논문 2005-42CI-1-4

다중 프로세서 환경에서 연결구조에 무관한 휴리스틱 부하평형 알고리즘

(A Topology Independent Heuristic Load Balancing Algorithm for Multiprocessor Environment)

송 의 석*, 성 영 락**, 오 하 령**

(Eui-Seok Song, Yeong-Rak Sung, and Ha-Ryoung Oh)

요 약

본 논문에서는 다중 프로세서 시스템을 위한 효율적인 휴리스틱 부하 평형 알고리즘을 제안한다. 제안 알고리즘은 부하이동을 여러 링크로 분산시켜, 사용하지 않는 링크의 수를 최소화하고 그에 따라 통신비용이 감소한다. 각각의 프로세서는 모든 이웃한 프로세서에게 단위부하를 보내거나 받는 과정을 반복적으로 시도한다. 그러나 실제의 부하 이동은 모든 계산이 이루어진 후 수행된다. 이것은 불필요한 부하 이동을 막아 전체적으로 부하이동의 수가 감소한다. 제안된 알고리즘은 약간의 수정만으로 다양한 연결 구조를 갖는 다중 프로세서 시스템에 적용할 수 있다. 본 논문에서는 하이퍼큐브 구조, 메쉬 구조, k-ary n-cube 구조와 일반 그래프 구조에 제안 알고리즘을 적용해 보았다. 알고리즘의 성능평가를 위하여 모의실험을 하였다. 제안된 알고리즘과 잘 알려진 알고리즘을 구현하여 비교하였다. 그 결과 제안된 알고리즘은 모든 경우에서 완전한 부하평형에 도달하였다. 또한 기존의 알고리즘과 비교하여 하이퍼큐브 구조에서는 약 77%, 메쉬 구조에서는 약 74%, 또한 k-ary 2,3-cube 구조에서는 약 73% 정도 통신비용을 감소시켰다.

Abstract

This paper proposes an efficient 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 all neighbor processors. However, real load transfer is collectively done after all load traffic is calculated. This prevents useless traffic and thus reduces the overall load traffic. The proposed algorithm can be employed in various interconnection topologies with slight modifications. In this paper, it is applied to hypercube, mesh, k-ary n-cube and general graph environments. For performance evaluation, simulation studies are performed. The proposed algorithm and the well-known existing algorithms are implemented and compared. The results show that the proposed algorithm always balances the loads perfectly. Furthermore, in comparison with the existing algorithms, it reduces the communication costs by 77%, 74% and 73% in the hypercube, the mesh, and *k-ary n-cube*, respectively.

Keywords : Multiprocessors system, Load Balancing, k-ary n-cube, General Graph

I. 서 론

최근 컴퓨터 시스템의 비약적인 발전에는 반도체 기술의 발달과 중앙처리장치의 다양한 설계방법의 적용(파이프라인, 수퍼스칼라, 윔홀 라우팅 등)이 크게 기여하고 있다. 그러나 컴퓨터의 고속의 연산 처리를 이용하는 분야(매트릭스 연산, 고차 방정식의 풀이, 기상 예측프로그램 등)나 데이터간의 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

* 정회원, 국민대학교 전자공학과 일반대학원
(Department of Electronics Engineering, Kookmin University)
** 정회원, 국민대학교 전자정보통신공학부
(School of Electrical Engineering, Kookmin University)
접수일자: 2004년9월16일, 수정완료일: 2005년1월10일

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 성능 향상에 중요한 역할을 한다. 각각의 프로세서가 처리하여야 할 작업을 부하(load)라 부른다. 따라서 시스템 내의 부하의 분포가 불균형한 상태로 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 하거나 예측 가능한 시스템 구조(하이퍼큐브, 메쉬, *k-ary n-cube* 등)를 가지고 있는 경우에는 정적 부하평형이라고 부른다. 반면에 문제 해결을 위한 연산 중에 부하 평형을 수행하거나, 예측 불가능한 구조를 가지는 시스템에서는 동적 부하평형이라고 부른다. 부하평형에서 특히 중요한 요소 중 하나는 부하 재분배에 소요되는 통신비용을 최소화하는 것이다[1-3].

기존의 부하 평형 알고리즘은 다중 프로세서의 연결구조에 따라 다양한 알고리즘이 연구되었다[3-5]. 그래서 특정 구조에 적합하게 개발된 알고리즘을 다른 연결 구조를 갖는 다중프로세서에 적용하기 어려웠다.

일반적으로 기존에 제안된 부하 평형 기법들은 크게 두 단계로 수행된다. 이중에서 전 단계에서는 상태정보를 주고받아 모든 프로세서들이 부하의 분포 상태를 알아내며, 기법에 따라 통신비용이 무시될 수 있다. 또한 후 단계에서는 이동할 부하의 양을 계산하고 부하 재분배를 수행한다. 이때 적용하는 방법에 따라 통신비용이 달라진다[3][6-8]. 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][14] 프로세서 사이의 데이터 교환을 위한 구조가 웜홀 라우팅의 구조라면 이용이 가능한 통신비용이다. 본 논문에서는 이를 채용한다. 후자의 경우에는 이동되는 부하의 총수로 정의한다[3].

