# 반복적인 부하 이동에 의한 휴리스틱 부하 평형 알고리즘

송 의 석[†] · 오 하 령[††] · 성 영 락[†††]


## 요 약

본 논문에서는 다중 프로세서 시스템을 위한 휴리스틱 부하 평형 알고리즘을 제안한다. 제안 알고리즘은 부하이동을 여러 링크로 분산시켜 사용하지 않는 링크의 수를 최소화시키고 그에 따라 통신비용을 감소시킨다. 각각의 프로세서는 모든 이웃한 프로세서에게 단위부하를 보내거나 받는 과정을 반복적으로 시도한다. 그러나 실제의 부하 이동은 모든 부하평형 계산이 이루어진 후 수행된다. 이는 불필요한 부하 이동을 최소화시킨다. 제안된 알고리즘은 약간의 수정만으로 다양한 연결 구조를 갖는 다중 프로세서 시스템에 적용할 수 있다. 본 논문에서는 하이퍼큐브 구조와 메쉬 구조에 제안 알고리즘을 적용해 보았다. 알고리즘의 성능평가를 위하여 모의실험을 하였다. 제안된 알고리즘과 잘 알려진 두 가지 종류의 알고리즘의 성능을 비교하였다. 그 결과 제안된 알고리즘은 모든 경우에서 부하평형의 목적을 달성하였다. 또한 기존의 알고리즘과 비교하여 하이퍼큐브 구조에서는 통신비용을 70~90% 감소시켰다. 메쉬 구조에서도 통신비용은 약 75% 감소되었다.


# A Heuristic Load Balancing Algorithm by using Iterative Load Transfer

Eui-Seok Song[†] · Ha-Ryung Oh[††] · Yeong-Rak Seong[†††]


## ABSTRACT

This paper proposes a heuristic load balancing algorithm for multiprocessor systems. The algorithm minimizes the number of idle links to distribute load traffic and reduces its communication cost. Each processor iteratively tries to transfer unit load to/from every neighbor processors. However, real load transfer is collectively done after complete load traffic calculation to minimize useless traffic. The proposed algorithm can be employed in various interconnection topologies with slight modifications. In this paper, it is applied to both hypercube and mesh environments. For performance evaluation, simulation studies are performed. The performance of proposed algorithm is compared to those of two well-known algorithms. The results show that the proposed algorithm always balances the loads perfectly. Furthermore, it reduces the communication costs by 70~90% in the hypercube ; and it reduces the cost by 75% in the mesh, compared to existing algorithms.


**키워드 : 다중 프로세서 시스템(Multiprocessor System), 부하 평형(Load Balancing), 하이퍼큐브(Hypercube), 메쉬(Mesh)**

## 1. 서 론

최근 컴퓨터 시스템의 비약적인 발전에는 반도체 기술의 발달과 중앙처리장치의 성능향상이 크게 기여하고 있다. 그러나 컴퓨터의 고속의 연산 처리를 이용하는 분야(매트릭스 연산, 고차 방정식의 풀이, 기상 예측프로그램 등)나 데이터 간의 독립적인 성질을 갖는 병렬 데이터 조작을 위한 문제(렌더링 문제, 시뮬레이션 모델링을 통한 과학적 예측문제 등)는 아직도 많은 시간이 소요된다[1]. 이에 따라 컴퓨터 시스템에 여러 개의 프로세서를 설치한 다중 프로세서 시스템을 이용하여 연산의 결과를 좀 더 빠르게 얻어내려는 연구가 활발하다. 다중 프로세서 시스템에서는 개개의 프로세서가 처리하여야 할 작업을 균등하게 분배하여 휴지 상태의 프로세서가 최소화되도록 하는 것이 시스템의 전체 성능향상에 중요한 역할을 한다. 각각의 프로세서가 처리하여야 할 작업을 부하(load)라 부른다. 따라서 시스템 내의 부하 분포가 불균형한 상태가 되면 부하를 적절히 재분배하여 한 프로세서에 부하가 집중되지 않도록 하거나 쉬고 있는 프로세서가 발생하지 않도록 해 주어야 한다. 이러한 작업을 부하 평형(load balancing)이라 한다. 작업 실행 이전에 부하 평형을 수행하거나 예측 가능한 시스템 구조(하이퍼큐브, 메쉬 등)를 가지고 있는 경우에는 정적 부하평형이라고 부른다. 반면에 문제 해결을 위한 연산 중에 부하 평형을 수행하거나, 예측 불가능한 구조를 가지는 시스템에서는 동적 부하 평형이라고 부른다. 부하평형에서 특히 중요한 요소 중 하나는 부하 재분배에 소요되는 통신비용을 최소화하는 것이다[1-3].

일반적으로 기존에 제안된 부하 평형 기법들은 크게 두 단계로 수행된다. 전 단계에서는 상태정보를 주고받아 모든 프로세서들이 부하의 분포 상태를 알아내며, 기법에 따라 통신비용이 무시될 수 있다. 또한 후 단계에서는 이동할 부하

† 준 회 원 : 국민대학교 일반대학원 전자공학전공
†† 정 회 원 : 국민대학교 전자정보통신공학부 교수
††† 종신회원 : 국민대학교 전자정보통신공학부 교수
논문접수 : 2004년 8월 11일, 심사완료 : 2004년 10월 7일

의 양을 계산하고 부하 재분배를 수행한다. 이때 적용하는 방법에 따라 통신비용이 달라진다[3-6]. 부하 평형은 일반적으로 두 가지 관점에서 최적화 될 수 있다. 첫 번째는 부하 평형에 소요되는 시간을 최소화하는 것이고, 두 번째는 통신량(payload)을 최소화하는 것이다. 전자의 경우에는 각각의 링크로 이동되는 부하의 수 중 최대값으로 통신비용을 정의하며[1] 본 논문에서는 이를 채용한다. 후자의 경우에는 이동되는 부하의 총수로 정의한다[3].

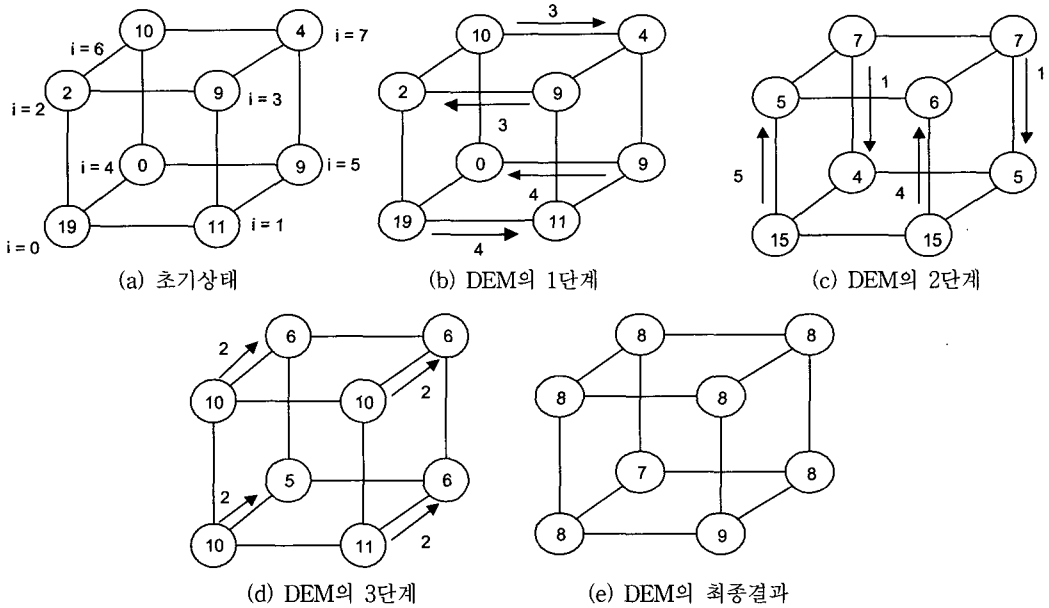
본 논문에서는 정적 휴리스틱(heuristic) 부하 평형 방법을 제안한다. 기존의 알고리즘들은, 하이퍼큐브 구조에서는 차원별로[3, 7], 그리고 메쉬 구조의 경우에서는 행, 열 별로 부하 평형을 수행한다[3]. 이 때 부하 이동이 없는 링크들이 생기면 통신시간이 커지는 요인이 된다. 본 논문에서 제안하는 알고리즘에서는 부하의 이동이 없는 링크의 수를 줄여 부하 이동을 여러 링크에 분산시키면 전체적인 통신시간을 줄일 수 있다는 점에 착안하였다. 이를 위하여 각 프로세서는 연결된 모든 링크에 대해서 단위 부하를 반복하여 이동시키는 기본기법과 부하의 분포가 한 노드에 집중되어 있을 때 한번에 여러 개의 단위부하를 이동시키는 고속이동기법을 이용한다. 각 프로세서는 자신이 보내거나 받아야할 부하의 수를 계산하게 되며 결과가 완성되면 일괄적으로 부하 평형을 실행한다. 제안된 알고리즘은 프로세서 사이의 연결 구조에 상관없이 같은 방법으로 부하 평형을 실행한다. 본 논문에서는 하이퍼큐브와 2차원 메쉬 구조를 예로 하여 실험하였다. 알고리즘 수행의 전제 조건으로 각 부하의 실행 시간과 크기가 동일하다고 가정한다. 즉 부하의 크기는 항상 정수로 주어지며, 이후로 단위 부하를 부하라고 부르겠다. 또한 프로세서의 각 통신 링크는 독립적으로 동작이 가능한 것으로 가정한다. 그러므로 프로세서에 연결된 링크의 수가 n이면, 그 프로세서는 n개의 이웃한 프로세서들과 동시에 통신할 수 있다.

