Myrinet Cluster 기반의 사용자 프로그래밍 환경

포항공과대학교 남경완 · 이승구

1. 서 론

고성능 마이크로프로세서와 고성능 네트웍의 발달은 PC로 연결된 클러스터 시스템을 새로운 병렬 패러다임으로 부각시키고 있다. 이러한 클러스터 시스템을 연결하는 네트웍으로는 ATM, Fibre Channel, Gigabit Ethernet, 그리고 Myrinet[1] 등이 있는데, 이 중에서 기가비트급 지역망인 Myrinet은 적은 비용으로 프로세서들 간에 빠른 통신을 지원할 수 있고 민간에서 상업적으로 개발되어 유통되고 있다는 장점으로 이를 이용한 많은 클러스터 시스템들이 개발되어왔다. 본 논문에서는 Myrinet을 기반으로 구축되어진 클러스터 시스템에서 실제로 사용자가 프로그래밍 할 수 있는 기존의 환경을 알아보고, 본 연구에서 구현한 사용자 프로그래밍 환경을 소개한다 또한 앞으로 보완이 되어야할 점을 알아보며 결론을 맺는다.

2. 기존의 사용자 프로그래밍 환경

Myrinet을 기반으로 구축된 기존의 클러스터 시스템에서 사용자가 프로그래밍할 수 있는 환경은 성능 측면과 프로그래밍의 편이성 측면에 따라 크게 4가지로 구분된다.

첫째, MyriAPI나 GM[2]과 같이 Myricom[2] 사에서 자체적으로 개발하여 제공하는 사용자 프로그래밍 환경이다. 초기의 MyriAPI의 성능 면을 보완한 GM은 현재 최신의 Myrinet 하드웨어(64-bit Myrinet/PCI with LANai 7) 상에서 대역폭 1.12 Gbps의 성능을 보인다. 가장 기본적으로 제공되는 환경으로 자동적으로 네트웍 구조를(topology) 탐지하는 mapping 기능과 순간적인 네트웍 결함을 회복하는 기능이 있다.

둘째는 Myricom사의 초기 MyriAPI의 낮은 성능을 보완하여 개발되어진 AM[3], FM[4], PM[5], BIP[6], Trapeze[7]와 같은 고성능 통신 라이브러리이다. AM과 FM은 능동적으로 메시지를 처리할 수 있는 핸들러라는 개념을 이용했으며, FM은 특히 작은 양의 메시지에서의 성능 향상에 초점을 두고 개발이 되었다. AM과 FM은 멀티 프로세스 환경을 지원하는 등의 성능 업그레이드를 통해 AM II[3]와 FM 2.x[8]로 개발이 되어져있다 BIP는 처음으로 사용자 프로세스 레벨에서 기가비트급의 대역폭을 제공한 통신 라이브러리로서, 메시지를 전송할 때 데이타 경로의 어느 구간에서도 병목이 되지 않도록 파이프라인 방식으로 전송되도록 설계되었다. 이러한 개념은 PM의 immediate sending과 Trapeze의 cut-through 전송과 유사하다. Trapeze는 고속 통신 라이브러리위에 TCP/IP 프로토콜을 올린 Trapeze TCP/IP[9]가 대역폭 1.147 Gbps의 성능을 보임으로써 TCP/IP 프로토콜로는 처음으로 기가비트급 성능을 보였다. 이를 위해 checksum offloading이나 zero-copy socket과 같은 개념을 이용했다. 이러한 고속 통신 라이브러리의 공통적 개념은 물리적으로는 네트웍으로 연결되어 있지만 가상적으로 사용자 프로세스간에 직접 연결되어진 것 같은 ULN(User Level Networking)으로 이는 Intel, Compaq 그리고 Microsoft에 의해 산업표준으로 제안된 VIA (Virtual Interface Architecture)[10,11]로 이어졌다.