부하 평형을 효과적으로 달성하기 위해서는 통신비용을 줄이면서 초과 부하들을 저부하 프로세서에 전송해야한다. 그러나 분할이 불가능한 부하에 대한 부하 평형 문제는 미니맥스 플로우 문제(minimax flow problem)과 동일하며 NP-Complete문제이다[14]. 따라서 본 논문에서는 정적 휴리스틱(heuristic) 부하 평형 방법을 제안한다. 기존의 알고리즘들은 프로세서의 연결구조에 따라 차원별 또는 행, 열 별로 부하 평형을 수행한다[3-5]. 이 때 부하 이동이 없는 링크들이 생기면 통신시간이 증가되는 요인이 된다. 본 논문에서 제안하는 알고리즘에서는 부하의 이동이 없는 링크의 수를 줄여 부하 이동을 여러 링크에 분산시켜 전체적인 통신시간을 줄일 수 있다는 점에 착안하였다. 이를 위하여 각 프로세서는 연결된 모든 링크에 대해서 단위 부하를 반복하여 이동시킨다. 각 프로세서는 자신이 보내거나 받아야할 부하의 수를 계산하게 되며 결과가 완성되면 일괄적으로 부하평형을 실행한다. 제안된 알고리즘은 프로세서 사이의 연결 구조에 상관없이 같은 방법으로 부하 평형을 실행한다. 본 논문에서는 하이퍼큐브 구조, 메쉬 구조, *k-ary n-cube* 구조와 일반 그래프 구조를 갖는 다중프로세서 시스템을 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. Ⅱ장에서는 기존의 연구를 살펴보고, Ⅲ장에서는 제안한 알고리즘을 설명하며, Ⅳ장에서는 실험을 통하여 결과를 분석하고, Ⅴ장에서는 결론을 내린다.

Ⅱ. 기존의 연구

하이퍼큐브 구조에서의 부하 평형 기법에 대하여 많은 연구가 발표되었다. 이 중에서 가장 대표적인 예로 DEM과 CWA를 들 수 있다. DEM (Dimension Exchange Method)[3-4]은 먼저 좁은 영역에서 부하 평형을 실시하고, 이를 점차 넓은 영역으로 확대 적용하여, 전체 시스템을 부하 평형에 도달하도록 한다. 그림 1(a)는 DEM 알고리즘을 수행한 후 차원별로 링크에 대하여 부하 이동량을 표시한 예이다. 자세한 수행과정은 [4]를 참고하기 바란다. DEM의 경우 알고리즘은 간단하나 부하 평형 후 모든 노드가 부하 평형에 도달 하지 못하는 단점이 있다. [3]의 연구에 따르면 DEM 수행 후의 부하 차이는 최대 하이퍼큐브의 차원 수만큼 될 수 있다.

CWA(Cube Walking Algorithm)[3][9]는 DEM 보다 조

금 더 복잡한 계산 과정을 거친다. 그림 1(b)는 CWA를 수행한 예이다. DEM과 같이 CWA도 차원별로 수행한 결과를 종합하여 표시하였다. CWA는 전체 이동되는 부하의 개수를 최적화시킨 알고리즘이다.

CWA는 매 단계마다 부하를 이동시키고 변화된 값을 이용하여 다시 계산하는 과정을 반복하는데 CWA를 개선시킨 [1]에서는 단계별 이동을 수행하지 않고 중간 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. 그림 1은 [1]의 방법을 이용하여 표시하였다.

메쉬 구조의 다중 프로세서 시스템에서의 부하 평형 알고리즘에 대해서도 많은 연구가 발표되었다[3][10 11]. 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 1(c)은 MWA의 수행결과를 종합하여 표시한 예이다.

k-ary n-cube 구조에서의 부하 평형 기법은 DEM(Dimension Exchange Method)[3 5][12]과 GDE (Generalized Dimension Exchange)[13]에 기초하여 구성한 DDE(Direct dimension Exchange)[5]를 들 수 있다. DDE는 세 가지의 서로 다른 알고리즘을 적용하여 *k-ary n-cube* 구조에 대하여 부하 평형을 이룬다. 실제 알고리즘의 구성은 [5]를 참고하기 바란다. 그림 1(d)은 DDE 알고리즘의 수행결과이다. DEM과 같이 DDE 알고리즘도 모든 노드가 부하평형에 도달하지 못하는 단점이 있다. [5]에 의하면 최대 부하의 차이는 n이 될 수 있다.

각 알고리즘의 통신비용을 계산해보자. 여기에서 표시한 통신비용은 실제 부하의 이동으로 인한 비용만으로 비교하였다. 알고리즘의 적용에 따라 부하이동에 소요되는 시간의 계산이 많은 차이를 보일 수 있다. 그 이유는 차원 단위로 또는 행 단위, 열 단위의 부하이동이 독립적으로 수행된다면 각 단계의 최대값를 합하여 부하이동에 필요한 통신비용을 계산할 수 있다. 다른 방법으로 부하의 이동량을 계산한 후 일괄적으로 이동시킨다면 각 프로세서의 통신링크 중 최대값를 통신비용으로 판단할 수 있으며 [1]에서 제안한 기법이다.

[1]의 기법을 이용하여 통신비용을 계산해보면 DEM의 경우 5, CWA의 경우 6으로 판단할 수 있다. 같은

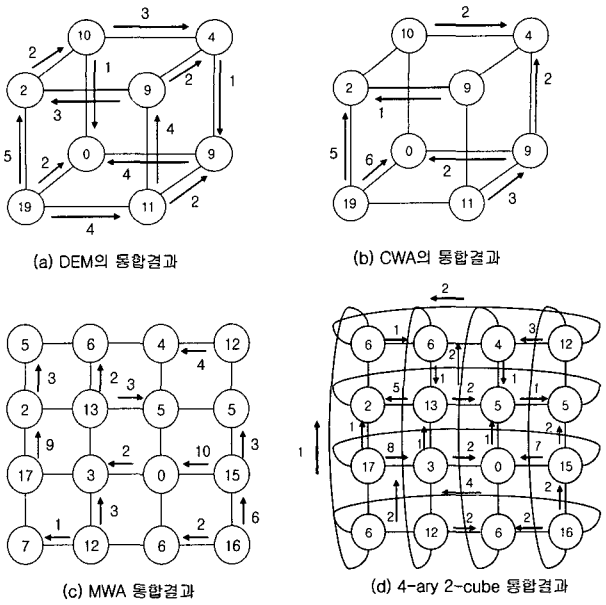


그림 1. 기존의 알고리즘에 대한 수행예제
Fig. 1. Examples in the previous algorithms.

