신속한 부하균등화를 위한 휴지링크의 최대 활용방법

(Method for Maximal Utilization of Idle Links for Fast Load Balancing)

임 화 경 † 장 주 욱 †† 김 성 천 ††
(HwaKyung Rim) (Juwook Jang) (Sungchun Kim)

요 약 본 논문에서는 병렬컴퓨터의 하이퍼큐브, 메쉬, 트리 위상에서 부하를 재분배할 때 소요되는 계산비용을 줄이기 위한 기법을 제안하였다. 기본 개념은 두가지이다. 첫 번째는 부하를 재분배할 때 각 단계마다 발생하는 idle한 링크를 결쳐서 사용하여 전송하는 방법이며, 두 번째는 이전의 방법에 파이프라인 형태로 부하의 이동을 수행하여 링크를 최대한 사용함으로써 보다 효과적으로 계산비용을 줄이는 방법이다. 즉, idle한 링크가 가능한 발생하지 못하게 부하의 전송에 링크가 적극 참여하도록 함으로써 계산비용을 은닉하는 방법이다. 실험을 통하여 병렬컴퓨터의 위상(하이퍼큐브, 메쉬, 트리)에 따라 기존의 기법들에 두 방법을 적용한 결과, 기존의 기법보다 최소 20%에서 최대 50% 의 계산비용이 감소됨을 알 수 있었다.

Abstract In this paper, we introduce new methods for hiding computation overheads involved in load redistributing for parallel computer of hypercube, mesh and tree topologies. The basic idea is either coalescing some phases of load redistributing to overlap the transfer on different links or dividing each phase into steps to pipeline the transfer of load unit by unit for maximum utilization of links. They proved effective in making links busy transmitting load as soon as possible, hence reducing the computation overheads involved in balancing. Proposed techniques experimented on hypercube, mesh or tree topologies reduce communication overheads by 20% to 50% compared with known methods.

1. 서 론

다중 컴퓨터 시스템에서 프로세서에 부하를 균등하게 재분배하는 문제는 전체 시스템의 성능을 향상시키는데 중요한 역할을 한다. 특히, 균등한 재분배를 수행하는 동안 소요되는 통신비용은 매우 중요한 요소이다. 일반적으로 이들 기법들은 다음과 같이 세 단계로 수행되며, 분배의 각 단계마다 다음의 과정을 반복 수행한다.

1단계는 프로세서 자신 또는 특정 제어 프로세서가 전담하여 프로세서간 상태정보(시스템의 크기, 할당된 부하의 수)를 주고받는 단계로 기법에 따라 통신비용이 생략될 수 있다. 2 단계와 3 단계는 부하의 균등한 재분배 결과를 얻기 위해 자신의 상태정보를 다른 프로세서들에게 송신하거나 수신한다. 이때 적용하는 기법에 따라 통신비용이 결정된다[1,2,3,4,5]. 여기서, 통신비용은 부하의 균등한 분배를 수행하는 동안 최대로 이동되는 총 부하의 수를 의미하며, 부하의 크기는 동일 함을 가정한다. 이유는 이러한 기법들은 주로 데이타간의 독립적인 성질을 갖는 병렬 데이타 조작을 위한 문제(렌더링 문제, 시뮬레이션 모델링을 통한 과학적 예측문제 등) 등을 해결하는 환경에 적합하기 때문이다.

예를 들어, 그림1은 하이퍼큐브 구조에서 CWA[6,7] 기법에 의해 균등한 분배를 하는 과정을 보이고 있다. 그림 1의 1은 초기 상태로 부하정보를 수집하여 계산을 수

· 본 논문은 정보통신부 대학기초과제의 연구비 지원에 의한 것임
† 정 회 원 : 경희대학교 전자정보학부 교수
ackyung@khu.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
jjang@ccs.sogang.ac.kr
ksc@arqlab1.sogang.ac.kr
논문접수 : 2000년 1월 13일
심사완료 : 2001년 9월 15일

행한다. 전체 부하의 수는 60, 밸런싱 후 프로세서 당 균등 분배될 부하의 수는 4개의 프로세서에 7, 4개의 프로세서에 8이다. 그림 1의 2는 첫 번째 분배로 101 프로세서에서 100 프로세서로 3의 부하가 이동되며, 최대 통신비용은 3이 된다. 이동이 완료되면, 두 번째 부하의 이동을 결정하기 위해 정보를 수집하고 계산한 후 C 분배를 수행한다(그림 1의 3). 이때 최대 통신비용은 2가 된다. 동일한 방법으로 세 번째 분배가 수행되며 이때 최대 통신비용은 5가 된다(그림 1의 4). 따라서, 총 통신비용은 10이 되며, 분배 단계마다 $\log n$(n은 시스템 크기)의 계산비용이 소요되므로 총 $\log^2 n$의 계산 비용이 소요된다.

그러나, 그림 1의 2에서 101에서 100으로 부하가 이동하는 동안 다른 링크들(000과 001, 010과 011, 110과 111)은 idle함을 알 수 있다. 이때, 그림 1의 2, 3, 4 에서 부하가 이동되는 링크들은 모두 상이하므로, 동시에 부하의 이동이 가능할 것이다. 따라서, 동시에 부하 이동을 수행한다면 통신비용은 최대로 이동되는 부하의 수 5로 감소할 것이다. 또한 동시에 부하를 이동하려면 매 단계마다 반복되는 재분배를 위한 계산은 밸런싱 전에 한번만 가능하므로 계산비용 또한 감소할 것이다. 즉, 단계별로 이루어지는 분배과정 때문에 발생하는 idle 링크를 적극 활용하여 통신비용 및 계산비용의 감소를 가져올 수 있다는 것이다. 이 의미는 통신비용의 감소는 계산비용의 감소가 반드시 이루어져야 가능하므로, 본 논문 이 후에 언급되는 통신비용은 계산비용과 동일한 의미로 사용한다.

