# 분산공유 메모리 시스템 상에서의 효율적인 자료분산 방법

민 옥 기†


## 요 약

자료 분산은 SPMD(Single Program Multiple Data) 형태의 병렬성을 제공하는 HPF(High Performance Fortran)의 주기능으로 구현 방법에 따라 컴파일러 성능을 좌우한다. 본 논문에서는 SPAX(Scalable Parallel Architecture computer based on X-bar network) 상에 자료 분산 기능을 제공하기 위한 설계 주안점과 효율적인 모델에 관하여 기술하였다. SPAX는 분산공유 메모리(DSM : distributed shared memory)를 사용한 계층적 클러스터링 구조를 가진다. 이러한 메모리 구조에서는 분산 메모리 자료 분산(DMDD : Distributed Memory Data Distribution)이나 공유 메모리 자료 분산(SMDD : Shared Memory Data Distribution) 방법으로는 시스템 가용성을 만족할 수 없다. 그래서 계층적 마스터-슬레브 형태의 분산공유 메모리 자료 분산(DSMDD : Distributed Shared Memory Data Distribution) 모델을 설계하였다. 이 모델은 각 노드에 원격 마스터와 슬레브들을 할당하고 노드내에서는 공유 메모리를 그리고 노드간에는 메세지 전달 인터페이스를 사용한다. 시뮬레이션을 수행한 결과, 시스템 성능 저하를 최소화하는 노드 크기로 DSMDD를 수행하였을 때 SMDD나 DMDD보다 훨씬 더 효율적이였다. 특히, 논리적 프로세서 갯수가 많을수록, 분산된 자료들 간의 자료 종속성이 적을수록 성능이 우수하였다.


# An Efficient Data Distribution Method on a Distributed Shared Memory Machine

Ok Gee Min †


## ABSTRACT

Data distribution of SPMD(Single Program Multiple Data) pattern is one of main features of HPF(High Performance Fortran). This paper describes design issues for such data distribution and its efficient execution model on TICOM IV computer, named SPAX(Scalable Parallel Architecture computer based on X-bar network). SPAX has a hierarchical clustering structure that uses distributed shared memory(DSM). In such memory structure, it cannot make a full system utilization to apply unanimously either SMDD(Shared Memory Data Distribution) or DMDD(Distributed Memory Data Distribution). Here we propose another data distribution model, called DSMDD(Distributed Shared Memory Data Distribution), a data distribution model based on hierarchical masters-slaves scheme. In this model, a remote master and slaves are designated in each node, shared address scheme is used within a node and message passing schemebetween nodes. In our simulation, assuming a node


※본 연구는 1995년부터 1998년까지 정보통신부와 과학기술처에서 시행한 "고속병렬컴퓨터 공동 개발 사업"의 일부분으로 수행하였다.

† 정 회 원 : 한국전자통신연구소 연구원

논문접수 : 1996년 5월 11일, 심사완료 : 1996년 8월 1일

size in which system performance degradation is minimized, DSMDD is more effective than SMDD and DMDD. Especially, the larger number of logical processors and the less data dependency between distributed data, the better performance is obtained.

## 1. 서 론

좀 더 빠른 컴퓨터를 만들기 위하여 지난 20여년간 병렬 컴퓨터에 대한 연구가 계속되어 왔다. 이와 더불어 병렬 프로그래밍 환경을 위한 연구가 컴퓨터 과학의 중요한 한 부분을 차지하게 되었다. 각 업체들은 자신들의 독자적인 구조에 적합한 병렬 프로그래밍 환경을 구축하였는데, 이로 인하여 사용자들은 각 시스템마다 새로운 프로그래밍 환경을 공부해야 하는 부담을 갖게 되었다. 이러한 문제를 해결하기 위하여 언어 수준의 표준화가 대두되었고, HPF[1, 7]는 이러한 움직임으로 인하여 발생된 최초의 표준화된 병렬 프로그래밍 언어이다. HPF는 Fortran 90에 병렬성을 제공하는 기능을 추가한 것으로 이미 많은 업체들이 구현하여 상품화하고 있거나 개발 계획을 가지고 있다.

SPAX[6] 개발 사업에서도 시스템 성능을 가시화할 수 있는 병렬 프로그래밍 환경을 개발하고 있는데, HPF 컴파일러 개발이 그 일부분이다. SPAX 상에서 HPF 컴파일러의 최대 관심사는 자료 분산 기능을 시스템 자원의 가용성을 높일 수 있도록 최적화하여 구현하는 것이다. SPAX는 크로스바로 연결되는 클러스터 기반 병렬 컴퓨터로써 각 클러스터는 4개의 프로세싱 노드와 4개의 입출력 노드 또는 통신 접속 노드로 구성되며, 각 프로세싱 노드는 4개의 인텔 펜티엄 마이크로 프로세서를 사용하는 대칭형 다중프로세싱 (SMP:Symmetric Multiprocessing) 구조로 버스를 이용한 공유 메모리를 사용한다.