방법으로 MWA와 *k-ary 2-cube*의 경우 통신비용은 각각 10과 8이 된다.

그림 1의 각 수행 결과를 보면 모든 링크가 부하 평형을 위하여 사용되지 못하였고 부하의 이동 또한 특정 링크에 집중되어 이동되는 현상을 볼 수 있다. 또한 알고리즘을 적용할 때 시작 위치(차원, 행 또는 열)의 선택을 어떻게 하느냐에 따라 결과가 상이해 질 수 있다. 그러므로 본 논문에서는 알고리즘의 적용 방법에 따른 통신비용의 변화를 최소화시키고 많은 링크가 부하 평형에 참여할 수 있는 휴리스틱 부하 평형 알고리즘을 제안한다.

III. 제안 알고리즘

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

각 부하의 실행 시간과 크기가 동일하다고 가정한다. 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.

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 1 만큼 차이가 날 수도 있다.

초기 조건 수집이 완료되면 여러 단계에 걸쳐서 부하 평형을 반복한다. 각 단계마다 각각의 프로세서는 자신의 현재 부하와 목표 부하를 비교하여 동작 방법을 결정한다. 즉 현재 부하가 목표 부하와 같지 않으면 불균등모드로, 같으면 중계모드로 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 $a+1$번째 이웃 프로세서를 검사한다. 링크에 대한 알고리즘을 라운드로빈으로 적용하면 특정링크로 부하이동이 집중되지 않도록 하고 부하가 특정 링크에서 진동하는 현상을 막을 수 있다.

비교 결과에 대한 처리는 프로세서의 동작모드에 따라 다르다. 불균등모드로 동작하는 프로세서는 자신보다 부하가 적은 이웃 프로세서에게 하나의 부하를 전달하거나 자신보다 부하가 큰 모든 이웃 프로세서로부터 각각 하나의 부하를 가져온다. 중계모드의 프로세서는 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.

이러한 알고리즘을 모든 프로세서에 차례대로 적용하면 한 단계가 끝난다. 단계를 마치면 각 프로세서가 목표 부하에 도달했는지 비교하여 다음 수행여부를 결정한다. 아직 부하 평형 상태에 도달하지 못하였다면 앞의 과정을 다시 적용한다. 이 때 다음 단계의 시작 프로세서는 라운드로빈하여 결정한다. 즉 전 단계의 시작 프로세서에 +1을 한 프로세서부터 알고리즘을 수행한다. 즉 어떤 단계에서 i번째 프로세서부터 알고리즘을 적용하였다면 다음 단계에서는 $i+1$번째 프로세서에서부터 알고리즘을 적용한다. 프로세서에 대한 라운드로빈 효과도 링크에서와 같은 효과를 얻을 수 있어 부하가 링크에 대하여 잘 분산될 수 있도록 한다. 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 불필요한 부하의 이동을 막을 수 있으며 통신비용도 줄일 수 있다.

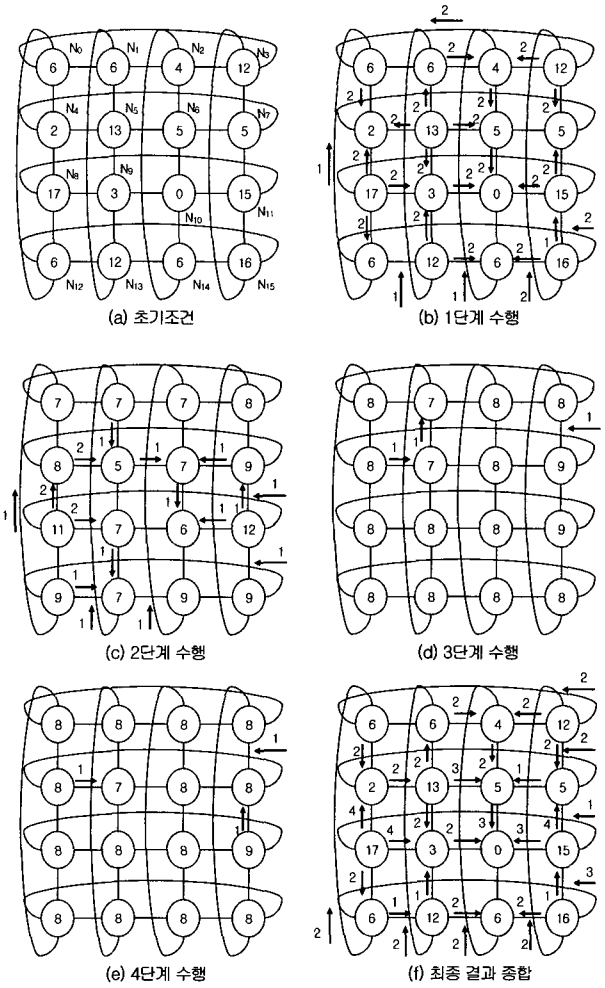


그림 2. 제안 알고리즘 수행 예제(*4-ary 2-cube*)
Fig. 2. Examples using the proposed algorithm.

그림 2는 그림 1(d)의 *4-ary 2-cube* 구조에 제안 알고리즘을 수행한 예이다. 전체의 총 부하는 128이고 16개의 프로세서가 있으므로 평균 부하는 8이다.