본 논문에서는 이러한 idle 링크를 재분배를 위한 부하 이동에 적극 참여시켜 통신비용을 줄이기 위한 두 가지 방법을 제안하였다. 첫 번째 기법은 중첩 기법으로 반복되는 정보의 수집과 이동될 부하 수의 계산을 한번만 수행하고 가능한 동시에 부하이동을 수행하여 통신비용을 줄이는 방법이다. 즉, 기존의 방법들은 위의 1,2,3 단계를 분배 단계마다 반복하여 수행하였는데, 이 기법은 1, 2단계를 한번만 수행하고 3단계만을 수행하는 방법이다. 두 번째 기법은, 중첩 방법에 부하의 이동을 파이프라인 형식으로 수행함으로써 idle 링크의 효율성을 보다 더 높임으로써 통신비용을 감소시키는 기법이다. 본 논문에서는 첫 번째 방법과 두 번째 방법을 기법1, 기법2라 명명하였다.

논문의 구성은 다음과 같다. 2장에서는 기법1과 기법2를 설명하고, 3장에서는 병렬컴퓨터 시스템의 위상(메쉬 구조, 트리 구조, 하이퍼큐브 구조)에 두 기법을 적용하였을 때 소요된 통신비용을 설명한다. 4장에서는 실험결과를 분석하고, 5장에서는 결론을 내린다.

2. 제안한 기법

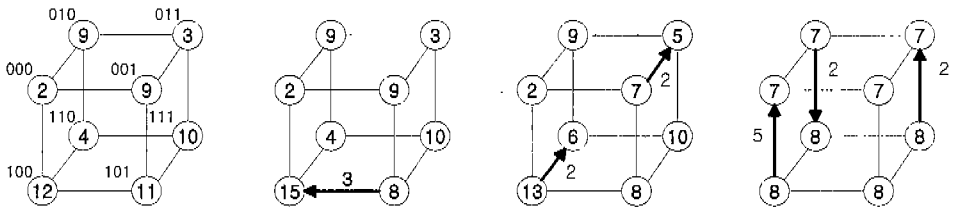
이 장에서는 부하의 균등한 재분배를 위해 소요되는 계산비용과 통신비용을 줄이기 위한 두 알고리즘을 소개하고, 시스템의 위상에 따라 기존의 기법에 제안한 기법을 적용하는 방법은 3장에서 설명한다.

2.1 기법1

기법1은 링크의 효율성을 높이기 위하여 부하 정보의 수집과 이동될 부하의 수를 한번만 계산한 후, 모든 링크에서 동시에 부하 이동을 수행하는 두 단계로 수행된다. 즉, 기존의 기법들이 이 두 단계의 수행을 균등 분배될 때까지 반복하는 것과 다른 점이다.

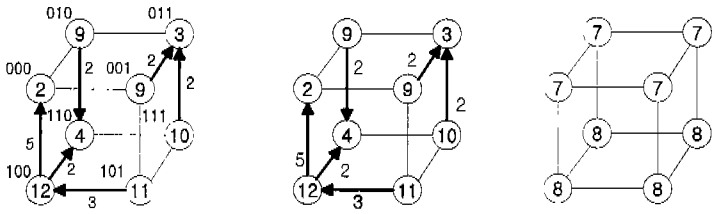
예를 들어, 그림 2는 그림 1에 기법1을 적용하여 수행한 경우를 보이고 있다. 이동될 부하의 계산을 미리 한 후 동시에 부하의 이동이 이루어졌기 때문에, 3단계의 수행이 한 단계로 감소하였고 그림 1의 통신비용 10이 5로 감소하였음을 볼 수 있다.

알고리즘은 다음과 같다. $Current(P_i)$는 프로세서 P_k에 할당되어 있는 부하의 수, $\mathrm{Togo}(P_{k,j})$는 $j = k \oplus 2^i$일 때 P_k에서 P_j로 이동되는 부하의 수를 의미한다. 밸런싱을 수행하기 전에 모든 프로세서는 현재의 부하 정보 $Current(P_k)$와 인접한 프로세서로 전송할 부하의 수 $\mathrm{Togo}(P_{k,j})$를 계산한다(알고리즘 0,1,2번줄). 계산이 완료되면 모든 프로세서에 대해 밸런싱을 시작한다(알고리즘 3번줄). 각 프로세서와 인접한 링크에 대해 전송할 부하의 수가 0보다 크고, 현재 갖고있는 로드의 양이 전



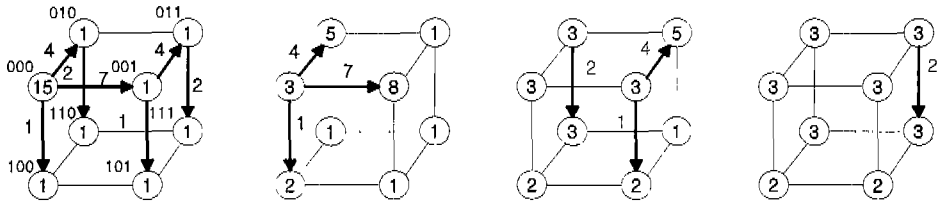
1. 초기상태 2. 0번째 digit에 의한 분배 3. 1번째 digit에 의한 분배 4. 2번째 digit에 의한 분배

