論文93-30B-4-4

하이퍼큐브 상에서의 부하 분산을 위한 최단 경로 할당 알고리듬

(A Shortest Path Allocation Algorithm for the Load Balancing in Hypercubes)

李 喆 源*, 林 寅 七*

(Chul Won Lee and In Chil Lim)

要 約

본 논문에서는 하이퍼큐브 시스템상에서 각각의 프로세서를 통한 최적의 모듈 할당 알고리듬 및 메시지 패싱 기법에 기초를 둔 최단 경로 할당 알고리듬을 제안한다. 멀티프로세서 시스템에서 발생하는 문제점은 수행하고자 하는 하나의 타스크를 어떻게 여러 개의 타스크로 나누어 수행시킬 것인가 하는 문제 및 각각의 수행 시간 예측 불가로 인하여 효율적인 수행을 할 수 없다는 것이다. 이를 해결하기 위하여 각각의 프로세스의 부하 분산과 융통성 있는 통신이 필요하게 된다. 이에 본 논문에서는 이러한 문제점을 해결하기 위하여 하이퍼큐브상에서 각각의 프로세서들이 액세스 가능한 지역메모리를 갖는 메시지 패싱 기법으로 프로세서들이 통신되는 프로세스들을 통하여 수행하는 병행성 프로그램 실현으로 최적의 모듈 할당을 위한 모듈들을 호출트리로 생성한다. 또한, 각 프로세서간에 가중치를 부여한 후 할당 그래프를 구축하여 최단 경로 할당 트리를 발견하고 하이퍼큐브로 사상, 내장 하므로써 부하분산을 고려한 최적의 할당을 실현한다.

Abstract

This paper proposes a shortest path allocation algorithm over the processors on the hypercube system based on the message passing techniques with the optimized module allocation. On multiprocessor systems, how to divide one task into multiple tasks efficiently is an important issue due to the hardness of the life cycle estimation of each process. To solve the life cycle discrepancies, the appropriate task assignment to each processor and the flexible communications among the processors are indispensible. With the concurrent program execution on hypercube systems, each process communicaties to others with the method of message passing. And, each processor has its own memory. The proposed algorithm generates a callable tree out of the module, assigns the weight factors, constructs the allocation graph, finds the shortest path allocation tree, and maps them with hypercube.

Ⅰ. 서론

하드웨어 설계기술과 소프트웨어 개발 기술은 보다 나은 컴퓨터 시스템을 구성하기 위해 발전하고 있다. 그 가운데 반도체 기술의 발달로 마이크로 프로세서와 메모리의 성능이 우수해지고 가격이 저렴해지면서 이 장점을 최대한으로 이용할 수 있는 다중(병렬)처리 시스템에 대한 관심이 높아지고 있다. 특히 병렬처리 하드웨어 분야가 많은 발전을 한 것에 비하여 병렬처리 소프트웨어 분야의 일종인 병행성(concurrency)프로

* 正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
接受日字: 1992年 10月 2日

그램 분야는 큰 성과를 거두지 못하고 있는 실정이며 앞으로의 주요 연구 분야이다. 또한, 기존의 대형 컴퓨터에 버금가는 다중 프로세서 시스템의 등장으로 병행 프로그래밍이 중요한 분야로 인식되고 있다. [1]

지난 10여년 동안 병행 프로그래밍은 두 가지 측면에서 크게 요구되어져 오고 있다. 첫째는 컴퓨터 프로그래밍 개념, 즉 이론적인 면의 발달로 병행 수행 및 각 프로세스의 동기화를 명시적으로 표현해 주는 새로운 프로그래밍 기법의 출현이고, 둘째는 하드웨어의 발달로 인해 프로세서의 가격이 낮아짐에 따라 다중 프로세서의 구축이 가능한 점이다. [1]

한편, 멀티프로세서 시스템에서 발생하는 문제로는, 먼저 하나의 타스크를 어떻게 여러개의 타스크로 나누어 수행시킬 것인가 하는 것이며, 또 각각의 수행 시간을 예측하기 어렵기 때문에, 효율적인 수행을 위해서는 각각의 프로세스의 부하를 균형있게 하기 위한 효율적이고 융통성 있는 통신 방법이 필요하게 된다. 따라서, 멀티프로세서 시스템은 여러 작은 프로그램들이 어떻게 공유 데이타를 처리하는 가에 따라 공유 메모리 방식과 메시지 전송 방식으로 나눌 수 있는데 사용자(프로그래머)측면에서 보면 하나의 공유 기억 공간을 제공하는 공유 메모리 방식이 프로그래밍하기가 편리하고 여러가지 프로그래밍 모델을 효율적으로 지원할 수 있지만, 하드웨어를 설계하는 입장에서는 여러 프로세서가 기억장치를 공유하는 것이 비용이 많이들고, 또 복잡한 상호 연결망이 필요하게 되므로 각 프로세서는 자신의 지역 기억 장치를 사용하고 타스크간의 통신은 메시지 전송을 통해 수행하는 방식을 이용한다. [2,9]

이러한 목적으로 사용되어 지는 것을 상호결합망(interconnection network) 이라 하며, 하이퍼큐브 및 메쉬의 여러가지 장점으로 인하여 현재 많이 연구되고 있다. 하이퍼큐브 멀티프로세서는 시스템 내의 PE(Processing Element) 수의 확장성이 매우 우수하므로 대규모(1000개 이상의 PE를 갖는 시스템) 멀티프로세서 시스템의 구현이 용이한 장점이 있다.

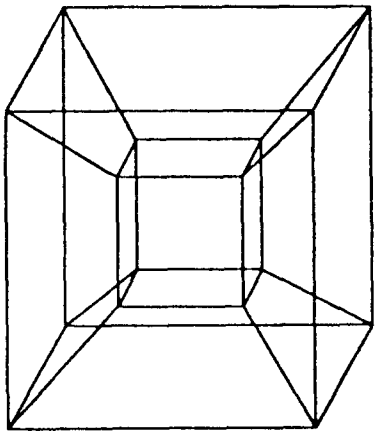
본 논문에서는 하이퍼큐브 시스템상에서 프로세서들을 통하여 최적의 할당된 프로그램의 모듈 상호간 통신 패턴을 그래프로 하고 각 프로세서마다 지역 메모리를 갖는 하이퍼큐브 상에서의 메시지 패싱 기법으로 인한 프로세스들 상호간의 통신에 IPC (Inter Processor Communication) 오버헤드 등의 부하분산을 고려한 효과적인 동적 프로그램 접근법인 최단 경로 할당 알고리듬을 제안한다. 제안한 알고리듬의 성능 검증을 위하여 모듈 실행과 통신에 대한 코스트의 최저값 할당을 할당 그래프에서 최저 가중치 할당 트리의 발견으로 그 효율성을 입증한다.

