特輯

고성능 병렬처리 컴퓨터 KAICUBE-860

朴 圭 晧
韓國科學技術院 電氣 및 電子工學科

본 연구 개발에서는 계속적으로 그 필요가 증가하는 대규모 계산을 위한 슈퍼급의 성능을 가지는 병렬처리 시스템의 구현, 구축을 목표로 하고 있다.우리는 슈퍼급의 컴퓨터 시스템을 개발하기 위해 최신의 i860 프로세서를 사용한 단위컴퓨터를 설계,제작하여 최대 40 MFLOPS의 성능을 가지는 단위 컴퓨터를 구현하였고, 8개의 단위컴퓨터를 연결하여 320 MFLOPS 컴퓨터를 개발하였다. 최종 목표는 128개의 단위 컴퓨터를 연결하여 5 GFLOPS 최대 계산능력을 갖는 7차원 하이퍼큐브 컴퓨터 시스템, KAICUBE-860을 구성하고자 한다. 대부분의 사람들은 그 생각방식이나 행동형태가 순차적인 구조를 가지고 있기 때문에 병렬적인 프로그램의 개발이 쉽지가 않고 때문에 사용자로 하여금 보다 쉽게 병렬 시스템에 접근할 수 있도록 사용자 환경을 적절히 구축하여야만 한다. 따라서 우리의 최종 목표는 슈퍼급 성능의 병렬 처리 시스템이고 그 사용 방법은 보통순차적 컴퓨터를 사용하듯이 간단한 사용자 환경을 제공하는 시스템이다.이를 위해 시스템 소프트웨어와 사용자 환경을 계속 개발 중에 있다.

Ⅰ. 서 론

반도체 산업의 괄목할만한 발전에 힘입어, 많은 마이크로프로세서들이 개발되어 산업의각 분야에 널리 활용되고 있다. 1970년대 초에 4 bit의 마이크로프로세서가 출현하여 현재 32 bit 이상의 고성능의 수퍼마이크로컴퓨터로 발전하여, 종래의 미니 컴퓨터의성능을 능가하며, 탁상위에 놓을 수 있을 정도로 소형경량화가 이룩되었다.이러한 마이크로프로세서 기술은 다종의 마이크로프로세서를 이용한 멀티프로세싱(multiprocessing) 의 구현이 가능하게 발전하였다. 하이퍼큐브 컴퓨터(Hypercube computer)에 대한 구조에 대해 제안된 이래 최초의 구현은 미국 California Institute of Technology에서 1983년부터 시작되었다. [1] 최초의 Cosmic cube라고 불리는 하이퍼큐브 컴퓨터는 대형의 수치 계산 문제에서 좋은 성능을 보였다.이러한 성공으로 즉시 4개의 회사가 하이퍼큐브 컴퓨터, Intel사의 iPSC, Ametek사의 System/14, Floating Point System사의 T series,Ncube사의 Ncube/ten을 출시하게 되었다. [2] 이후 현재까지 매년 하이퍼큐브 컴퓨터에 대한 Conference가 개최되고, 향상된 하이퍼큐브 컴퓨터가 출시되고 있다. 본 연구실에서는 1987년도 부터 하이퍼큐브 컴퓨터의 개발에 착수하여 KAICUBE-I, KAICUBE-II를 완성하고 현재 KAICUBE-860을 개발하고 있다. 3개의 컴퓨터에 대한 비교는 표1 과 같다.

표 1. KAICUBE 시스템의 제원

시 스 템		KAICUBE-Ⅰ	KAICUBE-Ⅱ	KAICUBE-860/8node
완성시기		1989년	1990년	1993년
성 능		16MIPS 4MFLOPS	64MIPS 64MFLOPS	320MFLOPS
단위컴퓨터수		8	32	8
단위컴퓨터 프로세서		MC68020	MC68020	i860
단위컴퓨터 성능	계산 성능	2MIPS 0.5MFLOPS	2MIPS 2MFLOPS	40MIPS 40MFLOPS
	통신성능	10Mbps	128Mbps	80Mbps
단위컴퓨터당 메모리		1Mbytes (up to 4M)	1Mbytes (up to 4M)	8Mbytes (up to 64M)
단위 컴퓨터당 통신 채널수		4	6	8

Ⅱ. 하이퍼큐브 컴퓨터에 대한 소개

하이퍼큐브 컴퓨터의 구조, 특징 그리고 컴퓨터 시스템으로서의 장점은 다음과 같다.

1. 구조 형태

Binary n-cube라고도 불리우는 하이퍼큐브는 하이퍼큐브 차원(dimension)이라는 파라미터에 의하여 특정 지워진다.[3] 이 파라미터가 하이퍼큐브의 노드 컴퓨터갯수와 각 노드 컴퓨터의 통신 링크(communication link)수를 결정한다. 예를 들자면 그림 1 에서 보듯이 n+1 dimension 하이퍼큐브는 n dimension 하이퍼큐브 두개를 구성하여 첫번째 하이퍼큐브의 각 노드에서 두번째 하이퍼큐브에 대응하는 노드에 링크(link)를 연결하여서 구성할 수 있다. n-dimemension하이퍼큐브의 각 노드는 n-binary digit ($a_{n-1},\ldots,a_0$)로 나타낼 수 있으며, 이들은 Hamming distance가 1인 node 사이에만 링크가 존재하게 된다. 예를들면 2차원 하이퍼큐브의 노드들 (00)와 (10)는 링크가있으나 노드들 (10)와 (01)에서 링크가 없다.

