Modular MIN에 관한 연구

A Study on Modular Min

장창수
여수대학교 컴퓨터공학부 교수

최창훈
상주대학교 컴퓨터공학부 조교수

유창하
여수대학교 대학원 컴퓨터공학부 병렬처리연구실

Chang-Soo Jang
Professor, Dept. of Com. Eng., Yuso National University
Chang-Hoon, Choi
Professor, Dept. of Com. Eng., Sangju National University
Chang-Ha, Yoo
Parallel Processing Lab., Dept. of Com. Eng., Yuso National University

중심어 : Shared-Memory Multiprocessor, MIN, 하이퍼큐브, BaseNet

요 약

비록 MIN이 짧은 직경을 갖고 있을지라도 지역화된 통신 형태를 갖는 병렬응용 프로그램에 있어서 hypercube와 tree구조를 비교했을 때 전체적인 시스템 성능은 떨어지게 된다. 그것은 MIN이 지역참조성의 활용할 수 있는 클러스터링 구조를 제공하는 것이 불가능하기 때문이다. 그러나 제안된 MIN은 잦은 데이터 통신 형태를 갖는 프로세서-메모리 클러스터의 내부에 짧은 경로 및 다중 경로를 제공하여 지역화된 통신 구조에 적합하도록 구성할 수 있다. 따라서 제안된 MIN은 지역화된 통신 형태를 갖는 병렬 응용 프로그램에 있어서 향상된 성능을 이룰 수 있게 된다.

Abstract

In parallel application programs with a localized communication, even if the MINs have low diameters, overall system performance degrades when compared to the hypercube and tree structure. The reason is that it is impossible for MINs to provide some mechanisms for clustering to exploit the locality of reference. However proposed MIN can be constructed suitable for localized communication by providing the shortcut path and multiple paths inside the processor-memory cluster which has frequent data communications. Therefore proposed MIN achieves enhanced performance in parallel application program with a localized communication.

Ⅰ. 서론

일반적으로 공유 메모리 시스템에서 범용의 응용 프로그램이 수행되는데 있어 프로그램의 요구에 따르는 다양한 통신 형태를 제공할 수 있게 하는 동적 연결 네트워크를 많이 사용하고 있다. 더우기 밀 결합 다중 프로세서 시스템(tightly coupled multiprocessor system)에서는 공유 메모리의 의존도가 약 결합 다중프로세서 시스템 (loosely coupled multiprocessor system)보다 높기 때문에 통신이 보다 많이 발생하게 되어 상호연결 네트워크에 대한 중요도는 상대적으로 한층 더해질 것이다. 이러한 밀결합 공유 메모리 다중 프로세서(shared-memory multiprocessor) 시스템에서의 동적 상호연결 네트워크로서 보다 효율적인 상호연결 네트워크의 구성을 위한 연구가 많이 진행되고 있다.

그러나 동적 연결 네트워크인 기존의 MIN (Multistage Interconnection Network)에서는 모든 프로세서-메모리간의 통신 거리는 항상 log2N(N×N MIN에서 스테이지 수)이기 때문에, 프로세서의 수의 증가로 인해 스테이지 수가 증가되어 지연 시간이 점점 더 길어지게 된다. 따라서 이러한 수많은 프로세서를 갖는 대형 시스템에서 각 프로세서, 메모리쌍 모두에 이렇게 동일한 길이의 연결 경로를 제공하기 보다는, 통신이 자주 발생되는 작은 그룹에 더 빠른 경로를 제공하여 통신의 지역 참조성을 활용할 수 있는 MIN을 개발함으로서 보다 향상된 시스템 성능을 얻을 수 있을

※ 이 논문은 정보통신연구진흥원 2001 대학 기초연구지원사업 연구비로 연구되었음.

것이다.

또한 지역 참조성을 활용할 수 있게 하여 통신 빈도수가 높은 지역 참조의 경우에는 보다 빠른 경로를 제공함으로써 자주 발생되는 통신에 대한 지연 시간을 줄일 수 있을 것이다. 따라서 프로세서와 메모리 모듈로 구성된 작은 크기의 클러스터에 높은 지역화 통신 분포를 가지며, 또한 통신 지연 시간이 네트워크의 조합 능력(combinatorial power)보다 더 중요하게 되는 병렬 응용 분야에 효과적으로 사용할 수 있는 상호연결 네트워크의 설계를 목표로 한다.