Ⅱ. 메시지 패싱을 위한 하이퍼큐브

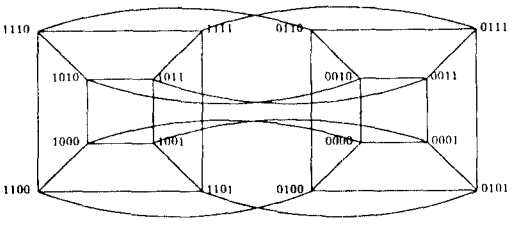
하이퍼큐브는 $N(=2^n)$개의 노드(node; 멀티프로세서 시스템에서는 PE에 해당)로써 n 차원의 이진 큐브를 형성하므로 2진 n-큐브 라고도 말한다. 하이퍼큐브의 차원이 3 이상일 경우에는 tree, grid, ring, torus등 임의의 연결구조를 갖는 시스템으로 노드 및 링크의 대응에 의한 전환이 가능하다. 따라서 하이퍼큐브는 다른 토폴로지에 비하여 구조적인 적응성이 매우 높다.

하이퍼큐브에는 몇가지 노드의 주소 지정방식이 있으나 현재까지는 구조적인 특성에 의하여 그레이(gray) 코드에 의한 방법이 많이 쓰이고 있다. 이때 각 노드 주소의 길이는 하이퍼큐브의 차원인 n으로 제한 되어질 수 있다. 그러므로 서로 이웃하는 두개의 PE 사이에는 그레이 코드에 의한 주소에 있어서 한 비트의 차이가 생기게 되고 이것은 인접 노드 사이의 경로선택에 사용된다. 또한 2 hop 이상 떨어져 있는 노드 사이에서도 두 노드의 주소에 의한 해밍 거리에 의하여 경로가 결정될 수 있으므로 경로선택에 의한 오버헤드가 없고 매우 간단하며 효율적인 경로선택이 가능하다. 그리고 통신 경로중 임의의 노드나 채널에 결함이 발생하여도 부가적인 채널로 다수의 우회 경로를 선택할 수 있다.

그림 1은 하이퍼큐브의 구조를 도시한 것이다.



(a)



(b)

그림 1. 하이퍼큐브의 구조
(a) 4차원 하이퍼큐브
(b) 평면상의 4차원 하이퍼큐브
Fig. 1. Hypercube Structure.
(a) 4-D Hypercube.
(b) 4-D Hypercube in Plane.

이러한 하이퍼큐브 시스템이 약결합 멀티프로세서 시스템으로서 갖는 장점중의 하나는 PE 사이를 연결하는 통신로의 고장 감내성(fault tolorency)이다. 통신로의 고장 감내성이란 임의의 두 PE를 연결하는 최단거리의 통신로가 고장에 의하여 통신기능을 수행할 수 없을 경우 최소의 Hop distance 증가에 의하여 다른 경로를 통하여 통신 할 수 있게 되는 것을 의미한다. 일반적으로 고장에는 노드의 고장과 선로의 고장이 있는데 노드의 고장에는 기계적인 고장과 통신 소프트웨어의 문제점에 의한 고장을 모두 포함시키고 선로의 고장에는 과도한 데이타의 송수신에 의한 통신병목 현상을 포함시킨다. 그러므로 다수의 PE를 장착하는 멀티프로세서 시스템에서 PE 사이에 일정하게 고정된 통신로 만을 이용하도록 한다면 시스템의 전체적인 통신 오버헤드는 매우 커질 것이다. 따라서 임의의 경로에 문제가 있을 경우 다른 우회경로를 허용하는 것이 필요하다. 임의의 선택된 두 PE의 그레이 코드화된 주소를 bitwise exclusive OR 하였을 경우 얻어지는 해밍거리는 PE 사이의 경로선택 결정에 사용된다. 여기서 말하는 해밍거리는 부하분산에서 부하분산의 거리로 대치되어질 수 있다. 만약 해밍거리에서 '1'의 갯수가 m개 발생하였을 경우 두 PE간의 최단의 통신로는 서로 중첩되지 않을 경우에는 m개, 그리고 가능한 최단 통신로의 갯수는 m! 개 가 된다. 그러나 이미 선택 되어진 통신로나 PE 사이에 어떠한 고장이 발생하여도 해밍거리(HDistance)에 의하여 나타낸 결과에서 '0'으로 표시된 비트 위치에 해당하는 임의의 한 비트를 선택하여 on/off 스윗칭(0 → 1, 1 → 0)을 하므로써 상대적인 통신로의 갯수는 해밍거리에 의하여 산출된 통신로에 해밍 거리에 의한 '0'의 갯수 만큼 각각 얻을 수 있다. 경우 1은 시작주소와 종단주소에 의한 해밍거리가 1일 경우 전체 주소의 길이 4에서 0의 갯수가 3 이므로 상대적인 우회 경로는 3개가 된다. 경우 2는 해밍거리가 2이고 0의 갯수가 2일 경우, 경로상의 노드 또는 채널상에 주어진 경로의 갯수는 2개가 된다. 이때 경로선택, 또는 노드 주소의 on/off의 순서를 임의로 할 수 있으므로 전체 우회 경로의 수는 8개 이다. 여기서 만약 하이퍼큐브의 차원이 높아지고 해밍거리의 값이 적을수록 상대적인 우회 경로의 수는 많아진다.

표 1은 하이퍼큐브에서의 상대적인 통신로 결정을 나타낸 표이다.