그림 1 하이퍼큐브 구조에서 CWA 기법을 적용하여 균등분배를 수행한 예



1. 초기상태:부하정보계산 2. 부하 이동 3. 균등분배 결과

그림 2 CWA기법에 기법1을 적용하여 부하재분배를 수행한 예



1. 초기상태:부하정보계산 2. 첫단계에 의한 분배 3. 두번째 단계에 의한 분배 4. 세번째 단계에 의한 분배

그림 3 CWA 기법에 기법1을 적용하여 부하 재분배를 수행한 예 : worst case

송할 양보다 많은 프로세서를 선택한다(알고리즘 4,5번줄). 조건을 만족하는 프로세서 쌍은 부하의 이동이 가능하므로 정보를 갱신한다(알고리즘 6,7번줄). 그리고, 더 이상의 부하 이동이 필요 없으므로 $\mathrm{Togo}(P_{k,j})=0$로 셋한다(알고리즘 8번줄). 이 수행을 인접한 모든 링크에 대해 반복한다(알고리즘 4번줄). 반복 수행후 모든 Togo $(P_{k,j})$ 가 0 인지를 판별한다(알고리즘 9번줄). 0 이면 모든 프로세서의 부하이동이 완료된 상태로 밸런싱을 종료하고, 그렇지 않으면, 일부 프로세서에서 부하이동이 남아 있으므로(즉, 알고리즘 4,5번줄의 조건을 만족하지 못함) 남은 프로세서에 대해 위의 과정을 반복 수행한다(알고리즘 10번줄).

Method1 :

0. For $i=0$ to $\log N-1$ do
1. For 모든 프로세서 P_k와 P_j, where $j=k\oplus 2^i$ do in parallel
2. Balancing Method(CWA, DEM, Tree algo., Mesh algo.)에 의해 $Current(P_k)$ $\mathrm{Togo}(P_{k,j})$의 초기값 계산
   // 밸런싱하기 전에 부하 정보계산
3. For 모든 프로세서 P_k do in parallel
4. For P_k와 $\mathrm{Togo}(P_{k,j})>0$인 조건으로 인접한 노드 P_j에 대해, $j=k\oplus 2^i$, $i=0,\log N-1$ do
5. if $\mathrm{Togo}(P_{k,j}) \le Current(P_k)$ then
6. $Current(P_k)=Current(P_k)-\mathrm{Togo}(P_{k,j})$
7. $Current(P_j)=Current(P_j)+\mathrm{Togo}(P_{k,j})$
8. $\mathrm{Togo}(P_{k,j})=0$
9. if $\forall \mathrm{Togo}(P_{k,j})=0$, End
10. goto 4 // $\mathrm{Togo}(P_{k,j})>Current(P_k)$ 인 조건으로 부하전송이 남아있는 링크가 있는 경우

기법1의 worst case

기법1을 적용한 경우 그림 2와 같이 기존의 방법보다 수행단계 및 계산비용, 그리고 통신비용이 감소되는 경우가 대부분이지만, 그림 3과 같이 수행 단계가 기존의 기법 그대로 통신비용이 소요되는 경우가 있다. 즉, idle 링크가 여전히 존재한다. 이러한 경우는 프로세서에 할당된 부하의 수가 인접한 프로세서로 이동할 부하의 양보다 적은 경우로 이웃 프로세서로부터 부하를 수신한 후 이동을 할 수 있기 때문이다. 이러한 경우는 부하의 분포가 급격한 차이를 나타낼 때 발생한다. 단지 수행단계가 감소되지 않았을 뿐이지 계산비용은 단 한번만 수행되기 때문에 기존의 방법보다 효과적이다. 기법1의 단점을 해결한 방법이 기법2이다.

2.2 기법2

기법2는 기법1의 worst case를 효과적으로 해결하기 위한 방법이다. 수행방법은 기법1을 그대로 적용하고, 단지 부하의 이동을 파이프라인 형태로 수행하는 방법이다. 즉, 프로세서는 부하의 한 단위를 수신하고 이웃

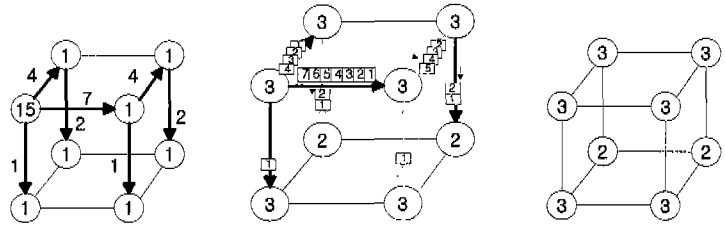


그림 4 기법1의 worst case에 기법2를 적용한 예

한 프로세서로 이동할 부하가 있을 경우 즉시 전송하는 방법이다. 그림 4는 기법1의 worst case인 그림 3의 예에 기법2를 적용한 예를 보이고 있다. 그림 3의 010, 001, 100 프로세서들은 평균부하 수보다 적은 양을 갖고있기 때문에, 000 프로세서로 부터 부하를 전달받은 후에만 다음 프로세서로 부하를 전달할 수 있다. 이러한 지연을 더욱 감소시키기 위해 기법2에서는 프로세서 자신이 갖고있는 부하 수와 평균 부하 수와 상관없이 자신이 부하를 갖고 있다면 무조건 인접 프로세서로 전달받고, 전달하는 방법을 취한다. 즉, 000, 010, 001, 100, 011 프로세서들은 그림 4에서 처럼 동시에 부하의 전달을 수행하게 된다. 따라서, 최종 통신 비용은 가장 많은 부하를 전송하는 프로세서에 의해 결정된다. 이 방법에 의해 그림 3은 통신비용이 13인데 반해 7로 상당히 감소했음을 알 수 있다.