II. 연구 배경

일반적으로 공유 메모리 다중 프로세서 시스템 환경하에서 많은 사용자들이 사용하는 대다수의 응용 프로그램들에서는 적은 수의 프로세서-메모리 그룹내에서의 통신되는 빈도수는 전체 통신양 중의 많은 부분을 차지하기 때문에, 이들 그룹에 대한 짧은 경로의 제공이 필요로 한다[1],[2],[5],[11]. 그러나 기존의 MIN에서는 모든 통신 쌍들간에는 스테이지 수와 동일한 거리가 항상 유지되기 때문에 지역 참조를 활용할 수 없게 된다. 이러한 지역 참조성의 손실은 시스템 성능을 저하시키는 한가지 요인이 될 수 있다. 일반적으로 단일 프로세서 시스템 환경에 있어서는 대부분의 메모리 참조는 메모리 위치상에서 아주 적은 부분에서만 발생하게 된다. 이러한 연구는 캐쉬를 기반으로 한 시스템(cache based system)의 발전을 성공적으로 이루게 되었다. 이와 유사하게 다중 프로세서 시스템 환경 하에서의 많은 대부분의 응용 프로그램에서는 프로세서간의 통신(interprocessor communication)은 주로 프로세서-메모리들의 작은 크기를 갖는 그룹에서 발생하게 된다[6][8],[10]. 따라서 수많은 프로세서를 갖는 대형 시스템에서 각 프로세서, 메모리 쌍간에 모두 동일한 길이의 연결 경로를 제공하기보다는 통신이 자주 발생되는 작은 그룹에 더 빠른 경로를 제공함으로써 보다 향상된 시스템 성능을 얻을 수 있을 것이다. 이러한 프로세서들간에서 통신 분포의 지역화를 본 논문에서는 지역 참조성이라는 표현으로 사용할 것이다. 예 1은 참고문헌[1]을 기초로 하여 기존의 MIN에서 통신의 빈도수가 높은 그룹에 빠른 경로를 제공함으로써 얻게되는 잇점을 보인 것이다.

예 1 한 시스템에 4개의 프로세서(P1 ~ P4)가 있다고 하자. 그리고 이들 프로세서간의 통신 빈도수를 측정한 결과 표 1에서와 같이 산출되었다고 하자.

표 1. 프로세서의 상호 통신 분포

Communication Between Processors (Total communication of each processor normalized to 1)				
	P1	P2	P3	P4
P1		0.7	0.2	0.1
P2	0.7		0.1	0.2
P3	0.2	0.1		0.7
P4	0.1	0.2	0.7	

각 프로세서 쌍에 따른 통신의 빈도수를 살펴보면, P1과 P2 그리고 P3과 P4가 각각 0.7로서 다른 프로세서 쌍들보다 높게 나타난 것을 볼 수 있다. 만약 이들을 2개의 그룹 {P1,P2}와 {P3,P4}로 나누어 구성시킨다면, 이들에 대한 상호연결 네트워크를 MIN으로 구성할 때 적용되는 스테이지 수는 $\log_2 p$개(여기서 p는 프로세서의 수)이므로 이들에 대한 평균 통신 지연은,

$0.7 \times \log_2 2 + (0.2 + 0.1) \times \log_2 4 = 1.3$으로써,

그룹화 시키지 않았을 경우의 통신 지연, $\log_2 4 = 2$ 보다 적은 통신 지연 시간을 얻을 수 있다. 따라서 일반적으로 많이 그리고 자주 사용되는 응용 프로그램의 통신의 형태 등을 추적 도구(tracer tool)를 이용하여 이들에 대한 통신 분포를 알아낼 수 있다면, 이러한 프로세서-메모리 그룹에 보다 짧은 경로를 제공함으로서 시스템 성능을 보다 향상시킬 수 있을 것으로 기대할 수 있다.

III. 메모리 검색 시간에 따른 오버헤드

병렬 처리 다중프로세서 시스템에서 원거리 메모리 참조에 따른 문제는 매우 중요한 문제 중에 하나이다. 표 2은 실제 시스템에서 원거리 메모리로부터 한 워드(word)를 검색하는데 걸리는 시간을 요약한 것이다.

예를 들어, 위와 같은 시스템들에서 근거리 메모리 참조에 대한 원거리 메모리 참조가 시스템에 미치는 영향을 평가하기 위해 32개 프로세서 시스템에서 한 개의 응용 프로그램을 가정하였다. 그리고 원거리 메모리 참조에 걸리는 시간이 2000ns라고 하자. 각 프로세서는 원거리 참조를 할 때는 정지 상태에 있고, 프로세서의 주기 시간(cycle time)은 10ns이라고 하고, 기본 CPI가 1.00이라 하자. 그리고 이 시스템에는 각 프로세서의 메모리 참조 중에서 0.5%가 원거리 메모리 참조를 수행한다고 하자.

표 2. 원거리 메모리로부터 한 개의 워드를 액세스하는데 걸리는 시간[11]

Machine	Communication Mechanism	Interconnection Network	# of Processors	Typical Remote Memory Access Time
SPARCCenter	Shared Memory	Bus	<20	1 μ
SGI Challenge	Shared Memory	Bus	<36	1 μ
Cray T3D	Shared Memory	3D torus	30-2048	1 μ
Convex Exemplar	Shared Memory	Crossbar+ring	8-64	2 μ
KSR-1	Shared Memory	Hierarchical Ring	32-256	2-6 μ
CM-5	Message Passing	Fat tree	32-1024	10 μ
Intel Paragon	Message Passing	2D mesh	32-2048	10-30 μ
IBM SP-2	Message Passing	Multistage Switch	2-512	30-100 μ

CPI(Cycle Per Instruction)[11]는

CPI = BaseCPI+Remote request rate×Remote request cost

= 1.0 + 0.5% × Remote request cost

Remote request cost = Remote access cost ÷ Cycle time

= 2000ns ÷ 10ns

= 200 cycles.

따라서 CPI는 아래와 같이 구할 수 있다.