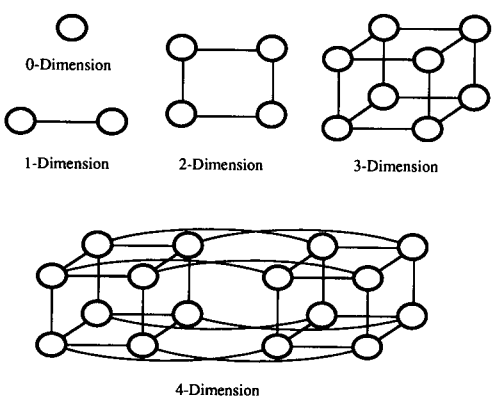

그림 1. 여러차원의 하이퍼큐브 구조

2. 하이퍼큐브 구조의 특성

하이퍼큐브 컴퓨터 구조의 성질은 대단히 다양하며 여러 연구에서 그 구조적 특성이 조명되었다. 여기에서는 간략히 이를 알아 보겠다. 하이퍼큐브 컴퓨터의 각 연산 단위는 노드(node)라고 불리는데 수십 - 수백의 노드가 서로 연결된 구조의 성질은 다음과 같다. n-dimensional 하이퍼큐브 시스템의 노드의 갯수는 $N=2^n$ 으로 표시 된다. 그러므로, 각 노드들은 총 n 개의 이진수(binary digits)로써 표현이 가능하고, 이 표현은 직접 각 노드를 지칭하는 번지가 된다.

- n-dimensional 하이퍼큐브의 각 노드는 n개의 이웃하는 노드들과 직접 연결되어 있다. 그러므로 각 노드당 통신 링크의 수는 n+1개(Host와 의연결을 고려) 이다.
- n-dimensional 하이퍼큐브의 가장 멀리 떨어진 두 노드간의 통신 거리는 n이다. 즉, 어떤 노드가 다른 한 노드와 서로 통신을 할때 최대로 거쳐야 할 중간 노드의 갯수는 n-1 이 된다.
- n-dimensional 하이퍼큐브의 각 노드들간의 평균 거리(즉, 두 노드 사이의평균 노드 갯수)는 n/2가 된다. 또한 전체 시스템의 총 통신 용량(Communication Bandwidth)는 $N/2 \log_2 N$이다.
- 이웃하는 두 노드 사이의 번지는 이진 자리수의 어느 한 bit만이 다르고 나머지 bit는 모두 같다. 그러므로, 어느 한 노드에서 다른 특정 노드로의 메세지 전송은 두 노드간의 번지의 bit가 다른 통신 링크를 통하여 점차적으로 옮겨가 최종 목적 노드에 이르게 된다. 이 경우 두 노드 사이의 최소거리(minimum distance)는 각 번지에서 서로 다른 bit의 수와 같다.
- 하이퍼큐브의 두 노드 사이에는 총 d!(여기에서 d는 두 노드간의 거리를 의미) 개의 병렬 path가 존재한다. 그러므로 5-dimensional하이퍼큐브의 최대 거리에 존재하는 두 노드간에는 총 120경로의 서로 다른 path가 존재한다.
- mesh, tree, ring등의 일반적인 multi컴퓨터 구조들은 하이퍼큐브 구조내에서 쉽게 구현되며 (embedded in 하이퍼큐브), 이에 대한 연구는 아직까지 도전 세계적으로 활발히 진행되고 있어, 가장 일반적인 연결구조로써의 하이퍼큐브 구조를 더욱 효율적으로 만들고 있다.

하이퍼큐브 각 노드는 통신을 위한 공유 메모리를 갖고 있지 않는 message-passing 구조이다. 그러므로 많은 노드를 통신 링크를 통하여 용이하게 연결할 수 있어 확장성이 좋다. 이러한 특성들은 다음의 소절에서 자세히 알아 보겠다.

3. 멀티컴퓨터로써의 하이퍼큐브의 특성

하이퍼큐브 Topology의 컴퓨터 시스템으로써의 여

러 특성은 다음과 같다.

- 많은 수치연산들의 연산들을 분할한 전체 구조가 하이퍼큐브 topology에 가장 잘 mapping 됨이 알려져 있다.
- 하이퍼큐브 Topology는 여러 가지 연결 topology중에서 가장 강력한 연결성을 가지며, 이러한 연결성으로 인해 ring, mesh, tree등의 일반적인 연결구조는 모두 하이퍼큐브 구조로써 쉽게 구현할 수 있다.
- 각 노드간의 최대 거리 및 평균 거리, 통신 용량의 관점에서 볼때 하이퍼큐브구조는 가장 유리한 특성을 나타낸다.
- 하이퍼큐브 구조하에서의 메세지 통신 algorithm은 아주 간단할 뿐만 아니라 여러 다양한 경로를 통한 통신이 가능하여, 통신 능력의 관점에서도 하이퍼큐브 구조는 아주 효율적이다.
- 일정한 확장 방법을 가진 하이퍼큐브는 하드웨어의 제작뿐 아니라 전체시스템의 확장에 있어서도 쉽고 간편한 특성을 가진다.

이상에서 하이퍼큐브 구조의 특징을 그래프 이론적 측면 뿐아니라,실제 컴퓨터의 관점에서도 알아 보았다.

Ⅲ. KAICUBE-860의 H/W

본 실험실에서 1993년도를 예정으로 설계 제작중인 하이퍼큐브 시스템은 하이퍼큐브 형태로 연결된 128개의 노드 컴퓨터(Node Computer)와 전체 시스템을 관장하고 사용자와 하이퍼큐브 컴퓨터사이를 연결해주는 호스트 컴퓨터 (Host computer)로 구성되어 있다. 각 노드컴퓨터는 40Mhz i860 Microprocessor를 사용하여 최대 40MIPS와 80MFLOPS의 연산능력을 갖는다. 병렬처리와 I/O를 위한 통신을 위해 각 노드 컴퓨터와 호스트 컴퓨터는 양방향 통신(full-duplex)방식의 16-bit의 데이타 라인을 통신 채널을 갖추었다. 7-차원의 구조를 가지기 때문에 각 단위 컴퓨터는 7 개의채널을 가지며 노드 0의 단위 컴퓨터는 호스트 컴퓨터와의 통신을 위하여 통신용 SRAM을 부가하였다. 각 단위 컴퓨터는 두 개의 Intel 82380 DMA를 가지고 send와 receive의 통신을 각각 수행하게 한다.하나의 82380은 총 8개의 채널을 제공할 수 있다. 그림 2는 전체 하이퍼큐브 시스템의 블록도(Block Diagram)을 나타내고 있다.