1 단계에서 N_0번 프로세서는 목표부하 8보다 적으므로 불균등모드로 동작한다. 그러므로 그림 2(a)의 노드

```
// Wi: 프로세서 i의 작업량
// Wq: 목표 작업량
// Wi,j: 프로세서 i의 j 번째 이웃 프로세서의 작업량
// LC: Loop 수행 회수 및 첫 번째 노드와 링크 선택
 1: LoadBalance()
 2: {
 3:   Wq = CalcAvgWorkLoad();
 4:   while (workload isn't evenly distributed) {
 5:     for (i = 0; i < number of node ; i++)
 6:       v = (LC + p) % number of node
 6:       if (Wv != Wq)
 7:         ImbalanceMode();
 8:       else if (Wv == Wq)
 9:         RelayMode();
10:    LC++;
11:   }
12: }
13: ImbalanceMode()
14: {
15:   for (p = 0; p < number of links; p++) {
16:     v = (LC + p) % number of links
17:     if ( Wi > Wi,v ) {
18:       Wi = Wi - 1;
19:       Wi,v = Wi,v + 1;
20:     }
21:     else if ( Wi < Wi,v ) {
22:       Wi = Wi + 1;
23:       Wi,v = Wi,v - 1;
24:     }
25:   }
26: }
27: RelayMode()
28: {
29:   BIG = SMA = NULL;
30:   for (p = 0; p < number of links; p++) {
32:     if ( Wp > Wi )
33:       BIG = Wp;
34:     else ( Wp < Wi )
35:       SMA = Wp;
36:     if ( BIG && SMA ) {
37:       BIG = BIG - 1;
38:       SMA = SMA +1;
40:       BIG = SMA = 0;
41:     }
42:   }
43: }
```

그림 3. 제안 휴리스틱 알고리즘의 의사코드
Fig. 3. Pseudo-code for the proposed heuristic algorithm.

번호에 대하여 N_{12}, N_1, N_4, N_3의 순서로 이웃 프로세서들의 부하를 검사한다. N_{12} 프로세서와의 비교에서는 동일한 부하를 가지고 있으므로 부하의 이동은 없다. 대신에 그림 2(b)를 보면 N_{12} 프로세서로부터 1개의 부하를 받도록 되어 있는데 이것은 N_{12} 프로세서에서 알고리즘이 수행될 때 이동되는 부하이다. N_3 프로세서에서도 2개의 부하가 이동되는 것으로 표시되어 있는데 이것은 N_0 프로세서에서 수행될 때 1개를 가지고 오며 N_3 프로세서에서 수행될 때 1개의 부하를 보내주어 2개가 된다(그림 2(b)). 다음은 N_1 프로세서의 차례가 된다. N_1 프로세서도 불균등모드이다. N_{13}, N_2, N_5, N_0의 순서로 이웃들을 검사한다. 이 때에도 N_{13}, N_2, N_5의 프로세서 사이에만 부하의 이동이 발생한다. 이 알고리즘을 남은 프로세서들에 순서대로 적용하면 그림 2(b)의 결과를 얻을 수 있다. 같은 방법으로 2, 3, 4단계에 수행한 예가 각각 그림 2(c),(d),(e)이다. 그림 2(d, e)의 3, 4 단계의 수행과정을 보면 중계모드의 역할을 쉽게 확인해 볼 수 있다. 즉 중계모드의 경우 알고리즘 수행의 마무리단계에서 부하를 평형 시켜주는 역할을 한다. 그림 2(d)에서는 N_4 프로세서가 중계모드 역할을 수행하여 N_7 프로세서로부터 N_5 프로세서로 부하를 이동시켜 준다. 그림 2(f)는 각 단계별 알고리즘의 계산 결과이다. 이 예에서는 4단계 만에 알고리즘의 수행이 종료된다. 지금까지의 알고리즘의 수행에 의한 부하이동은 가상적인 것이었으므로 계산된 링크 당 이동 부하를 실제로 이동시키면 부하 평형이 완료된다. 최종결과를 보면 통신비용은 4번 노드와 8번 노드간의 링크를 통하여 이동하는 4가 됨을 알 수 있다. 한편 총 이동 부하는 65이다. 그림 1(d)의 결과와 비교해 보면 통신비용이 50% 감소되었음을 확인할 수 있다.

그림 3은 제안 알고리즘의 의사 코드를 기술한 것이다. 주 알고리즘 LoadBalance()가 수행되면 초기 조건을 수집하여 평균 부하를 계산하고(그림 5의 3), 부하 평형을 이룰 때까지 모든 프로세서들에서의 부하 이동을 차례대로 수행한다(그림 3의 4-11). 이 과정에서 각 프로세서는 불균등모드(ImbalanceMode()), 중계모드(RelayMode())로 구분되어 계산을 수행한다. 알고리즘에서 LC는 반복된 횟수를 저장하는 변수로서 프로세서들이 이웃들을 검사할 때 라운드로빈 방식으로 검사 순서를 바꾸기 위해 사용된다.

Ⅳ. 시뮬레이션 및 실험결과

1. 하이퍼 큐브 구조에서의 실험결과

제안된 알고리즘의 성능을 검증하기 위하여 모의실험을 하였다. 하이퍼큐브 구조에서는 프로세서의 개수

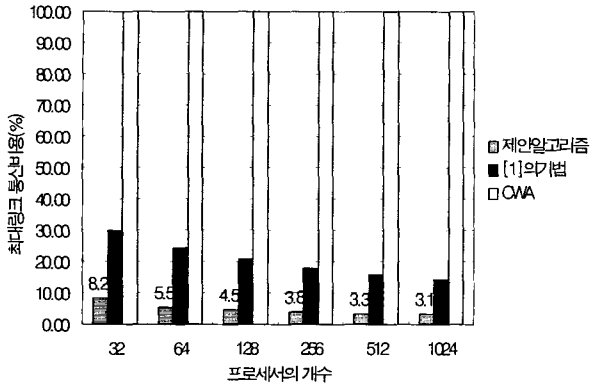


그림 4. 하이퍼큐브 구조에서의 실험결과
Fig. 4. Simulation results in hypercube.