CPI = 1.0 + 0.5% × 200 = 2.0

이와 같이 인스트럭션(instruction)을 수행시키는데 필요로 하는 시간을 측정하는 중요한 요소인 CPI에서도 잘 나타나듯이, 만약 지역 메모리 참조를 가지 시스템에서는 원거리 메모리 참조하게 하는 시스템에서 보다 2배의 성능 향상을 볼 수 있게 된다.

이러한 지역 메모리 참조에 대한 효율성을 활용하기 위하여 일반적으로 자주 사용되는 응용 프로그램에 대한 통신 확률 분포 함수(communication probability distribution function)를 찾을 수 있다면, 프로세서-메모리 쌍들에서의 통신에 대해 지연 시간을 최소화하는 프로세서-메모리 모듈쌍들에 대한 클러스터링(clustering)의 크기를 결정할 수 있을 것이다.

Abraham과 Davidson[1]은 이러한 통신 분포(communication distribution)를 통하여 통신 지연 시간을 최소화시킬 수 있는 클러스터 크기를 결정하는 연구가 있었다. 여기서의 통신 분포 함수(communication distribution function)를 위한 curve는 통신의 빈도 수에 대한 등급(rank)으로써 그려질 수 있다. 앞에서 언급된 표 1에서 통신 빈도 수 0.7 은 등급 1에 해당되고 통신 빈도 수 0.2로, 그리고 통신 빈도 수 0.1은 등급 3이므로, 그에 대한 점(point)은 (1, 0.7), (2, 0.2), (3, 0.1)으로써 형성할 수 있어, 통신 분포 곡선(communication distribution curve)을 그릴 수 있으므로, 이에 대한 통신 분포 함수 f(q)를 얻을 수 있다. 따라서 통신 분포 곡선 f(q)가 주어진다면, 아래와 같이 한 응용 프로그램에 있어 클러스터 크기 c를 갖는 네트워크에서의 평균 지연 D_c는 아래와 같이 구할 수 있다[1].

$$D_c = \log c \cdot \int_0^{c-1} f(q)dq + \log p \cdot \int_c^{p-1} f(q)dq$$

그리고 위의 식의 목적은 주어진 프로세서 갯수 p에 대해 $Min_c D_c$를 달성하는 것이므로, 다음과 같은 식을 얻을 수 있다.

$$Max_c[(\log p - \log c) \cdot \int_0^{c-1} f(q)dq]$$

따라서 지연시간을 최소로 하는 최대 크기의 클러스터를 찾을 수 있는 것이다. 이에 관련된 또 다른 연구로서, 병렬처리 시스템에서 자주 사용되는 프로그램에 대해서 수행시간을 최소화하기 위한 프로세서 그룹을 결정하는 연구가 있었다[2]. 여기에서는 다음과 같이 3가지 접근 방법을 통한 작업 크기의 분할 및 메모리 액세스 전략을 정의하였다[2].

1. Naive Approach:
   입력 데이타를 모두 공평히 분할하고 데이타 액세스들에 대해서는 어떠한 동기화도 되어 있지는 않다.
2. Semi-Naive Approach:
   입력 데이타는 모두 공평하게 분할하고 데이타 액세스들에 대해서 동기화 시킨다.
3. Optimal Approach
   데이타 액세스들은 모두 동기화 되어 있으며, 입력 데이타는 모두 동일하게 공평히 나누어지는 것이 아니고, 다음과 같은 방법으로 분할된다[1].

Min T subject to

$$T \geq \sum_{i=1}^{k} a(s_i) + y(s_k) + \sum_{i=k}^{n} b(s_i)$$

for k=0, ..., n

$$\sum_{i=0}^{n} s_i = 1$$

여기에서 $a(\cdot), b(\cdot), y(\cdot)$는 각각 입력 통신 시간, 출력 통신 시간, 컴퓨팅(computing) 시간을 나타낸다. 이들 중에서 $a(\cdot), b(\cdot)$는 미리 알려진 네트워크의 통신 속도로 부터 직접 얻을 수 있다. 또한 $y(\cdot)$는 주어진 알고리즘에 의해 결정되어지게 되는데, 이것에 대한 추정 시간은 단일 프로세서에 다양한 여러 크기의 작업을 할당시켜 산출하여 얻어 낼 수 있다. 따라서 위의 식을 이용하여 주어진 작업을 최적의 크기로 분할시킬 수 있을 것이다.

또한 이러한 이론을 평가하기 위해, R. Agrawal, H.V. Jagadish[1]는 2개의 100×100 정수 행렬의 곱(integer matrices multiplication)을 입력 데이타를 이용하여 이를 실험하였다. 그림 1과 표 3는 이러한 행렬의 곱을 프로세서 수를 변화시키면서 최적 접근 방법에 대한 이론적인 예상 결과 및 3가지 접근 방법에 대한 실험 결과를 보인 것이다. 이 그래프에서 살펴볼 수 있듯이, 최적의 접근 방법에서도 100 × 100 정수 행렬의 곱을 5개의 프로세서에서 수행시킬 때까지는 수행시간이 감소하다가, 5개 이상의 프로세서 수에 대해서는 통신의 부하영향으로 전체 수행 시간은 점차 증가하게 된다.