현재 KAICUBE-860에서 구현된 노드 컴퓨터는 하이퍼큐브 컴퓨터 시스템의 최소 구성 단위가 되는 컴퓨터로서 노드 컴퓨터의 성능이 전체 하이퍼큐브 컴퓨터 시스템의 성능과 직결되므로 이에 대한 설계는 매우 중요하다. 하이퍼큐브 시스템의 핵심이 되는 부분은 강력한 연산 능력과 고속의 통신 능력을 갖춘 노드 컴퓨터이다. 또한 각 노드 컴퓨터는 연산 모듈(Computation Module)과 통신 모듈(Communication Module)로 구성된다. 128개의 단위 컴퓨터의 최대 성능은 40 Mhz에서 동작 시켜 최대 5 GFLOPS의 성능을 목표로 하고있다. 그리고 100Mbps 이상의 통신 속도를 구현하여 연산 모듈간의 효율적이고도 빠른 통신이 될 수 있도록 하고자 한다. 고성능 마이크로프로세서와 주변장치 칩을 사용하여 연산 속도와 통신 속도의 성능을 극대화 할 수 있는 설계를 하였다. 현재 구현된 단위컴퓨터는 단위컴퓨터 운영 커널등의 부담을 고려하여 40 MFLOPS의 성능을 보이고 있으며 통신 속도는 80 Mbps이다. 그림3에서는 노드 컴퓨터의 블록 다이어그램을 나타냈다.노드 컴퓨터는 다음과 같이 구성되어 있다.

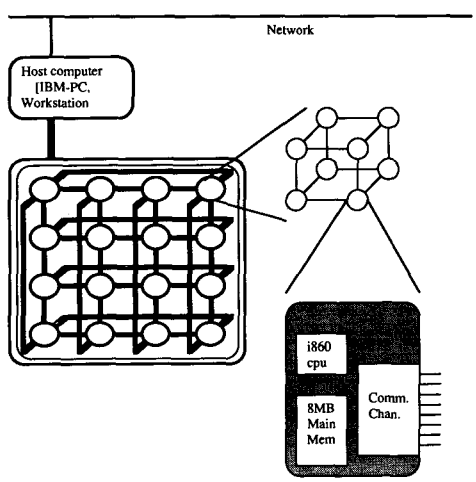

그림 2. KAICUBE-860의 전체 구조

- 64-bit i860 마이크로프로세서 (30 MHz)
- DRAM Main Memory (8MB, 64-bit)
- 2개의 82380 Integrated System Peripheral (ISP)

- Bootstrap용 EPROM (64KB, 8-bit)
- SRAM 통신용 버퍼 메모리 (128KB, 32-bit)
- 8개의 통신 채널
- 82530 Serial Communication Controller (SCC)
- 시스템 제어와 상태 점검을 위한 Control/Status Ports

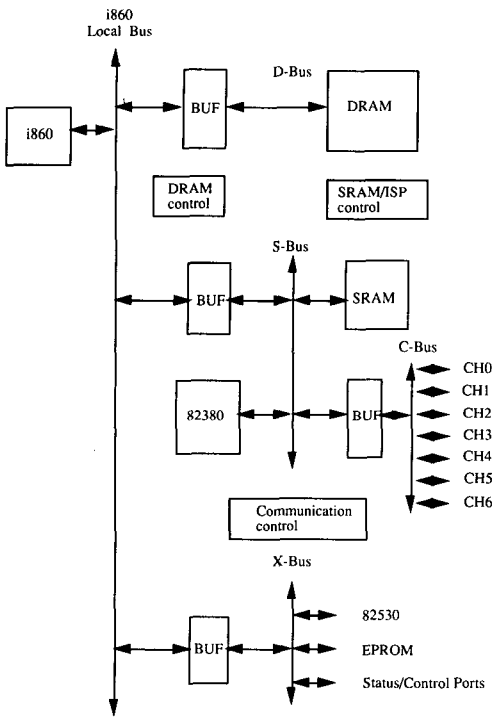

그림 3. 노드 컴퓨터의 블럭 다이어그램

Ⅳ. KAICUBE-860의 S/W

KAICUBE-860의 system S/W와 Application S/W는 기존에개발된 KAICUBE-II용 S/W를 KAICUBE-860의 바뀌어진 H/W에맞게 변형하여 사용할 수 있다. 그래서 이 절에서는 KAICUBE-II의 전체적인S/W와 현재 개발 중에 있는 병렬 디버거, 병렬 트랜스래이터에 대해 설명한다.

1. 개요

병렬처리 시스템을 사용하기 위하여서는 기본적으로 시스템 운용소프트웨어와 프로그램 개발 환경 등 크게두 종류의 소프트웨어가 필요하게 된다.

1) 시스템 운용 소프트웨어

시스템 운용 소프트웨어는 기본적으로 시스템을 구동시키기 위한 소프트웨어이다. 이를 위해서는 병렬 처리 시스템의 각 하드웨어를 초기화하고 시스템의 오류를 검정하며 이상 유무에 대한 정보를 사용자에게 제공하고 기본적인 기능들을 보다 상위의 구조에 제공하는 모니터(monitor)프로그램, 또 컴퓨터의 자원 (메모리, 입출력, 통신, 계산 능력)을 관리하고 이들이 효율적으로 사용되어질 수 있도록 여러 가지의 가능들을 제공하는 작은 규모의 운용 체제인 노드 커널(node kernel) [4] 이 필요하며 병렬 처리에서 필수적인 통신을 효율적으로 할 수 있게하는 통신 커널(communication kernel) [5] 이 존재하여야 한다.