셋째는 MPICH-GM[2], MPI-AM[3], MPI-FM[8], MPICH-PM[12], MPI-BIP[13]와 같이 고성능 통신 라이브러리위에 병렬 프로그래밍 환경인 MPICH[14]를 포팅한 환경으로 앞에서 언급한 환경들의 대부분이 MPI 환경을 지원하고 있다. 또한, VIA를 기반으로 한 MPI 환경도 개발되어졌으며 이는 VIA 호환성의 하드웨어에서 동작이 가능하다[15]. 이렇게 개발되어진 MPI 환경들은 통신 라이브러리의 속도를 작게는 3.3 us 정도만 저하시키면서도 MPI 환경을 제공하는 장점이 있다. 그러나, 앞서 언급한 프로그래밍 환경들은 성능 측면에서 고성능을 제공하는 반면에 사용자가 메시지 전달에 일일이 신경을 써야하므로 프로그래밍의 편이성 측면에서 단점이 있다. 이를 보완하고자 분산 네트웍 환경에서 공유 메모리 프로그래밍 환경을 제공하는 개념인 분산공유메모리(DSM: Distributed Shared Memory. 이후 DSM)가 1986년 Kai Li에 의해 제안되었다[16].

네번째 환경인 SCASH[17]와 DOSMOS[18]가 고성능 통신 라이브러리위에 DSM을 구현한 프로그래밍 환경이다. SCASH는 PM[5]을 기반으로 구현된 DSM 환경으로 adaptive coherence protocol에 대한 실험적 결과를 보였으며, DOSMOS는 BIP[6]를 기반으로 쓰레드를 이용해서 구현한 것이다. 이러한 공유메모리 프로그래밍 환경을 위해서 FM의 shmem put/get와 Global Array[8], Trapeze의 GMS(Globally-mananged Memory System)[19]와 같은 관련 연구가 제안되었다. 그러나, 고성능 통신 라이브러리를 이용한 DSM 환경들은 편이성에 비해 아직 성능면에서 Myrinet의 성능을 최대한 이용하지 못하고 있으며 고속 통신망에 맞는 실험이 부족한 상황이다. 이를 위해 본 연구에서는 Myrinet 클러스터 기반에서 고성능 통신 라이브러리인 BIP를 이용하여 DSM 환경인 LIGHT (LIGHTweight DSM)을 구현했으며 그 구조와 성능평가를 다음 장에서 살펴본다.

3. LIGHT(LIGHTweight DSM)

LIGHT는 Myrinet 클러스터 기반에서 Myrinet의 기가비트급 성능을 최대한 이용하기 위해 구현된 DSM 환경으로 다음과 같은 특징을 갖고 있다.

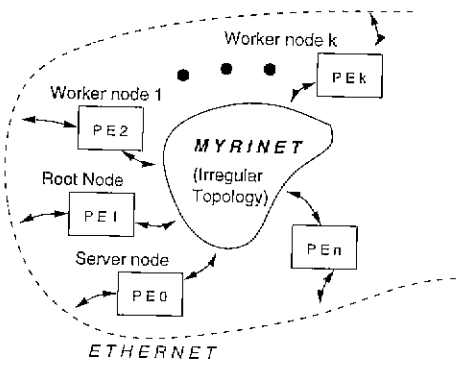


그림 1 LIGHT의 Multi-Network 구조

첫째, 이름 그대로 lightweight한 시스템으로 시스템에 거의 부하를 주지않으며 다른 시스템에 쉽게 이전이 가능하다. 둘째, 멀티 네트웍 환경을 이용하고 있다 즉, 100 Mbps 이더넷의 버스구조의 장점과 1.28 Gbps Myrinet의 스위치 구조의 장점을 모두 이용하는 멀티 네트웍 환경을 이용한다(그림 1). 이는 Myrinet이 이미 연결되어진 이더넷에 덧붙여서 구현이 되어진 경우를 고려하여 착안했으며 그런 환경을 메시지의 특성에 맞게 이용하고 있다. 셋째, 하드웨어의 성능을 최대한 이용하기 위해 현재까지 발표된 가장 고성능의 통신 라이브러리인 BIP를 이용하고 있다 (2000년 1월 현재 BIP는 1.18 Gbps의 대역폭과 5 us이하의 latency를 보인다). 다음 각 장에서는 LIGHT의 구조와 기능 그리고 동작 원리를 살펴본다.

3.1 LIGHT의 전체 구조 및 메모리 관리

LIGHT는 사용자 라이브러리 형태로 구현되었다. 커널 수정을 하지 않음으로써 쉽게 포팅할 수 있는 면과 시스템에 거의 부하를 주지 않는 lightweight한 특성을 반영한다. LIGHT를 이용한 실제 응용 프로그램의 수행은 그림 1에서 보듯이 응용 프로그램을 수행하는 Root 프로세스와 Worker 프로세스, 그리고 DSM 정보를 관리하는 Server 프로세스로 나뉜다 각 프로세스는 멀티 네트웍 환경인 Myrinet과 Ethernet을 메시지의 종류에 따라 모두 이용하게 된다. Myrinet을 이용한 네트웍 구조는 비정형화 구조

(irregular topology)로 확장이 용이한 구조이며 고성능 통신라이브러리인 BIP를 메시지 전달에 사용한다. Ethernet은 버스 구조이며 통신을 위해서 UDP/IP를 사용한다.