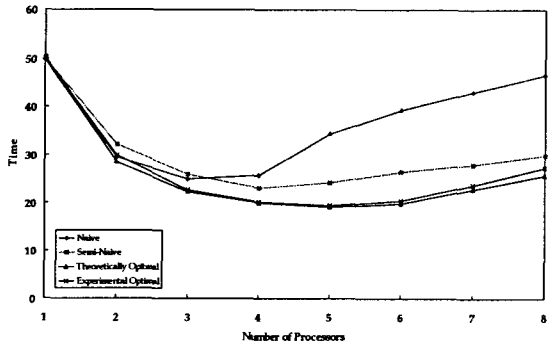


그림 1. 100×100 정수 행렬 곱에 대한 실행 시간[2]

표 3. 3가지 접근 방법 대한 실행 시간의 비교[2]

Processors	Execution Time (second)			
	Naive	Semi-Naive	Optimal	
	Experimental	Experimental	Theoretical	Experimental
1	49.91	50.32	50.06	50.32
2	29.52	32.23	28.55	29.98
3	24.98	26.01	22.34	22.75
4	25.75	23.05	19.94	20.22
5	34.39	24.18	19.07	19.55
6	39.36	26.41	19.92	20.52
7	43.12	27.92	22.80	23.67
8	46.76	29.88	25.68	27.34

IV. 병렬 프로그램의 수행에 대한 통신 오버헤드

병렬 프로그램을 수행에 대한 성능을 결정하게 되는 주된 요소로서 통신에 대한 계산의 비율을 들 수 있다. 만약 그 비율이 높다면, 이 응용 프로그램은 각 데이타에 대한 데이타의 통신에 소요되는 시간 보다 계산에 소요되는 시간이 많은 부분을 차지하고 있다는 의미로서 평가되는 것이다. 이러한 통신은 병렬 프로그램에 있어 값비싼 비용을 치르는 것이 된다. 따라서 병렬처리 환경에 있어서 문제의 크기의 증가 또는 문제 해결을 위해 사용되는 프로세서의 증가에 따른 계산 대비 통신 비율의 변화에 대한 정보는 응용 프로그램을 효율적으로 빠른 수행을 보장하게 될 것이다. 일반적으로 P개의 프로세서를 사용하는 병렬 프로그램을 처리하는데 소요되는 총 시간인 T_P는 아래와 같이 나타낼 수 있다.[11]

$$T_P = TCOMP_P + TCOMM_P + other\ terms$$

여기서 $TCOMP_P$는 P개의 프로세서로 병렬 처리하는데 소요되는 계산 시간이고, $TCOMM_P$는 P개의 프로세서를 사용할 때 통신의 총 시간을 나타낸다. 만약 순차처리 시간이 T_S라고 하자. 그리고 병렬 처리 환경에서 이상적인 모델로서 완전한 병렬 수행을 가정하였을 때 P개의 프로세서를 사용하였을 경우 주어진 문제에 대한 수행 시간은 $TCOMP_P = T_S/P$이다.

따라서 speedup의 방정식은

$$\frac{1}{S_P} = \frac{T_P}{T_S} = \frac{1}{P} + \frac{TCOMM_P}{T_S}$$

$$= \frac{1}{P} + \frac{1}{r_P} \text{ 이 된다.}$$

여기서 $r_P = T_S / TCOMM_P$이다. 이것은 P개의 프로세서를 갖는 시스템에서 통신 시간에 대한 계산 시간의 비율을 의미한다. 여기서의 성능향상(speedup)에 대한 식을 살펴보면, 주어진 문제 해결을 위해 많은 수의 프로세서를 사용하면 계산에 걸리는 시간은 줄어들지라도 통신을 위해 필요로 하는 지연 시간 또한 증가되어 문제 해결을 위한 전체의 성능향상은 나쁘게 나타나게 된다는 사실을 알 수 있다. 따라서 주어진 응용 프로그램 및 문제의 크기에 따라 적당한 프로세서의 수를 적용하므로써 빠른 문제 해결을 이룰 수 있게 된다. 이에 대한 한가지 예로써, 표 4는 병렬 컴퓨터에서 성능을 결정하게되는 각 응용 프로그램에 대한 계산에 대한 통신 비율을 나타낸 것이다.

표 4. 병렬컴퓨터의 성능을 결정하는 계산, 통신의 확장성 및 그의 비율[6]

Application	Scaling of Computation	Scaling of Communication	Scaling of Computation Communication
FFT	$\frac{n\log n}{p}$	$\frac{n}{p}$	$\log n$
LU	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Barnes	$\frac{n\log n}{p}$	$\frac{\sqrt{n}(\log n)}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}\ \frac{\sqrt{n}}{\sqrt{p}}$
Ocean	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$

이러한 사실은 고정된 크기의 문제를 해결하기 수행시키기 위해 많은 프로세서의 사용은 프로세서간의 통신의 증가로 인해 오히려 비효율적이라는 것을 보여주는 것이다. 또한 병렬처리 응용 프로그램들은 주어진 문제의 크기에 대해 각 알고리즘 설계자가 요구하는 최적의 시간을 발휘할 수 있는 프로세서의 수는 정하여지게 되어 병렬 처리 컴퓨터에 최적의 수로 할당할 수 있게 된다. 표 5는 문제의 크기가 n일 때 공유 메모리 환경 하에서 각 알고리즘들의 최적의 성능과 이를 만족시킬 수 있는 프로세서의 수를 나타낸 것이다.