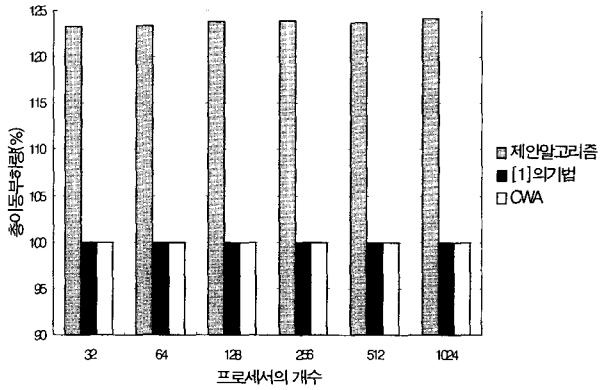


그림 5. 하이퍼큐브 구조에서 총부하이동량(%)
Fig. 5. Total Task-Hop in Hypercube.

표 1. 하이퍼큐브 구조에서 평균 수행시간(단위 ms)
Table 1. Avg. processing time in hypercube environment.

프로세서의 개수	32	64	128	256	512	1024
제안알고리즘	1.87	3.93	8.75	14.40	39.87	82.15
[1]의 기법	1.56	4.21	11.09	28.90	73.28	189.21

를 32, 64, 128, 256, 512, 1024로 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 4는 하이퍼큐브 구조에서의 실험 결과이다. 제안 알고리즘의 성능은 CWA와 CWA를 개선시킨 [1]의 기법과 비교하였다. CWA의 통신비용은 원래의 알고리즘을 적용하였을 때 통신비용이다. 이 때의 비용을 100이라 하였을 때 소요되는 통신비용을 표시하였다. [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 22%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.

표 1는 제안 알고리즘과 [1]의 기법을 이용한 수행방법에서 그림 3의 결과를 얻기까지의 수행시간을 표시한 표이다. 수행시간의 측정은 펜티엄4 2.8㎓ 컴퓨터에서 10회 측정 후 평균을 구한 수행시간이다. 수행시간의 비교를 보면 차원이 증가할수록 제안 알고리즘의 수행시간이 [1]의 기법보다 더 짧은 시간이 요구되었다. 이는 늘어난 통신링크를 잘 활용하여 부하평형이 수행된 것으로 판단된다.

그림 5는 총부하이동량을 비교한 그림이다. 총부하이동량은 기존의 알고리즘에서 통신비용으로 판단한 방법으로 제안알고리즘의 통신비용과 다른 방법으로 측정한다. 실험결과는 CWA의 총부하이동량을 100으로 환산하여 제안 알고리즘과 비교하였다. 실험결과 CWA보다 23~25% 정도 전체 이동되는 부하량이 증가 하였다. 이것은 CWA의 경우 하이퍼큐브 구조에서 최적 총부하이동량을 계산하는 알고리즘으로 개발된 결과라 판단된다. 이외의 나머지 결과에서는 총부하이동량 또한 제안 알고리즘이 개선된 결과를 볼 수 있다.

2. 메쉬 구조에서의 실험결과

메쉬 구조에서의 성능은 MWA와 [1]의 기법과 비교하였다. 실험에 사용한 프로세서 구조는 8x8, 16x16, 24x24 32x32의 구조에 대하여 하이퍼 큐브에서와 같은 부하 분포와 작업집합에 대하여 실험하였다. 그림 6는 메쉬 구조에서의 실험 결과이다. 제안 알고리즘이 MWA 대하여 평균 약 14.6%의 통신비용으로 부하평형을 이룰 수 있었다. [1]의 기법과의 비교에서도 평균 약 25.7%의 통신비용이 요구되었다. 하이퍼큐브 구조보다는 감소의 폭이 줄어들었는데 이것은 메쉬 구조의 특성

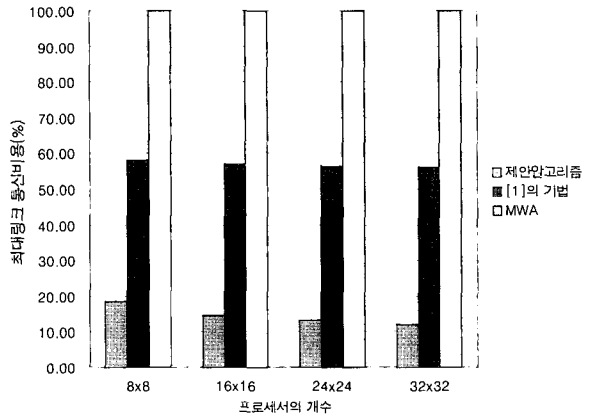


그림 6. 메쉬 구조에서의 실험결과
Fig. 6. Simulation results in mesh environment.

표 2. 메쉬 구조에서 평균 수행시간 비교(단위 ms)
Table 2. processing time in Mesh environment.