논문의 구성은 다음과 같다. 2장에서는 기존의 연구를 살펴보고, 3장에서는 제안한 알고리즘을 설명하며, 4장에서는 실험을 통하여 결과를 분석하고, 5장에서는 결론을 내린다.

## 2. 기존의 연구

### 2.1 하이퍼큐브 구조에서의 부하 평형 알고리즘

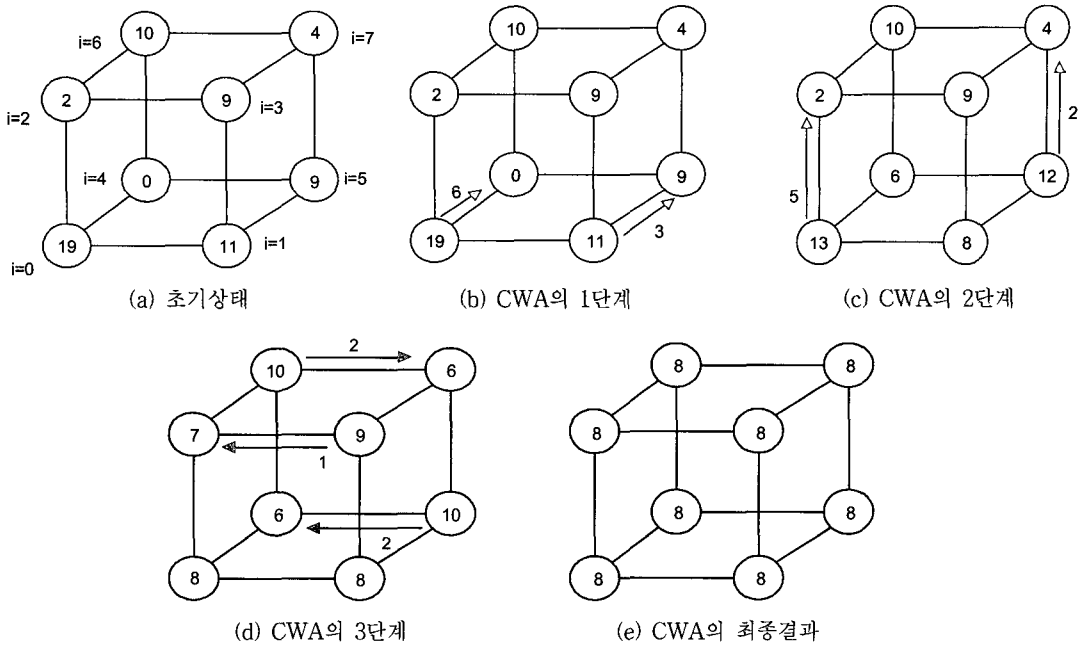
하이퍼큐브 구조에서의 부하 평형 기법에 대하여 많은 연구가 발표되었다. 이 중에서 가장 대표적인 예로 DEM과 CWA를 들 수 있다. DEM(Dimension Exchange Method)[3, 7-9]은 먼저 좁은 영역에서 부하 평형을 실시하고, 이를 점차 넓은 영역으로 확대 적용하여, 전체 시스템을 부하 평형에 도달하도록 한다. (그림 1)은 DEM 알고리즘을 수행한 예이다. (그림 1)(a)에서 0번 프로세서와 1번 프로세서 사이의 부하의 차이로부터 이동되어야 할 부하를 계산하고 ($\lfloor (19-11)/2 \rfloor = 4$) 두 프로세서 사이의 경로를 통해 이동한다. 같은 방법을 (2, 3), (4, 5), (6, 7) 프로세서들 사이에도 적용하면 (그림 1)(c)의 결과가 나타난다. 이제 단계에 따라 차원을 바꾸어 가며 동일한 과정을 수행한 결과가 각각 (그림 1)(d)와 (그림 1)(e)이다. DEM은 부하의 전체적인 분포에 대해 조사하므로 구현이 간단하지만, (그림 1)(e)에서 나타난 바와 같이 불균등한 부하의 분포가 발생할 수도 있다. [3]의 연구에 따르면 DEM 수행 후의 부하 차이는 최대 하이퍼큐브의 차원 수만큼 될 수 있다.




(a) 초기상태 (b) DEM의 1단계 (c) DEM의 2단계

(d) DEM의 3단계 (e) DEM의 최종결과

(그림 1) DEM 알고리즘의 수행 예제




(a) 초기상태 (b) CWA의 1단계 (c) CWA의 2단계

(d) CWA의 3단계 (e) CWA의 최종결과

(그림 2) CWA의 수행 예제

CWA(Cube Walking Algorithm)[3, 10]는 DEM 보다 조금 더 복잡한 계산 과정을 거친다. (그림 2)는 CWA의 수행 예이다. 초기상태에서 i = 0, 1, 2, 3으로 구성된 면과 i = 4, 5, 6, 7로 구성된 면사이의 부하 차이를 계산하고 부하를 이동한다. (그림 3)(a)에서 두 면의 총 부하가 각각 41과 23이므로 9개의 부하를 전달하게 된다. 각 프로세서에서 몇 개의 부하를 전달해야 하는 지를 계산하는 과정은 설명하기에 매우 복잡하므로 [3]을 참고하기 바란다. 이 경우에는 0번 프로세서는 6개의 부하를, 1번 프로세서는 3개의 부하를 각각 4번과 5번 프로세서로 보낸다. (그림 2)(c)가 그 결과이다. 그 다음에 차원을 바꾸고 같은 계산 과정을 반복하면 (그림 2)(e)의 결과를 나타낸다. 이와 같이 CWA의 경우에는 최종 결과가 부하 평형상태에 도달하게 된다.

DEM과 CWA는 알고리즘에 의하여 이동시킬 부하를 계산하고 해당 차원에 대하여 실제로 부하이동이 발생한다. 모든 부하이동이 완료되면 다음 차원에 대하여 이동시킬 부하를 계산하고 부하의 이동이 발생하여 계산과 이동이 반복적으로 차원만큼 수행한다. 이 과정에서 알고리즘에 의한 이동 부하의 계산은 짧은 시간에 수행되지만 실제 부하 이동에 소요되는 시간은 부하의 크기나 시스템의 구성에 따라 알고리즘 수행시간 보다 더 많은 시간이 소요되어 부하 평형에 도달하기까지의 시간은 부하의 이동시간이 큰 영향을 준다.

CWA는 매 단계마다 부하를 이동시키고 변화된 값을 이용하여 재 계산하는 과정을 반복하는데 CWA를 개선시킨 [1]에서는 단계별 이동을 수행하지 않고 중간 단계에서의 결과를 저장하였다가 최종단계에서 동시에 부하를 이동시키는 기법(overlapping)과 전송에 필요한 부하가 충분하지 않을 때 파이프라인 기법까지 적용한 기법(overlapping & pipelining)을 제안하여 CWA의 성능을 개선하였다.

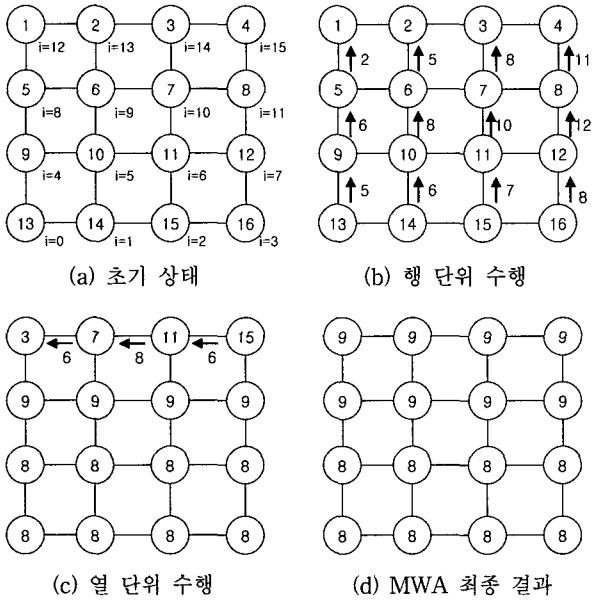
DEM과 CWA의 성능을 통신비용의 측면에서 비교해 보자. 수정 CWA 기법[1]을 적용한다면 DEM의 통신비용은 5 (= max(4, 5, 2))이고, CWA의 통신비용은 6(= max(6, 5, 2))이다. 또한 전체 이동 부하는 DEM이 33이고, CWA는 21이다. 이렇게 DEM이 전체 이동 부하가 많지만 통신비용이 줄어드는 이유는 DEM이 부하 이동에 많은 수의 링크들을 사용하기 때문이다. 본 예제의 경우에도 DEM은 12개의 링크를 사용하지만 CWA에서는 7개의 링크만이 부하 평형에 사용되었다. 그러나 앞에서도 언급한 것처럼 DEM은 완전한 부하 평형에 도달하지 못하는 경우가 발생할 수 있다는 단점이 있다.