표 1. 하이퍼큐브에서의 상대적인 통신로 결정
(a) 경우 1 (b) 경우 2
Table 1. Relative Path Determination in Hypercube.
(a) Case 1, (b) Case 2.

상대적인 통신로 결정 시작 주소: 1100, 종단 주소: 1110, EXOR:0010							
1	1100	→	0100	→	0110	→	1110
			1st bit off				1st bit on
2	1100	→	1000	→	1010	→	1110
			2nd bit off				2nd bit on
3	1100	→	1101	→	1111	→	1110
			4th bit on				4th bit off

(a)

상대적인 통신로 결정 시작 주소 : 0100, 종단 주소 : 0010, EXOR : 0110 고정 경로 : 0100 → 0000 → 0010, 0100 → 0110 → 0010									
1	0100	→	1100	→	1000	→	0000	→	0010
			1st bit on				1st bit off		
2	0100	→	1100	→	1110	→	0110	→	0010
			1st bit on				1st bit off		
3	0100	→	0000	→	1000	→	1010	→	0010
			1st bit on				1st bit off		
4	0100	→	0110	→	1110	→	1010	→	0010
			1st bit on				1st bit off		
5	0100	→	0101	→	0001	→	0000	→	0010
			4th bit on				4th bit off		
6	0100	→	0101	→	0111	→	0110	→	0010
			4th bit on				4th bit off		
7	0100	→	0000	→	0001	→	0011	→	0010
					4th bit on				4th bit off
8	0100	→	0110	→	0111	→	0011	→	0010
			4th bit on						4th bit off

(b)

Ⅲ. 하이퍼큐브 상에서의 부하 분산

1. 하이퍼큐브 상에서의 타스크 할당

하이퍼큐브 상에서 타스크의 분산 처리시 관심있는 문제중에 하나로 이질성의 분산 처리 프로세서들을 통하여 하나의 모듈라 프로그램을 최적의 모듈로 할당하는 문제다. 만일 모듈라 프로그램의 그래프가 최적의 할당에 확실한 방법이라면 그것은 다항시간(polynomial time)에 많은 수의 프로세서들을 만들 수 있는 시스템을 통한 효과적인 동적 프로그래밍 접근법을 사용하여 최적의 할당을 발견하는 것이 가능하다. 만일 그 그래프를 어떤 트리로 나타낸다면, 최단 트리 알고리듬을 사용하여 최적의 할당을 산출할 수 있다.

이와 같이 극대의 성능으로 프로세서들이 타스크들을 할당하는 것을 부하분산(load balancing)이라 하고 부하분산은 멀티프로세서 혹은 일반적인 분산 시스템을 구현하는데 중요한 요소로 작용한다.

2. 부하분산을 위한 메시지 패싱

하이퍼큐브 시스템에서는 PE간의 통신을 위하여 메시지 패싱을 사용한다. 따라서 부하분산의 프로세스 이동시에는 IPC 오버헤드를 줄이기 위하여 충분히 간단한 형태의 메시지 패싱 프로토콜을 필요로 한다. 메시지 패싱은 기본적으로 트랜스포트 프로토콜, 제어정보 그리고 전송되어야 할 프로세스 데이타로 구성된다. 트랜스포트 계층을 위한 요소로는 PE간의 통신에 필수적인 버퍼 할당이 해당되며 프로세스 데이타 버퍼의 크기는 512 바이트로 설정하고 제어 요소로는 통신 초기화를 위한 클라이언트 PE의 선택과 확인(ack) 신호를 주고 받음으로서 데이타 송수신을 위한 초기화 및 데이타의 전송이 종료 되었을 때 확인 신호로 끝내는 일련의 과정에 해당한다. 또한 데이타는 이동 실행이 가능한 선택되어진 프로세스의 오브젝트 이미지를 클라이언트 PE의 프로세스 제어부에서 해체시켜 서버 PE로 전송하는 역할을 담당한다. 그림 2는 두 PE 사이의 부하분산의 프로세스 이동을 위한 메시지 패싱의 순서를 나타낸 것이다.

NOS의 부하분산에서는 IPC 오버헤드로 인하여 주로 인접 PE간의 프로세스 전송만을 허용하는 것이 보통이다. 그러나 하이퍼큐브 멀티프로세서의 메시지 패싱은 부하분산의 입장에서 보았을 때, 특히 server-initiative 한 방법을 적용할 경우 커넥션 오리엔티드 (connection-oriented) 형태를 기본으로 하며 부하분산 이외의 목적의 데이타 전송에서는 물론 PE의

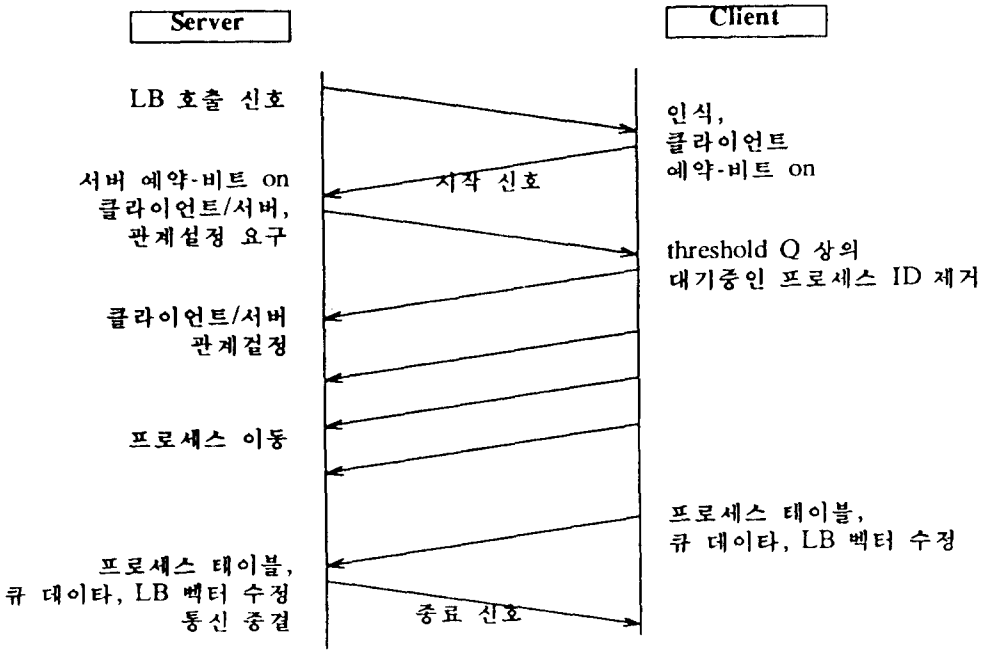


그림 2. 부하분산을 위한 메시지 패싱
Fig. 2. Message Passing for Load Balancing.