기법1은 모든 프로세서에서 균등분배를 위해 이동될 부하 중에서 최대 수가 최대 통신비용이 되기 때문에 통신비용의 상당한 감소의 효과를 가져올 수밖에 없다.

알고리즘은 다음과 같다. 기법2에서는 기법1의 기본 변수는 그대로 사용한다. Togo(P_k)는 임의 프로세서 P_k가 이웃한 프로세서들로 전송해야하는 부하의 수를 의미하다. 수행과정은 다음과 같다. 먼저, 방법1과 동일한 방법으로 현재의 부하정보 *Current*(P_k)와 인접한 프로세서로 전송할 부하의수를 Togo(P_k)계산한다(알고리즘 0,1,2번줄). 계산이 완료되면 모든 프로세서에 대해 밸런싱을 시작하기 위하여, 모든 프로세서들이 전송해야되는 부하가 있는지 판단한다(알고리즘 3번줄). 이 부분이 추가된 이유는 밸런싱 종료를 확인하기 위함이다. 각 프로세서와 인접한 링크에 대해 전송할 부하의 수와 현재 갖고있는 부하의 양이 0보다 큰지를 판단한다. 이것은 방법2는 하나씩 부하이동이 연속하여 이루어지기 때문에 전송할 부하의 수와 현재의 부하 수의 크기 비교를 할 필요가 없음을 의미한다. (알고리즘 5번줄). 조건을 만족하는 프로세서 쌍은 부하의 이동이 가능하므로 정보를 갱신한다(알고리즘 6,7,8번줄). 이 과정을 3번줄의 조건이 만족될 때까지 반복 수행한다.

Method2 :

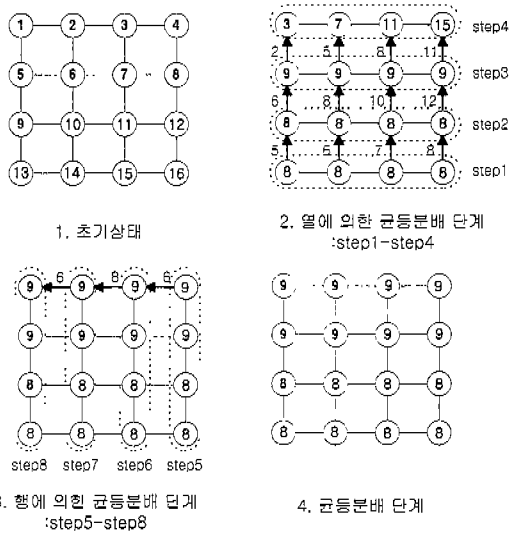
0. For $i=0$ to $\log N-1$ do
1. For 모든 프로세서 P_k와 P_j, where $j=k\oplus 2^i$ do in parallel
2. Balancing Method(CWA, DEM, Tree algo., Mesh algo.)에 의해 *Current*(P_k) Togo(P_k)의 초기값 계산 // 밸런싱 하기전에 부하 정보 계산
3. While ($\sum_k$Togo(P_k)>0) do // 밸런싱 필요
4. For 모든 P_k에 대해 do in parallel
5. For *Current*(P_k)>0 and Togo($P_{k,j}$)>0 인 P_k와 인접한 P_j에 대해 do
6. *Current*(P_k) = *Current*(P_k) − 1
7. Togo($P_{k,j}$) = Togo($P_{k,j}$) − 1
8. *Current*(P_j) = *Current*(P_j) − 1
9. end(while)

3. 위상에 따른 두 기법의 적용

이 장에서는 메쉬 구조, 트리 구조, 하이퍼큐브 구조를 위한 부하 재분배 기법에 제안한 두 기법을 적용하는 방법을 설명한다.

3.1 메쉬 구조

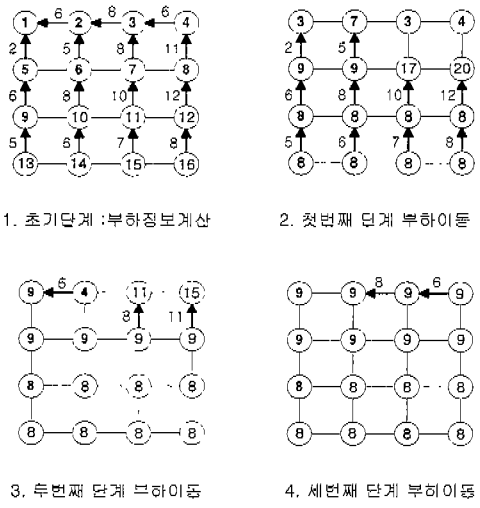
메쉬 구조는 프로세서들 중에서 종단 프로세서(terminal node)들간은 직접 경로가 없는 acyclic 링크를 갖는다. 따라서, 이들간에 부하 이동은 반드시 다른 프로세서들을 거쳐야 하기 때문에 통신비용 증가의 원인이 된다[1,4,6]. 예를 들어, 임의의 한 행을 구성하는 4개의 프로세서가 갖는 부하가 6-4-4-2인 경우(ideal case: 4-4-4-4일 때), 종단 프로세서간의 링크가 존재하지 않기 때문에 왼쪽의 부하 6 으로부터 이웃 프로세서로 2, 2, 2의 부하를 전송하여야 한다. 따라서, 최종 통신비용은 6이 된다.



