는 점에서 보다 바람직하다고 할 수 있다.

표 1 BSPLib 기반의 BSP 비용 모델

P	s (Mflop)	L(maximum)		g(all-to-all)	
		flop/word	us/word	flop/word	us/word
2	89	30088.69	338 05	38.27	0 43
4		44108 96	496.08	35 85	0 40
6		55902.55	625 18	43.59	0 49
8		58324 30	655.49	44 00	0.50

표 2 VBSP 기반의 BSP 비용 모델

P	s (Mflop)	L(maximum)		g(all-to-all)	
		flop/word	us/word	flop/word	us/word
2	86	15275.48	176 86	41.57	0.48
4		21563.76	250 47	37.55	0 44
6		27742.78	322 53	36 91	0.43
8		33970 00	394.91	36 78	0 43

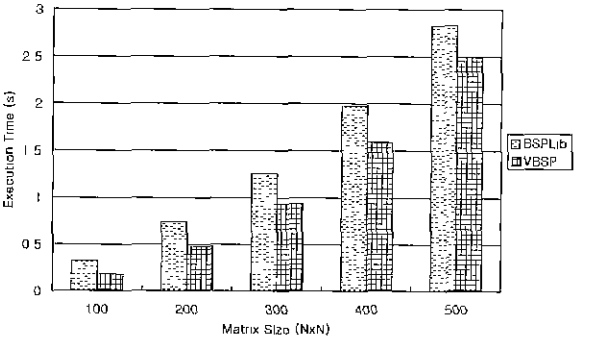


그림 12 4(2x2) 프로세서에서의 LU 분해 수행 시간

마지막으로, 몇 가지 응용 프로그램을 통하여 VBSP와 BSPlib의 성능을 비교한다. 그림 12는 행렬의 크기를 늘리면서 4개의 프로세서에서 수

행한 LU 분해 실험 결과이다. 이 LU 프로그램은 프로세서의 개수가 p^2이라고 가정한다. LU 분해 프로그램의 특징은 행렬의 크기가 커짐에 따라 계산, 통신, 동기화의 양이 더불어 증가한다.

표 3 리눅스 클러스터에서의 실행 시간(sec)
(정렬 원소 수 = 700,000, grid solver의 행렬 크기 = 100)

		p=2	p=4	p=6	p=8
Sorting	BSPlib	0.440	0.313	0.192	0.151
	VBSP	0.427	0.261	0.189	0.138
Grid Solver	BSPlib	0.585	0.657	1.204	0.996
	VBSP	0.545	0.498	0.472	0.558

표 3에는 700,000개의 원소에 대한 정렬과 100x100 행렬에 대한 grid-solver 프로그램의 수행 결과가 제시되어 있다. 정렬의 경우, 계산이 통신에 비해 상대적으로 많은 비중을 차지하기 때문에 통신 성능의 향상에 의한 영향이 두드러지지는 않다. 그러나, 이 결과는 VBSP가 프로세서의 개수에 대하여 확장성이 있다는 것을 보여준다. 반면, grid-solver 커널의 경우에는 통신이 계산보다 많은 비중을 차지한다. VBSP는 BSPlib보다 평균 68% 정도의 우수한 성능을 보인다.

7. 결론 및 토의

이상의 실험으로부터 우리는 VIA를 기반으로 한 MPI가 효과적으로 구현되지 못함을 볼 수 있다. 이에 대한 원인으로 VIA의 수신측에서 먼저 포스팅을 해야 한다는 제약 조건을 들 수 있다. VIA의 무복사 정책이 올바르게 동작하기 위해서는 송신자가 데이터를 보내기 전에 수신자가 받을 준비가 되어 있어야 한다. 그러나, MPI가 송/수신을 위해 가정하고 있는 비동기 통신에 있어 수신자 측에서 이러한 무복사 프로토콜을 직접 적용할 수는 없다. 더욱이, 무복사 통신을 위한 오버헤드는 복사 오버헤드와 비등할 수도 있다. 특히 소프트웨어 에뮬레이션을 사용했을 때 성능은 그다지 좋지 않음을 볼 수 있다. 따라서, 빠른 통신 프로토콜을 이용하여 표준 라이브러리를 구현할 때에는 통신 프로토콜과 프로그래밍 라이브러리 사이의 구조적인 유사성에 주목해야 한다.