### 2.2 메쉬 구조에서의 부하 평형 알고리즘

메쉬 구조의 다중 프로세서 시스템에서의 부하 평형 알고리즘에 대해서도 많은 연구가 발표되었다[3, 11, 12]. 이 중에서 가장 대표적인 알고리즘으로는 MWA(Mesh Working Algorithm)[3]를 들 수 있다. MWA는 행 단위로 부하를 이동한 다음 열 단위로 부하 평형을 한다. (그림 3)은 간단한 수행 예이다. 1단계에서는 행 단위의 부하평형을 한다. 초기상태에서 i = 0, 1, 2, 3 행의 부하가 58이므로 행에 대한 균등부하(32)를 초과하는 부하를 다음 행으로 전달한다. 이 때 각 프로세서의 이동 부하의 계산 방법은 [3]을 참고하라. 알고리즘에 의하면 각각 0번부터 3번 프로세서는 5, 6, 7, 8의 부하를 이동시킨다. 다음 행인 i = 4, 5, 6, 7은 부하가 68이므로 초과되는 36개의 부하를 다음 행으로 전달한다. 같은 방법으로 남은 행에 대하여 계산하면 (그림 3)(c)의 결과를 얻

는다. 그런 다음 이 과정을 열에 대해서도 반복하면 (그림 3)(d)의 최종 결과를 얻는다.

MWA의 경우에도 [1]에서는 동일한 기법으로 MWA의 성능을 개선하였다. [1]의 기법을 이용하여 통신비용을 계산해 보면 12(= max(12, 8))가 된다. 또한 이동된 총 부하는 108이다. MWA의 경우도 DEM, CWA와 같이 부하 평형에 사용되지 않는 링크들이 발생한다.




(그림 3) MWA의 수행 예제

## 3. 제안 알고리즘

제안알고리즘은 세 개의 부분으로 구성되어진다. 먼저 초기조건 수집을 위한 단계가 수행되어 작업집합의 평균부하를 계산하여 각 노드의 목표부하를 결정한다. 초기조건의 수집이 완료되면 고속기법을 이용하여 전처리과정을 수행하는데 부하의 분포상황에 따라서 수행이 제외될 수 있다. 마지막 단계에서 기본기법을 이용하여 부하평형을 완료한다. 본 논문에서는 제안한 휴리스틱 알고리즘의 기초가 되는 기본기법부터 다음의 절에서 설명한다.

### 3.1 기본 기법

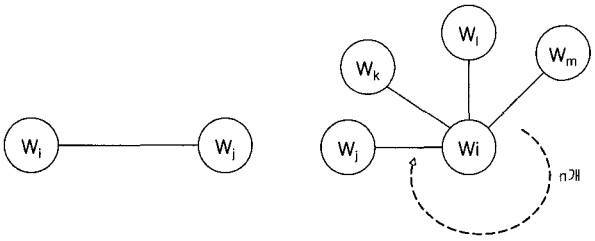
앞서 설명한 바와 같이 CWA와 MWA에서는 프로세서에 연결된 링크들 중에서 부하가 이동하지 않는 링크가 발생한다. 그러므로 부하의 이동이 분산되지 않아 부하평형에 소요되는 시간이 길어진다. 또한 DEM의 경우에는 완전한 부하 평형에 실패하는 경우도 있다. 본 논문에서는 성공적으로 부하 평형을 수행하면서, 그 과정에서 부하의 이동이 없는 링크를 줄여 전체 통신비용을 줄이는 휴리스틱 알고리즘을 제안한다. 제안하는 부하 평형 알고리즘에서는 각 프로세서에서 연결된 모든 링크를 이용하여 최대 하나의 부하를 이동시키는 것을 반복하는 방식으로 이동 부하를 계산한 다음에 일괄적으로 이동하는 방법을 취한다.

제안된 부하 평형 알고리즘을 위해서는 초기 조건 수집 단계가 필요하다. 초기 조건 수집 단계에서는 전체 시스템의 총 부하를 계산하고, 이를 프로세서의 수로 나누어 평균 부하를 산출한다. 이 평균 부하는 각 프로세서의 목표 부하가 된다. 단 본 논문에서는 부하가 정수로 주어지는 것으로 가정하였으므로, 일부 프로세서의 목표 부하는 나머지 프로세서들의 목표 부하에 비하여 1만큼 클 수도 있다.

초기 조건 수집이 완료되면 여러 단계에 걸쳐서 부하 평형을 반복한다. 각 단계마다 각각의 프로세서는 자신의 현재 부하와 목표 부하를 비교하여 동작 모드를 결정한다. 즉 현재 부하가 목표 부하보다 많으면 소스모드로, 같으면 중계모드로, 그리고 적으면 싱크모드로 동작한다. 동작 모드가 결정되면, 각 프로세서는 자신의 이웃 프로세서의 부하를 검사하고 자신의 부하와 비교한다. 이때 이웃 프로세서들을 검사하는 순서는 단계별로 라운드로빈(*round robin*)한다. 즉, 어떤 단계에서 a번째 이웃 프로세서를 검사하였다면, 다음 단계에서는 a+1번째 이웃 프로세서를 검사한다. 또한 한 단계에서 한 링크로 하나 이상의 부하가 이동하는 것이 방지하기 위하여, 각 링크에 그 링크가 사용되었음을 알리는 플래그를 둔다. 이 플래그들은 각 단계가 시작되기 전에 모두 사용 가능한 상태로 설정된다. 그런 다음 알고리즘이 진행되면서 어떤 링크를 통해 부하의 이동이 발생하면 사용 불가 상태로 설정된다. 어떤 프로세서가 이웃 프로세서들을 검사하는 도중에 사용할 수 없는 링크를 발견하면, 그 링크를 통하여 연결된 이웃 프로세서와의 비교 및 이동과 관련된 계산은 금지된다.

비교 결과에 대한 처리는 프로세서의 동작모드에 따라 다르다. (그림 4)(a)을 살펴보자. 두개의 프로세서가 하나의 링크를 가지고 연결되어있고 두 프로세서의 부하가 균등하지 않다면 연결된 링크를 통하여 부하가 많은 곳에서 적은 곳으로 이동을 시켜 부하 평형 상태로 만들 수 있다. 이 때, $W_i > W_j$ 이면 $W_i$는 $W_i - W_{avg}$ 만큼의 부하를 링크를 통하여 이동시키게 될 것이다. 링크가 하나밖에 없다면 부하를 반복적으로 이동시켜 부하평형에 도달할 수 있다. 그러나 (그림 4)(b)와 같이 연결된 링크의 개수가 n개 있다면 각 링크를 통하여 부하를 이동시키는 것이 부하이동에 소요되는 시간을 감소시킬 수 있다. 그러므로 (그림 4)(b)에서 프로세서 i가 소스모드로 판단되었다면 프로세서 i는 자신보다 부하가 적은 이웃 프로세서에게 하나의 부하를 전달한다. 프로세서 i가 싱크모드로 판단되었다면 소스모드와 반대로 자신보다 부하가 큰 모든 이웃 프로세서로부터 각각 하나의 부하를 가져온다. 프로세서 i가 목표부하와 일치되어 더 이상 부하의 이동이 필요하지 않은 상태라면 중계모드로 판단하여 이웃한 프로세서들 중에서 목표 부하보다 현재 부하가 많은

프로세서와 적은 프로세서를 찾아, 많은 프로세서에서 적은 프로세서로 하나의 부하를 이동시킨다. 이렇게 하면 자신의 부하를 그대로 유지하면서 자신을 통해 연결되는 두 프로세서 사이의 부하를 이동할 수 있다.




(a) 기본모델 1 (b) 기본모델 2

(그림 4) 프로세서 기본모델

이러한 알고리즘을 모든 프로세서에 차례대로 적용하면 한 단계가 끝난다. 한 단계가 끝나면 모든 링크의 플래그를 지워 다시 다음 단계에서 알고리즘이 반복될 수 있도록 한다. 그리고 다음 단계의 시작 프로세서는 전 단계의 시작 프로세서에 +1을 한 프로세서부터 알고리즘을 수행한다. 즉 어떤 단계에서 i번째 프로세서부터 알고리즘을 적용하였다면 다음 단계에서는 i+1번째 프로세서에서부터 알고리즘을 적용한다. 그런데 앞의 설명에서의 부하의 이동은 가상적인 것으로, 실제 부하의 이동은 부하 평형이 완료될 때까지 연기된다. 즉, 중간 단계에서는 부하가 이동되는 것으로 가정하면서 알고리즘을 반복하고, 가상적으로 부하 평형이 이루어져 알고리즘이 종료 된 다음에 일괄적으로 실제 부하를 이동한다. 이렇게 함으로서 링크에 대하여 이동 부하를 판단하여 통신비용을 줄일 수 있다.