표 5. 공유 메모리 알고리즘의 성능[3]

Problem	Number of Processors	Asymptotic Complexity
Maximum	$\frac{n}{\log n \log n}$	$\log\log n$
Merging	$\frac{n}{\log n}$	$\log n$
Merging	$\frac{n}{\log n \log n}$	$\log\log n$
Sorting	$\frac{n}{\log n}$	$\log^2 n$
Sorting	n	$\log n$
Median	n^{ε}	$n^{1-\varepsilon}$
Median	$\frac{n}{\log n \log\log n}$	$\log n \log\log n$
Connected Component	$\frac{n^2}{\log^2 n}$	$\log^2 n$
Minimum Spanning Tree	$\frac{n^2}{\log^2 n}$	$\log^2 n$
Minimum Spanning Tree	n^2	$\log n$
Minimum Spanning Tree	$\frac{n^2}{\log n \log\log n}$	$\log n \log\log n \log\log\log n$
Biconnected Component	$\frac{n^2}{\log^2 n}$	$\log^2 n$
Biconnected Component	n^2	$\log n$
Euler Tour	n^2	$\log n$
String Match	$\frac{n}{\log n}$	$\log n$
Parsing Arithmetic Exp.	$\frac{n}{\log n}$	$\log n$
Max Flow	n	$n^2 \log n$

이러한 통신이 빈번한 프로세서 그룹 내에서의 지역 참조성을 활용할 수 있는 상호연결 네트워크로서 위에서 언급된 정적 네트워크인 hypercube[4],[5],[17]를 예를 들 수가

있는데 이는 임의의 두 노드간의 직경(diameter)이 n으로서 안정적인 거리를 갖고 있을 뿐만 아니라 지역 참조성(잦은 통신 빈도 수)이 높은 노드와는 거리가 1인 이웃(direct neighbor)으로 연결되어, 자주 발생되는 통신에 대하여 짧은 지연을 보장하게 하였다.

또한 정적 네트워크로서 이진 트리(binary tree)구조에서도 지역 참조성이 높은 노드와는 거리가 1인 이웃노드로 연결될 수 있어, 지역 참조 역시 활용할 수 있다. 이러한 트리 네트워크구조는 병렬처리 데이타 베이스 및 여러 응용 분야에서 많이 사용되고 있다 [5],[9].

V. UNIT module을 기반으로한 점진적 확장 설계 기법

본 절에서 정의될 UNIT module은 4개의 입력 포트와 4개의 출력 포트를 갖고 있으며, 3개의 스테이지(입력 스테이지, 중간 스테이지, 출력 스테이지)로 구성된 한 개의 네트워크이다. 여기서 입력과 출력 스테이지에서는 각각 2개의 스위칭 소자를 갖고 있으며, 중간 스테이지에서는 한 개의 스위칭 소자로 구성되어 있다. 여기서 UNIT module의 구성의 규칙을 정의하기로 하겠다. UNIT module에서의 입력 스테이지 및 출력 스테이지에서는 각각 2개의 스위칭 소자로 구성되어 있기 때문에 스위칭 소자의 번호는 1-bit 2진 형태인 b_1으로 표현될 수 있다. 따라서 입력 스테이지와 출력 스테이지에 위치한 스위칭 소자의 링크들의 2진 표현 형식은, $b_1 b_0$가 된다.

또한 Modular MIN의 확장은 이러한 UNIT module과 아래와 같이 정의된 BaseNet으로 부터 설계가 이루어지게 된다. 아래의 규칙은 재귀 관계식(recurrence relation equation)으로서 Modular MIN의 확장에 관련된 topology describing function을 나타낸다. 따라서 Modular MIN의 제 k차 (kth)확장에 대한 규칙을 아래와 같이 정의한다.

Modular($2^k \times 2^k$):= BaseNet ‖ Add_UNIT

BaseNet := Modular($2^{k-1} \times 2^{k-1}$)

Add_UNIT := 2^k/4 UNIT

(여기서 만약 k=2이면, 제2차 확장일 때, 제1차 확장으로서 BaseNet, Modular($2^1 \times 2^1$)은 단지 한 개의 SE를 갖게 된다.)

Modular MIN의 확장 과정에서 추가되는 2^n/4개 Add_UNIT의 각 입력(출력) 스테이지의 스위칭 소자들은 위(top)에서 부터 시작하여 2진 형태 표현으로써 bℓ ... b1, (ℓ=n-1)와 같이 부여될 수 있다. 위의 식에서 operator ‖ 는 2^n/4개 UNIT 모듈을 BaseNet에 연결시키기 위한 연결 동작을 의미한다. 그리고 이러한 확장 규칙들은 아래에서 정의되는 매핑 함수(mapping function) f^i에서, 상단 출력 포트인 Up에 대한 연결 함수 f^0와 관련이 있게 된다. f^i (i=0 또는 1이며, 여기서 0은 스위칭 소자의 입/출력의 Up 포트를 의미하며, 1은 Down 포트를 의미함)은 UNIT module을 위한 매핑 함수라고 하자. 또한 function $f^1(b_\ell \ldots b_1)_{UNIT_{I/O}} = (b_1)_{UNIT_M}$는 UNIT module의 입력(출력) 스테이지에 있는 스위칭 소자 $b_\ell \ldots b_1$의 Down 출력 링크(입력 링크)는 그 UNIT module내의 중간 스테이지에 있는 스위칭 소자의 입력(출력) 링크 b_1에 연결되어 있음을 의미한다.