hop count가 2 이상인 경우도 허용하지만 부하분산에서 부하벡터 및 데이타 이동은 IPC 오버헤드의 감소를 위하여 최대 평균 거리를 2로 제한하고 특히 하이퍼큐브 시스템의 연구에 있어서 PE간의 거리가 멀어질수록 IPC 오버헤드가 늘어남으로 인하여 시스템의 성능저하를 가져오고 있으므로 이를 해결하기 위한 방안을 찾고 있다.

하이퍼큐브를 구성하는 PE 상호간의 통신은 light-weighted 프로토콜의 기능을 필요로 한다. 특히 부하분산에서는 방대한 양의 프로세스 데이타의 이동이 필요하기 때문에 IPC 오버헤드로 인한 영향이 매우 크다. 특히 NOS를 바탕으로 하는 분산 시스템에서는 IPC의 설계, 구현을 대체적으로 TCP/IP (Transmission Control Protocol/Internet Protocol)에 의한 소켓을 이용하여 프로세서간의 통신에 이용하였다. 그러나 멀티프로세서 시스템의 IPC를 위하여 PE의 커널에 이러한 기능을 첨가하는 것을 상당한 오버헤드를 감수하여야 하고 또한 충분히 간단한 커널 구조를 필요로 하는 각각의 PE를 상대적으로 복잡하게 한다. 따라서 기존의 인터네트워킹을 위한 통신 프로토콜을 변형하여 멀티프로세서 시스템의 PE 통신에 이용하는 것은 바람직하지 않으며 간단하고 효율적인 통신 프로토콜을 독립적으로 설계하여야 할 필요가 있다.

일반적으로 두 프로세서간에 상호 데이타를 주고 받을 때, 송신측이 수신측에 비하여 많은 오버헤드를 받는다. 그러므로 멀티프로세서 시스템에서 PE 사이의 통신 오버헤드를 줄이기 위한 적절한 메커니즘을 필요로 한다. 기존의 IPC에서는 일차적으로 송신측에서 수신측의 큐의 끝에 송신할 데이타의 정보를 붙이고 수신측에 시그널을 보낸 후 수신측에서 데이타를 접수하는 일련의 과정을 진행한다. 그러나 이때도 송신측은 패킷 framing, sequencing 등을 위한 인스트럭션의 실행을 중지할 수 없는 반면에 server-initiative 한 부하분산에서는 데이타를 송신하는 측이 과부하, 수신하는 측이 저부하 상태에 있게 되고 클라이언트/서버의 통신 설정에 관한 요구를 서버(수신측)에서 요구하게되므로 과부하상태에 있는 클라이언트(송신측)의 부하를 보다 줄일 수 있다. 따라서 클라이언트의 위치에서 보았을 때 자신의 버퍼에 대한 서버의 액세스는 DMA와 유사하다. 그러므로 부하분산 데이타에 한정 되지만 서버가 클라이언트 PE의 메모리를 직접 액세스하므로 효율적인 통신을 할 수 있다. 그림 3은 하이퍼큐브의 부하분산에서 이용되는 IPC 메카니즘을 나타낸 것이다.

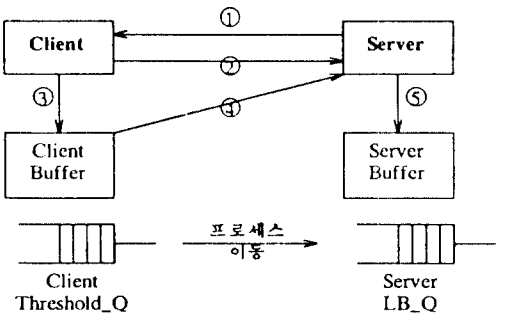


그림 3. 부하분산의 IPC 메커니즘
Fig. 3. IPC Mechanism of Load Balancing.

하이퍼큐브 시스템에서의 메시지 패싱에 있어서 구현의 복잡성을 줄이고 이식성을 높이기 위하여 실행을 PE의 역할을 담당하는 LBPE(Load Balancing PE) 프로세스 사이에서 이루어지게 한다.

동작의 순서는 다음과 같다.

① 서버 PE에 의한 부하분산 요구
② 클라이언트 PE에 의한 부하분산 수락
③ 클라이언트의 프로세스 선택기에 의한 프로세스 분류
④ 서버 PE에 의한 프로세스 데이타 이동
⑤ 서버 PE의 LB큐에 프로세스를 적재하는 일련의 과정을 수행한다.

3. 하이퍼큐브의 선행 트리 사상

사상(mapping)은 무방향 그래프와 같은 일반적인 특징을 갖는 타스크를 노드 타스크 모듈을 나타내고, 링크로 모듈 사이에 통신을 위하여 필요한 것을 나타낸다. 사상 프로세스는 두개의 개념상 구분되는 부분을 구성한 후 그 타스크 그래프를 그 크기와 토폴로지 양쪽에서 프로세서 네트워크로 부터 다르게 할 수 있다. 따라서, 사상 프로세스는 모듈들의 수 충당으로 수축성있는 타스크 그래프를 이루고 한 그래프로 타스크 그래프를 내장하는 상호연결 네트워크의 병렬 프로세서를 나타낸다. 사상 문제상 중요한 작업은 하이퍼큐브와 같이 고정된 프로세서 네트워크는 트리같은 특수한 구조와 함께 타스크 그래프의 맵으로 내장하는 함수의 구성을 수반한다.

사상 문제 형식에서 병렬 구조는 그래프 $G_H=(V_H, E_H)$로 나타내고, 여기서 V_H는 프로세서의 집합을, E_H는 통신 링크들의 집합을 나타낸다.