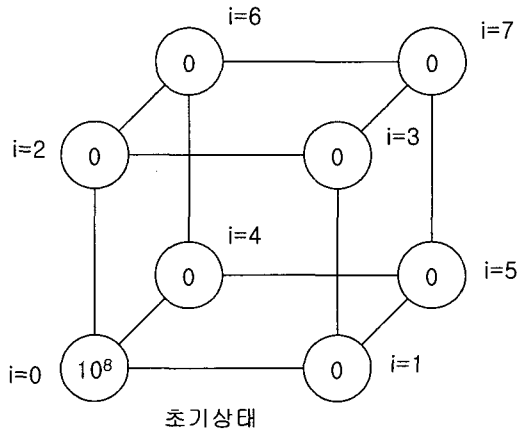
(그림 5)는 제안 알고리즘의 기본기법의 의사 코드를 기술한 것이다. 주 알고리즘 LoadBalance( )가 수행되면 초기 조건을 수집하여 평균 부하를 계산하고((그림 5)의 1) 부하 평형을 이룰 때까지 모든 프로세서들에서의 부하 이동을 차례대로 수행한다((그림 5)의 5-16). 이 과정에서 각 프로세서는 소스모드(SourceMode( )), 싱크모드(SinkMode( )), 중계모드(RelayMode( ))로 구분되어 계산을 수행한다. 알고리즘에서 LC는 반복된 횟수를 저장하는 변수로서 프로세서들이 이웃들을 검사할 때 라운드로빈 방식으로 검사 순서를 바꾸기 위해 사용된다.

```
// Wi : 프로세서 i의 작업량
// Wq : 목표 작업량
// Wi,j : 프로세서 i의 j번째 이웃 프로세서의 작업량
// Li,j : 프로세서 i와 j번째 링크의 사용 여부
// LC : Loop 수행 회수 및 첫 번째 이동 링크 선택
 1 : Wq = CalcAvgWorkLoad( ) ;
 2 : LoadBalance( )
 3 : {
 4 :  LC = 0 ;
 5 :  while (workload isn't evenly distributed) {
 6 :     for (i = 0 ; i < number of node ; i++)
 7 :        v = (LC + i) % number of node
 8 :        if (Wv > Wq)
 9 :           SourceMode( ) ;
10 :        else if (Wv < Wq)
11 :           SinkMode( ) ;
12 :        else if (Wv == Wq)
13 :           RelayMode( ) ;
14 :     ClearLinkFlags( ) ;
15 :     LC++ ;
16 :  }
17 : }
18 : SourceMode( )
19 : {
20 :  for (p = 0 ; p < number of links ; p++) {
21 :     v = (LC + p) % number of links
22 :     if ( ! Li,v && (Wi > Wi,v)) {
23 :        Wi = Wi - unit load ;
24 :        Wi,v = Wi,v + unit load ;
25 :        Li,v = TRUE ;
26 :     }
27 :  }
28 : }
29 : SinkMode( )
30 : {
31 :  for (p = 0 ; p < number of links ; p++) {
32 :     v = (LC + p) % number of links
33 :     if ( ! Li,v && (Wi < Wi,v)) {
34 :        Wi = Wi + unit load ;
35 :        Wi,v = Wi,v - unit load ;
36 :        Li,v = TRUE ;
37 :     }
38 :  }
39 : }
40 : RelayMode( )
41 : {
42 :  BIG = SMA = NULL ;
43 :  for (p = 0 ; p < number of links ; p++)
44 :     if ( !(Li,p)) {
45 :        if ( Wp > Wi )
46 :           BIG = Wp ;
47 :        else ( Wp < Wi )
48 :           SMA = Wp ;
49 :        if ( BIG && SMA ) {
50 :           BIG = BIG - unit load ;
51 :           SMA = SMA + unit load ;
52 :           LBIG = Lsma = TRUE ;
53 :           BIG = SMA = NULL ;
54 :        }
55 :     }
56 : }
```

(그림 5) 제안 부하 평형 알고리즘의 기본기법의 의사코드

### 3.2 고속기법

3.1의 기본 방법은 매 단계마다 이웃한 링크와 한 개의 부하만을 주고받는다. 그러므로 인하여 어떤 특정노드에 많은 부하가 집중되어 있다면 부하 평형의 단계까지 많은 반복 수행이 요구된다. (그림 6)의 경우를 보자. 노드 i = 0에 1억 개의 부하가 집중되어 있다. 이 때 평균 부하는 12,500,000이 되어 노드 i = 0에서는 한 링크에 대하여 근사적으로 판단하여보면 (100,000,00 - 12,500,000)/3 = 29,166,667개 정도의 부하가 이동하여야 한다. 또한 근사적으로 제안 알고리즘은 이 회수만큼 반복 수행하여야 하므로 결과를 얻기까지 많은 시간이 소요될 수 있다. (그림 6)에서는 3차원 하이퍼큐브를 보였는데 차원을 확장하여 10차 하이퍼큐브의 특정 노드에 모든 부하가 집중되어 있다면 더욱더 많은 알고리즘 수행 시간이 요구된다. 이 때에는 초기 부하의 분포상태를 고려하여 부하의 이동 개수를 한 개로 제한하지 않고 링크 당 여러 개의 부하를 이동시키도록 한다. 이 때 제안알고리즘에서는 두 가지 요소를 고려하여 부하 평형의 중간 단계인 고속기법을 수행할 수 있다. 첫 번째 요소는 각 노드에서 현재 부하와 목표부하의 차이를 나타내는 불균등 부하 요소(imbalance_factor)이고 두 번째는 링크당 보내고자 하는 부하의 개수(transfer_load)이다. 고속기법에서는 먼저 모든 노드들을 방문하면서 각 노드가 불균등 부하 요소의 범위에 들어오는지 확인한다. 만약 불균등 부하 요소의 값보다 작으면 부하 이동을 위한 어떤 동작도 하지 않는다. 그러나 불균등 부하 요소의 값보다 크면 이웃 노드의 상황을 판단하여 설정된 링크당 부하를 보내거나 받는다. 알고리즘 수행 노드 i가 소스모드라면 이웃 노드가 싱크모드이거나 보내고자 하는 부하를 받았을 때 노드 i보다 크지 않은 이웃 노드들에게만 부하를 보낸다. 반대로 노드 i가 싱크모드라면 이웃 노드가 소스모드이거나 이웃 노드로부터 부하를 받았을 때 이웃 노드가 노드 i보다 작지 않은 이웃 노드들로부터 부하를 가지고 온다. 고속기법을 적용한 후에는 모든 노드의 부하의 차이는 평균 부하에 대하여 불균등 부하 요소 이내로 조정된다. 이러한 상태에서 다시 기본기법을 적용하여 부하 평형을 수행하면 알고리즘 수행의 반복회수 및 수행시간을 줄일 수 있다.




(그림 6) 부하가 한 노드에 집중된 경우

(그림 7)은 고속기법을 적용한 의사 코드이다.

(그림 7)의 기법은 (그림 5)의 기본 기법에서 1행과 2행 사이에서 수행하면 설정된 불균등 부하의 수에 따라서 수행 여부를 판단할 수 있다. 즉 초기의 부하 상태가 설정된 불균등 부하의 수보다 작으면 고속 기법은 한번도 수행되지 않고 기본기법만으로 부하 평형을 수행할 수 있다. (그림 7)의 행 4, 12, 23, 36는 고속기법의 계속 적용여부를 판단하는 플래그로 이용한다.