1. 초기상태 2. 열에 의한 균등분배 단계 :step1-step4

3. 행에 의한 균등분배 단계 :step5-step8 4. 균등분배 단계

그림 5 4×4 메쉬 구조에서 부하 재분배를 수행한 예



1. 초기단계 :부하정보계산 2. 첫번째 단계 부하이동

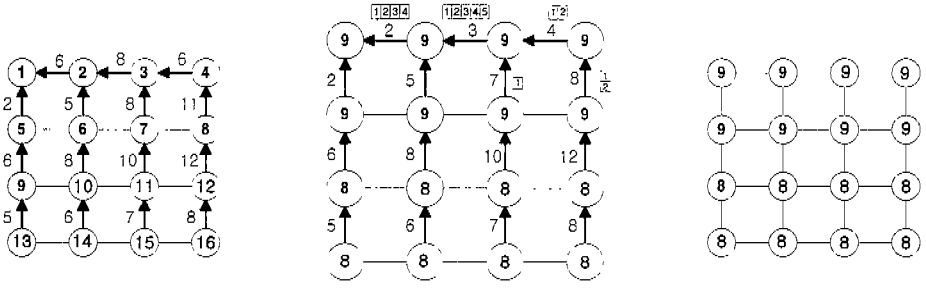
3. 두번째 단계 부하이동 4. 세번째 단계 부하이동

그림 6 4×4 메쉬구조에서 기법1로 부하 재분배를 수행한 예

메쉬 구조를 위한 부하균등 기법들은 전체 프로세서에 할당되어 있는 부하 정보를 수집하여 계산하고, 행(또는 열), 열(또는 행)에 의한 순서로 균등한 분배가 이루어질 때까지 세 단계를 반복 수행한다[7]. 그림5는 4×4 메쉬 구조에서 균등분배의 예를 보이고 있다. 각 행에서 부하이동시 발생한 통신비용은 8+12+11이 되며, 열을 수행 시 발생한 통신비용은 6-8-6으로 총 통신비용은 51이 된다. 따라서, n×n 메쉬일 때 계산비용은 2(n-1)이 된다.

이러한 통신비용을 감소시키기 위해 기법1을 적용한다. 먼저, 한번에 모든 프로세서의 부하 정보를 수집하고 계산을 한다. 그리고, 이동될 부하의 수보다 많은 부하를 갖고 있는 프로세서들을 선택하여 동시에 부하이동을 수행한다. 두 번째 작업은 부하가 균등하게 분배될 때까지 반복된다. 즉, 기존의 기법과 다른 점은 매 단계마다 부하의 이동으로 인해 변화된 부하의 정보 계산을 한번만 수행한다는 것이다. 그림 6은 그림 5의 예에 기법1을 적용한 예를 보이고 있다. 균등한 분배를 위해 소요된 통신비용은 31, 계산비용은 3으로 감소되었음을 알 수 있다.

기법1을 적용하였을 때에도 각 수행 단계에서 idle한 링크가 여전히 발생한다. 원인은 이동될 부하의 수를 충분히 갖고있는 프로세서만 부하이동이 이루어지기 때문이다. 이렇게 발생하는 idle 링크를 적극적으로 부하이동에 참여시키기 위하여 기법2를 적용하였다. 그림 7은 기법2를 적용한 예를 보이고 있다. idle 링크의 적극적인 부하이동의 참여로 총 통신비용이 상당히 감소함을 볼 수 있다.