BSP의 경우는 MPI와 달리 긍정적인 결과를 보이고 있다. 그런데, 한 가지 기억해야 할 점은 현재 구현이 M-VIA라는 소프트웨어 에뮬레이션을 사용하고 있기 때문에 VIA의 특징을 충분히 활용한 것은 아니라는 점이다. 가장 심각한 것은 M-VIA가 RDMA 기능을 제공하고 있지 않다는 점이다. BSPlib의 DRMA 루틴을 구현하기 위해 전통적인 메시지 전달 함수들을 사용하였기 때문에, 우리는 우선-포스팅(preposting) 조건과 불필요한 흐름 제어 문제를 해결해야 했다. 이들은 빈번한 스레드 전환과 상호 배제, 흐름 제어에 의해 대기 상태를 유발한다. 우리는 VIA의 RDMA 루틴을 사용함으로써 이러한 오버헤드를 상당히 줄일 수 있을 것으로 기대한다.

또 하나의 병목의 원인으로 메모리 등록에 의한 오버헤드를 들 수 있다. 보호와 메모리 스와핑을 방지하기 위해, VIA는 메모리 등록 메커니즘을 사용하고 있다. VipRegisterMem 함수가 이러한 용도로 사용된다. M-VIA의 경우 VIPL 루틴을 모두 시스템 호출을 통해 구현하였다. 이는 모든 통신 루틴을 호출할 때마다 적어도 두 번-한 번은 메모리 등록을 위해, 다른 한 번은 메모리 등록 해제를 위해-의 시스템 호출이 일어난다는 것을 의미한다. 이들은 복사 오버헤드에 준하는 부담이 된다. 비록 우리는 미리 등록된 복합 버퍼를 통해 이 등록 오버헤드를 피하였지만, VIA가 다른 메시지 전달 라이브러리에 적용될 때에는 이 오버헤드가 심각한 문제를 일으킬 수 있다. 따라서, VIA를 위한 소프트웨어 에뮬레이션의 경우 통신 루틴뿐만 아니라 등록 루틴에 대한 최적화를 할 필요가 있다.

마지막으로 고려해야 할 사항으로 VIA 프로토콜을 하드웨어로 구현한 VIA-NIC을 고려해야 한다. 현재 M-VIA의 구현은 리눅스 Ethernet 드라이버에 전적으로 의존하고 있다. 따라서, M-VIA와 인텔의 에뮬레이션 사이에도 성능의 차이를 보인다. 이전의 인텔의 보고서에서 예측했던 것처럼[5] 단말 사이의 지연 시간은 VIA 네트웍 카드를 사용함으로써 10us까지 줄일 수 있을 것이고, 하드웨어로의 구현은 VIA의 데이터 링크 층이나 네트웍 층의 최적화를 도모할 수

있을 것이다. 더욱이, 지능적인 네트웍 인터페이스 카드는 CPU의 통신 부담을 경감시킬 수 있고, 이는 계산과 통신의 중첩 효과를 발휘하게 할 것이다.

참고문헌

[1] Gregory F. Pfister, "In Search of Clusters," Prentice Hall PTR, 1998.

[2] David E. Culler et al., "Parallel Computing on the Berkeley NOW," JSPP'97, 1997.

[3] Thomas Sterling et al., "An assessment of Beowulf-class computing for NASA requirements' Initial findings from the first NASA workshop on Beowulf-class clustered computing," IEEE Aerospace Conference, Mar. 1998.

[4] J. Kay and J. Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP," IEEE/ACM Transactions on Networking, Vol. 4, No. 6, pp. 817-828, Dec. 1996.

[5] D. Dunning et al., "The Virtual Interface Architecture," IEEE Micro, Vol. 18, Num. 2, pp.66076, Mar./Apr. 1998

[6] T. von Eicken et al., "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," Operating Systems Review, Vol. 29, No. 5, pp. 40-53, Dec. 1995.

[9] R. A.F. Bhoedjang, Tim Ruhl, and Henri E. Bal, "User-Level Network Interface Protocols," IEEE Computer, Vol. 31, Num. 11, pp.53-60. Nov. 1998.

[10] Message Passing Intertace Forum, "MPI: A Message Passing Interface Standard," Technical Report Version 1.1, Univ. of Tennessese, Knoxville, Tenn, 1995.