프로세서의 개수	8x8	16x16	24x24	32x32
제안알고리즘	2.12	12.44	48.28	76.74
[1]의 기법	0.62	2.81	7.03	11.40

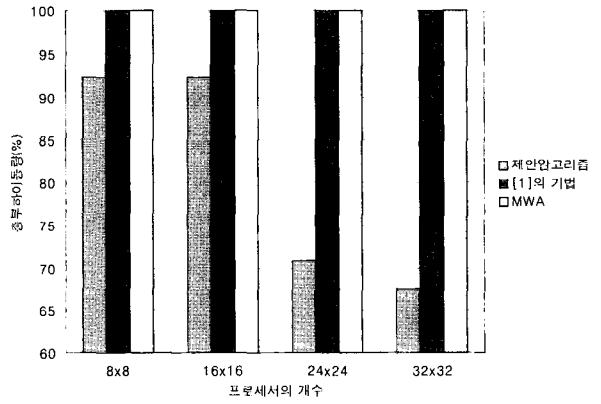


그림 7. 메쉬 구조에서 총부하이동량(%)
Fig. 7. Total Task-Hop in Mesh.

에 기인한다. 하이퍼큐브 구조의 경우 차원이 늘어나면 링크의 수가 함께 증가하지만 메쉬 구조의 경우 링크의 수는 노드 당 일정하기 때문이다.

표 2는 평균 수행시간을 비교한 표이다. 연결 링크의 개수 제한으로 인하여 제안알고리즘은 [1]의 기법보다 약 6.4배가량 증가한 수행시간이 요구되었다.

그림 7는 메쉬 구조에서 총부하이동량을 비교한 그림이다. 실험결과 MWA보다 8~32% 정도 총부하이동량이 감소하였다. 그러므로 제안 알고리즘이 메쉬 구조에서는 기존의 알고리즘보다 개선된 결과를 얻었다.

3. *k-ary n-cube* 구조에서의 실험결과

k-ary n-cube 구조에서는 큐브의 개수가 2인 구조에

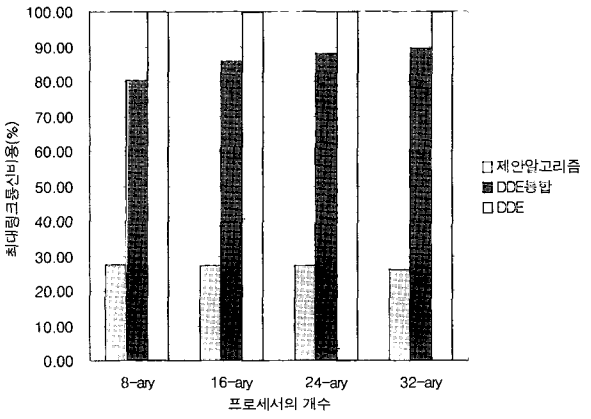


그림 8. *k-ary 2-cube* 구조에서의 실험결과
Fig. 8. Simulation results in *k-ary 2-cube*.

표 3. *k-ary 2-cube* 구조에서 평균 수행시간(단위 ms)
Table 3. Avg. processing time in *k-ary 2-cube*.

프로세서 구조	8-ary	16-ary	24-ary	32-ary
제안알고리즘	1.77	11.4	51.56	80.77
DDE통합기법	0.78	4.53	13.75	22.34

대하여 *8, 16, 24, 32-ary* 의 구조로 바꾸어 가며 실험하였다. 큐브의 개수가 3인 구조에서는 *8, 12, 16-ary* 의 구조로 바꾸어 가며 실험하였다. 부하의 분포 및 작업집합의 개수는 하이퍼큐브 구조에서와 동일하게 실험하였다. 실험 결과 모든 경우에서 제안된 알고리즘이 성공적으로 부하 평형을 달성하였다.

k-ary 2-cube 구조는 보통 토러스(torus) 구조라 한다. 그림 8는 토러스 구조에서의 실험 결과이다. 제안 알고리즘의 성능을 DDE와 비교하였다. DDE는 동적 부하평형알고리즘으로 제안되어 알고리즘 수행 중 부하이동이 발생한다. 그래서 본 논문에서는 DDE의 수행방법을 변형하여 두 가지 실험값과 비교하였다. 첫 번째는 원래의 DDE를 수행하였을 때 통신비용이고 두 번째는 DDE를 모의 실험한 후 링크에 대하여 최대 값을 통신비용(DDE통합)으로 판단한 것이다. 이러한 기법은 [1]의 논문에서 제안한 기법을 DDE 알고리즘에 적용하였다.

그림 8는 최대 비용이 요구되는 통신링크의 통신비용을 나타낸 것이다. 통신비용은 DDE에 대하여 약 26.98%의 비용만으로 부하평형이 가능하였다. DDE 통합 기법에 대해서도 평균 약 31.4%의 비용만으로 부하평형의 목적을 달성할 수 있었다.

표 3은 알고리즘의 수행시간을 비교한 표이다. *2-cube* 의 경우 각 노드당 4개의 링크를 가지고 있게 되므로 메쉬 구조와 유사한 수행시간이 요구되었다.

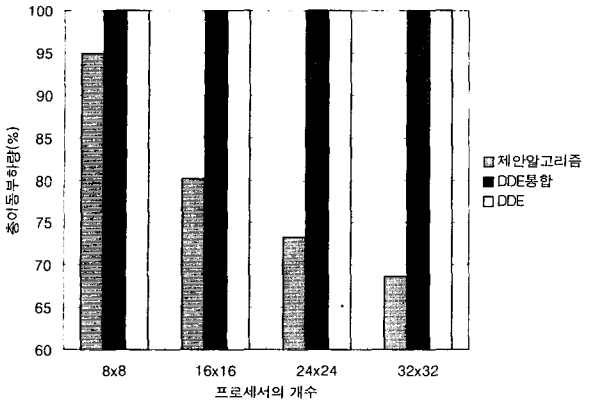


그림 9. k-ary 2-cube 구조에서 총부하이동량(%)
Fig. 9. Total Task-Hop in k-ary 2-cube.