1. 초기상태:부하정보계산 2. 파이프라인 유형으로 부하이동 3. 균등분배 결과

그림 7 4×4 메쉬구조에서 기법2로 부하 재분배를 수행한 예

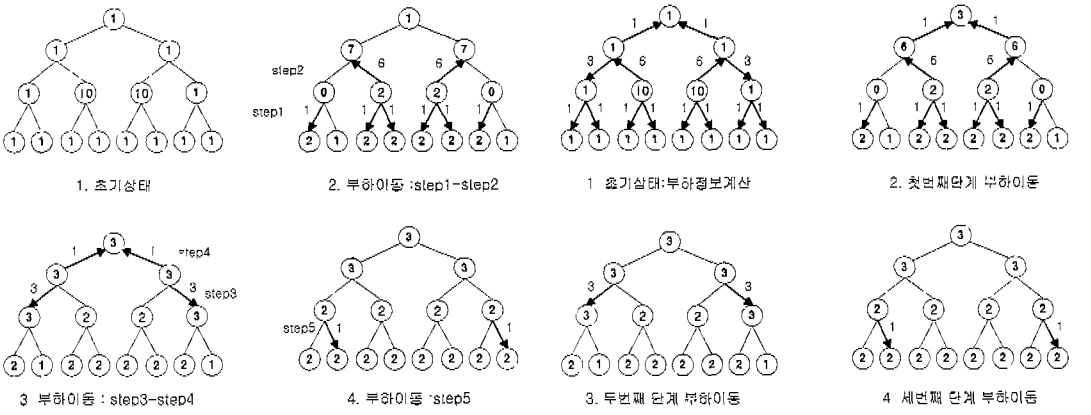


그림 8 트리구조에서 균등한 분배를 수행한 예

그림 9 트리 구조에서 기법1로 균등한 분배를 수행한 예

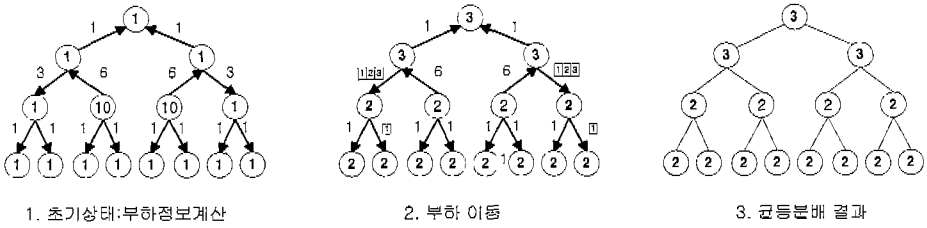


그림 10 트리 구조에서 기법2로 균등한 분배를 수행한 예

3.2 트리구조

트리 구조는 프로세스들 간의 연결이 부모노드와 자식노드로 이루어져있기 때문에 프로세서 자신의 위치에 따라 부하의 이동 경로가 일정하지 않다. 즉, acyclic 하게 구성되어 있기 때문이다. 예를 들어, 종단 프로세서인 경우에 루트 프로세서로의 부하 이동은 직접 경로가 없기 때문에 log n의 경로를 거쳐야한다. 따라서, 루트와 종단 프로세서간의 부하의 이동은 항상 단방향으로만 진행되기 때문에 균등한 분배를 수행할 때도 부하의 이동 방향이 동일하다.

메쉬 구조와 같이 트리 구조에서의 균등 분배 기법도 트리의 레벨에 따라 이동될 부하정보의 계산과 이동이 균등분배될 때까지 반복 수행된다[1,4,6]. 그림 8의 예와 같이 5단계의 분배가 수행되며 통신비용은 12가 된다. 그림 9는 그림 8의 예를 기법1에 의해 수행되는 과정으로 그림 8에서 5단계의 수행이 3단계로 감소되었으며, 통신비용 또한 10으로 감소되었다. 하지만, 여전히 idle 링크는 존재한다. 이 여분의 링크를 적극활용하기 위하여 기법2를 적용한 예가 그림 10에 나타나 있다. 통신비용이 6으로 급격하게 감소하였음을 알 수 있다.

3.3 하이퍼큐브 구조

하이퍼큐브 구조는 메쉬나 트리 구조와 다르게 차원에 의한 수행으로 각 프로세서는 같은 차원의 인접한 프로세스들과 병렬 수행함이 원칙으로 균등분배도 이를 기반으로 수행된다. 또한 모든 프로세서들은 logn개의 이웃과 연결되어 있어 모든 프로세서의 연결경로는 cyclic 하며, logn 의 직경으로 모든 프로세서에 도착할 수 있는 장점을 갖는 구조이다. 따라서, 기법1의 적용이 다른 구조보다 효과적으로 통신비용 감소에 영향을 주게된다.

하이퍼큐브 구조를 위해 적용한 밸런싱 방법은 DEM (Dimension Exchange Method)[8], CWA(Cube Walking Algorithm)[6] 이다. DEM 기법과 CWA 기법을 예로 기법1과 기법2를 적용한 경우를 초기 부하의 분포상태에 따라 다음 두가지로 나누어 설명한다.

초기 부하의 불균형이 완만한 경우

그림 11은 DEM기법으로 밸런싱을 수행한 경우이다. 모든 프로세서가 밸런싱을 위해 이동될 부하의 수보다 많은 부하의 수를 갖고 있는 경우는 그림 12와 같이 한번의 기법1로 수행을 완료할 수 있다. 이때 통신비용은 기존의 기법인 경우는 7(3+2+2)이 되고, 기법1에서는 3으로 감소되었음을 알 수 있다. 그림 13은 기법2를 적용