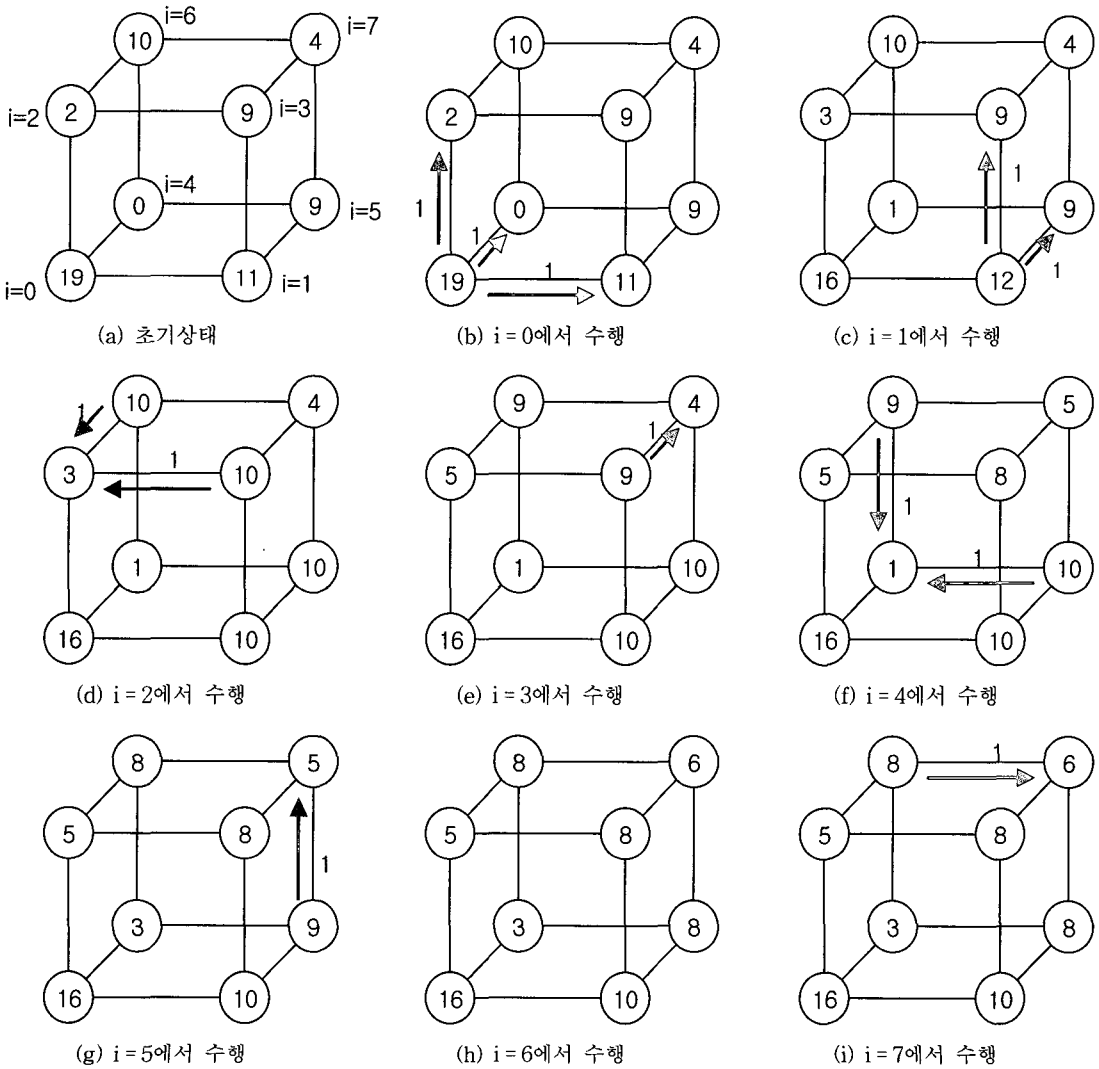
```
// IMBALANCE_FACTOR 목표부하와 현재부하의 차
// TRANSFER_LOAD 보내거나 받을 부하의 수

 1 : LoadBalanceExpress( )
 2 : { LC = 0 ;
 3 :   do {
 4 :     flag = FALSE ;
 5 :     for ( i = 0 ; i < number of node ; i++ )
 6 :       if ( | Wi - Wq | > IMBALANCE_FACTOR )
 7 :         if ( Wi > Wq )
 8 :             SourceModeEx( )
 9 :         else if ( Wv < Wq )
10 :             SinkModeEx( ) ;
11 :     ClearLinkFlags( ) ; LC++ ;
12 :   } while (flag) ;
13 : }
14 : SourceModeEx( )
15 : {
16 :   for ( p = 0 ; p < number of links ; p++ ) {
17 :     v = (LC + p) % number of links
18 :     if ( ! Li,v )
19 :       if ( (Wq > Wi,v ) && (Wi - Wi,v ) > TRANSFER_LOAD){
20 :         Wi = Wi - TRANSFER_LOAD ;
21 :         Wi,v = Wi,v + TRANSFER_LOAD ;
22 :         Li,v = TRUE ;
23 :         flag = TRUE ;
24 :       }
25 :   }
26 : }
27 : SinkModeEx( )
28 : {
29 :   for ( p = 0 ; p < number of links ; p++) {
30 :     v = (LC + p) % number of links
31 :     if ( ! Li,v )
32 :       if ( (Wq < Wi,v) && ( Wi,v - Wi ) > TRANSFER_LOAD){
33 :         Wi = Wi + TRANSFER_LOAD ;
34 :         Wi,v = Wi,v - TRANSFER_LOAD ;
35 :         Li,v = TRUE ;
36 :         flag = TRUE ;
37 :       }
38 :   }
39 : }
```

(그림 7) 제안 부하 평형 알고리즘의 고속기법의 의사코드

### 3.3 하이퍼큐브 구조에서의 부하 평형

(그림 8)는 (그림 1)과 같은 조건에 대하여 제안 알고리즘의 기본기법을 한번 수행한 예이다. 전체의 총 부하는 64이고 8개의 프로세서가 있으므로 평균 부하는 8이다. 1단계에서 0번 프로세서는 소스모드로 동작한다. 그러므로 1, 2, 4의 순서로 이웃 프로세서들의 부하를 검사하고, 모든 이웃 프로세서들의 부하가 자신보다 적으므로 각각 하나의 부하를 보낸다((그림 8)(b)). 다음은 1번 프로세서의 차례가 된다. 1번 프로세서도 목표 부하보다 많은 부하를 가지고 있으므로 소스모드이다. 0, 3, 5번의 순서로 이웃들을 검사한다. 그러나 0번 프로세서와의 링크는 조금 전의 부하 이동에 의해서 사용 불가능한 상태이므로, 3, 5번 프로세서들과의 비교만 이루어진다. 그 결과 두 프로세서들로 각각 하나의 부하를 보낸다((그림 8)(c)). 2번 프로세서는 싱크모드로 0, 3, 6번의 순서로 이웃 프로세서들을 검사한다. 그 중에서 0번 프로세서는 제외되며, 3, 6번 프로세서들이 모두 자신보다 부하가 많으므로, 그들로부터 하나씩의 부하를 가져온다((그림 8)(d)). 이 알고리즘을 남은 프로세서들에 순서대로 적용하면 (그림 8)(i)의 결과를 얻을 수 있다.

(그림 9)는 각 단계별 알고리즘의 계산 결과이다. 이 예에서는 5단계만에 부하 평형이 완료된다. 그런데 (그림 9)(a)에서는 6번 프로세서에서 7번 프로세서로 부하가 이동되고, (그림 9)(b)에서는 반대로 부하가 이동된다. 만약 각 단계가 수행될 때마다 실제로 부하를 이동한다면, 그들 사이의 링크의 이동 부하는 2가 될 것이다. 이것은 아무런 의미도 없이 통신비용의 증가나 총 이동 부하의 증가만을 초래한다. 앞서 언급한 대로 제안된 알고리즘에서는 각 단계에서의 부하의 이동은 가상적으로만 계산하고, 실제 부하의 이동은 계산이 끝난 후에 일괄적으로 처리하여 이 문제를 해결한다. (그림 9)(f)는 그 결과이며, 2번과 6번 프로세서 사이의 링크로는 부하 이동이 없다. 한편 통신비용은 4가 되며 총 이동 부하는 31이다.