병렬 처리 시스템 내부에는 많은 수의 노드 프로세서들이 존재하며 사용자는 이 중 전부 혹은 일 부분을 할당받아 개발한 프로그램을 수행시키게 된다. 만약 시스템이 임의의 시간에 하나의 사용자에게 만 주어진다면 그 시스템의 사용률은 낮아지게 될 것이다. 그러므로 시스템의 사용률을 높이기 위해 다중 사용자 환경이 제공되어야 한다. [6] [7] 이 다중 사용자 환경은 호스트 컴퓨터에서 동작하며 현재 일이 수행되어지고 있는 노드 프로세서들과 일이 할당되지 않은 프로세서들에 관한 정보를 관리하며 병렬 처리 컴퓨터 사용의 요청이 들어올 때마다 현재의 노드 프로세서 사용에 관한 정보를 이용하여 적절한 부분의 프로세서들을 할당하여 주는 역할을 담당하며 반대로 어떠한 일의 수행이 끝날 경우 그 일이 수행되어진 프로세서의 정보를 수집하여 다음 일의 수행에 다시 사용되어지게 하는 일도 수행한다.

다중 사용자 환경에서의 프로그램 개발자는 자신의 프로그램이 병렬컴퓨터 중 어느 부분의 프로세서에서 수행되어 질지를 알지 못하게 된다. 그러므로 프로그램의 개발자는 자신의 프로그램이 언제나 임의의 특정 프로세서 부분에서 시작한다는 가정에서 프로그램을 개발하며 이 프로그램의 수행시 자동적으로 사용자가 정의한 프로세서의 이름을 실지 할당된 프로세서의 이름으로 변환시키는 방법이 존재하여야 한다. 이 부분은 노드 프로세서의 커널에서 담당하게 한다.

2) 프로그램 개발 환경

이미 언급한 바와 같이 병렬 프로그램을 작성하려

면 시스템의 병렬성을 잘 사용할 수 있는 언어가 필요하다. 이같은 프로그램은 각 태스크 간의통신이 필수적이므로 병렬 언어는 통신을 위한 primitive들을 포함하여야 한다. 이를 위해 기존의 C 언어를 병렬 프로그래밍용으로 확장하고 이를 위한 컴파일러가 존재하여야 한다.

이러한 언어로 짜여진 프로그램이 수행되어지면 통신 등의 부분은 노드 커널 내부에 존재하는 기능을 사용하게 되며 이를 위해 사용자프로그램과 시스템 소프트웨어 간에 기능 수행 요청 및 수행 결과를 돌려주는 방법이 존재하여야 하며 이는 노드 커널과 컴파일러의 정해진 규약에 의해 결정된다.

```
/*
 * Host program of simple communication testing:
 * host -> node0 -> node1 -> ... -> node(n-1) -> host.
 */

#include <stdio.h>
#include <hcube/host.h>

/* some message types */
#define T_START 10
#define T_INPUT 100
#define T_DATA 110
#define T_DATA 200

int snode, supid;
int stype;

main()
{
 int i;
 int data;
 int pid = getpid();
 int dim = getdim();
 int nodes = nodesofdim(dim);

 /* first synchronization */
 for (i=0; i<nodes; i++)
   hcsend(i,1,T_START,&pid,sizeof(pid),0);
 for (i=0; i<nodes; i++)
   hcrecv(&snode,&supid,&stype,&data,sizeof(data),0);

 /* sent input data to node0*/
 hcsend(0,1 T_INPUT,&data,sizeof(data),0);

 /* receive result data from node(n-1)*/
 hcrecv(&snode,&supid,&stype,&data,sizeof(data),0);

 printf("\nResult data = %d\n", data);
}
```

그림 4. Host 프로그램

일반적으로 병렬처리 컴퓨터에서 특정 문제를 풀기 위해서는 두개의 응용 프로그램을 개발 해야한다.그 중의 하나는 호스트 컴퓨터에서 운용되는 호스트 프로그램이고, 다른 하나는 노드 컴퓨터에서 운용되는 노드 프로그램이다.

```
/*
 * Node program of simple communication testing:
 * host -> node0 -> node1 -> ... -> node(n-1) -> host.
 */

#include <stdio.h>
#include <hcube/node.h>

/* some message types */
#define T_START 10
#define T_INPUT 100
#define T_DATA 110
#define T_DATA 200

#define MYUID 1

main()
{
 int data;
 int nid = getvnid();
 int nodes = nodesofdim(getdim());
 int snode, supid, hostpid;

 /* first synchronization */
 recvw(&snode,&hostpid,T_START,&data,sizeof(int));
 sendw(-1,hostpid,T_START,&pid,sizeof(int));

 if (nid == 0) { /* node 0 */
   recvw(&snode,&supid,T_INPUT,&data,sizeof(data));
   data++;
   sendw(nid+1,MYUID,T_DATA,&data,sizeof(data));
 } else if (nid < (nodes - 1)) {
   recvw(&snode,&supid,T_DATA,&data,sizeof(data));
   data++;
   sendw(nid+1,MYUID,T_DATA,&data,sizeof(data));
 } else { /* last node */
   recvw(&snode,&supid,T_ DATA,&data,sizeof(data));
   data++;
   sendw(HOSTID,hostpid,T_RESULT,&data,sizeof(data));
 }
}
```