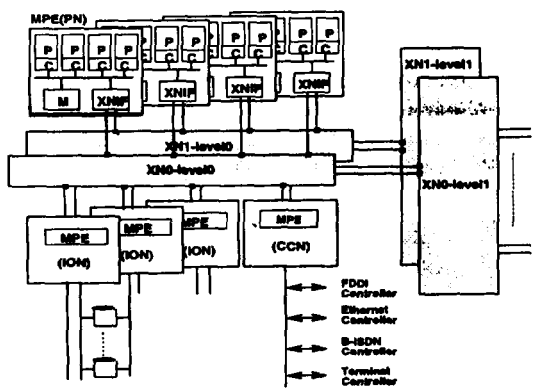
만일 SPAX에서 SMDD 모델이나 DMDD 모델을 사용하면 시스템 자원을 충분히 활용할 수 없게 된다. 왜냐하면 SMDD 모델은 SMP 형태의 한 노드밖에 사용할 수 없게 되고, DMDD 모델은 한 노드에 하나의 논리적 프로세서 만을 할당할 수 밖에 없기 때문이다. 본 논문에서 제시한 자료 분산 모델은 SPAX 구조를 고려하여 공유 메모리와 메세지 전달 인터페이스를 같이 사용하는 계층적 마스터-슬래브 형태의 분산공유 메모리 자료 분산(DSMDD) 모델이다. 이 모델은 SPAX와 같은 형태의 DSM 구조를 갖는 시스템에서 시스템 자원의 가용성을 높이는 효율적인 방법으로 쉽게 적용될 수 있다.

본 논문에서는 HPF 컴파일러 구현이 시스템 구조에 종속적이므로 먼저 SPAX 시스템 자체에 대하여 설명하였다. 그런 후에 3장에서는 HPF 컴파일러에 대하여 전반적인 구성과 SPAX 상에서 구현하는 데 대한 주안점을 설명하고, 4장에서는 DSMDD 모델과 이 모델의 프로세서들과 노드 상에 태스크 할당 방법, 그리고 분산된 자료들 간에 통신 정의 방법에 대하여 논하였다. 그리고 5장에서는 시뮬레이션을 통하여 모델들의 성능을 비교하여 그 결과를 기술하였다. 마지막으로 6장에서 이 논문을 결론지었다.

## 2. SPAX

### 2.1 SPAX의 구조

(그림 1)에서 보는 바와 같이, SPAX는 크로스바로 연결되는 클러스터 기반 병렬 컴퓨터이다. 클러스터는 4개의 프로세싱 노드를 포함하고 있으며 각 노드는 대칭형 다중프로세싱(SMP:symmetric multiprocessing) 구조로 4개의 인텔 펜티움 프로세서를 포함한다. SPAX는 국제 표준을 만족하는 개방형 시스템




(그림 1) SPAX의 구조
(Fig. 1) Architecture of SPAX

구조와 대용량 병렬 컴퓨터 구조로 되어 있으며, 클러스터 단위로 자유롭게 첨가 또는 제거가 가능하다.

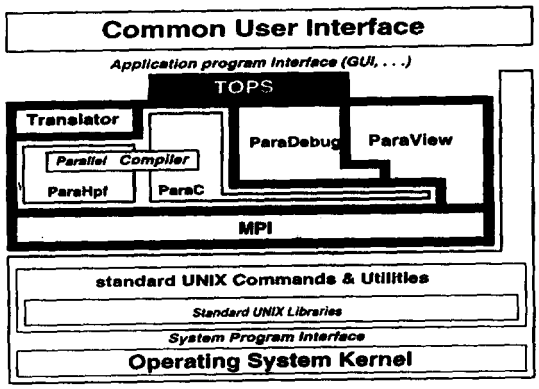
SPAX는 최대 16개의 클러스터까지 연결할 수 있으며, 이는 256개의 프로세서와 64개의 입출력 노드(ION:Input/Output Node)와 64개의 통신접속 노드(CCN:Communication Connection Node)로 구성된다. 그래서 최대 20GIPS의 자료 처리 능력과 10,000 tpmC 트랜잭션 처리 능력을 보유한다. 시스템의 최소 구성은 1개의 프로세싱 노드와 1개의 ION 그리고 1개의 CCN으로 구성된다. 클러스터나 노드간 통신은 크로스바를 사용하여 메세지 전달 인터페이스를 사용한다. 노드내에서는 버스를 이용한 공유 메모리를 사용하여 통신을 수행한다.

SPAX의 운영체제(MISIX)는 마이크로 커널 기반의 단일 시스템 이미지를 가지며, Unixware 2.0과 Chorus 마이크로 커널을 기반으로 한다. 운영 체제 커널은 마이크로 커널을 기반으로 여러 서버들로 구성되는 구조이다. 각 노드들에 운영체제 커널의 복사본을 따로 가지고 노드 내의 자원들을 관리한다. 다른 노드들에 대해서는 분산공유 메모리 정책에 의하여 자원을 공유한다. 그리고 다른 노드의 자원을 사용할 수 있는 단일 시스템 이미지를 제공하여 편리한 병렬 프로그래밍 환경을 만들어 준다.

### 2.2 프로그래밍 환경

TOPS(TOols for Parallel System)는 SPAX를 위하여 제공되는 병렬 프로그래밍 환경 시스템이다. TOPS는 (그림 2)와 같이 병렬 컴파일러, 병렬 디버거, 병렬 프로그램 성능 모니터를 중심으로 구성되어 있다. 또한, 운영체제 위에 메세지 전달 인터페이스(MPI), 쓰레드 라이브러리를 설치하여 프로그래밍 환경으로 제공된다.