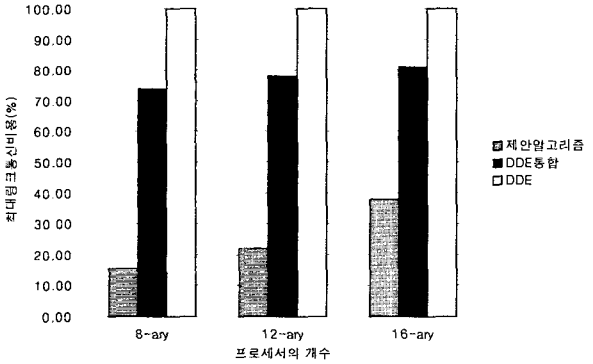


그림 10. *k-ary 3-cube* 구조에서의 실험결과
Fig. 10. Simulation results in *k-ary 3-cube* environment.

표 4. *k-ary 3-cube* 구조에서 평균 수행시간(단위 ms)
Table 4. Avg. processing time in *k-ary 3-cube*.

프로세서의 구조	8-ary	12-ary	16-ary
제안알고리즘	3.56	35.84	188.11
DDE통합기법	2.14	21.11	53.86

그림 9는 *k-ary 2-cube* 구조에서 총부하이동량에 대한 실험결과이다. *k-ary 2-cube* 구조에서도 기존의 알고리즘보다 약 5~32% 정도 적은 총부하이동량을 나타내었다.

그림 10은 *k-ary 3-cube* 구조에서의 실험 결과이다. 통신비용을 분석해 보면 DDE에 대하여 약 25% 정도의 비용만으로도 부하평형의 목적을 이룰 수 있음을 알 수 있다. 이것은 *3-cube* 구조가 *2-cube* 구조보다 링크의 개수가 증가하여 제안 알고리즘에서 증가한 링크 개수만큼 부하의 분산이 잘 이루어진 결과라 볼 수 있다. 표 4의 수행시간에서는 노드 수의 증가로 인하여 수행시간이 늘어나는 결과를 얻었다.

그림 11은 *k-ary 3-cube* 구조에서 총부하이동량에 대한 실험결과이며 기존의 알고리즘보다 약 7~16% 정

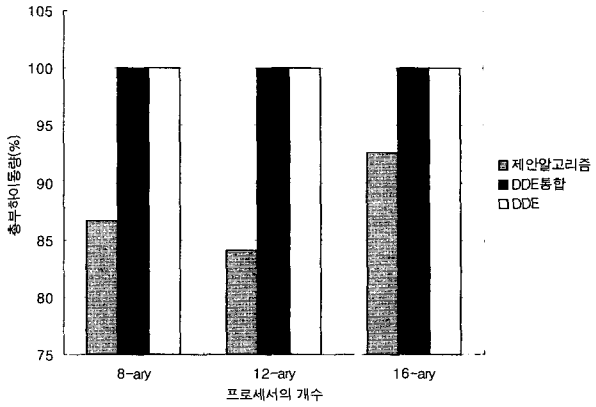


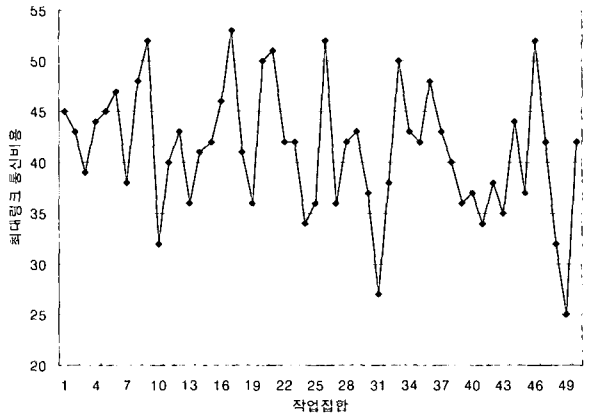
그림 11. k-ary 3-cube 구조에서 총부하이동량(%)
Fig. 11. Total Task-Hop in k-ary 3-cube.

도 적은 총부하이동량을 나타내었다.

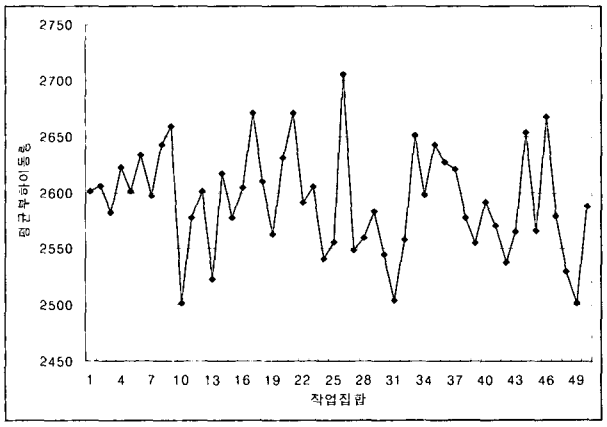
4. 일반 그래프 구조에서의 실험결과

제안 알고리즘을 일반 그래프 연결 구조로 구성되어 있는 다중 프로세서 구조에서 모의실험 하였다. 일반 그래프를 만들기 위하여 네트워크 그래프 발생기를 이용하였다[15-18]. 사용한 그래프 발생기는 다양한 네트워크 구조를 만들어 네트워크 프로토콜이나 통신 환경을 평가하기 위한 프로그램이다. 본 논문에서는 그래프 발생기를 이용하여 100개의 노드를 갖는 50개의 서로 다른 그래프를 발생시켜 에지 정보만 추출하였다. 이 정보를 이용하여 일반 그래프 구조의 다중 프로세서 시스템을 만들고 평균 1000개의 부하를 갖는 포와송 분포의 부하를 할당하여 각 그래프 구조마다 100회 실험하였다. 실험결과 50개의 그래프 모두 부하 평형을 이루었다.