LIGHT에서 일관성 유지의 단위(granularity)는 페이지를 기본으로 하며 가상메모리 페이지 프로텍션 하드웨어를 이용하여 공유변수에 대한 읽고 쓰기를 감지한다. 읽기가 가능한 공유변수는 여러 개를 허용하나 쓰기 변수에 대한 소유권은 하나의 노드만이 가질 수 있다. 쓰기 권한을 가질 때는 Lock을 획득하고 페이지 무효화(invalidation)신호를 보냄으로써 하나의 노드만이 쓰기를 할 수 있는 것을 보장한다.

3.2 LIGHT의 세부 구조 및 기능

LIGHT에서 Root와 각 Worker 프로세스는 응용프로그램 프로세스와 LIGHT 사용자라이브러리, 통신 모듈로 구성되고 Server 프로세스는 서버 역할을 하는 프로세스를 중심으로 구성된다(그림 2). 각 구성 요소를 살펴보면 다음과 같다.

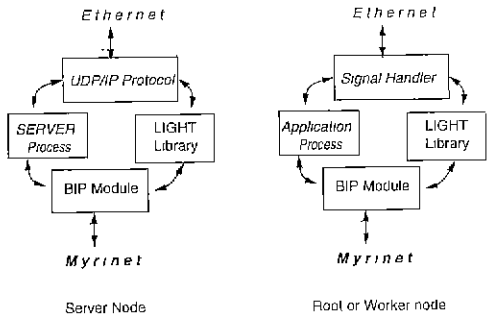


그림 2 Root, Worker, 그리고 Server 프로세스의 구조

3.2.1 LIGHT 서버

LIGHT 서버 프로세스는 DSM 정보를 관리하는 역할을 하며 특정 노드에 초기에 할당되며 그 기능은 다음과 같다. 첫째, 공유 메모리의 할당과 해제를 관리한다. 둘째, 공유 메모리의 순차성을 보장하는 Lock이나 Barrier의 할당과 해제를 관리한다. 중앙집중식 서버이며 스위치 단위의 확장성(Switch-Level Scalability)을 갖는 분산 서버로 개발중이다. 그런데, LIGHT 시스템에서 LIGHT 서버는 DSM 메타데이타(metadata)를 관리할 뿐이다. 실제로 공유변수나 Lock, Barrier를 할당하는 것은 Root 프로세스의 역할이다.

3.2.2 LIGHT 사용자 라이브러리

LIGHT는 사용자 레벨의 DSM으로 사용자가 이용할 수 있는 사용자 라이브러리를 제공한다. LIGHT에서 제공되는 사용자 함수는 다음과 같다(각 함수의 initial은 ldsm을 이용하고 있다).

void ldsm_init(void) : LIGHT를 초기화 시켜주는 함수로서 반드시 호출해야하는 함수이다. 이 함수를 호출하면 LIGHT의 전역변수나 페이지테이블 등과 같은 메타데이타와 멀티네트웍 연결이 초기화 된다.

char *ldsm_malloc(int shmsize) : 공유변수를 동적 할당할 때 호출하는 함수이다.

void ldsm_wait_barrier(int situation) : 연산의 동기화때 호출되는 함수이다 situation 변수가 '0'이면 제어신호만 주고받는 동기화이고, '1'이면 Root 프로세스가 모든 데이터를 모으면서 데이터 일관성을 유지하는 작업도 한다.

void ldsm_free(void *shmaddr) : 할당되어진 동적 공유변수를 풀어주는 함수이다.

void ldsm_lock_acquire(int lockid) : lockid를 얻을 때 호출하는 함수이다.

void ldsm_lock_release(int lockid) : lockid를 풀어줄 때 호출하는 함수이다.

사용자 라이브러리는 ldsmlib.a 형태로 제공되며 시스템에 거의 부하를 주지않으며 쉽게 포팅이 되는 LIGHT의 lightweight 특성을 보여준다.