[11] A. Geguelin et al,. "A Users' Guide to PVM(Parallel Virtual Machine)," Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, 1991.

[12] J. M.D. Hill et al., "BSPlib' The BSP programming Library," Parallel Computing, Vol 24, Num. 14, pp. 1,947-1,980, 1998.

[13] R. Ceceres et al., "Characteristics of Wide-Area TCP/IP Conversations," Proceedings of the SIGCOMM'91 Symposium on Communications Architectures and Protocols, pp. 101-112, Aug. 1991.

[14] J. Kay and J. Pasquale, "Importance of Non-Data Touching Processing Overheads in TCP/IP," Computer Communication Review, Vol. 23, No 4, pp.256-268, Oct. 1993.

[15] D.B Skillicorn, J M.D. Hill, and W.F. McColl, "Questions and Answer about BSP," Techincal Report PRG-TR-15-96, Oxford University Computing Laboratory, Nov. 1996.

[16] R Miller, "A Library for Bulk-Synchronous Parallel Programming," Proc. British Computer Soc. Parallel Processing Specialist Group Workshop General Purpose Parallel Computing, pp. 100-108, Dec. 1993.

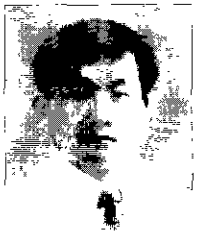
[17] Mark W. Goudreau et al., "Portable and Efficient Parallel Computing Using the BSP Model." IEEE Transactions on Computers, Vol 48, Num.7, pp. 670-689, July 1999.

[18] Olaf Bonorden et al, "The Paderborn University BSP(PUB) Library-Design, Implementation and Performance," Proceedings of the 13th International Parallel Processing Symposium and 19th Symposium on Parallel and Distributed Processing, Apr. 1999.

[19] NERSC, "M-VIA: A High Performance Modular VIA for Linux," http://www.nersc.org/research/FTG/via.

[20] Patrick Bridges et al., "Installation Guide to mpich, a Portable Implementation of MPI," Jun. 1995.

하 순 회

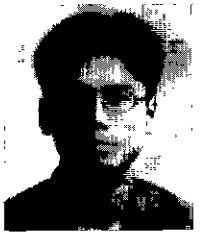


1985 2 서울대학교 전자공학과 학사
1987 2 서울대학교 전자공학과 석사
1992 5 U C Berkeley EECS 박사
1993~1994 현대 전자 근무
1994~현재 서울대학교 컴퓨터공학과 조교수
관심 분야:하드웨어-소프트웨어 통합 설계, 신호 처리를 위한 설계 방법론, 병렬 처리, 마이크로 프로세서 구조
E-mail:sha@iris snu ac kr

기 양 석

1996.2 서울대학교 컴퓨터공학과 학사
1998.2 서울대학교 컴퓨터공학과 석사
1998 3~현재 서울대학교 컴퓨터공학과 박사 과정
관심 분야 병렬 프로그래밍 환경, 클러스터 시스템, 시각 프로그래밍, 네트워 디바이스, 분산 서비
E-mail:enigma@iris snu ac.kr

김 선 재



1996 2 동국대학교 컴퓨터공학과 학사
1997 2~현재 서울대학교 컴퓨터공학과 석사 과정
관심 분야 클러스터 시스템, 리눅스, 디바이스 드라이버
E mail sjkim@iris snu ac kr

2000년 논문지 편집회의 일정

시스템 및 이론 : 홀수달 마지막주 금요일 16시
소프트웨어 및 응용 : 짝수달 마지막주 금요일 (4, 10월 제외) 16시
데이터베이스 : 2, 5, 8, 11월 셋째주 수요일 16시
정보통신 : 2, 5, 8, 11월 넷째주 수요일 16시
위원장단회의 : 2, 5 8, 11월 *표시한 날짜 15 - 16시

	시스템 및 이론	소프트웨어 및 응용	데이타베이스	정보통신
1월	28(금)			
2월		25(금)	*16(수)	22(화)
3월	24(금)			
4월		21(금)		
5월	26(금)		17(수)	24(수)
6월		30(금)		
7월	28(금)			
8월		*25(금)	16(수)	23(수)
9월	29(금)			
10월		20(금)		
11월	*24(금)		15(수)	23(수)
12월		29(금)		

※ 회의일정은 사정에 따라 변경될 수 있음