ParaC(Parallel C)는 ANSI C 상에 병렬성을 제공하는 예약어를 확장하여 설계하고 구현하였다. Parahpf (Parallel HPF)는 HPF 표준을 기반으로 SPAX에 효율적인 형태로 제공되는 병렬 Fortran 컴파일러이다. ParaDebug(Parallel Debugger)는 병렬 프로그램의 비결정성을 재실행 기법을 이용하여 결정적 수행 환경으로 변경시켜 주므로써 디버깅을 가능하게 설계하였다. ParaView(Parallel Program Performance Monitor)는 ParaC로 작성된 쓰레드 기반 병렬 프로그램 및




(그림 2) 병렬 프로그래밍 환경
(Fig. 2) Parallel Programming Environment

MPI로 작성된 메세지 전달 병렬 프로그램의 성능 디버깅 및 오류 디버깅을 가시화기를 통하여 제공한다.

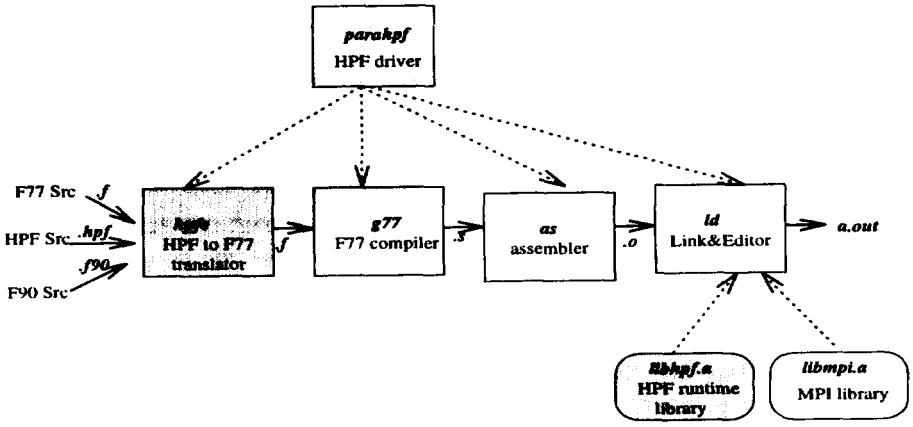
## 3. HPF 컴파일러

### 3.1 개 요

HPF 컴파일러는 (그림3)과 같은 구조로 구성되어 있다.

SPAX용 HPF 컴파일러[8, 9]는 PGI(Portland Group Inc.)의 HPF 컴파일러 PGHPF[3, 4]를 기반으로 한다. (그림 3)에 표시되어 있는 HPF 드라이버, HPF to F77 트랜스레이터 그리고HPF 실행시간 라이브러리가 PGI HPF 컴파일러 구성요소이다. 이들 대부분은 시스템 종속성이 없어 그대로 사용되며 실행시간 라이브러리를 제외하고는 약간의 튜닝 작업만이 있었다. g77은 Fortran 77 컴파일러로 별도로 존재하며, 기타 as, ld는 C에서 제공하는 도구를 사용한다. MPI [2] 라이브러리는 ANL(Argonne National Lab.)의 mpich를 기반으로 HPF 컴파일러의 실행시간 라이브러리와 함께 SPAX에 적합한 모델로 이식되었다.

병렬 Fortran 컴파일러 드라이버(parahpf)는 컴파일러 전 과정을 제어하고 각 단계의 입력과 출력을 정의한다. 드라이버는 우선 입력 화일의 확장자를 확인하여 입력 화일의 종류를 분별하고 사용자가 사용한 각 선택항목을 처리해주며 각 단계의 도구들을 구동시켜주고, 연결 편집 과정에서 연결해야 하는 각 라이브러리들을 정의하고 연결한다. HPF to F77 트



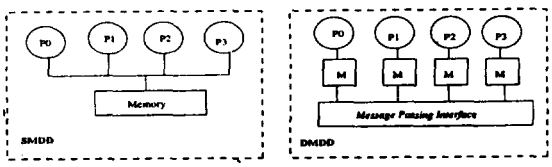

(그림 3) HPF 컴파일러 구조
(Fig. 3) Structure of HPF compiler

랜스레이터(hpfc)는 입력 화일의 확장자가 .hpf인 화일을 입력으로 여러 처리 과정을 거친 후, HPF 실행시간 라이브러리를 호출하는 Fortran 77 프로그램으로 변환하여 출력한다. F77 컴파일러(g77)는 hpfc에서 출력한 Fortran 77 화일을 입력으로 하여 어셈블리 코드를 출력한다. 실행시간 라이브러리(libhpf. a)는 자료 분산과 프로세서간 통신을 수행하는 프리미티브들을 포함한 많은 기능들을 수행하는 프리미티브들의 집합이다.

### 3.2 SPAX에서의 주안점

SPAX 상에서 HPF 컴파일러에 관한 최대 관심사는 최적의 성능을 얻을 수 있는 자료 분산 방법을 찾는 데 있다. 이미 앞에서도 말한 바와 같이 SPAX는 SMDD 모델이나 DMDD 모델 한가지 방법만으로는 일괄적으로 적용시킬 수 없다. (그림 4)의 SMDD를 SPAX 상에서 사용하면 공유 메모리를 사용하고 있는 하나의 SMP에서만 수행이 가능하다. 그러므로 여러개의 노드로 구성된 시스템을 한개의 노드밖에 사용하지 못하는 단점을 갖게된다.