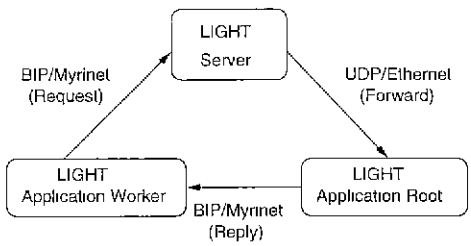


그림 3 Multi-Network에서의 메시지 경로

3.2.3 LIGHT의 통신 모듈

LIGHT의 통신 모듈은 LIGHT의 특징인 멀티네트웍과 고성능 통신 라이브러리가 반영되어 있다. 그림 3에서 보듯이 Forward 메시지는 UDP/

Ethernet 네트웍을 이용하고, Request와 Reply 메시지는 BIP/Myrinet을 이용한다. Request 메시지에는 페이지 결함(page fault)에 의한 페이지 요구 메시지와 페이지 소유권(ownership)을 요구하는 메시지 등이 있다. Request 메시지는 서버에서 순차적으로 처리된다. 서버는 Request 메시지를 분석하여 해당 노드로 Forward 메시지를 UDP/Ethernet으로 보낸다. Forward를 수신한 노드는 요구에 맞게 처리한 후 Reply 메시지를 BIP/Myrinet을 통해 보낸다.

멀티 네트웍 구조를 이용하는 것은 2가지 이유가 있다. 첫째, LIGHT에서는 페이지 결함 처리나 상대 노드로의 요구 메시지 처리를 시그널(Signal)을 이용해서 처리하는 방식을 사용하기 때문이다. 시그널은 Unix Socket에서 가능하므로 UDP/Ethernet 네트웍이 이용된다. 한편, DOSMOS[18]에서는 쓰레드를 이용해서 구현했다. 둘째, UDP/Ethernet은 버스 구조이므로 페이지 무효화와 같은 메시지를 다중전송(multicast)할 때 효율적이다. 이것은 Myrinet과 같은 스위치 구조에서는 다중전송이 지원되지 않는 점을 보완한다.

3.3 LIGHT의 동작 : 읽기와 쓰기

LIGHT는 초기에 LIGHT 서버 프로세스를 특정 노드에서 실행하고 Root 노드에서 응용 프로그램을 실행함으로써 동작된다. 각 노드에는 고유한 노드 ID가 할당되며 이러한 네트웍 구조(topology)는 ldsm.hosts라는 컨피규레이션 파일로 미리 지정한다. 실행이 되면 Root 프로세스는 지정된 수만큼의 Worker 프로세스를 rsh을 이용해 실행시킨다. 그리고 공유변수를 생성해서 LIGHT 서버에 등록하고 초기화 시킨다. 공유변수는 동적 메모리 할당이 되며 이때 mmap() 시스템 함수가 이용된다. 공유 메모리는 페이지 단위로 관리되며 각 페이지 플래그(flag)는 mprotect() 시스템 함수에 의해 제어된다. rsh에 의해 실행된 Worker 프로세스도 공유변수를 할당하나 mprotect() 함수에 의해 무효화(NO R/W)상태로 초기화됨으로써 페이지를 읽거나 쓰면 페이지 결함이 발생한다.

LIGHT에서 읽기 결함이 발생했을 때의 한 예가 그림 4에 나와있다. Worker 프로세스 1에서 임의의 무효화된 페이지를 읽으면 세그멘테이션 결함(segmentation fault)이 발생하여 SIGSEGV 시그널을 발생시킨다[그림 4의 (1)]. 이 시그널은 연산 중인 사용자 프로세스로 전달되어 미리 signal() 시스템 함수에 의해 등록시켜놓은 sigsegv_handler()로 CPU의 실행권한을 옮긴다. 핸들러는 발생한 에러가 읽기결함인지 쓰기결함인지를 확인하고 결함이 발생한 주소를 확인하여 Request 메시지를 만들어 LIGHT 서버로 보낸다[그림 4의 (2)]. 서버는 Request 메시지를 분석하여 페이지를 갖고 있는 프로세스에게 Forward 메시지를 UDP/Ethernet을 통해 보낸다[그림 4의 (3)]. 이때 읽기 모드의 페이지는 동시에 여러개가 존재할 수 있는데 노드 ID가 가장 작은 노드에게 보낸다. 해당 노드에 도착한 Forward 메시지는 SIGIO 시그널을 발생시켜 연산중인 사용자 프로세스를 잠시 중단시키고 sigio_handler() 핸들러로 실행권한을 옮긴다. 핸들러는 Forward 메시지를 분석하여 해당 페이지를 Reply 메시지로 요구한 노드에게 보낸다[그림 4의 (4)]. Reply를 받은 sigsegv_handler()는 페이지에 대해 처리를 한 후 중단되어있던 사용자 프로세스에게 실행권한을 넘긴다. 사용자 프로세스는 다시 그 페이지를 읽어 연산을 수행한다.