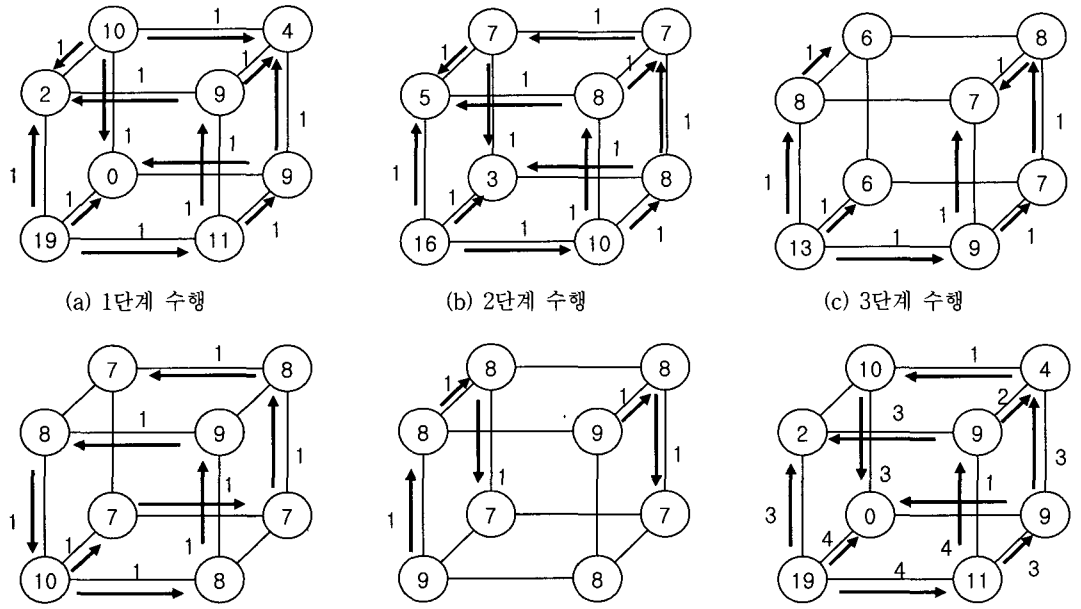



(a) 초기상태 (b) i = 0에서 수행 (c) i = 1에서 수행

(d) i = 2에서 수행 (e) i = 3에서 수행 (f) i = 4에서 수행

(g) i = 5에서 수행 (h) i = 6에서 수행 (i) i = 7에서 수행

(그림 8) 제안 알고리즘의 1단계 수행 예



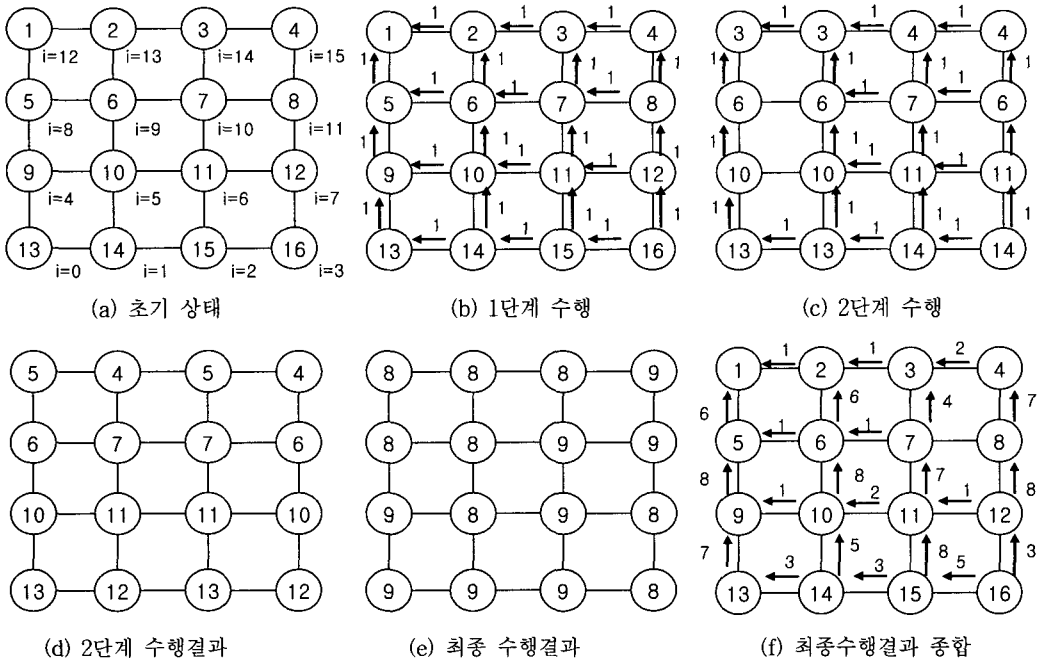
(a) 1단계 수행 (b) 2단계 수행 (c) 3단계 수행

(d) 4단계 수행 (e) 5단계 수행 (f) 각단계의 결과 종합

(그림 9) 하이퍼큐브 구조에서 제안알고리즘의 수행 예

### 3.4 메쉬 구조에서의 부하 평형

초기 조건 수집이 완료되면 하이퍼큐브 구조와 마찬가지로, 작업이동을 수행할 프로세서에서는 부하의 상태에 따라 동작 모드를 결정한다. (그림 10)은 메쉬 구조에 대한 제안 알고리즘의 수행 예이다. 전체 부하가 136이고 프로세서의 개수가 16이므로 평균 부하는 8 또는 9이고, 이 값은 각 프로세서들의 목표 부하가 된다. 1단계에서 0번 프로세서에게 이웃한 프로세서가 1번과 4번 프로세서이며, 부하가 목표 부하보다 많으므로 소스모드가 된다. 그런데 1번 프로세서는 자신보다 부하가 많고 4번 프로세서는 적으므로, 4번 프로세서에게 하나의 부하를 내보낸다. 1번 프로세서에서 0번 프로세서로의 부하의 이동은 1번 프로세서에 대해 알고리즘을 수행할 때 이루어진다. 실제 수행 단계는 초기 상태에 대하여 8번을 수행하여야 하지만 지면상 간략화 하였다. 결과적으로 통신비용 8이며 MWA보다 33%정도 감소하였으며, 전체 이동한 부하의 수도 98로 MWA보다 감소된 것으로 나타났다.



(a) 초기 상태 (b) 1단계 수행 (c) 2단계 수행

(d) 2단계 수행결과 (e) 최종 수행결과 (f) 최종수행결과 종합

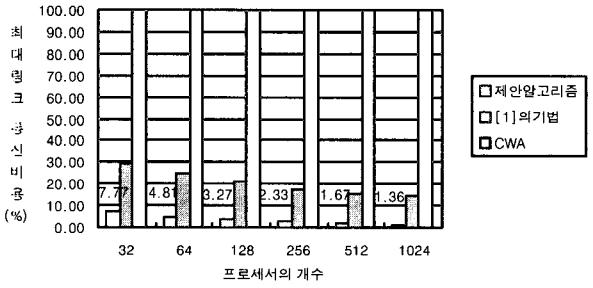
(그림 10) 제안 알고리즘의 메쉬 구조에서 수행한 예제

## 4. 시뮬레이션 및 실험결과

### 4.1 기본기법을 이용한 시뮬레이션 및 실험

제안된 알고리즘의 성능을 검증하기 위하여 모의실험을 하였다. 하이퍼큐브 구조에서는 프로세서의 개수를 32, 64, 128, 256, 512, 1024로, 메쉬 구조에서 8×8, 16×16, 24×24, 32×32으로 바꾸어 가며 실험하였다. 부하의 분포는 포아송 분포(Poisson Distribution)를 사용하였다. 각 프로세서의 평균 부하는 1,000개로 하였으며 1,000개의 작업 집합에 대하여 실험하였다. 제안된 알고리즘은 적용되는 프로세서의 순서에 따라 결과가 상이해 질 수 있는데 실험에서는 각 프로세서의 일련번호 순서대로 알고리즘을 적용하였다. 실험 결과 모든 경우에서 제안된 알고리즘이 성공적으로 부하 평형을 달성하였다.

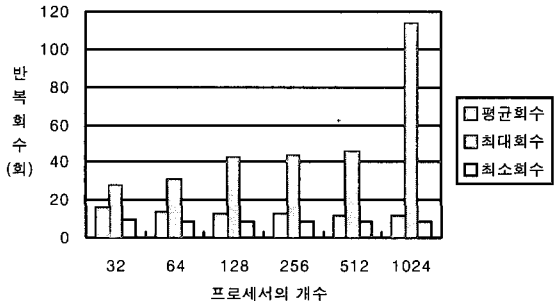
(그림 11)는 하이퍼큐브 구조에서의 실험 결과이다. 제안 알고리즘의 성능은 CWA와 CWA를 개선시킨 [1]의 기법과 비교하였다. (그림 11)는 통신비용의 비율, 즉 부하 이동에 소요되는 시간의 평균을 CWA를 100%라 보았을 때 비율을 나타낸 것이다. [1]의 기법과 제안 알고리즘은 프로세서의 개수가 늘어날수록 통신비용이 감소함을 알 수 있다. 그러나 감소 비율을 보면 제안알고리즘이 더 우수함을 알 수 있고 [1]의 기법보다도 평균 약 17%의 비용만으로 부하평형을 이룰 수 있다. CWA와 [1]의 기법의 이러한 특성은 CWA가 알고리즘 수행 시에 각 차원에 대하여 초과된 부하를 한 번에 내보내려 하기 때문이다. 즉 프로세서의 개수가 증가함에 따라 각 프로세서의 링크의 개수가 늘어나는 하이퍼큐브 구조의 특징을 잘 이용하지 못하는 것이다. 이에 비하여 제안 알고리즘은 늘어난 링크들을 충분히 활용하였기 때문에 각 링크 당 이동할 부하의 수가 감소하는 추세를 보였다. 그 결과 제안된 알고리즘은 [1]의 기법에 비하여 무려 약 75~90% 정도의 통신비용을 줄였다. (그림 12)은 제안알고리즘의 반복회수를 표시한 그림이다.




(그림 11) 하이퍼큐브 구조에서의 실험 결과

평균 반복회수는 프로세서의 개수가 증가할수록 감소함을 알 수 있다. 이것은 부하의 분포 상태가 포와송 분포를 따르기 때문으로 판단된다. 그러나 제안알고리즘의 휴리스틱한 특성으로 인하여 최대반복회수는 증가됨이 관찰되었다.

<표 1>은 알고리즘의 수행시간을 비교한 수치이다. 수행시간의 측정은 펜티엄4 2.8GHz CPU를 사용하는 윈도우 XP 시스템에서 (그림 11)의 실험결과가 도출되기까지의 시간을 10회 측정하여 평균 시간을 산출하였다. 프로세서의개수가 증가할수록 수행시간의 비율이 줄어들어 프로세서의 개수가 많을수록 더 짧은 수행시간이 요구되었다.