또한, (그림 4)의 DMDD 모델을 SPAX 상에서 적용할 경우, 각 노드가 하나의 프로세서 역할을 하게 된다. 즉 크로스바 스위치로 메세지 전달 인터페이스를 수행하는 각 노드에 한개씩의 태스크만을 할당하게 된다. 이는 소수의 태스크만을 사용하는 단순한 프로그램을 수행하는 데에도 비용이 비싼 여러개의 노드를 사용해야 하므로 엄청난 시스템 성능 저하를 가져올 수 있다.



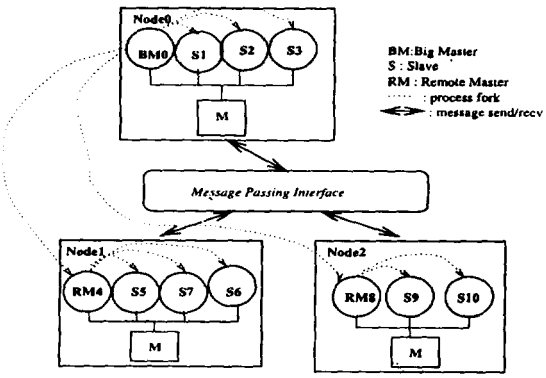

(그림 4) SMDD와 DMDD 모델
(Fig. 4) Model of SMDD and DMDD

이와같이 SMDD나 DMDD 모델은 SPAX 상에서 효율적으로 적용될 수 없으므로, 두 모델을 혼합하여 사용하여야만 한다. 두 방법을 모두 사용하여 SPAX의 자원 가용성을 높이고, 성능을 최적화시킨 것이 DSMDD 모델이다.

## 4. SPAX에서의 자료분산 방법

### 4.1 DSMDD 모델

SPAX 상에서의 자료 분산 처리 모델은 (그림 5)와




(그림 5) DSMDD 모델
(Fig. 5) Model of DSMDD

같은 DSMDD 모델이다.

DSMDD 모델은 여러개의 노드에 논리적 프로세서들의 집합을 분산시키고, 각각의 노드에 지역 마스터를 할당한다. 처음 프로그램을 구동한 프로세서가 프로그램 전체의 마스터가 되고 이를 다른 노드의 마스터와 구분하기 위하여 빅 마스터라 한다. 빅 마스터를 제외한 각 노드의 마스터를 원격 마스터라 한다. 모든 마스터들은 각 노드에 분산되어 있는 슬래브들을 관리하며 원격 마스터들은 빅 마스터의 통제를 받는다. 마스터와 슬래브들 모두는 순차적인 논리 식별자를 가지며, 같은 노드에 할당된 프로세서 식별자들은 연속적이다. 같은 노드에 할당된 모든 프로세서는 공유 메모리를 사용하며 공유 메모리를 통하여 프로세서간 통신(IPC : Inter Processor Communication)을 수행한다. 서로 다른 노드에 위치한 프로세서들 간에는 서로 메모리 영역이 달라지므로 메세지 전달 인터페이스를 사용하여 IPC를 수행한다. (그림 5)는 Node0와 Node1에 4개의 논리적 프로세서와 Node2에 3개를 할당한 예를 보여준다. 빅 마스터는 프로세서 0이며 원격 마스터는 4와 8이 된다. 점선은 프로세서 구동 관계를 나타낸다.

### 4.2 자료 분산

SPAX에서의 분산은 두가지 측면에서 볼 수 있다. 하나는 배열을 논리적 프로세서들의 집합에 분산시키는 것이고, 또 하나는 논리적 프로세서 집합을 각 노드에 적당한 크기로 할당하는 것이다. 클러스터에 분배하는 것은 노드에 분산하는 것과 동일하므로 클러스터에 할당하는 것은 고려하지 않았다.

사용자가 정의한 배열을 $A$라하고, 배열 첨자를 표현하는 인덱스 집합을 $I_A$ 그리고 프로세서의 집합을 $P$라 하자. 그러면 분산은 다음과 같은 함수로 표현될 수 있다[5].

$$d_A : I_A \rightarrow P$$

$d_A^{-1}(p)$는 한 프로세서 $p$에 대한 배열 $A$의 분할된 배열에 해당한다. 배열 $A$를 N개의 요소를 갖는 1차원 배열이라 하고, 프로세서 집합 $P$는 M개의 프로세서로 구성되어 있고, 그리고 배열 $A$가 $P$에 블록 분산(block distribution)을 한다고 가정하자. 그러면 배열 $A$의 분산 $d_A$은 다음과 같이 정의될 수 있다. 이 때, $I_A$는 배열 원소들의 집합으로 $i$는 배열 $A$의 한 원소이고, $d_A$는 배열 $A$의 각 원소들에 대한 자료 분산을 의미한다.

$$I_A = \{1, 2, ..., N\}\ P = \{1, 2, ..., M\}$$

$$d_A \rightarrow (i * M - 1)/N + 1$$

그래서 각 프로세서에 할당된 배열의 부분은 다음과 같이 계산된다. 프로세서 집합 $P$의 원소인 임의의 한 프로세서 $p$에 할당되는 자료 원소들의 집합들은 다음 $d_A^{-1}(p)$와 같이 표현될 수 있다.

$$d_A^{-1}(p) = \{(p-1) * N/M + 1, ..., p * N/M\}$$

각 프로세서들의 집합을 노드에 할당하는 것도 이와 유사하다. 노드들의 집합을 $\Phi$라 하고 프로세서들의 식별자를 표현하는 집합을 $J_P$라 하자. 그러면 프로세서들이 노드에 할당되는 것은 다음과 같이 표현될 수 있다.

$$s_P : J_P \rightarrow \Phi$$

M개의 프로세서가 L개의 노드에 할당되는 것은 다음과 같이 정의될 수 있다. 이 때, $s_P$는 임의의 프로세서 $j$가 노드들의 집합에 할당되는 것을 의미한다.