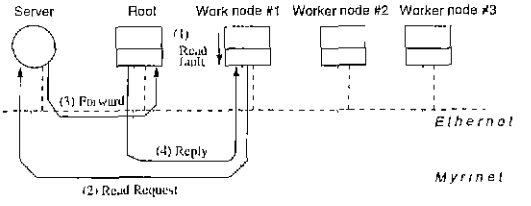


그림 4 읽기결함이 발생했을 때의 메시지 경르

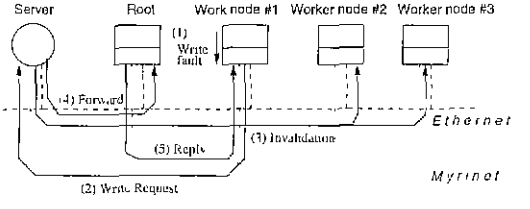


그림 5 쓰기결함이 발생했을 때의 메시지 경로

그림 5는 Worker 프로세스 1에서 쓰기결함이 발생했을 때의 예이다. 무효화되어있거나 혹은

읽기모드로만 되어있는 페이지에 쓰기를 하면 세그멘테이션 결함이 발생하여 SIGSEGV시그널을 발생시킨다[그림 5의 (1)]. 이 시그널은 연산중인 사용자 프로세스를 중단시키고 미리 등록되어있는 sigsegv_handler()로 실행권한을 옮긴다. 핸들러는 쓰기결함인지를 확인하고 해당 페이지에 대한 Request 메시지를 서버로 보낸다[그림 5의 (2)]. 서버는 Request 메시지를 받은 후에 쓰기결함인 경우에 해당 페이지의 소유권을 확인한다. 임의의 한 노드가 소유권을 가지고 있으면 그 노드로 Forward 메시지를 보낸다[그림 5의 (4)]. 그러나, 여러개의 복사본(copyset)이 있을 경우에는 가장 낮은 ID의 노드를 제외한 복사본 노드에게 무효화 신호를 보낸 후에[그림 5의 (3)] 남은 노드에게 Forward 메시지를 보낸다[그림 5의 (4)]. Forward 메시지를 받은 노드는 해당 페이지를 Reply 메시지로 보내고[그림 5의 (5)] 해당 페이지를 무효화 시킨다. 이렇게 함으로써 쓰기권한은 하나의 노드만이 갖는다. Reply 메시지를 받은 sigsegv_handler()는 해당 페이지에 대한 처리를 한 후에 실행권한을 중단되어있던 사용자 프로세스로 넘긴다. 사용자 프로세스는 다시 그 페이지를 읽으면서 연산을 수행한다.

```
/*-------------------------------------------------------*/
/*                  L I G H T                            */
/*      Matrix Multiplication Pseudo Code                */
/*-------------------------------------------------------*/
#include "ldsm.h"
typedef int element_t;
element_t * matA, *matB, *matC;    /* Shared memory area */
void main(void)
{
    int i,j,k;
    int product;

    ldsm_init();    /* ldsm initialization */

    matA = (element_t *) ldsm_malloc(matsize*matsize*sizeof(element_t));
    matB = (element_t *) ldsm_malloc(matsize*matsize*sizeof(element_t));
    matC = (element_t *) ldsm_malloc(matsize*matsize*sizeof(element_t));
    if(ldsmnodeid == 0) {
        init_matrices();
    }
    start_row = (matsize/ldsmnumnodes)*ldsmnodeid;
    if(ldsmnodeid == 0) start_row = 0;
    end_row = (matsize/ldsmnumnodes)*(ldsmnodeid+1);

    if(ldsmnumnodes > 1) ldsm_wait_barrier(0);
    for(i=start_row;i<end_row;i++)
        for(j=0;j<matsize;j++){
            product = 0;
            for(k=0;k<matsize;k++)
                product += read_elem(matA,i,k) read_elem(matB,k,j);
            write_elem(matC,i,j,product);
        }
    if(ldsmnumnodes > 1) ldsm_wait_barrier(1);

    if(ldsmnodeid == 0) {
        verify_result();
    }
    ldsm_free(matA);
    ldsm_free(matB);
    ldsm_free(matC);
}
```