그림 5. Node 프로그램

그림 4과 그림 5에 예제를 보여준다. 이 예에서 호스트 프로그램은 data값을 노드0 컴퓨터에 준다음노드(n-1)에게서 data를 받아 출력한다. 노드 프로그램은노드 i.d.에 따라 data를 다음 노드 또는 호스트 컴퓨터로 전달한다. 호스트 프로그램은 프로그램의 외부 세계와의 상호 작용을 통제하여 병렬 프로그램의 전체 행동을 조정한다. 프로그래밍의 이러한 방식은 메세지 전달 방식의 병렬 컴퓨터에 자연스러운 방식이다. 왜냐하면 병렬 컴퓨터는 직접적인 I/O 방식이 제공되지 않고 다만 통제용 컴퓨터를 통해서만 간접적으로 I/O가 가능하기 때문이다. 이러한 방식에서 응용 프로그램은 직접적인 I/O를 제공받을 때보다 프로그램의 길이가 길어 질 수 있고 또한 에러도 발생하기 쉽다. 또한, 그런 프로그램의 유지, 보수,디버깅이 어렵다. 그러므로 호스트 프로그램/노드 프로그램 방식에 대한 하나의 제안으로 사용자가 노드 프

로그램만을 작성하며 호스트와의 입출력은 직렬형 방식과 유사한 방식을 취할 수 있도록 하는 방법이 개발되었으며 이를 통해 사용자는 보다 쉽게 병렬 프로그램을 작성할 수 있게 된다. 이것은 CUBIX [10] 을 통해서 구현하였다.

2. 병렬 디버거

병렬 시스템은 그 프로그램의 복잡함과 설계에 있어서 새로이 도전해야할 문제들을 던지고, 특히 디버깅을 매우 복잡하게 만든다. 병렬성(Parallelism) 표현에 있어서 추가된 어려움 때문에 병렬 프로그램의 디버깅은 순차 프로그램에 비해서 어렵다. 본 하이퍼큐브 컴퓨터에서 실행되는 응용 프로그램은 동시에 다른 노드에서 수행되는 많은 수의 프로세스들로 구성된다.각 프로세스는 메세지를 주고 받도록 확장된 순차 프로그램의 경우로, 이 병렬 프로그램의 디버깅은 전통적인 순차 디버깅의 모든 것을 포함한다. 그것은 또한 병렬 어플리케이션의 분산된 본연의 특성, 즉 한 어플리케이션에 많은 수의 프로세스들이 존재하고 프로세스 간에 정보를 교환하고 동기를 맞추기 위해 메세지의 사용 등으로 부터 해결 해야할 일들을 포함한다. 전형적인 병렬 어플리케이션은 4에서 32개의 프로세스들이 수백 혹은 수만 번의 메세지를 교환하면서 동시에 실행된다. 이러한 어플리케이션을 적당하지 않은 도구로써 디버깅하는 것은 무척 어렵다.

따라서 KAICUBE-II 프로그래머가 병렬 프로그램을 디버깅하는 것을 도와주는 소스-레벨의 병렬디버거가 설계 구현되었다. [8] [9] 구현된 병렬디버거(kdb)는 순차 디버거를 많은 수의 프로세스들이 동시에 존재하는 상황을 처리할 수 있도록 확장하여 설계되었다. 이 kdb은 C 언어로 쓰여진 KAICUBE-II 응용 프로그램을 높은 레벨의 심볼 표현을 사용하여 디버깅할 수 있게 해준다. 또한 이 병렬 디버거는 발전된 소스-레벨의 순차 디버거들이 제공하는 주요한 기능들 뿐만 아니라, 병렬 프로세스들의 분산된 본성으로 부터 해결해야 할 기능들을 제공한다.

1) 병렬 디버거의 사용 환경

본 병렬 디버거는 앞에서 이야기한 다중사용자 환경 아래에서 호스트 컴퓨터에서 실행된다. 디버깅할 어플리케이션은 호스트와 노드를 위한 프로그램을 각각 따로 있고, 더 나아가 한 프로그램은 한 노드에 한 프로세스 만을 생성한다고 가정한다. 이러한 환경에서 kdb은 다음과 같은 명령으로써 동작을 시작한다.

% kdb -h 호스트-프로그램 -n 노드-프로그램 -d demension

〉

(여기서 %는 UNIX의 프롬프트이고, 〉는 kdb의 프롬프트이다.)

먼저 디버거는 dimension의 subcube을 할당하고 호스트-프로그램와 노드-프로그램을 load하여 실행시킬 준비를 마친다. 그런 다음 명령 대기 상태를 나타내는 프롬프트를 출력하여,다음에 나열된 여러 명령들을 사용하여 원하는 디버깅을 할 수 있게 해준다.

2) Context

많은 수의 프로세스들이 동시에 존재하는 상황에서, 프로그래머가 관심의 촛점을 그 어플리케이션의 어떤 부분으로 집중시키는 것이 요구된다. 또한 빠르고 쉽게 그 촛점을 다른 부분으로 이동 시키는 것이 절실히 필요하다. 이를 위해 본 디버거는 context의 개념을 사용한다. 한 context은 디버깅 명령의 대상이 되는 프로세스들의 집합을 나타낸다. current context은 현재 촛점이 맞추어진 프로세스들을 나타내는데, 이 프로세스들은 프롬프트로 표시되며,명령 context을 사용하여 이 current context을 바꿀 수 있다.예를들어 명령,

〉context (0,1,2)

(0,1,2) 〉

은 current context에 아무것도 없는 상태에서 노드 0,1,2 프로세스가 현재의 current context로 바뀜을 보여준다.

(nodes)〉contex (host, odds)

(host,odds)〉context (all)

(all)〉

3) Breakpoints

Breakpoints은 디버거의 가장 기본적인 기능으로, 본 디버거는 동시에 존재하는 많은 수의 프로세스들의 문장(statements) 혹은 행(line), 함수(function) 혹은 라벨(label) 등에 breakpoints을 설정할 수 있게한다. 예를 들어 두 개의 기본 명령,

break (0,1,2) sub1( )

break (nodes) #100은 각각, 노드 0,1,2 에서 실행되고 있는 프로세스들의 함수 sub1에,모든 노드에서 실행되는 프로세스들의 100 번째 행(현재 하일의)

에 breakpoints을 설정한다.