$J_P=\{1, 2,...,M\}\Phi=\{1,...,L\}$
$s_P \rightarrow (j*L-1)/M+1$

그리고 임의의 노드 $r$에서 수행되는 프로세서들의 집합은 다음 $s_\Phi^{-1}(r)$와 같이 얻을 수 있다.

$$s_\Phi^{-1}(r)=\{(r-1)*M/L+1,...,r*M/L\}$$

빅 마스터는 항상 시작하는 노드의 처음 프로세서가 되며, 각 노드의 원격 마스터는 ((r-1)*M/L+1)이 된다. 예를 들어, 255개의 요소를 가진 배열 A를 12개의 논리적 프로세서들의 집합에 블록 분산 시키고 이것을 3개의 노드에서 수행시킨다고 하면, 각 배열의 항목들은 다음과 같은 노드와 프로세서에서 수행될 것이다.

〈표 1〉 분산 예
〈Table 1〉 Distribution example

| 배열 범위 | 프로세서 | 노드 |
|---|---|---|
| (N=255) | (N=12) | (L=3) |
| A(1:22) | P(1) | $\Phi$(1) |
| A(23:44) | P(2) | |
| A(45:66) | P(3) | |
| A(67:88) | P(4) | |
| A(89:110) | P(5) | $\Phi$(2) |
| ..... | ..... | |
| A(155:176) | P(8) | |
| A(177:198) | P(9) | $\Phi$(3) |
| ..... | ..... | |
| A(255) | P(12) | |

이 때 빅 마스터는 노드 $\Phi$(1)에 있는 $P$(1)이고 노드 $\Phi$(2)의 원격 마스터는 $P$(5), 노드 $\Phi$(3)의 원격 마스터는 $P$(9)이다.

### 4.3 통 신

자료 분산 후 분산된 자료들간 간에 자료 종속성이 있을 경우 프로세서들 간의 통신이 필요하다. 가령, $A$와 $B$가 프로세서 집합인 $P$에 분산되어 있다고 하자.

$d: I_A \rightarrow P \quad d: I_B \rightarrow P$

여기서 배열 $A$를 계산하기 위해서 필요한 배열 $B$의 영역을 $S_B$라 하면, 이 때의 자료 종속성은 다음 함수 $m$으로 표시한다.

$m: S_B \rightarrow I_A$
where, $S_B \subseteq I_B$

특정 프로세서 $p$에서 배열 $A$를 수행함에 있어 프로세서 $q$로부터 배열 $B$의 일부를 전송 받아야 할 때, 프로세서 $p$와 프로세서 $q$의 통신은 다음과 같이 나타낼 수 있다.

$$C_{p \leftarrow q}=m_B^{-1}(d_A^{-1}(p)) \cap d_A^{-1}(q)$$

여기서 $C_{p \leftarrow q}$가 공집합이 아니면 두 프로세서 간에는 자료 종속성이 존재하며, 자료 송수신이 필요하다.

만일, 프로세서 $p$와 $q$가 속하는 노드 $r$과 $t$로 서로 다른 노드 영역에 있으면, 노드 $r$과 $t$사이에는 노드간 통신(INC: Inter-Node Communication)을 수행해야 한다. 프로세서 집합 $P$와 $Q$가 노드들의 집합 $\Phi$에 할당되었다고 하자.

$s_P: J_P \rightarrow \Phi \quad s_Q: J_Q \rightarrow \Phi$

$Q$의 부분집합인 $s_Q$와 프로세서 집합 $P$ 사이에 프로세서 종속성을 함수 n으로 나타내면,

$n: s_Q \rightarrow J_P$
where, $s_Q \subseteq J_P$

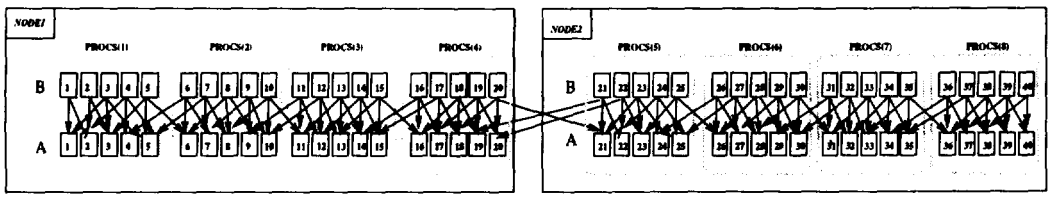
노드 r과 t 사이에 메세지를 전달해야 하는 프로세서들의 $INC$는 다음과 같이 나타낼 수 있다.

$$INC_{r \leftarrow t}=n_Q^{-1}(s_\Phi^{-1}(r)) \cap s_\Phi^{-1}(t)$$
where, $r \neq t$ for every pair $(r, s)$

예를 들어, 다음 프로그램1을 2개의 노드에서 수행했다고 가정하자.