그림 6 LIGHT 매트릭스 곱셈 Pseudocode

그림 6은 LIGHT 사용자라이브러리를 이용한 매트릭스 곱셈 프로그래밍 예제이다. 이 프로그램 예제를 보면 단일 컴퓨터 매트릭스 프로그램과 유사함을 느낄 수 있다. 차이점은 초기화를 위해서 ldsm_init()을 호출하고, 동적 메모리와 관련하여 ldsm_malloc()과 ldsm_free() 함수를 이용한다. 또한, 프로그램에서 연산의 시작과 끝 부분에 Barrier를 사용하여 연산의 순차성을 보장하고 각 노드를 구별하기 위한 ldsmnodeid란 전역 변수를 사용하고 있다. 그러나, 이러한 정도의 차이는 사용자가 직관적으로 이용할 수 있으며, 무엇보다도 메시지 전달과 같은 통신에 대해 고려하지 않고 프로그램 할 수 있다는 측면에서 장점을 지닌다 그림 6의 pseudocode는 가장 우수한 DSM으로 평가받고 있는 TreadMarks[20]의 코드와 유사하다 DSM은 분산 네트웍 환경에서 가상적인 공유 메모리 프로그래밍 환경을 만듬으로써 사용자에게 프로그래밍의 편이성을 제공한다. 사용자가 느끼기로는 조금의 편이성과 조금의 성능향상이라고 해도 그것을 위해서 지금까지 많은 DSM[21]이 내부 구조적으로 대규모로 변화되며 개발되어져왔다 다음 장에서는 LIGHT의 성능평가를 다룬다.

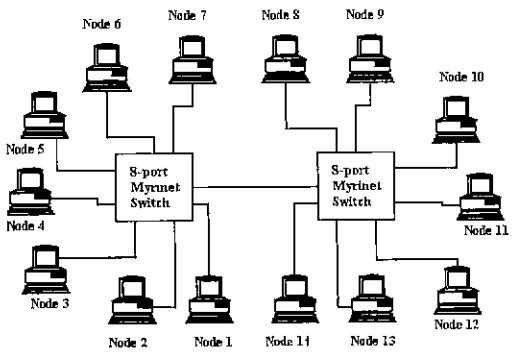


그림 7 Myrinet 클러스터 환경

4. 성능 평가

본 연구에서 구현된 LIGHT의 테스트 환경은 그림 7과 같다. 시스템은 14대의 컴퓨터가 1.28 Gbps의 Myrinet과 100 Mbps의 Ethernet으로 각각 연결되어져있다. Myrinet 보드는 32-bit/33-MHz PCI 보드이며 LANai 4.1 버전이다. 각 컴퓨터는 펜티엄 III-500MHz이고 메모리는 각각 64M이다. BIP는 0.95d 버전이며 현재 시스템에서 1.005 Gbps의 대역폭과 6.7 us의 단방향 latency를 보인다. 운영체제는 Linux 2.0.35/RedHat 5.0이다 테스트한 응용프로그램은 그림 6과 같은 매트릭스 곱셈이며 gcc를 이용하여 ldsmlib.a와 BIP 라이브러리를 -O -Wall 옵션으로 컴파일 및 링크하여 수행한다. 성능 평가 지표는 Speedup(단일 머신에서의 수행시간/LIGHT에서의 수행시간)이며 2 노드, 4 노드 그리고 8 노드에서 수행했다.