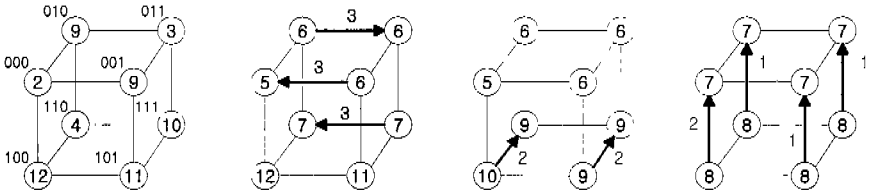


그림 11 DEM 기법으로 부하 재분배를 수행한 예

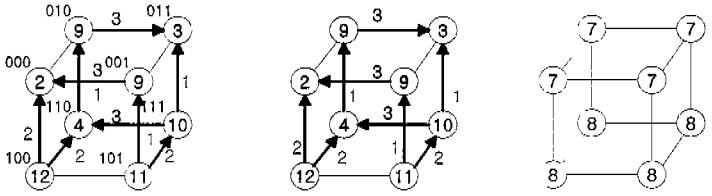


그림 12 DEM 기법에 기법1을 적용하여 균등한 분배를 수행한 예

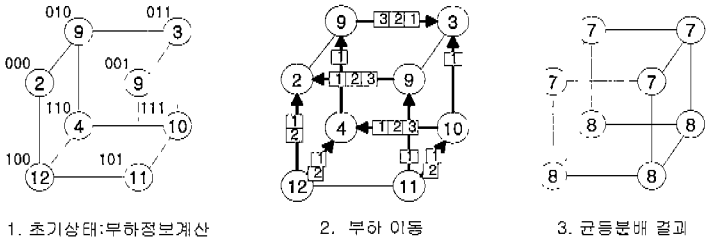


그림 13 DEM 기법에 기법2를 적용하여 균등한 분배를 수행한 예

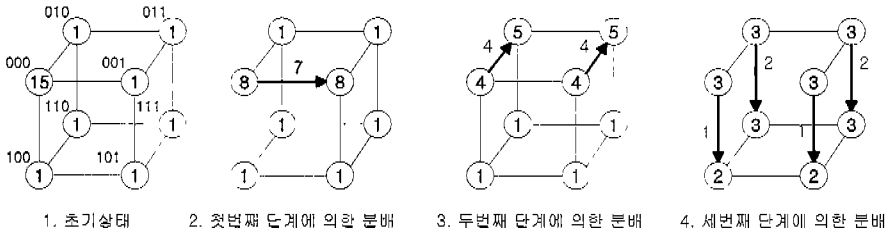


그림 14 CWA 기법으로 부하 재분배를 수행한 예

한 경우로 기법1과 동일한 통신비용을 소요함을 알 수 있다. 따라서, 균등한 분배를 하기 전에 부하의 불균형(imbalance)정도가 완만한 경우 기법1이 효과적임을 알 수 있다.

초기 부하의 불균형이 급격한 경우

위의 경우와 반대로 균등한 분배를 수행하기 전 부하의 불균형 정도가 급격한 경우는 한번의 균등분배가 어렵게 된다. 즉, 이동될 부하 수가 현재 가진 부하의 수보다 많은 경우 다른 프로세서로부터 부하를 받은 후에 전송해야하기 때문에 한번의 밸런싱은 어렵게된다. 즉, 기법1은 통신비용의 감소에 기여하지 못함이다.

그림 14는 CWA기법을 적용한 경우의 예를 보이고 있다. 여기에 기법1을 적용한 경우는 그림 3에서 볼 수 있는데 통신비용이 동일하게 소요됨을 알 수 있다. 이를

보완하기 위한 기법으로 기법2를 적용한 경우, 그림4에서 보듯이 총 통신비용 7로 밸런싱이 완료됨을 알 수 있다. 따라서, 밸런싱 전의 부하의 불균형 정도가 급격한 경우는 기법2가 더 효과적임을 알 수 있다.

4. 실험 및 분석

시뮬레이터는 SLAMⅡ[1]를 사용하여 이산적 사건(discrete-event)을 모델로 실험을 수행하였다[1]. 실험환경으로 프로세스의 도착 분포는 포아송 분포(Poisson distribution)를 사용하였으며, 서비스 시간은 지수분포(exponential distribution)를 사용하였다. 밸런싱을 수행하는 시기를 모니터하는 방법으로는 중앙 집중 제어 방식을 사용하였으며, 밸런싱 시기를 결정하는 방법으로 전체 시스템을 동기화하면서 시간 주기적 방법을 사용하였다. 또한, 밸런싱을 수행하는 동안 밸런싱 정보에 대한 변화는 없는 것으로 가정하였다. 시스템의 크기는 트리 구조는 16, 32 프로세서, 메쉬 구조는 4×4, 5×5 프로세서, 하이퍼큐브 구조는 8, 16, 32 프로세서로 하였다. 부하의 불균형 정도는 두 가지 환경으로 나누어 병합하여 수행하였다. 먼저, 전체 프로세서들 중에서 20%, 40%, 60%, 80% 의 프로세서에 다른 프로세서에 할당된 부하의 양보다 30%, 50%, 70%, 100%, 400%, 1000% 의 비율로 부하의 불균형 정도를 증가시키면서 발생하는 통신비용을 측정하였다. 실행은 각 %마다 20개의 seed 값을 사용하였으며, 각 100,000 시간단위를 수행하였다. 실험결과로 얻은 그래프는 20%, 40%, 60%, 80%의 불균형 프로세서를 기준으로, 불균형(imbalance)에 따른 통신비용들의 평균값으로 나타내었다. 또한 시스템의 크기별로 얻은 결과들을 평균하여 위상에 따라 나타내었다.

그림 15는 메쉬 구조에 대한 실험결과이다. 두 기법 모두 예측한데로, 기법1을 적용한 경우 약 50%정도, 기법2를 적용한 경우는 약 60%정도로 통신비용이 급격히 감소했음을 볼 수 있다. 이것은 행 또는 열에 의한 수행으로 부하이동의 조건을 만족하는 프로세서가 수행할 행이나 열에 속해 있지 않으면 부하 이동을 하지 못하기 때문에 기법2로 idle한 링크사용을 허락함으로써 통신비용감소에 커다란 영향을 주었음을 보여주는 결과이다.

그림 16은 트리 구조에 대한 실험결과이다. 트리 구조에 적용한 기법 역시 메쉬 구조의 기법의 성질과 유사하기 때문에 결과도 마찬가지로 기법1은 약 50% 정도, 기법2는 약60% 정도로 통신비용의 상당한 감소를 가져왔다.

하이퍼큐브 구조에서 CWA, DEM 기법에 제안한 두 기법을 적용한 실험결과는 그림 17, 18 이다. 이미 설명한 바와 같이 하이퍼큐브 구조는 위상의 특성상 초기 부하의 분포에 따라 두 기법에 따른 통신비용의 감소가 다른 양상으로 나타났다. 첫째로, 프로세서에 할당된 부하의 불균형 분포가 완만한 경우 기법2가 효과적으로 기존의 방법보다 20% 정도 통신비용의 감소를 가져왔다. 기법1은 프로세서에 할당된 부하의 불균형 분포에 상관없이 기존의 방법보다 50%정도 통신비용의 감소 효과를 가져왔다.