프로그램 1:

```
1:          real A(40), B(40), S
2:!HPF$  template T(40)
```




(그림 6) 프로세서 및 노드들 간의 통신
(Fig. 6) Communication between processors and nodes

```
3:!HPF$  align (I) with T(I):: A, B
4:!HPF$  processors PROCS(8)
5:!HPF$  distribute T(BLOCK) onto PROCS
6:       forall (I=1:40:1)
7:          A(i)=I*I
8:          B(I)=A(I+2)+A(I)+A(I-1)
9:       end forall
```

프로그램1에 대한 자료 종속성은 다음과 같이 구할 수 있다.

$d_A^{-1}(1)=\{1, 2, 3, 4, 5\},...,d_A^{-1}(4)=\{16,...,20\}$,
$d_A^{-1}(5)=\{21,...,25\},...,d_A^{-1}(8)=\{36,...,40\}$
$m_A^{-1}(d_B^{-1}(1))=\{-1, 1,...,7\},...,m_A^{-1}(d_B^{-1}(4))=\{15,...,22\}$
$m_A^{-1}(d_B^{-1}(5))=\{20,...,27\},...,m_A^{-1}(d_B^{-1}(8))=\{35,...,42\}$
$\Phi(1)=\{1, 2, 3, 4\}\ \Phi(2)=\{5, 6, 7, 8\}$

각각의 IPC를 수행하는 배열 요소들 및 *INC*를 수행하는 프로세서 id들은 다음과 같다.

$C_{4\leftarrow5}=\{21, 22\}\quad C_{5\leftarrow4}=\{20\}$
$INC_{1\leftarrow2}=\{5\}\quad INC_{2\leftarrow1}=\{4\}$

배열 A와 배열 B 간의 자료 종속성으로 인한 IPC 및 INC는 (그림 6)과 같다.

## 5. 성능 평가

### 5.1 시뮬레이션

SPAX 시스템은 현재 개발 단계에 있으므로 실제 환경 하에서의 시험은 이루어지지 않았다. 단지, 자료 분산을 수행하는 부분에 영향을 줄 수 있는 요소들을 고려하여, 자료 분산 모델의 성능을 시뮬레이션하였다. 시뮬레이션은 참고문헌 [10]의 분석 결과를 토대로 성능에 영향을 주는 〈표 2〉와 같은 요소들의 범위를 설정하고 계산적으로 상대적인 성능을 측정하였다. 시뮬레이션은 2개의 펜티엄 프로세서를 가진 Compaq Proliant 4500 하드웨어에 Unixware 운영체제가 탑재된 환경 하에서 수행하였으며, GNU Plot를 이용하여 계산 결과를 도식화 하였다. 자료 분산은 1차원 배열을 각 논리적 프로세서에 블록 단위로 분산시키고 각 노드에 동일한 논리적 프로세서 갯수가 할당되었다고 가정하였다. 또한, HPF 자료 분산에서의 논리적인 프로세서는 시스템 상에서의 프로세스(process) 단위로 구성되므로 그 갯수를 제한하지 않았으며, 노드의 갯수는 시스템 특성에 맞추어 16개까지 확장할 수 있도록 하였다.

〈표 2〉 시뮬레이션 요소들
〈Table 2〉 Factors of simulation

| 요소 | 의미 |
|---|---|
| *task* | 논리적 프로세서의 갯수 |
| *A* | 배열 크기 |
| *sa* | 한 프로세서에 분할된 배열 크기 |
| *NN* | 노드의 갯수 |
| $t_{op}$ | 한 배열 요소에 대한 실행 시간 |
| $deg_p$ | 프로세서 당 성능 저하율 |
| $t_{com}$ | 통신 시간 |
| *rd* | sa에서의 종속성 비율 |
| $deg_n$ | 노드 당 성능 저하율 |

고려된 시뮬레이션 요소들을 요약하면 〈표 2〉와 같다. 성능 저하 요소는 정확히 측정할 수 없으나 다른 시뮬레이션을 통하여 노드를 증가시키는 것이 프로세서 관리 비용에 비하여 월등히 큰것으로 나타났다[10]. 본 시뮬레이션에서는 논리적 프로세서 70-100개가 증가하였을 때 성능이 2배로 감소하는 것을 기준으로 하였으며, 노드는 프로세서에 비하여 3배-70배 정도의 비용이 필요한 것으로 가정하였다. 이외에도 다른 사용자들에 의한 부하가 요소가 될 수 있으나 본 시뮬레이션 요소에서는 제외시켰다. 시뮬레이션 대상으로는 SMDD, DMDD, DSMDD 모델의 스피드업(speedup)을 비교하였다. 이 중 DSMDD 모델을 DSMDD_R과 DSMDD_O로 분류하였는데, DSMDD_R은 임의의 노드 갯수를 사용하였을 경우이며, DSMDD_O는 성능 저하를 최소화하는 노드 갯수를 사용한 것을 의미한다.