4) 분산된 데이타의 검색

여러 노드들에 분산된 프로세스의 변수의 값을 print 명령을 사용하여 조사해볼 수 있다. 예를들어 명령,

print (0,1) A [1]

은 노드 0와 1에서 실행되고 있는 프로세스의 배열 변수 A의 1 번째 원소의 값을 보여준다. 이 때, 다른 모든 명령에서도 마찬가지로, 변수의 지시는 symbolic reference, 즉 프로그램에 선언된 이름을 그대로 사용한다.종종 변수의 값을 임의로 바꿀 수 있는 기능이 요구되는데, 명령 assign이 이를 가능케 한다.

assign (0,1) var 100

이 명령은 노드 0와 1에서 실행되고 있는 프로세스의 변수 var에 100을 할당시킨다.

5) 메시지 큐의 검사

실제 많은 에러들이 메세지를 잘못 사용하여 발생한다(예를 들에, 메세지를 잘못된 곳으로 보내지거나, 잘못된 타입으로 인식하거나, 보내는 중에 data가 변하거나, 제대로 보냈어도 영원히 받는곳이 없는 등등).따라서 어떠한 메세지들이 어떤 노드에 도착하여 있으며, 어떠한 프로세스들이 메세지를 기다리고 있는 지 조사할 수 있는 기능이 요구된다. 명령 mesgq은 각 프로세스에 도착하여 가져가기를 기다리는 메세지들을 출력한다. 이때 출력되는 정보는 메세지를 보낸 프로세스,메세지의 type와 크기(size) 등이다.

(0,1)〉 mesgq

For From Type Size

0 host 10 4

0 1 2 100

1 2 3 100

2 empty

다른 노드로부터 메세지를 기다리는 프로세스들은 메세지가 올 때까지 실행을 멈추고 대기 큐(receive queue)에서 기다린다. 명령 recvq은 각 노드에서, 이 큐(queue)에 있는 프로세스들을 출력한다.

recvq (nodes)

Node Message type

(1) 101

(2) 102

위는 노드 1과 2의 프로세스가 각각 101와 102의 타입(type)의 메세지를 기다리고 있는 것을 나타낸다.이 두 통신 상태 검사 명령은 사용자로 하여금 전체 노드들 간의 통신상태를 알 수 있게 해준다.

6) 그 외의 기능들

앞에서 기술한 것들 이외에도, 노드와 호스트 프로세스를 load, start 그리고 reset 시키거나, 프로세스들의 실행을 run, stop, step, kill 그리고 continue 시키고, 프로세스들의 현재 상태와 멈춘 위치를 보여주고, 소스 코드(code)를 조사하고, UNIX 명령을 실행시키고, 디버거 명령들의 사용법을 보여주는 on-line help,command alias 등등의 기능들이 제공된다.

3. 병렬 언어 변역기- KAPPA

메세지 전달 방식의 많은 상업적 병렬 컴퓨터들은 단일 프로그램 다중 데이타(single program multiple data, SPMD) 형태의 프로그램 기술방식을 택하고 있다.[11] 이 방식은 여러 단위 컴퓨터가 같은 프로그램을 수행하면서 각 컴퓨터의 노드ID에 따라 자신의 로칼 메모리에 있는 데이타에 대해 프로그램의 다른 부분이 수행을 하는 형태이다. 현재 KAICUBE 860에서도 병렬 프로그램 설계의 용의성으로 이방식을 택하고 있다. 그러나 이러한 방법은 비정규적 형태의 문제에 경우각 단위 컴퓨터간의 부하균등을 이루는 것이 쉽지않다. SPMD 형태의 프로그램 개발은 크게 두가지 방식으로 나눌 수 있다. 한가지 방식은이러한 문제가 너무 복잡해서 프로그래머가 모두 해야한다는 것이고 다른 하나는 순차적 프로그램을 재구성 컴파일러로 병렬성을 추출하여 병렬 프로그램으로 변환할 수 있게 해야한다는 방식이다. 전자의 방식에서는 아주 전문적인 프로그래머가 아닌 이상 프로그램의 개발이 그리 간단하지 않기 때문에 많은 연구가 병렬 프랜스레이터 또는 컴파일러에 대해 진행되고 있다. 실제로 공유메모리 방식의 병렬 컴퓨터에서는 for loop 분할과 같은 자동 병렬 변환에 대하여 많은 업적이 이루어져 실제로 사용되고 있는 상황이다. 그러나 분산메모리 시스템의 경우에는 통신에 의한 과부담으로 인해 course-grain형태의 트랜스레이터가 필요하다.

우리는 병렬 프로그램을 개발하는데 있어서 두가지 방식의 절충적 입장을 취해 문제가 주어졌을 때 기본적인 병렬 알고리즘은 사람이 생각해내고 이 병렬 알고리즘에 대해 병렬 시스템이 수행할 수 있도록 문제를 분할하고 스케줄하는 일은 자동화하도록 병렬 트

랜스레이터를 구성한다. 즉, 설계자가 적절한 알고리즘을 설계하고 분할을 수행한 뒤에 몇개의 프로시져로 구성된 프로그램을 작성한다. 이 프로그램은 순차적 프로그램 형태이기 때문에 디버깅을 위해서 순차적 머신에서 수행을 시킬 수도 있다. 이러한 프로그램은 병렬 트랜스레이터의 병렬 코드 합성과 최적화를 통해 메세지 전달 방식인 KAICUBE 860에서 수행될 수 있는 병렬프로그램으로 바뀐다.