그림 12는 일반 그래프에서의 실험 결과이다. 일반 그래프 구조에서도 제안된 알고리즘이 성공적으로 부하평형을 달성하였다. 그림 8(a)는 최대링크 통신비용을 나타낸 그림이다. 평균 통신비용은 25~52으로 나타났으며 50개 그래프의 평균은 42.22로 계산되었다. 가장 적은 부하 이동을 가진 그래프의 경우에는 리프(leaf) 노드가 없는 그래프였다. 즉 링크를 노드에 대하여 두 개 이상 가진 그래프였다. 가장 많은 부하 이동을 가진 그래프는 노드에 대하여 링크의 개수가 1~6개에 집중된 그래프였다. 즉 제안 알고리즘은 링크의 개수가 많은 구조에서 더 우수한 결과를 얻을 수 있었다. 그림 8(b)는 전체 평균 부하이동량을 나타낸다. 전체 평균 할당 부하의 수는 99994개였으며 이중 최소 2502개부터 최대 2705개의 부하 이동이 발생하였다. 평균 약 2592



(a) 최대링크 통신비용



(b) 전체 평균 부하 이동량

그림 12. 일반 그래프 구조에서의 실험 결과
Fig. 12. Simulation results in general graph.

개의 부하 이동이 발생하였으며 약 2.59 %를 나타낸다. 그러므로 제안 알고리즘은 구조에 무관하게 부하 평형을 수행할 수 있음을 알 수 있다.

V. 결 론

다중 프로세서 시스템에서는 부하의 균등한 분배가 시스템의 성능 향상에 중요한 역할을 한다. 기존의 부하 균등 재분배 방법에서는 매 단계마다 변화된 각 프로세서의 부하를 계산하여 부하의 이동을 결정한다. 이에 따라 부하의 이동이 발생하지 않는 링크가 발생하는데 이러한 링크가 발생하면 통신비용이 증가되는 원인이 된다. 본 논문에서 제안한 알고리즘은 링크에 대하여 한 개의 부하를 반복적으로 이동함으로서 부하 평형에 사용되지 않는 링크의 수를 줄였다. 또한 프로세서의 작업 이동 방법을 각 프로세서의 상태에 따라 균등 부하와 같지 않은 노드에 대하여 이웃 노드의 부하를 판단하여 현재 자신보다 부하가 많은 노드의 경우 부하를 1개 가지고 오고 반대로 적게 할당되어 있는 노드에게는 부하를 1개 전달해 주도록 동작한다. 부하가 균등 상태에 있을 때에는 주위의 노드 상태에 따라 부하를 많이 가지고 있는 노드에서 적게 가지고 있는 노드 쪽으로 부하를 이동시키도록 하였다. 제안된 알고리즘의 성능을 비교하기 위하여 하이퍼큐브 구조, 메쉬 구조, *k-ary 2,3-cube* 구조와 일반 그래프 구조에 대하여 시뮬레이션 하였다. 실험 결과 제안된 알고리즘은 모든 경우에서 완전한 부하 평형에 성공하였다. 하이퍼큐브 구조에서 제안 알고리즘은 기존의 알고리즘에 비해 통신비용을 약 77% 줄이면서 부하 평형을 이루었고, 메쉬 구조에서는 약 74%, *k-ary 2,3-cube* 구조에서는 약 73% 정도 줄이면서 부하평형을 이룰 수 있었다. 또한 일반 그래프 구조를 갖는 다중 프로세서 환경에서도 제안 알고리즘은 성공적으로 부하평형의 목적을 달성할 수 있었다.

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

[5] 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

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

[7] 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.

[8] M. Mitzenmacher, "How useful Is Old

Information?," *IEEE Trans. on Parallel and Distributed Systems,* Vol.4, No.9, pp.979-993, 1993.

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

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

[11] 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

[12] S. Ranka, Y. Won, and S. Sahni, "Programming a Hypercube Multicomputer," *IEEE Software,* pp. 69-77, Sept. 1988.

[13] Xu, C. Z., and Lau, F. C. M.. "Analysis of the generalized dimension exchange method for dynamic load balancing," *Journal of Parallel and Distributed Computing 16,* pp.385-393, Dec, 1992.

[14] Ching-Chih Han, Kang G. Shin, Sang Kyun Yun, "On Laod Balancing in Multicomputer /Distributed Systems Equipped with Circuit or Cut-Through Switching Capability," *IEEE Trans. on Computers,* Vol.49, No.9, pp.947-957, 2000.

[15] Megan Thomas and Ellen W. Zegura. "Generation and Analysis of Random Graphs to Model Internetworks." *Technical Report GIT-CC-94-46,* College of Computing, Georgia Tech.

[16] Ellen W. Zegura, Ken Calvert and S. Bhattacharjee, "How to Model an Internetwork", *Proceedings of IEEE Infocom '96,* San Francisco, CA.

[17] Ken Calvert, Matt Doar and Ellen W. Zegura, " Modeling Internet Topology," *IEEE Communications Magazine,* June 1997.

[18] Ellen W. Zegura, Kenneth Calvert and M. Jeff Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking,* December 1997.

저 자 소 개

송 의 석(정회원)
1993년 국민대학교 전자공학과 학사 졸업
1995년 국민대학교 일반대학원 전자공학과 석사 졸업
2005년 국민대학교 일반대학원 전자공학과 박사 졸업
<주관심분야: 병렬처리, 내장형시스템, 영상처리>

성 영 락(정회원)
국민대학교 부교수
2004년 제41권 CI편 제6호 참조

오 하 령(정회원)
국민대학교 교수
2004년 제41권 CI편 제6호 참조