### 5.2 성능 측정

자료 분산 수행의 스피드업($\mu$)은 다음과 같이 나타낼 수 있다.

$$\mu = \frac{Tsingle}{Tparallel}$$

$T_{single}$은 배열 크기(A)와 한 배열 요소의 실행 시간 $t_{op}$의 곱으로 나타낼 수 있으며, $T_{parallel}$은 다음과 같이 나타낼 수 있다.

$$T_{parallel} = DEG_{PERF} * (OH_{OP} + OH_{COM})$$

$DEG_{PERF}$는 1개의 노드에서 1개의 태스크를 수행하는 것을 1로 하였을 때, 노드와 태스크의 증가에 따른 성능 저하를 나타내는 것으로 다음과 같다.

$$DEG_{PERF} = 1 + (task-1) * deg_p + (NN-1) * deg_n$$

$deg_p$는 $0.001 \le deg_p \le 0.015$ 범위에서 수행하였으며, $deg_n$은 $0.02 \le deg_n \le 0.07$ 범위에서 수행하였다. $OH_{OP}$는 한 태스크에 할당된 블록의 크기와 한 배열 요소에 대한 실행 시간을 곱한 것으로 나타낼 수 있다.

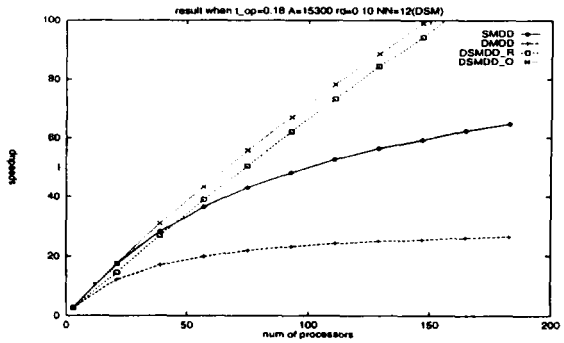
$$OH_{OP} = sa * t_{op}$$

$sa$는 배열 크기 A와 태스크 갯수에 종속적이며, $t_{op}$는 $0.05 \le t_{op} \le 0.80(\mu s)$ 범위에서 수행하였다. $OH_{COM}$은 태스크의 블록 크기와 종속성 비율 그리고 한 요소에 대한 통신 시간의 곱으로 나타내었다.

$$OH_{COM} = sa * rd * t_{com}$$

$rd$는 한 블록에서 비지역(non local) 자료의 비율로 $0.0 \le rd \le 0.3$의 범위로 하였고, $t_{com}$은 시스템의 노드간 통신 대역폭이 266 Mbytes/sec인 점을 고려하여 $0.12(\mu s)$로 정의하였으며, 노드 내에서의 태스크간 통신 시간은 무시하였다.

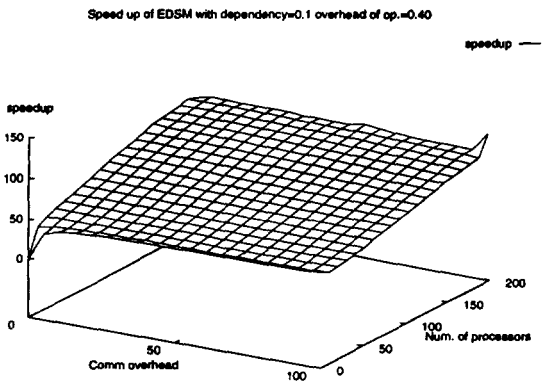
### 5.3 결 과




(그림 7) 시뮬레이션 결과
(Fig. 7) A result of simulation

(그림 7)은 프로세서에 대한 저하 요소를 0.01, 노드 수에 대한 저하 요소를 0.03으로 하고, 배열 크기를 15300, 실행 시간 0.18($\mu s$) 그리고 자료 종속성 비율을 0.01로 시뮬레이션했을 때, 각 모델의 스피드업을 보여준다. 그림에서 볼 수 있듯이 DSMDD 모델은 SMDD나 DMDD보다 훨씬 좋은 스피드업을 보여준다. 그러나 노드 갯수를 임의로 12개를 사용한 DSMDD_R의 경우 프로세서 갯수가 적은 곳에서는 오히려 SMDD보다 스피드업이 좋지 않다. 이것은 프로세서 갯수에 비하여 노드 갯수가 지나치게 많기 때문이며, 프로세서 갯수에 따라 성능 저하를 최소화하는 노드 갯수를 정한 후에 할당하는 DSMDD_O는 전체적으로 높은 스피드업을 얻을 수 있음을 알 수 있다.

(그림 8)은 DSMDD 모델의 통신 오버헤드에 따른 스피드업을 보여준 것인데, 통신 오버헤드는 동일한 프로세서 갯수를 사용했을 때, 가장 큰 오버헤드를 나타내는 기점을 기준으로 백분율로 나타내었다.




(그림 8) DSMDD 모델의 스피드업
(Fig. 8) Speedup of DSMDD model

DSMDD 모델은 프로세서 갯수가 증가할수록, 통신 오버헤드가 적을수록 성능이 우수함을 알 수 있었다. 예를 들어 프로세서 수가 151개일 때, 통신 오버헤드가 90인 경우의 speedup은 87. 5이며 10인 경우의 speedup은 105. 7로 나타났다. 그러나 그림에서 보이듯이 통신 오버헤드보다는 프로세서 갯수에 따라 더 민감한 성능 차이를 보이고 있다. (그림 7)과 (그림 8)을 종합하여 볼 때, 분산 메모리와 공유 메모리를 혼용하는 SPAX와 같은 시스템 상에서는 DSMDD 모델이 DMDD나 SMDD보다 월등히 우수함을 알 수 있다. 이는 SMDD 모델은 프로세서가 커지면 하나의 SMP에 과부하가 걸리게 되고, DMDD 모델은 노드간의 비싼 통신 오버헤드를 감수해야 하기 때문이다. DSMDD 모델의 경우는 이 모든 경우를 감안하여 착안되었기 때문에 효율적으로 운용될 수 있다. 가령, 프로세서가 적은 경우는 한 노드 내에서만 수행되며, 노드간 통신 오버헤드 이상의 과부하가 생기면 적정 노드 수를 선택하여 프로세서들을 분산시켜 줌으로써, 시스템의 가용성과 성능을 최적화시킬 수 있는 것이다.