또한, $f^0(b_\ell \ldots b_1)_{UNIT_{I/O}} = (b_\ell \ldots b_1)_{BaseNet_{I/O}}$는 UNIT module의 입력(출력) 스테이지에 있는 스위칭 소자 $b_\ell \ldots b_1$의 Up 출력(입력) 링크는 BaseNet에 있는 스위칭 소자의 입력(출력) 링크 $b_\ell \ldots b_1$에 연결되는 것을 의미한다.

초기 상태, 즉 확장이 이루어지지 않은 기본 단위로서의 UNIT module에 대한 표현에 있어서 입력(출력) 스테이지에 있는 스위칭 소자 b_1의 상단 출력 링크인 Up 출력 링크(입력 링크)는 그 UNIT module 밖에 있는 다른 모듈의 스위칭 소자와 연결시키기 위한 것으로서 UNIT module내에서 입력(출력) 스테이지 스위칭 소자의 UPP 링크들은 네트워크 확장에 사용될 링크들이므로 이 UNIT 모듈내의 어느 스위칭 소자에도 연결이 이루어지지 않은 상태인 것을 의미하여 $f^0(b_1) = outside$와 같이 나타낼 수 있다.

따라서 위의 두 함수 f^0와 f^1에 대해 Modular MIN의 확장을 고려했을 경우, 즉 $f^0(b_\ell \ldots b_1)_{UNIT_{I/O}} = (b_\ell \ldots b_1)_{BaseNet_{I/O}}$ 와 $f^0(b_\ell \ldots b_1)_{UNIT_{I/O}} =$

$(0)_{UNIT_M}$ 으로 각각 표현될 수 있다. 또한 입력(출력) 스테이지에 있는 스위칭 소자 b_1의 출력(입력) Up 포트는 다른 UNIT module인 BaseNet의 모듈들에 대한 연결을 지원한다. 즉 네트워크가 확장되어질 때 사용되어 질 것이다.

VI. 프로세서 클러스터링

이러한 확장 법칙은 각 UNIT module들의 입력(출력) 스테이지에 있는 스위칭 소자들의 Up 링크를 통해서 이루어짐을 알 수 있다. 따라서 Modular MIN에서의 n차 확장 f^0_{nth}에 형성 규칙은 아래와 같이 정리될 수 있다.

○ Expansion n : $f^0_{nth}(b_{\ell} \ldots b_1)_{UNIT_{I/O}}$

○ Expansion n-1 : $\leftarrow f^0_{n-1th}(b_{\ell} \ldots b_2)_{UNIT_{I/O}}$

○ Expansion n-2 : $\leftarrow f^0_{n-2th}(b_{\ell} \ldots b_3)_{UNIT_{I/O}}$

⋮ ⋮

○ Expansion 2 : $\leftarrow f^0_{2nd}(b_{\ell})_{UNIT_{I/O}}$

○ Expansion 1 : $\leftarrow f^0_{1st}(0)_{UNIT_{I/O}}$

위의 정의에 따르면 n차 확장까지의 상단 출력 링크로의 연결은 n-1차 확장 네트워크가 먼저 이루어져야 하며, 또한 n-1차 Modular MIN을 위해서는 n-2차 확장이 먼저 선행되어야 한다. 이러한 과정은 1차 확장, 즉 Modular MIN의 최소 단위인 UNIT까지 전개된다. 위와 같은 확장 규칙을 이용하여 설계된 4x4 Modular MIN은 그림 2와 같이 구성할 수 있다.

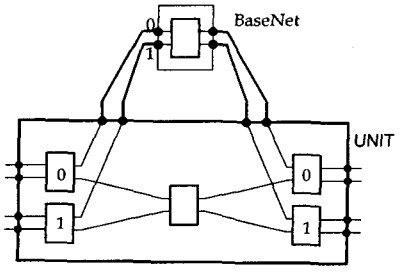


그림 2. 4x4 Modular MIN

각 클러스터 내에는 4개에서 최대 N개의 메모리 모듈을 포함하고 있으며, 또한 선택 가능한 중복 경로가 2개에서 최대 n개가 존재하게 된다. 또한 이들 클러스터에 따라 프로세서 및 메모리간의 거리도 최단 거리로서 3에서 부터 2n-1개까지 다양한 길이를 가지게 된다. 표 6은 16×16 Modular MIN의 각 프로세서에 대해 각각 3가지의 클러스터들을 예로써 표현한 것이다. 프로세서 P0는 클러스터 1에 해당하는 것으로서 M0,M1,M2,M3을 갖고 있으며, 또한 클러스터 2에 해당하는 것으로서 M0에서 M7까지 8개의 메모리를 포함하고 있다. 그리고 16×16 Modular MIN에서의 최대 크기인 클러스터 3에서는 M0에서 M15인 메모리 모듈 전부를 포함하게 된다.