V_H의 한 정점이 프로세서 노드에 해당하고 한 에지 $(vi,vj)\in E_H$는 프로세서들의 쌍, $vi,vj\in V_H$ 사이에 양방향적 링크를 나타낸다.

타스크 그래프는 $G_T=(V_T,E_T)$로 나타내며, 정점의 집합 V_T는 프로세스 노드들을 나타내고 에지의 집합 E_T는 프로세스 사이에 요구된 통신을 나타낸다.

한 타스크 선행 그래프는 비순환 그래프를 관리한다. 즉 E_T의 에지들을 관리한다. 여기서 만일 $(vi, vj)\in E_T$라면 프로세스 vj가 암시하는 것은 프로세스 vi가 완전하게 될 때까지 프로세스 vj는 수행을 시작할 수 없다.

각각의 프로세스 노드는 일반적으로 어떤 데이타 저장과 어떤 출력을 계산하며 사상함수는 프로세서 집합 V_H로 프로세스 노드의 집합 V_T를 맵으로 만든다.

P_H는 G_H에 모든 경로들의 집합을 나타내고 노드 vi 와 vj 사이의 거리 $d_H(vi,vj)$는 두 노드 사이에 최단 경로의 거리이다.

G_T로부터 G_H까지 사상 $\psi : V_T \rightarrow V_H$, 프로세서들로 프로세스 노드들을 사상하며, 또한 G_H의 경로들로 G_T의 에지들 맵을 사상 ϕ로, $\phi : E_T \rightarrow P_H$를 유도 한다. 사상 $\psi : V_T \rightarrow V_H$는 G_H에서 G_T를 내장한다고 부른다. 만일 ψ 가 1 대 1 로 사상한다면, 사상의 효과 평가로 사용된 것은 팽창 코스트 δ (타스크 그래 프에 묘사된 잠재적인 통신 실현으로 요구된 통신 네트워크에 링크수)와 확장 코스트 ϵ 이다.

사상 ψ 의 팽창 코스트 $\psi : V_T \rightarrow V_H$는 G_T의 이웃하는 G_H에 최대 거리와 같이 정의된다. 즉, $\delta = \max(vi,vj) \in E_T\{d_H(\psi(vi), \psi(vj))\}$이며, 또한 확대 코스트 ϵ 는 다음과 같이 정의된다.

$$\epsilon = \frac{|V_H|}{|V_T|}$$

또 다른 측정은 프로세서 부하 λ_p(단일 프로세서로 할당된 타스크 모듈의 수)와 링크 부하 λ_l(edge 밀집)를 포함한다. 여기서 $\lambda_p=\max(v\in V_H)|\psi^{-1}(v)|$ 이다. 링크부하가 최대인 G_H에 모든 에지들을 통하여 타스크 그래프 통신의 수를 통신 네트워크의 단일 링크를 통하여 패싱을 요구한다.

가중된 타스크 선행 그래프 $G_T(V_T,E_T)$는 가중치와 함께 하나의 선행 그래프의 에지들과 프로세스 노드들로 할당된다. 즉, 거기에는 함수 $w_1 : V_T \rightarrow N$과, $w_2 : E_T \rightarrow N$ 로 나타낸다. 여기서 $w_1(vi)$는 프로세스 vi에 의해 요구된 프로세싱의 양을 나타내고, $w2(vi,vj)$는 vi 부터 vj 까지 전송된 통신의 양을 나타낸다. 그러나 그러한 정보가 없을 경우에, 즉 가중치 함수가 특별한 것이 없을 때, 우리는 모든 노드들과 에지들은 같은 가중치를 갖는다고 가정한다.(부가적으로 간소함을 위해 모두 단위 가중치를 갖는 것으로 가정한다.)

가중된 타스크 선행 그래프를 고려할 때, 병렬 구조로 사상되는 알고리듬은 복잡한 주소에 위치하게 된다. 따라서 사상의 효과가 보다 더 나은 평가로 허용된 것을 측정하여 정의할 수 있으며 어떤 것은 병행 구조로 맵되어질 때 타스크의 복잡성이 차례로 결정된다. 타스크 그래프는 자체의 고유의 복잡성으 로부터 타스크 그래프 사상의 복잡성 코스트 들을 분리하는 것을 도와주며 타스크 그래프 내에서 $P_m = T_{mp} / T_{gp}$와 $C_m = T_{mc} / T_{gc}$를 정의한다.

여기서 변수 T_{gp}는 타스크 그래프에 함축된 순수한 병렬처리 복잡성을, T_{mp}는 사상으로 부터 결과 T_{gp}에 변화를, T_{gc}는 타스크 그래프에서 통신 복잡성이고 T_{mc}는 사상으로 부터 결과 T_{gc}의 변화를 나타낸다. 단, T_{mc}는 음수일 수 있음을 주목하여 타스크 그래프의 축소가 곧 통신의 요구를 감소할 수 있게 된다.

P_m은 사상에 의한 처리 복잡성에 대한 변화로 $0\leq P_m\leq(T_s / T_{gp})$범위에서 타스크의 프로세싱 복잡성상에 사상된 대체 타스크 그래프의 효과의 명백한 비교를 제공한다. T_s는 타스크의 순차적 복잡성이다

C_m과 $C_m \geq -1$은 사상으로 부터 통신관계의 복잡성 결과 변화의 측정으로 어떤 타스크 그래프를 위하여 산출할 수도 있고, 산출된 것의 처리 복잡성과 통신 복잡성의 합으로 타스크 그래프를 위한 총 실행 복잡성 T_g를 정의에 의해 사상 하고, 유사하게 맵된 알고리듬을 위하여 총 실행 복잡성 T_p는 그것의 처리 복잡 성 T_{pp}와 그것의 통신 복잡성 T_p의 합계다.

관련된 그래프들로 부터 타스크를 위한 T_g, T_{gp}와 $T_{gc} = T_g - T_{gp}$를 발견하고 만일 그것이 타스크 그래프의 토폴로지를 정확하게 병렬 구조상에서 실행 한다면 어떤 사상을 위한 T_p, T_{PP} 그리고 $T_{pc} = T_p - T_{pp}$를 발견한다. 정의로 부터 $T_{mp} = T_{pp} - T_{gp}$와 $T_{mc} = T_{pc} - T_{gc}$이므로 이제 위에서 정의된 P_m과 C_m을 계산할 수 있다.