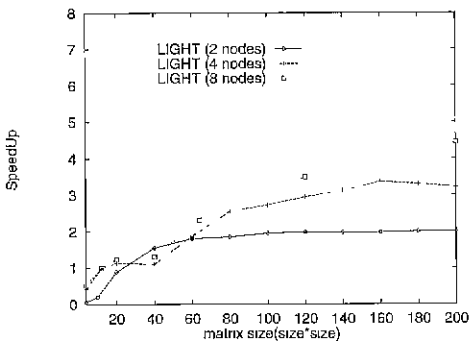


그림 8 2-,4-,8-노드에서의 Speedup

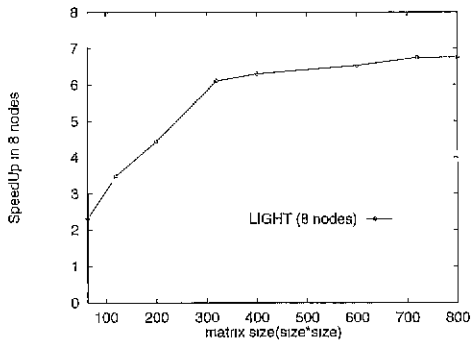


그림 9 8-노드에서의 Speedup

그림 8과 9는 LIGHT의 Speedup 결과를 보여준다. 2 노드에서는 200 x 200의 매트릭스 사이즈에서 거의 2에 가까운 이상적인 Speedup을 보여준다. 그러나, 8 노드로 가면서 페이지 결함 요구가 Server 노드에 집중적으로 많아지면서 네트웍 부하가 증가하게 되므로 성능 저하가 현저히 일어난다. 그림 8에서 800x800 매트릭스 사이즈에서 LIGHT는 6.7의 Speedup을 보이며 실제 시간은 28초가 소요된다. 그러나, 똑같은 연산을 단일 컴퓨터에서 수행시에는 2분 30초의 시간이 소요되므로 멀티 네트웍과 BIP/Myrinet과 같은 고성능 통신 라이브러리를 이용하는 것이 효율적임을 보여준다. 또한, 앞선 그림 6과 같이 프로그래밍 환경도 사용자에게 직관적인 편리함이 있다.

5. 결 론

본 논문은 기가 비트급 지역망인 Myrinet을 기반으로 구현되어진 기존의 클러스터 환경에서 실제로 사용자가 이용할 수 있는 프로그래밍 환경을 소개하였다. 각 환경을 크게 4가지로 나누어 특징과 장단점을 설명했으며 프로그래밍의 편이성 측면에서 장점을 갖는 DSM 개발이 미비한 점을 지적했다. 그리고 그런 면에 착안하여 본 연구에서 개발한 DSM인 LIGHT(LIGHT-weight DSM)를 소개했다.

LIGHT는 크게 2가지 측면에서 그 의미를 지닌다. 첫째, LIGHT는 Myrinet과 Ethernet의 멀티 네트웍 환경을 최대한 이용하는 하나의 모델을 제시했으며, 성능평가에서의 실험 결과는 이러한 고성능 멀티 네트웍을 그대로 반영하며 2 노드의 경우에는 거의 2에 가까운 Speedup을 보이고 8 노드에서도 6.7의 Speedup을 보인다. 둘째, 기존의 10 Mbps 네트웍을 중심으로 만들어진 DSM의 주요 개념들이 기가비트급 네트웍 모델에서는 어떻게 동작하는지를 분석할 수 있는 발판으로 이용될 수 있다. 그러나, 앞으로 중앙집중식 서버 모델을 스위치 기반의 분산 모델로 확장하고 SPLASH[22]와 같은 벤치마크를 통해 성능 평가를 해봄으로써 성능 보완이 이루어져야 한다.

참고문헌

[1] N.J. Boden, D. Cohen, R.E. Felderman,

A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su, "Myrinet: A Gigabit-per-Second Local Area Network," IEEE Micro, pages 29-35, Feb 1995.

[2] Myricom Homepage, http://www.myri.com.

[3] D.E. Culler, A.A. Dusseau, R. Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong, "Parallel Computing on the Berkeley NOW," Joint Symp. Parallel Processing, 1997.

[4] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," Supercomputing '95, San Diego, California.

[5] H. Tezuka, A. Hori, and Y. Ishikawa, "Design and Implementation of PM : A Communication Library for Workstation Cluster," JSPP, 1996.

[6] L. Prylli and B. Tourancheau, "BIP : A New Protocol Designed for High Performance Networking on Myrinet," Workshop PC-NOW, IPPS/SPDP, 1998.

[7] K.G. Yocum, J.S. Chase, A.J Gallatin, and A.R. Lebeck, "Cut-through Delivery in Trapeze : An Exercise in Low-Latency Messaging," IEEE Symp. High-Performance Distributed Computing (HPDC), 1997.

[8] A. Chien, S. Pakin, M. Lauria, B. Buchanan, K. Hane, L. Giannini, and J. Prusakova, "High Performance Virtual Machines(HPVM): Clusters with Supercomputing APIs and Performance," SIAM Conf. Parallel Processing for Scientific Computing, 1997.

[9] Trapeze/Myrinet TCP/IP Performance http://www.cs.duke.edu/ari/trapeze/ip/

[10] T. von Eiken and W. Vogels, "Evolution of the Virtual Interface Architecture," IEEE Computer, Nov. 1998.