표 6. 16×16 Modular MIN에서 클러스터의 예

Cluster \ processor	$P_0 \sim P_3$	$P_4 \sim P_7$	$P_8 \sim P_{11}$	$P_{12} \sim P_{15}$
Cluster 1	$M_0 \sim M_3$	$M_4 \sim M_7$	$M_8 \sim M_{11}$	$M_{12} \sim M_{15}$
Cluster 2	$M_0 \sim M_7$		$M_8 \sim M_{15}$	
Cluster 3	$M_0 \sim M_{15}$			

Modular MIN에서 각 프로세서는 각각의 계층을 통해 연결될 수 있는 메모리 모듈들에 대해서는 경로의 수가 1개씩 추가되는 것을 알 수 있고, 또한 계층 2에서 부터 새로이 추가 되는 통신 가능한 메모리 수는 각 계층 번호에 대한 2의 멱승 단위로 증가하게 되므로, 각 프로세서당 계층에 따른 메모리 모듈의 갯수와 중복 경로의 수는 표 7과 같다.

표 7. 각 클러스터에 따른 메모리 수와 중복 경로 수

Cluster	메모리 모듈 갯수	중복 경로의 수
Cluster 1	4	n
Cluster 2	4	n-1
Cluster 3	8	n-2
⋮	⋮	⋮
Cluster n-1	2^{n-1}	2

따라서 Modular MIN에서 중복 경로의 평균 수는

$$\frac{4\times n+\sum_{i=1}^{n-1}2^i\times(n-i+1)}{4+\sum_{i=1}^{n-1}2^i}$$ 이다.

그림 3는 동일 네트워크 크기를 갖는 기존의 MIN과 Modular MIN에 대하여 프로세서-메모리 쌍들에 대한 평균 경로의 수를 그래프화한 것이다. 여기에서도 볼 수 있듯이 기존의 MIN에 대해서는 네트워크의 크기 변화에 관계없이 항상 1개의 유일 경로만 존재하지만, Modular MIN에서는 네트워크 크기가 커질 수록, 중복 경로의 수가 증가함을 볼 수 있다.

따라서 Modular MIN에서는 통신의 빈도가 높은 적은 크기의 프로세서-메모리에 대해 짧은 경로를 제공할 뿐만 아니라 많은 수의 중복 경로를 제공하기 때문에 매우 효율적인 네트 워크라고 할 수 있을 것이다.

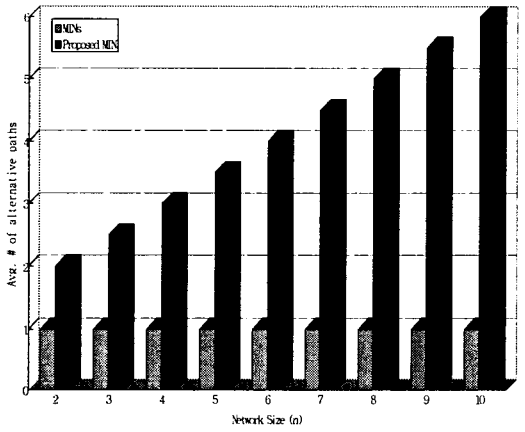


그림 3 평균 중복 경로의 수 비교

다음은 Modular MIN의 하드웨어 복잡도의 대부분을 차지하고 있는 스위칭 소자의 갯수는 초기의 N×N Modular MIN에서 사용되는 스위칭 소자 보다 $2^{n-1}-1$개의 2×2 스위칭 소자를 추가한 것으로서, $N\times N$ Modular MIN에서 사용되는 스위칭 소자의 갯수인 2N-3에 $2^{n-1}-1$를 더하면 된다. 따라서 Modular MIN에서 사용되는 스위칭 소자의 총 갯수는 아래의 식과 같이 구할 수 있다.

$$2N-3+2^{n-1}-1=2\cdot2^{n}-3+2^{-1}2^{n}-1$$
$$= 2.5N-4$$

따라서 Modular MIN에서 스위칭 소자의 수는 계속해서 O(N)을 유지할 수 있고, 또한 기존의 MIN에서의 스위치 소자의 하드웨어 비용, O(NlogN)에 비해 훨씬 적은 비용으로 써 FAC를 만족할 뿐만 아니라, 네트워크의 크기의 증가에 중복 경로의 수는 증가하고, 또한 계층이 낮을 수록 많은 중복 경로를 제공할 수 있어 최대 n개의 중복 경로를 제공할 수 있다. 또한 이들에 대해서는 짧은 경로를 제공할 수 있게 된다.

VII. 결론

본 논문에서는 공유 메모리 다중 프로세서 시스템에서 지역화된 통신 형태를 갖는 병렬 응용 프로그램을 효율적으로 수행시킬 수 있도록 하기 위한 상호연결 네트워크로서 Modular MIN을 제안하였다. Modular MIN은 통신이 빈번하게 발생되는 클러스터 내부에 보다 빠른 경로를 제공한다.