## 6. 결론 및 향후 계획

본 논문에서는 첫째 DSM을 사용하는 SPAX 구조에 가장 적합한 자료 분산 모델로써 DSMDD 모델을 제시하고, 둘째 이 모델에서 배열을 프로세서에 할당하는 방법 및 프로세서들을 노드에 할당하는 방법을 기술하였으며, 셋째 서로 다른 프로세서에 있는 자료들 간에 자료 전송이 요구될 때 사용할 프로세서간 통신(IPC)과 서로 다른 노드에 위치한 프로세서들간에 자료 통신을 수행하는 노드간 통신(INC)을 정의하였다. 그리고 마지막으로 시뮬레이션을 통하여 이 모델이 SPAX와 같은 구조에서는 매우 효율적임을 증명하였다.

제시된 DSMDD 모델은 마스터-슬레브 모델을 확장한 계층적 마스터-슬레브 구조로 노드 내에서는 공유 메모리를 사용하고 노드간에는 메세지 전달을 사용한다. 이 모델은 SPAX 시스템의 자원의 가용성을 최적화할 수 있는 모델로 이와 유사한 DSM 구조를 사용하는 시스템에 쉽게 적용될 수 있다. 시뮬레이션 결과 DMDD 모델은 SPAX와 같은 DSM 시스템에서 현격하게 저조한 성능을 보이며, SMDD 모델은 적은 크기의 태스크에서는 유리하나 태스크가 커질수록 불리하다. DSMDD 모델의 경우에는 임의의 노드수에 분산했을 때, 만일 노드 수가 태스크 수에 많으면 오히려 SMDD보다도 성능이 저조하나 시스템 저하 요소를 최소화하는 노드 갯수를 사용한 DSMDD 모델은 전체적으로 가장 좋은 성능을 얻었다. 특히, DSMDD 모델은 프로세서 갯수가 클수록 통신 오버헤드가 적을수록 좋은 성능을 얻는다.

현재 2개의 펜티엄 프로세서를 장착한 Compaq ProLiant 4500 하드웨어에 Novell Unixware 2.01을 설치한 시스템과 Solaris 2.4를 설치한 Sun 워크스테이션, 그리고 Unixware를 설치한 펜티엄 PC를 네트워크으로 연결한 시험 환경에서 DSMDD 모델을 시험하고 있다. HPF 컴파일러의 하부 구조로는 소켓과 SYSV IPC를 사용한 p4 버전의 MPI 라이브러리를 사용하고 있다. 앞으로는 소켓 대신에 운영체제에서 제공하는 노드간 통신 프리미티브를 이용한 SPAX용 DSMDD 모델을 구현할 예정이며, SPAX 시스템 상에서의 시험은 97년 초에 수행할 예정이다.

## 참 고 문 헌


[1] HPFF, High Performance Fortran Languages Specification, Version 1.0, Rice University, May 1993.

[2] MPIF, MPI: A Message-Passing Interface Standard, Message Passing Interface Forum, April 15, 1994.

[3] R. Babb II, A. Choudhary, L. Meadows, S. Nakamoto and V. Schuster, "Reatargetable High Performance Fortran Compiler Challenges," *Proc. IEEE*, pp. 137-146, 1993.

[4] Z. Bokus, L. Meadows, S. Nakamoto and V. Schuster, "Retargetable HPF Compiler Interface," *4th Workshop on Compilers for Parallel Computers*, Delft Univ., Dec. 1993.

[5] T. Brandes, "Compiling Data Parallel Programs to Message Passing Programs for Massively Parallel MIMD Systems," GMD Web site, 1993.

[6] Y. W. Kim, S. W. Oh and J. W. Park, "Design Issues and the System Architecture of TICOM-IV, A Highly parallel Commercial Computer," *Proceeding of Euromicro Workshop on Parallel and Distributed Procesing*, pp219-226,January 1995.

[7] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr. and M. Zosel, *The High Performance Fortran Handbook*, The MIT press, 1994.

[8] 김학영, 민옥기, 이재경, "고속병렬컴퓨터 용 병렬 Fortran 컴파일러 설계," 1995년도 한국정보과학회 가을 학술발표 논문집, Vol. 22 No. 2, pp857-860, 1995.

[9] 민옥기, 김학영, 지동해, "고속병렬컴퓨터 용 병렬 Fortran 언어 설계," 1995년 한국정보처리학회 춘계 학술발표 논문집, 제2권 제1호, pp467-470, 1995.

[10] 홍철의, 안대영, "병렬처리 시스템에서의 합성부하 모델 분석," 1995년 정보처리학회 추계학술발표 논문집, 제2권 제2호, pp323-329, 한국정보처리학회, 1995.



**민 옥 기**

1988년 충남대학교 계산통계학과(학사)

1992년 충남대학교 계산통계학과(석사)

1988~현재 한국전자통신연구소 연구원

관심분야: 컴파일러(특히, 병렬 컴파일러 및 최적화), 컴퓨터 그래픽스, 객체지향 등임.