[11] P. Buonadonna, A. Geweke, and D.E. Culler, "An Implementation and Analysis of the Virtual Interface Architecture," Supercomputing, 1998.

[12] RWCP Laboratory, http://pdswww.rwcp.or.jp/

[13] LHPC Project, http://lhpca.univ-lyon1.fr/

[14] MPICH Homepage, http://www-unix.mcs.anl.gov/mpi/mpich/

[15] http://www.ncsa.uiuc.edu/General/CC/ntcluster/VIA/MPI-FM-VIA/fm-via.htm

[16] Li, K., and Hudak, P., "Memory Coherence in Shared Virtual Memory Systems," ACM Trans. on Computer Systems, Vol. 7, pp. 321-359, Nov. 1989.

[17] H. Harda, H. Tezuka, and A Hori, S. Sumimoto, T. Takahashi, and Y. Ishikawa, "SCASH : Software DSM using High performance network on commodity hardware and software". In 8th Workshop on Scalable Shared-memory Multiprocessors, pages 26-27.

[18] L. Lefevre and O. Reymann, " Combining low-latency communication protocols with multithreading for high performance DSM systems on clusters," EUROMICRO PDP 2000

[19] G.M. Voelker, E.J. Anderson, T. Kimbrel, M.J. Feeley, J.S. Chase, A.R. Karlin, and H.M. Levy, "Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System," ACM SIGMETRICS on Performance, Measurement, and Evaluation, 1998.

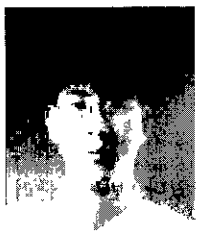
[20] C. Amza, A.L. Cox, S Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks : Shared Memory Computing on Networks of Workstations," IEEE Computer, Feb. 1996

[21] DSM Home page at http://www.cs.umd.edu/users/keleher/dsm.html

[22] Singh, J., Weber, W., and Gupta,A.

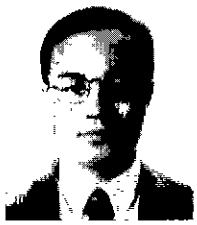
"SPLASH: Stanford Parallel Application for Shared-memory," Technical Report CSL-TR-91-469. Stanford University. Palo Alto, CA, April 1991.

남 경 완



1994 중앙대학교 전자공학과 학사
1996 포항공과대학교 전자전기공학과 석사
1996 ~ 포항공과대학교 전자전기공학과 박사과정
관심분야:부하균등화법, 분산공유메모리
E-mail nky@postech.ac.kr

이 승 구



1985 University of Kansas 전기공학 학사
1987 Universty of Michigan 전자전기 및 전자계산학 석사
1990 University of Michigan 전자전기 및 전자계산학 박사
1990~1991 University of Delaware 전기공학과 조교수
1997~1998 IBM T.J. Watson 연구소 방문 연구원
1991 ~ 포항공과대학교 전자전기공학과 부교수
관심분야 병렬 및 결함포용 컴퓨팅, 분산 클러스터 시스템
E-mail slee@postech.ac.kr

2000년 정례회의 및 주요행사 연간일정표

월 별	이 사 회		편집위원회	총회 · 학술발표회		송년회
	상 임	정 례	학회지	임시 · 춘계	정기 · 추계	
1월	14일(금) 16:00		21일(금) 16:30			
2월	14일(월) 16:00	18일(금) 16:00	18일(금) 16:30			
3월	10일(금) 16:00		17일(금) 16:30			
4월	7일(금) 16:00	21일(금) 16:00	21일(금) 16:30	28(금)~29(토) 대구효성가톨릭대		
5월	12일(금) 16:00		19일(금) 16:30			
6월	9일(금) 16:00	23일(금) 16:00	16일(금) 16:30			
7월	7일(금) 16:00		21일(금) 16:30			
8월	4일(금) 16:00	25일(금) 16:00	18일(금) 16:30			
9월	8일(금) 16:00		15일(금) 16:30			
10월	6일(금) 16:00	20일(금) 16:00	20일(금) 16:30		27(금)~28(토) 숙명여대	
11월	3일(금) 16:00		17일(금) 16:30			
12월	1일(금) 16:00	15일(금) 16:00	15일(금) 16:30			12일(화) 18:00

※ 회의일정은 사정에 따라 변경될 수 있음