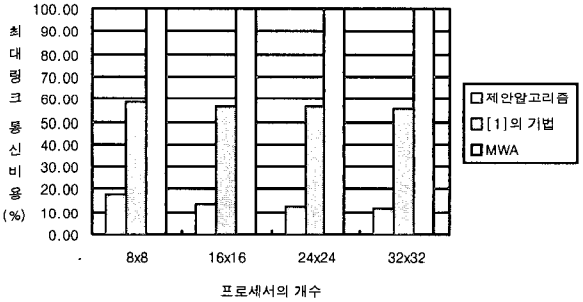



(그림 12) 하이퍼큐브 구조에서 제안알고리즘의 반복회수(회)

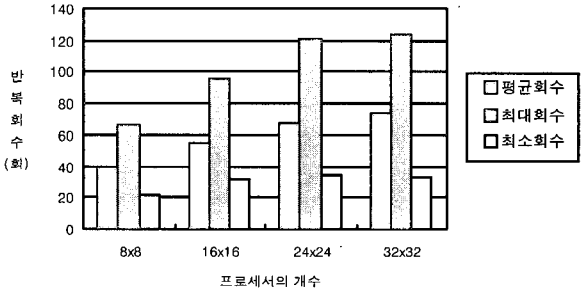
〈표 1〉 하이퍼큐브 구조에서 알고리즘의 수행 시간 비교

(단위 msec/workset)

| 프로세서의 개수 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|
| 제안알고리즘 | 0.312 | 0.640 | 1.390 | 2.218 | 6.875 | 16.234 |
| [1]의 기법 | 0.156 | 0.421 | 1.109 | 2.890 | 7.328 | 18.921 |




(그림 13) 메쉬 구조에서의 실험결과




(그림 14) 메쉬 구조에서의 제안알고리즘의 반복회수(회)

메쉬 구조에서의 성능은 MWA와 [1]의 기법과 비교하였다. (그림 13)은 메쉬 구조에서의 실험 결과이다. (그림 13)의 통신비용에 대한 비교를 살펴보면, 제안된 알고리즘의 부하 이동시간이 [1]의 기법에 대하여 통신비용이 평균 약 75% 가까이 줄어들었음을 알 수 있다. 이와 같이 통신비용

의 감소가 크게 나타난 것은 MWA에 비해 많은 수의 링크들이 부하 이동에 사용되어 부하 이동이 분산되었기 때문이다. (그림 14)는 제안알고리즘의 반복회수를 표시한 그림이다. 하이퍼큐브 구조의 경우와는 달리 프로세서의 개수가 증가함에 따라 반복회수가 증가하는 추세를 보였다. 이것은 메쉬 구조는 프로세서의 수가 늘어나도 프로세서 당 링크의 개수가 변화하지 않기 때문이다. 수행시간을 표시한 <표 2>에서도 이러한 메쉬 구조의 특성이 반영되었다. 프로세서의 개수가 늘어나면 제안 알고리즘의 수행시간 증가율이 [1]의 기법에 대하여 더 많이 요구되었다.

〈표 2〉 메쉬 구조에서 수행시간 비교

(msec/workset)

| 프로세서의 개수 | 8×8 | 16×16 | 24×24 | 32×32 |
|---|---|---|---|---|
| 제안알고리즘 | 0.859 | 4.609 | 13.906 | 24.937 |
| [1]의 기법 | 0.062 | 0.281 | 0.703 | 1.140 |

### 4.2 고속기법을 이용한 시뮬레이션 및 실험

고속기법의 효과를 실험하기 위하여 부하의 할당은 2보다 크고 100,000보다 작은 난수발생기를 이용하여 100개의 작업집합에 대하여 실험하였다. 먼저 하이퍼큐브 구조의 경우 1024개의 노드를 가진 하이퍼큐브 구조에서 고속기법과 기본기법을 적용하여 알고리즘을 수행하였다. 이 때 효과를 비교하기 위하여 먼저 기본기법만으로 알고리즘을 수행하였을 때 결과를 <표 3>에 정리하였다. 기본기법을 이용하여 수행한 결과가 기존의 비교 알고리즘보다 적은 부하이동량을 나타내었다. 그러나 다른 경우보다 과다한 수행시간이 요구된다. 이것은 3.2절에서 언급했듯이 기본기법이 단위부하 만을 반복적으로 이동시킴으로 인하여 알고리즘의 반복회수가 많기 때문이다. 이에 대하여 고속기법을 이용하면 통신비용은 약간 증가하지만 반복회수에 따른 수행시간은 현저히 감소시킬 수 있다.

〈표 3〉 10차 하이퍼큐브 구조에서 제안알고리즘의 기본기법 적용

| | 평균부하이동량(개) | 평균 수행시간(초) | 평균 반복 회수(회) |
|---|---|---|---|
| 기본기법 | 6,267 | 3.440 | 6268 |
| [1]의 기법 | 96,205 | 0.021 | N/A |
| CWA | 637,152 | N/A | N/A |

<표 4>는 10차 하이퍼큐브 구조에서 고속기법과 기본기법을 적용한 모의실험 결과이다. 고속기법 적용시에는 두가지 요소를 통하여 수행 방법을 결정할 수 있으므로 이 두 가지 요소를 바꾸어 주면서 실험하였다. 각 노드의 평균부하량과의 차이 즉 불균등 부하의 수는 16~1024까지 변화시키면서 이때 이동시키는 부하의 수를 16~1024개로 변화시켰다. 빈칸은 정상적이지 못한 부하이동이므로 수행치 않았다. <표 4>에는 세 가지의 실험결과를 표시하였다. 평균부하이동(%)은 <표 3>의 평균부하이동량을 기준으로 하여 증가된 부하이동량(%)를 나타낸다. 평균반복회수는 <표 3>의 평균반복회수를 기준으로 하여 증가된 반복회수를 백분율로 표시하였다. 평균수행시간은 한 작업집합 당 필요한 수행시간을 초 단위로 기술하였다. 고속기법을 추가하면 기본기법만을 적용했을 때보다 부하이동량이 약 0.6~50% 정도 증가되었다(<표 4>의 평균부하이동). 그러나 반복회수는 기본기법 대비 1.3~12% 정도만으로 수행이 가능하였다. 또한 한 개의 작업집합을 부하평형 시키는데 필요한 수행시간의 경우 기본기법만으로 수행하였을 때 약 3.4초가 소요되었으나 고속기법을 함께 적용하였을 때에는 약 0.045~0.145초 정도 요구되었다. 그러므로 기본기법만 수행했을 때보다 반복회수와 수행시간을 현저히 감소시킬 수 있음을 확인할 수 있다.

〈표 4〉 10차 하이퍼큐브 구조에서 고속기법 적용시 실험결과

| 이동부하수 \ 불균등부하수 | | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|
| 16 | 평균부하이동(%) | 0.702 | 0.654 | 0.734 | 0.957 | 1.324 | 1.995 | 3.032 |
| | 평균반복회수(%) | 6.445 | 6.557 | 6.828 | 7.419 | 8.408 | 10.003 | 12.556 |
| | 평균수행시간(초) | 0.239 | 0.236 | 0.243 | 0.257 | 0.285 | 0.333 | 0.415 |
| 32 | 평균부하이동(%) | | 1.468 | 1.388 | 1.548 | 1.851 | 2.425 | 3.383 |
| | 평균반복회수(%) | | 3.382 | 3.590 | 4.148 | 5.185 | 6.844 | 9.445 |
| | 평균수행시간(초) | | 0.121 | 0.128 | 0.142 | 0.172 | 0.223 | 0.309 |
| 64 | 평균부하이동(%) | | | 3.143 | 2.920 | 3.032 | 3.494 | 4.213 |
| | 평균반복회수(%) | | | 1.914 | 2.345 | 3.335 | 5.073 | 7.722 |
| | 평균수행시간(초) | | | 0.068 | 0.080 | 0.109 | 0.163 | 0.252 |
| 128 | 평균부하이동(%) | | | | 6.399 | 5.936 | 5.984 | 6.303 |
| | 평균반복회수(%) | | | | 1.324 | 2.090 | 3.797 | 6.573 |
| | 평균수행시간(초) | | | | 0.045 | 0.069 | 0.121 | 0.212 |
| 256 | 평균부하이동(%) | | | | | 14.138 | 11.888 | 10.898 |
| | 평균반복회수(%) | | | | | 1.324 | 2.680 | 5.456 |
| | 평균수행시간(초) | | | | | 0.043 | 0.085 | 0.172 |
| 512 | 평균부하이동(%) | | | | | | 27.302 | 22.100 |
| | 평균반복회수(%) | | | | | | 1.883 | 4.212 |
| | 평균수행시간(초) | | | | | | 0.061 | 0.134 |
| 1024 | 평균부하이동(%) | | | | | | | 50.918 |
| | 평균반복회수(%) | | | | | | | 3.191 |
| | 평균수행시간(초) | | | | | | | 0.105 |

메쉬 구조에서도 하이퍼큐브 구조와 동일한 조건에서 실험하였다. 16×16 메쉬 구조에 기본기법만으로 수행하였을 때의 결과가 <표 5>와 같다.

〈표 5〉 16×16 메쉬 구조에서 제안알고리즘의 기본기법 적용

| | 부하이동량(개) | 평균수행시간(초) | 반복회수(회) |
|---|---|---|---|
| 기본기법 | 37,996 | 0.985 | 38,004 |
| [1]의 기법 | 186,226 | 0.00015 | N/A |
| MWA | 326,235 | N/A | N/A |

메쉬 구조에서도 하이퍼큐브 구조와 마찬가지로 제안 알고리즘의 기본 기법이 비교 알고리즘보다 더 적은 부하이동량을 나타내었다. 그러나 기본 기법의 많은 반복수행으로 인하여 과다한 수행시간이 요구되었다. 메쉬 구조에서 고속기법을 적용한 실험 결과를 <표 6>에 표시하였다.