어떤 타스크가 정확한 타스크 그래프 위상과 함께 프로세서 네트워크상에 동일하게 내장하는데서 내장된 것은 $\delta = \epsilon = \lambda_p = \lambda_l = 1$ 과 $P_m = Cm = 0$을 갖고 사상 평가로 δ , ϵ , λ_p, λ_l, P_m 과 C_m을 측정하는데 사용하며 그 사상의 목적은 극한까지 속도를 증가하여 어떤것은 극소까지 축소된 P_m, C_m을 수반한다.

Ⅳ. 부하분산을 위한 최단 경로 할당 알고리듬

1. 알고리듬의 흐름도

그림 4는 부하분산을 위한 최단 경로 할당 알고리

듬의 흐름도를 나타낸 것이다.

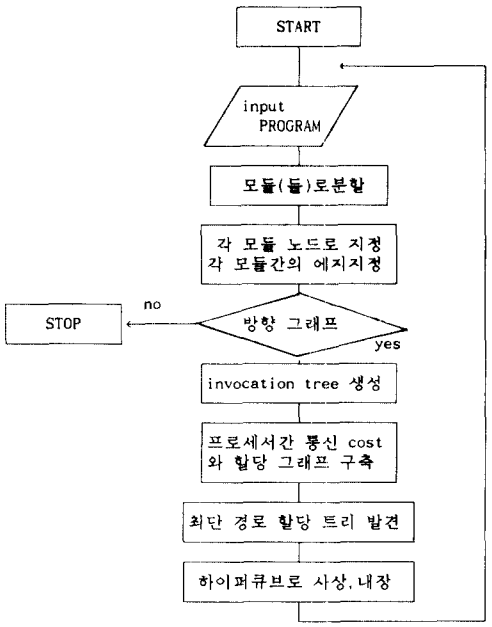


그림 4. 부하분산을 위한 최단 경로 할당 알고리듬의 흐름도

Fig. 4. Shortest Path Allocation Algorithm for Load Balancing.

2. 알고리듬의 구현

그림 5는 최단 할당 경로 알고리듬을 기술한 것이다.

```
shortest_path_allocation()
{
  a modular program to be distributed of a number of modules;
  each node represents a module;
  an edge from node i to node j and if module in existence direction;
  if (a module calls only the other modules) {
     call InvoPro;
        /* 모듈들의 연결상태로 Invocation 트리 생성 알고리듬 호출 */
     call CommAsPro;
        /* 통신 코스트와 할당그래프 결정 알고리듬 호출 */
     call ShortPro;
        /* 최단 경로 할당 트리 알고리듬 호출 */
     call MapPro;
        /* 하이퍼큐브로의 사상 알고리듬 호출 */
  }
}
```

그림 5. 최단 경로 할당 알고리듬

Fig. 5. Shortest Path Allocation Algorithm.

그림 6은 invocation tree 생성 알고리듬을 기술한 것이다.

```
InvoPro()
{
  input G=(V,E), v    /* G : 방향 그래프, v   G의 노드 */
  /* 다음의  postWORK와 함께 DFS를 사용 */
  postWORK;
  for (all edges(v,w)) {
     if (w was unmarked) {
        Depth Firth Search(G,w);
        add the edge(v,w) to T;
     }
  }
  generate of T;    /* T는 G의 DFS 트리,  T의 초기치는  empty */
}
```

그림 6. Invocation Tree 생성 알고리듬

Fig. 6. Invocation Tree Generation Algorithm.

그림 7은 통신 코스트 및 할당 그래프 결정 알고리듬을 기술한 것이다.

```
CommAsProf()
{
  input  tree.    /* invocation tree */
  n is the number of prossor;
  m is the number of module;
  dij is the total amount of data transmitted between the module i and j;
  eij is the cost of executing module i on processor j ;
  Spq(dij) is the cost of transmitting data dij between processor p & q;
  Spq is the cost of transmitting a unit amount of data ;
  Spq = Sqp;
  Spp = 0;
  m×n nodes generate for a problem invoking m modules and n processors;
  add source node s and terminal nodes; /* 할당 그래프 구축 */
  each node is labeled with a pair of numbers(i,j)
  each node represents the assignment of module I to processor j;
  if ( outdegree of nodes ) 1 )
     forknode;
  while ( node set ) 1 ) {
     weight of all edges incident on the terminal nodes = 0;
     weight of edges joining source node to node (i,j) = eij;
     weight of the edge joining node (i,q) to node (j,p)   ejp+Spq(dij)
  }
  find the minimum weight assignment tree in the assignment graph;
  /* 할당 그래프의 각각의 계층은 invocation 트리의 한 노드에 해당 */
  /* each assignment of the m modules to the n processors there
     corresponds some subset of nodes of the assignment graph */
}
```

그림 7. 통신 코스트 및 할당 그래프 결정 알고리듬

Fig. 7. Communication Cost and Graph Determination Algorithm

그림 8은 최단 경로 할당 트리 알고리듬을 기술한 것이다.

```
ShortPro()
{
  input graph;  /* 할당 그래프 */
  /* TSET is the set of all terminal nodes */
  /* FSET is the set of all forksets */
  while (ABS(TSET)) 1) {
    to each terminal node t in TSET apply procedure SHORT;
    remove t from TSET;
    for (each exposed forkset f in FSET) {
      temporarily disconnect all outgoing edges;
      create a pseudoterminal node tf;
      join all nodes in f to tf with edges that have weights equal to the
      sum of the several shortest paths to the several discarded terminal
      nodes;
      remove f from FSET;
      add tf to TSET;
    }
  }
  find the shortest path from the last terminal node to s;
  /* length of this path equals weight of shortest tree */
  reconnect all disconnected edges;
  traverse graph from s to all terminal nodes by following points set up
  by procedure SHORT;
  /* each node encountered is part of the shortest tree */
}
```

그림 8. 최단 경로 할당 트리 알고리듬

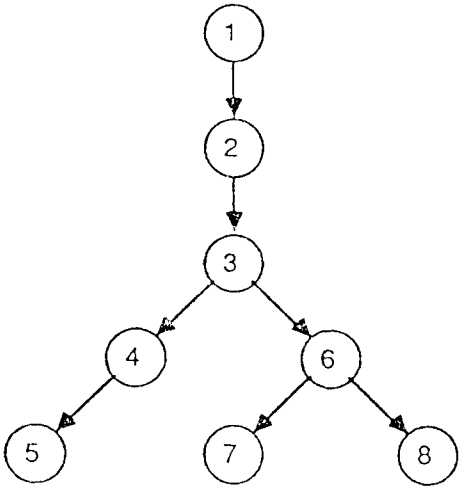
Fig. 8. Shortest Path Allocation Tree Algorithm.