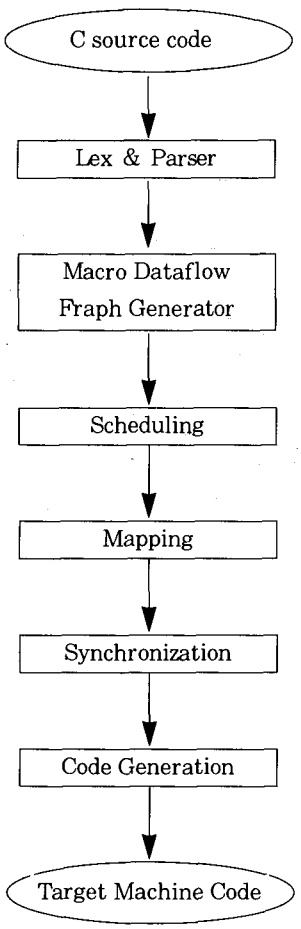

그림 6. 병렬 트랜스래이터의 구성

그림 6은 병렬 트랜스레이터의 구성을 보여준다. lexer와 parser는 주어진 프로그램으로 부터 프러시저 단계의 데아타 연관성(data dependencies)을 찾아내고 이를 바탕으로 graph generator는 스케줄과 매핑(mapping)을 위한 directed graph을 만들어 낸다. 여기서 각 노드는각 프러시져를 나타내고 그 값은 프러시저가 수행될 때의 수행시간을 가지게 된다. 그리고 각 에지는 각 프로시저간의 데이타 연관성을 의미하여 그 값은 두프러시저간의 통신량이 된다. 그다음 scheduling은 각각의 노드를 수행속도가 최소로 될 수 있도록 각 프로세스에 할당하는 일을 한다. 매핑은 할당된 프로세스를 통신시간을 최소화할 수 있게 각 단위 컴퓨터에 할당하는 일을 한다. 그 뒤에 synchronization은 올바른 수행을 위해 통신 프리미티브(primitive)을 삽입하는 일을하고 최종적으로 code generator가 병렬 프로그램을 만들어낸다.

Ⅴ. 응용프로그램 수행 및 성능

개발된 8 node KAICUBE-860의 유용성을 보기 위하여 여러가지 응용 프로그램을 수행하였다.

1. 병렬언어변역기- KAPPA

병렬트랜스레이터 KAPPA의 성능을 측정하기 위하여 두가지 프로그램을 수행하였다. 그 첫번째는 250 X 250 행렬의 곱셈으로서 KAICUBE-860 을 잘 이해하는 프로그래머가 만든 프로그램과 병렬언어 변역기 KAPPA를 통해서 생성한 KAICUBE-860 프로그램을 각각 수행하였고 이를 또한 Sparc II 에서 수행 시간을 비교하였다. 두번째로 3차원 물체의 시간에 따른 열확산 문제를 각각 위의 경우와 같이 KAICUBE-860 의 전문가가 만든 프로그램, KAPPA를 통한 수행, 및 Sparc II 에서의 수행에 대하여 비교하여 표2에 수행시간을 비교하였다.

표 2. KAPPA의 성능 측정

수행대상	250X250 행렬곱	열분포해석
KAICUBE 전문가	셈	1.19초
KAPPA에 의한 프로그램	2.22초	3.52초
Sparc-Ⅱ	2.22초	10.32초

표2에서 프로그램 크기가 작은 행렬 곱셈의 경우는 전문가가 작성한 프로그램이나 KAPPA를 통하여 생성한 것이다. 같은 수행시간을 보였으나 프로그램 크기가 크고 복잡한 두번째의 경우는 KAPPA에 의한

것이 2배 이상의 시간 부담이 있음을 보이고 있다. 즉 사용자가 KAICUBE-860 에 대한 구체적인 지식 없이 사용할 수 있는 대신 효율은 그만큼 떨어진다.

2. 신경회로망학습

그 응용범위가 넓은 신경회로망중에서 Kohonen 의 Self Organizing Map의 학습을 수행하였다. 그 크기는 400 X 400 뉴턴을 학습시키는 것으로서 Sparc-II, Cray 2S, 및 KAICUBE-860 에서 각각 시간을 측정하였다. 이때 vector 슈퍼컴퓨터인 Cray 2S 의 경우는 Autovectorization에 의하여 프로그램을 수행하였다. KAICUBE-860은 전문가가 프로그램하여 수행하였으므로 Cray-2S 의 경우도 전문가가 프로그램하면 그 수행 시간이 다를 것으로 보인다.

표 3. 신경회로망 학습

수행대상	400X400 뉴런학습
Sparc Ⅱ	265분
Cray 2S	141분
KAICUBE 860	16분

3. 3차원 물체 열분포 해석

공학 분야에서 많이 쓰이는 Finite Difference Method(FDM) 에 의한 3차원 물체 해석을 하여 보았다. 이 경우도 역시 Cray 2S 는 Autovectorization에 의하여 수행하였다.

표 4. 3차원 물체 열 분포 해석

수행대상	3차원 물체
Sparc-Ⅱ	861초
Cray 2S	318초
KAICUBE-860	154초

4. 행렬곱셈

행렬곱셈은 백터화에 적합한 것으로서 다음과 같은 성능을 보였다.

수행대상	200X200 행렬곱셈	400X400 행렬곱셈
Sparc-Ⅱ	16초	133초
Cray 2S	0.5초	4초
KAICUBE-860	5초	21초