〈표 6〉 16×16 메쉬 구조에서 고속기법 적용시 실험결과

| 이동부하수 \ 불균등부하수 | | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|
| 16 | 평균부하이동(%) | 0.674 | 0.713 | 0.779 | 0.969 | 1.432 | 2.584 | 5.064 |
| | 평균반복회수(%) | 6.915 | 7.149 | 7.639 | 8.641 | 10.636 | 14.435 | 21.329 |
| | 평균수행시간(초) | 0.072 | 0.076 | 0.080 | 0.090 | 0.103 | 0.134 | 0.192 |
| 32 | 평균부하이동(%) | | 1.350 | 1.426 | 1.600 | 2.063 | 3.095 | 5.403 |
| | 평균반복회수(%) | | 4.326 | 4.810 | 5.813 | 7.812 | 11.659 | 18.627 |
| | 평균수행시간(초) | | 0.046 | 0.051 | 0.060 | 0.073 | 0.104 | 0.164 |
| 64 | 평균부하이동(%) | | | 2.874 | 3.008 | 3.390 | 4.429 | 6.751 |
| | 평균반복회수(%) | | | 3.815 | 4.786 | 6.776 | 10.557 | 17.517 |
| | 평균수행시간(초) | | | 0.039 | 0.047 | 0.062 | 0.095 | 0.150 |
| 128 | 평균부하이동(%) | | | | 6.272 | 6.698 | 7.253 | 8.814 |
| | 평균반복회수(%) | | | | 5.084 | 6.976 | 10.675 | 17.469 |
| | 평균수행시간(초) | | | | 0.046 | 0.062 | 0.094 | 0.151 |
| 256 | 평균부하이동(%) | | | | | 12.246 | 12.651 | 13.141 |
| | 평균반복회수(%) | | | | | 8.546 | 12.030 | 18.469 |
| | 평균수행시간(초) | | | | | 0.074 | 0.102 | 0.157 |
| 512 | 평균부하이동(%) | | | | | | 21.363 | 20.892 |
| | 평균반복회수(%) | | | | | | 14.838 | 20.556 |
| | 평균수행시간(초) | | | | | | 0.126 | 0.175 |
| 1024 | 평균부하이동(%) | | | | | | | 31.332 |
| | 평균반복회수(%) | | | | | | | 24.108 |
| | 평균수행시간(초) | | | | | | | 0.205 |

<표 6>을 살펴보면 하이퍼큐브와 같이 약 0.6~30%정도의 추가 부하이동이 발생하였다. 그러나 기본기법만 적용하였을 때보다 기본기법 대비 반복회수는 6.9~24%로 감소하였고 수행시간도 약 0.04~0.2초 소요되어 더 적은 수행시간이 요구됨을 확인할 수 있다.

이상의 다양한 실험결과로부터 제안알고리즘이 기존의 알고리즘보다 적은 통신비용으로 부하 평형의 목적을 달성할 수 있음을 확인하였다. 그러나 다중프로세서의 구조나 부하의 분포 상태에 따라 제안 알고리즘의 수행시간이 기존의 알고리즘보다 더 요구되나 이것은 실제 부하 이동시간이 더 적게 요구되므로 전체 부하 평형 시간은 감소될 것으로 판단된다.

## 5. 결 론

다중 프로세서 시스템에서는 부하의 균등한 분배가 시스템의 성능 향상에 중요한 역할을 한다. 기존의 부하 균등 재분배 방법에서는 매 단계마다 변화된 각 프로세서의 부하량을 계산하여 부하의 이동을 결정한다. 이에 따라 부하의 이동이 발생하지 않는 링크가 발생하는데 이러한 링크가 발생하면 통신비용이 증가되는 원인이 된다. 본 논문에서 제안한 휴리스틱 부하 평형 알고리즘은 링크에 대하여 기본기법을 이용하여 한 개의 부하를 반복적으로 이동함으로서 부하평형에 사용되지 않는 링크의 수를 줄였다. 또한 프로세서의 작업 이동 방법을 각 프로세서의 상태에 따라 하이퍼큐브 구조의 경우 균등 부하량보다 많은 노드에 대해서는 부하를 내보내는 소스모드로 동작하고, 균등 부하량보다 적은 노드에 대하여서는 부하를 가지고 오는 싱크모드로 동작하도록 하였다. 부하가 균등상태에 있을 때에는 주위의 노드 상태에 따라 부하를 많이 가지고 있는 노드에서 적게 가지고 있는 노드 쪽으로 부하를 이동시키도록 하였다. 또한 부하의 분포가 한 노드에 집중되어 있거나 불균등 부하의 수가 큰 경우에는 고속기법을 이용하여 부하의 불균등 분포를 완만하게 조절한 후 기본기법을 적용하여 기본기법만으로 수행시 보다 빠르게 부하평형의 목적을 달성할 수 있다. 제안된 알고리즘의 성능을 비교하기 위하여 하이퍼큐브 구조와 메쉬 구조에 대하여 시뮬레이션 하였다. 실험 결과 제안된 알고리즘은 모든 경우에서 완전한 부하 평형에 성공하였다. 하이퍼큐브 환경에서 제안알고리즘은 기존의 알고리즘에 비해 통신비용을 70~90% 정도 줄이면서 부하평형을 이룰 수 있었다. 또한 메쉬 환경에서도 기존의 알고리즘에 비해 통신비용을 약 75% 감소시키는 성능을 보였다.

## 참 고 문 헌


[1] 임화경, 장주욱, 김성천, "신속한 부하균등화를 위한 휴지링크의 최대활용방법", 정보과학회논문지, 시스템 및 이론, 제28권 제12호, pp.632-641, 2001.

[2] M. Y. Wu and D. D. Gajski, "Hypertool : A Programming Aid for Message-Passing Systems," IEEE Trans. on Parallel and Distributed Systems, Vol.1, No.3 pp.330-343, July, 1992.

[3] M. Y. Wu, "On Runtime Parallel Scheduling for Processor Load Balancing," IEEE Trans. on Parallel and Distributed Systems, Vol.8, No.2, pp.173-185, Feb., 1997.

[4] C. Hui, S. Chanson, "Hydrodynamic Load Balancing," IEEE Trans. on Parallel and Distributed Systems, Vol.10, No.11, pp.1118-1137, 1999.

[5] I. Ahnad, Y. Kwok, "On Parallelizing the Multiprocessor Scheduling Problem," IEEE Trans. on Parallel and Distributed Systems, Vol.10, No.4, pp.414-432, 1999.

[6] M. Mitzenmacher, "How useful Is Old Information?," IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.9, pp.979-993, 1993.

[7] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," J. Parallel and Distributed Computing, Vol.7, pp.279-301, 1989.

[8] S. Ranka, Y. Won and S. Sahni, "Programming a Hypercube

Multicomputer," IEEE Software, pp.69-77, Sept., 1988.

[9] Min-You Wu, Wei Shu, "DDE : A Modified Dimension Exchange Method for Load Balancing in k-ary n-cube," Journal of Parallel and Distributed Computing, 44, pp.88-98, 1997.

[10] Kyungwan Nam, Jaewon Seo, Sunggu Lee and Jong Kim, "Synchronous Load Balancing in Hypercube Multicomputers with Fault Nodes," Journal of Parallel and Distributed Computing, 58, pp.26-43, 1999.

[11] Leonid Oliker, "PLUM : Parallel Load Balancing for Adaptive Unstructured Meshed," Journal of Parallel and Distributed Computing, 52, pp.150-177, 1998.

[12] Marlin H. Mickle and Jo Ann M. Paul, "Loading Balancing Using Heterogeneous Processors for Continuum Problems on a Mesh," Journal of Parallel and Distributed Computing, 39, pp.66-73, 1996.

**송 의 석**

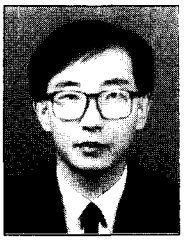
e-mail : euiseok@kookmin.ac.kr

1993년 국민대학교 전자공학과(공학사)

1995년 국민대학교 일반대학원 전자공학전공(공학석사)

2002년 국민대학교 일반대학원 전자공학전공(박사수료)

관심분야 : 병렬처리, 내장형시스템, 영상처리



**오 하 령**

e-mail : hroh@kookmin.ac.kr

1983년 서울대학교 전기공학과(공학사)

1983년~1986년 삼성전자 종합연구소

1988년 한국과학기술원 전기전자과 컴퓨터공학전공(공학석사)

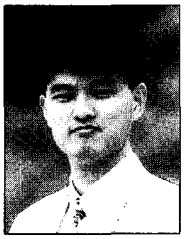
1992년 한국과학기술원 전기전자과 컴퓨터공학전공(공학박사)

1992년~1996년 국민대학교 전자공학부 조교수

1996년~2001년 국민대학교 전자공학부 부교수

2001년~현재 국민대학교 전자정보통신공학부 교수

관심분야 : 병렬처리, 내장형 시스템, 고장감내



**성 영 락**

e-mail : yeong@kookmin.ac.kr

1989년 한양대학교 전자공학과(공학사)

1991년 한국과학기술원 전기 및 전자공학과(공학석사)

1995년 한국과학기술원 전기 및 전자공학과(공학박사)

1995년~1996년 한국과학기술원 위촉연구원

1996년~1998년 국민대학교 전자공학부 전임강사

1998년~2002년 국민대학교 전자공학부 조교수

2002년~현재 국민대학교 전자정보통신공학부 부교수

관심분야 : 시뮬레이션, 고장감내, 내장형 시스템