그림 9는 하이퍼큐브로의 사상 알고리듬을 기술한 것이다.

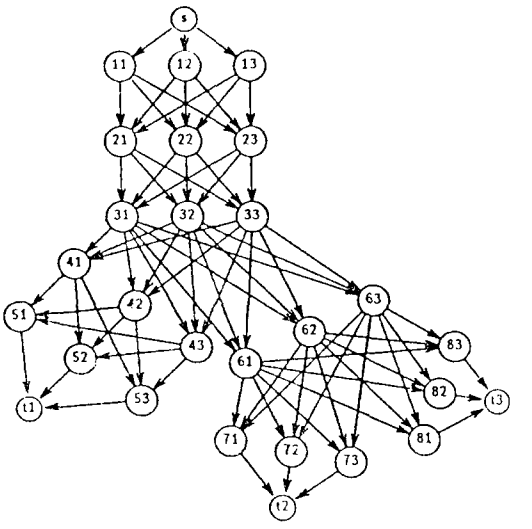
```
MapProf()
{
  input GT = (VH,EH) /*병렬 구조 그래프*/
  /* VH:set of processor(node),EH:set of communication linkedge; */
  input GT = (VT,ET) /* 타스크 그래프 */
  /* VT process node, ET edge */
  if ((Vi,Vj)∈ET) {
    process Vj cannot start till process Vi has completed;
    PH is the set of all paths in GH
    dH(vi,vj) is length of shortest path(in GH) between the two nodes;
    ψ : VT → VH;
    /* from GT to GH mapping process nodes to processors */
    φ : ET → PH
    /* maps edges in GT to paths in GH */
    if (ψ is a one to one mapping) {
      /* evaluate the efficiency of the mappings, are
         the dilation cost δ and the expansion cost */
      δ = max(Vi,Vj)∈ET[dH(ψ(Vi),ψ(Vj))];
          VH
      ε =
          VT
      /* processor load λP and link load λL */
      λP = max(Vi)[ψ-1(Vi)];
      the link load is the maximum, over all edges in GH, of the no.
      of task graph communication requirment passing through a single
      link in the communication network.
      W[Vi] = /* amount of processing required by process Vi */
      W2[ET] = /* amount of communication to be transfer from Vi to Vj */
      PM = TPP/TPP;
      /* TPP = the pure parallel processing complexity
         TMP = the change in TPP resulting from the mapping */
      CM = TMC/TPC ;
      /* TPC = the communication complexity in the task graph
         TMC = the change in TPC resulting from the mapping */
      if (TMC < 0) {
          contracting the task graph = 통신 요구 감소 */
          0 ≤ PM ≤(TPC/TPP);
          PM is change in processing complexity caused by the mapping
          /* TS = the sequential complexity of the task */
      }
    }
  }
}
```

그림 9. 사상 알고리듬
Fig. 9. Mapping Algorithm.

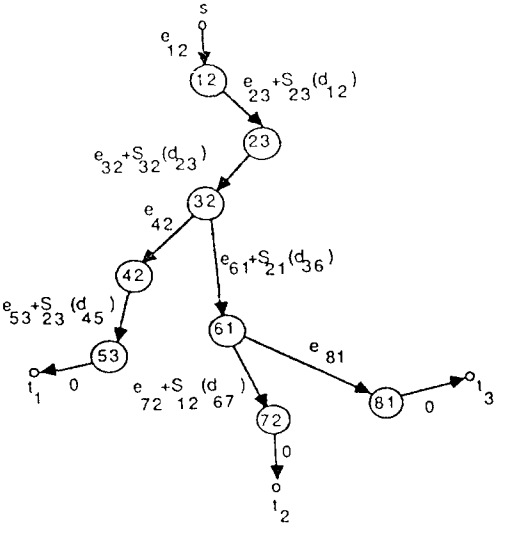
본 논문에서 제안한 알고리듬을 3차원 하이퍼큐브에 적용한 예는 다음과 같다.



(a)



(b)



(c)

그림 10. 3차원 하이퍼큐브 시스템상에서의 부하 분산
(a) 호출 트리
(b) 할당 그래프
(c) 할당 트리

Fig. 10. Load Balancing on 3-D Hypercube System.
(a) Invocation Tree,
(b) Allocation Graph,
(c) Allocation Tree.

즉, 3차원의 하이퍼큐브의 호출 트리로 변환하면 그림 10(a)와 같이 나타 낼 수 있으며, 또한 변환된 호출 트리에 의해서 할당 그래프를 결정하면 그림 10(b)와 같다. 이 할당 그래프는 3개의 프로세서 시스템(3차원 하이퍼큐브의 모델)으로 결정되어질 수 있으며, 또한 각 노드의 가중치를 결정할 수 있다. 그림 10(c)는 최단 경로 할당 트리 알고리듬에 의해서 구해진 할당 트리이다. 구해진 각각의 할당 트리의 가중치는 해당하는 모듈의 할당 코스트와 같다.

Ⅴ. 수행 결과

현재의 하이퍼큐브 시스템에서는 PE 사이의 부하를 균일하게 유지하는 부하 분산의 기능이 제공되어 있지 않기 때문에 부하분산은 전적으로 사용자 프로그램 레벨에서 시행되어 왔다. 따라서 시스템 레벨의 부하분산은 보다 효율적인 부하분산 기능을 제공할 수 있으며 사용자로 하여금 단순히 프로세스의 분할만을 생각하게 한다.