Ⅵ. 결론

본 연구 개발에서는 계속적으로 그 필요가 증가하는 대규모 계산을 위한 슈퍼급의 성능을 가지는 병렬처리 시스템의 구현, 구축을 목표로 하고 있다. 현재 단일 CPU를 사용하는 슈퍼급의 컴퓨터 시스템의 개발은 그 기술적 어려움으로 인하여 전 세계적으로 몇 개 안되는 나라에서만 개발이 가능한 데에 비해 다수의상용 CPU를 사용하는 다중 프로세서 시스템은 효율적인 통신 모듈만 개발한다면 쉽게 구성할 수가 있기 때문에 현재 우리나라의 기술 수준하에서는 가장 바람직한 연구 방향이라 할 수 있다. 이러한 시스템은 세계적으로 많은 곳에서 연구가되고 있고 실제로 상업화까지 되어 있다. 다중 컴퓨터의 여러 구조중에서 하이퍼큐브 구조는 많은 연구를 통해서 그 특성상 많은 장점을 가지고 있음이 입증되었다. 주요 장점중의 하나는 하이퍼큐브 구조는 다른 형태의 구조, 즉 트리(tree) 형태, 메시(mesh) 형태, 원(ring) 형태 등을 포함할 수 있기 때문에 다른 형태의 구조에 적합한 응용 프로그램이 쉽게 하이퍼큐브 시스템에서 수행될 수가 있다. 우리는 슈퍼급의 컴퓨터 시스템을 개발하기 위해 i860 프로세서를 사용한 단위컴퓨터를 설계,제작하여 최대 40 MFLOPS의 성능을 가지는 단위 컴퓨터를 개발하였고, 단위 컴퓨터를 효율적인 통신 모듈을 이용하여 현재 8개의 단위 컴퓨터를 연결하여 320 MFLOPS 최대계산 능력을 갖는 하이퍼큐브 컴퓨터 시스템, KAICUBE-860을 구현하였다. 향후 128개의 단위컴퓨터를 연결하여 5 GFLOPS 성능의 컴퓨터 구현을 목표로 한다. 병렬 시스템에서는가장 중요한 성능 저하의 원인이 통신 부분에 있기 때문에 이에 대한 설계 및 구현이 하드웨어 부분에서 가장 중요한 부분이라 할 수 있다. 우리는 이를 위하여DMA를 사용하여 병렬 통신으로 구현 하였다. 현재 통신 방법으로 router 제작을 진행중이다. 이것이 완성되면 좀더 효율적인 통신이 이루어 질것이다. 또한 이와 관련된 시스템소프트웨어와 사용자 환경을 구현하였다. 그리고 몇 가지의 응용 프로그램을 설계된 사용자 환경을 사용하여 수행 시켜서 이들의 유용성을 확인하였다.

이러한 연구의 결과로서 다중 컴퓨터 형태의 병렬처리 컴퓨터 시스템, 특히 하이퍼큐브형 컴퓨터를 효율적으로 수행되도록 하는 시스템 소프트웨어와 사용

자가 용이하게 이 컴퓨터를 사용할 수 있게하는 사용자 환경에 대한 개발의 기본 능력을 갖추었다고 보며 개발된 결과를 통하여 병렬처리 컴퓨터의 효용성을 증대시킬 수 있었다고 본다. 이러한 하이퍼큐브 컴퓨터를 위한 시스템 소프트웨어와 사용자 환경의 개발은 임의의 다중 컴퓨터를 위한 시스템 소프트웨어 개발에 대한 중요한 기반 기술이 될 것이다.이를 바탕으로 시스템 소프트웨어와 사용자 환경을 개선, 보완하여, 효율적인 병렬 처리 컴퓨터 시스템의 운용 기술을 확보하고 사용자를 위한 더 용이한 프로그램 환경을 제공할 필요가 있다고 본다. 특히 병렬 처리의 overhead가 되는 task allocation, load balancing 및 parallel language 등의 연구가 가속화 되어야 한다. 대부분의 사람들은 그생각방식이나 행동형태가 순차적인 구조를 가지고 있기 때문에 병렬적인 프로그램의 개발이 쉽지가 않고 때문에 사용자로 하여금 보다 쉽게 병렬 시스템에 접근할 수 있도록 사용자 환경을 적절히 구축하여야만 한다. 따라서 우리의최종 목표는 슈퍼급 성능의 병렬 처리 시스템이어야 하고 그 사용 방법은 보통순차적 컴퓨터를 사용하듯이 간단한 사용자 환경을 제공하는 시스템이다.

參 考 文 獻

[1] C.L.Seitz,"The Cosmic Cube," *Comm. ACM*, vol.28,pp.22-23,1985.

[2] Daniel A.Reed and Richard M. Fujimoto, *Multicomputer Networks*,The MIT Press,1987.

[3] Y.Saad and M.H.Schultz,"Topological Properties of Hypercube,"*IEEE Trans. Comput.*, Vol.37,pp.867-872,1988.

[4] 이승섭, *Implementation of kernel for Parallel processing in hypercube computer systems*, M.S.Thesis, E.E. Dept.,KAIST,1989.

[5] 김종욱, *Implementationof communication module fora hypercube multicomputer*, M.S.Thesis, E.E.Dept., KAIST,1990.

[6] 서경룡, *A Processor Allocation Strategy for the Hypercube Computer*, M.S. Thesis, E.E.Dept.,KAIST,1990.

[7] 이준용, *Implementation of Multiuser Environment on theHypercube Computer System*, M.S.Thesis, E.E.Dept., KAIST, 1991

[8] 심규현, *Implementation of Node debugger on theHypercube Computer SystemM.S.Thesis*, E.E.Dept.,KAIST, 1991.

[9] 이형석, *Implementation of Parallel Debugger on Hypercube Computer System", M.S.Thesis*, E.E.Dept., KAIST,1991.

[10] G.Fox,M.Johnson and et al., *Solving Problems on the Concurrent Processors*, Prentice-Hall,1988.

[11] M.Y. Wu and D.D.Gajski,"Hypertool: A Programming Aid for Message-PassingSystems,"*IEEE Tr. on Parallel and Distributed Systems*, vol. 1, no. 3,,pp.330-343, July 1990.

筆 者 紹 介

朴 圭 晧
1950年 10月 19日生
1973年 2月 서울대학교 공과대학 전자공학과 졸업(학사)
1975年 8月 한국과학원 전기 및 전자공학과 졸업(석사)
1983年 3月 프랑스 파리 대학 졸업 (박사)

1975年 8月 ~ 1978年 8月 동양정밀 개발실
1983年 3月 ~ 현재 한국과학기술원 전기 및 전자공학과 교수

주관심분야 : 병렬처리 컴퓨터, 컴퓨터 비젼