일반적으로 다중 프로세서 시스템 환경하에서 대부분의 응용 프로그램들은 프로세서와 메모리로 구성된 작은 크기의 클러스터 내부에서 상호 통신이 주로 발생할 수 있도록 작업들을 그룹화하여 할당할 수 있는 것으로 알려져 왔다. 따라서 동적 상호연결 네트워크인 MIN에서 이러한 지역 참조성을 활용할 수 있는 방법으로서 지역 참조성 활용에 대한 장점을 갖고 있는 정적 네트워크인 트리 위상을 적용하여, 이들 두 위상의 장점을 결합하는 새로운 부류의 형태의 MIN인 Modular MIN을 유도하였다.

Modular MIN의 확장 설계는 동일 크기인 UNIT 모듈로써 점진적 확장 기법을 통해 이루어지기 때문에 미리 설치되었던 적은 시스템 크기의 Modular MIN을 그대로 재사용을 할 수 있고, 사용중인 Modular MIN과의 연결 방법이 매우 단순하기 때문에 실제 상용 다중 프로세서 시스템에서 쉬운 확장 구현이 이루어질 수 있을 것이다.

Modular MIN은 정적 네트워크인 이진 트리 위상의 장점과 동적 위상인 MIN의 장점을 결합함으로서 지역 참조성의 활용과 적은 수의 스위칭 소자로써 대체 경로를 제공하고, 목적지 주소를 이용한 간편한 분산적 자기경로 제어 라우팅을 적용시킬 수 있을 뿐만 아니라 또한 계층 버스[12]에서와 같은 트래픽의 고립화를 MIN에서도 적용시킬 수 있어 통신의 효율성을 높일 수 있게 되었다. 따라서 Modular MIN은 하드웨어 비용과 성능면을 고려하였을 때 기존의 MIN보다 효율적인 네트워크이며, 또한 적은 수의 프로세서

와 메모리로 구성된 클러스터 내에서 통신의 빈도가 높게 지역화 된 MPP 시스템 환경 하에서 동일 크기의 기존의 MIN 보다 우수한 상호연결 네트워크 구조라 할 수 있다.

참 고 문 헌

[1] S.G. Abraham, and E.S. Davidson, "A Communication Model for Optimizing Hierarchical Multiprocessor System," In Proc. Int'l Conf on Parallel Processing, pp.467~474, 1986.

[2] R. Agrawal and H.V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," IEEE Trans. Compt., Vol. C-37, pp.1627~1634,. Dec., 1988.

[3] G.S. Almasi and A. Gottlieb, Highly Parallel Computing, The Benjamin/Cummings Publishing Company Inc., 1994.

[4] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," IEEE Trans. Compt., Vol. C-39, pp.775~785, 1990.

[5] A.L. Decegama, The Technology of Parallel Processing : Parallel Processing Architectures and VLSI hardware Vol. I, Prentice-Hall International Editions, 1989.

[6] M. Dubois and S.S. Thakkar, Cache and Interconnect Architectures in Multiprocessors, Kluwer Academic Pub., 1990.

[7] C.S. Ferner and K.Y. Lee, "Hyperbanyan Networks: A New Class of Networks for Distributed-Memory Multiprocessor," In Proc. the Fourth Symposium on the Frontiers of Massively Parallel Computation, pp.254~261, Oct., 1992.

[8] J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," Proc. 10th Symp., Computer Arch., pp.124~131, June, 1983.

[9] J.R. Goodman and C.H. Sequin, "Hypertree: A multiprocessor Interconnection Topology," IEEE Trans. Compt., Vol. C-30, pp.923~933, Dec., 1981.

[10] N. Suzuki, Shared Memory Multiprocessing, The MIT Press, 1992.

[11] D.A. Patterson and J.L. Hennessy, Computer Architecture A Quantitative Approach, Morgan Kaufmann Pub., 1996.

[12] A.W. Wilson, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," In Proc. 14th Int'l Symp. on Compt. Architecture, pp.244~252, 1987.

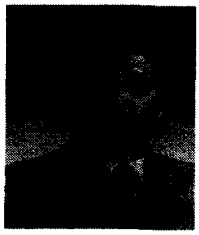
장창수(Chang-Soo Jang) 정회원

1980년 2월 : 조선대학교 전자공학과(공학사)
1982년 8월 : 건국대학교 전자공학과(공학석사)
1997년 2월 : 서강대학교 컴퓨터공학과(공학박사)
1999년 1월 ~ 2000년 2월 : 교환교수(California State Polytechnic University)
1984년 ~ 현재 : 국립 여수대학교 IT학부 컴퓨터공학과 교수
<관심 분야> : 병렬처리구조, 컴퓨터네트워크, DSP

최창훈(Chang-Hoon Choi) 정회원

1988년 2월 : 명지대학교 전자계산학과(학사)
1990년 2월 : 서강대학교 대학원 전자계산학과(공학석사)
1997년 8월 : 서강대학교 대학원 전자계산학과(공학박사)
1990년 : 대우통신(주) 기술개발부
1995년 ~ 1996년 : 미국 AT&T(NCR) 기술파견 연구원
1997년 9월 ~ 현재 : 국립상주대학교 컴퓨터공학부 교수
<관심분야> : 컴퓨터구조, 병렬처리시스템, 컴퓨터시뮬레이션

유창하(Chang-Ha Yoo) 정회원



2001년 2월 : 여수대학교 컴퓨터공학과(공학사)
2001년 ~ 현재 : 여수대학교 컴퓨터공학과 대학원