멀티 프로세서를 구성하는 PE의 갯수가 매우 많을 경우 전체 PE의 집합을 부하분산 영역으로 하면 PE 사이의 IPC 오버헤드는 증가한다. 따라서 전체 PE의 집합을 일정한 크기의 부하분산 영역으로 분할하고 그 내부에 있는 모든 PE는 일정한 시간 간격으로 프로세스 값의 교환에 의한 IPC 오버헤드가 증가하게 되고 임의의 경로를 따라서 일주하며 프로세스 값을 읽게 된다.

표 2. 알고리듬 적용 결과
Table 2. Computation Result.

과부하 PE의 갯수		실행시간
무부하분산	부하분산	
340	45	16 × 500
430	57	16 × 500
361	47	16 × 500
408	37	16 × 500
1539	186	16 × 2000

〈 평균 과부하 프로세서의 감소정도 = 87.91% 〉

본 논문에서 제안한 알고리듬을 SUN SPARC(4.3 BSD UNIX) 상에서 C언어로 실현한다. 또한 수행에서는 모듈이 단위 시간에 도착할 수 있는 수를 최대 20개로 하고 상호간 선행관계를 나타내는 invocation tree로 구축한 후 발생 분포 간격을 exp(-0.3)에서 exp(-6.0)까지 20개의 구간으로 프로세서에 할당하여 할당 그래프를 형성한 후 프로세서간 발생하는 통신코스트를 부여하고 최단 경로 할당 트리를 발견하여 하이퍼큐브로 사상하므로써 부하분산을 고려한 모듈들로 최적의 할당을 하게되며 64개의 PE를 갖고 있는 하이퍼큐브 멀티프로세서 시스템에서 전체의 영역을 각각의 부하분산 영역 단위로 분할하여 16 500번의 주행을 4회 실행한다. 여기서 시간 단위는 실제의 시스템에서 리스케쥴 시간 간격이 대응된다. 표 2는 알고리듬 적용 시키기 전후의 과부하 프로세서 감소 정도를 나타낸것이다.

하이퍼큐브 시스템내의 임의의 PE가 과부하 상태로 전이하는데 소요되는 시간을 보면 알고리듬을 적용한 후의 결과가 매우 짧은 시간을 소모하게 된다.

표 3은 시스템에 있는 임의의 PE를 선택하여 프로세스의 변화 상태를 기록 한 것이다.

표 3. 부하 분산에 의한 과부하 극복시간
Table 3. Diminution Time of Overloading by Load Balancing.

	최대	최소	평균
무부하분산	20	3	6.08
부 하 분 산	5	1	1.5

본 논문에서 제안된 알고리듬을 실행 시킨 결과 최단 경로 할당 트리를 하이퍼큐브로 사상 내장 시키므로써 통신 방법에서도 더욱 융통성 도모와 함께 타스크의 부하분산을 고려한 최적의 할당으로 수행속도의 향상을 가져옴을 볼 수 있다.

Ⅵ. 결 론

본 논문에서는 분산처리 시스템상에서 프로세서를 통하여 하나의 모듈라 프로그램을 최적으로 할당하기 위한 효과적인 동적 접근법으로 최단 경로 할당 알고리듬을 제안하였다.

한 개 이상의 PE로 구성되어 있는 분산 시스템에서 프로세서가 할당되어 실행 대기상태에 있을 때, 프로세스가 특정 PE에 과다하게 주어져서 실행되므로써 시스템 전체의 처리 효율이 떨어지는 것을 방지하기 위하여 프로세서에 할당된 모듈들로 호출트리를 생성하고 프로세서간에 초래한 통신 코스트를 가중치로 한 할당 그래프를 구축한 후 최단 할당 경로 트리를 발견하므로써 최적의 할당을 얻을 수 있었다. 한편, 실제 필요한 만큼 병행 프로세서를 설계하는데 우수한 하이퍼큐브로 사상, 내장하여 프로세서간의

통신을 메시지 패싱으로 연결하므로써 보다 더 효율적으로 수행 속도를 향상할 수 있었다.

參 考 文 獻

[1] K.Hwang and F.Brigs, Computer Architecture and Parallel Processing, McGraw-Hill, 1984

[2] Timothy C.K.Chou and J.A.Abraham, "Load Balancing in Distributed Systems", *IEEE Trans. on Software Engineering*, vol.SE-8, no.4, pp.401-412, 1982.

[3] S.H.Bokhari, "A Shortest Tree Algorithm for Optimal Assignment Across Space and Time in a Distributed Processor System", *IEEE Trans on Software Engineering*, vol. SE-7, no.6, pp.583-589, 1981.

[4] D.Towsley, "Allocating Programs Containing Branches and Loops Within a Multiple Processor System ", *IEEE Trans. on Software Engineering*, vol. SE-12, no.10, pp.1018-1024, 1986.

[5] B.Narahari,et.al., "Mapping Binary Precedence trees to Hypercubes and Meshes", Proceeding of the Second IEEE Symposium on Parallel and Distributed Processing, pp.838-841, 1990.

[6] Xiaoshu Qian and Q.Yang, "Load Balancing on Generalized Hypercubes and Mesh Multiprocessors with LAL ", PROCEEDINGS, the 11th Inter. Conference on Distributed Computing Systems, pp.838-841, 1990.

[7] N.Wirth, "Toward a Discipline of Real-time Programming ",Commun. ACM, vol.20, no.8, pp.577-583, 1977.

[8] M.A. Chughtai, "Complete Binary Spanning Trees of the Eight Nearest Neighbor Array", *IEEE Trans.on Computers*, vol.C-34,no.6, pp.547-549, 1985.

[9] P.E.Krueger, T.H.Lai and V.A. Radiya, "Processor Allocation vs. Job Scheduling on Hypercube Computers", the 11th International Conf. on Distributed Computing System, pp.394-401,1991.

著 者 紹 介

李 喆 源 (正會員) 第 28卷 B編 6號 參照
현재 한양대학교 대학원 박사과정 재학 중

林 寅 七 (正會員) 第 28卷 4號 參照
현재 한양대학교 전자공학과 교수