결론적으로, 메쉬나, 트리 구조는 위상의 특성 때문에 부하의 불균형 정도에는 그리 영향을 받지 않고 부하균등 기법에 의해 통신비용의 감소가 영향을 줌을 알 수 있었다. 그러나, 하이퍼큐브 구조는 부하의 불균형 정도가 작을수록(100%에 가까울수록) 기법1의 효과가 우수함을 알 수 있었고 기법2는 부하의 불균형 분포 정도와 상관없이 모든 위상에서 통신비용감소에 매우 효과적인 결과를 가져왔다.

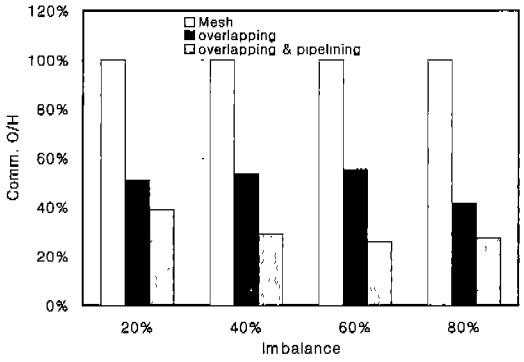


그림 15 메쉬 구조에서 두 기법을 적용한 경우 소요된 통신비용(overlapping : 기법1, overlapping & pipelining : 기법2)

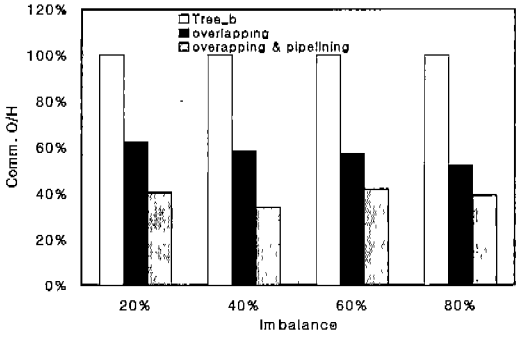


그림 16 트리 구조에서 두 기법을 적용한 경우 소요된 통신비용

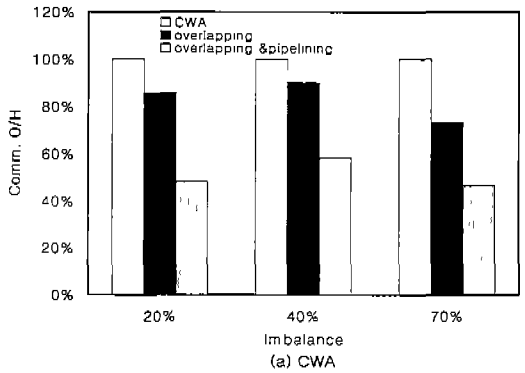


그림 17 하이퍼큐브 구조에서 CWA 기법에 두 기법을 적용한 경우 소요된 통신비용

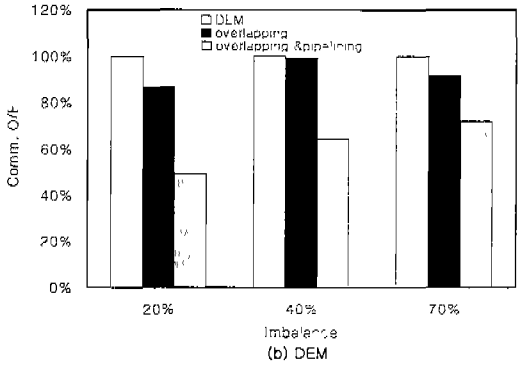


그림 18 하이퍼큐브 구조에서 DEM 기법에 두 기법을 적용한 경우 소요된 통신비용

5. 결 론

기존의 부하 균등 재분배 기법들은 부하를 균등하게 분배하기 위하여 매 단계마다 부하의 이동으로 인한 프로세서의 변화된 부하의 정보수집과 계산, 부하의 이동을 반복하여 수행을 완료한다. 즉, 매 단계마다 부하의 이동이 모두 끝나야 다음 단계의 밸런싱을 수행하기 때문에, 지연되는 시간은 가장 긴 부하에 의한다. 이로 인해 idle한 링크가 발생하고 매 단계마다 정보를 이용한 계산으로 통신비용에 부담을 주게된다. 본 논문에서는 이러한 계산비용과 idle 링크를 부하이동에 적극 참여시킴으로써 통신비용을 감소시키는 기법을 제안하였다. 첫 번째 기법1은 매 단계마다 반복되는 부하의 정보수집 및 계산비용을 한번으로 단축시키고, 조건을 만족하는 모든 프로세서가 가급적 동시에 부하이동을 하도록 하여 총 통신비용을 감소시키는 기법이고, 두 번째 기법2는 프로세서에 부하가 도착하면 즉시, 다음 프로세서로 계속 이동시키는 방법으로, 여전히 존재하는 idle한 링크의 효율성을 증가시켜 통신비용을 더욱 감소시킨 기법이다. 이 두 기법은 메쉬, 트리, 하이퍼큐브 구조에서 기존의 기법보다 20%에서 최대 50%까지 통신비용 감소를 가져왔으며, 특히, 기법2는 모든 경우에 있어서 매우 효과적인 통신비용감소를 가져왔다.

참 고 문 헌

[1] B. Veeravallie, G. Debasish, V. Mani, G. Thomas, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, 1996.

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

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

[4] M.H. Willebeek-LeMair, "Strategies for Dynamic Load Balancing on Highly Parallel Computers, "IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.9, pp979-993, 1993.

[5] M. Mitzenmacher, "How Useful Is Old Information?," IEEE Trans. on Parallel and Distributed Systems, Vol.11, No.1, 2000.

[6] M.Y. Wu, "On Runtime Parallel Scheduling for Processor Load Balancing," IEEE Trans. on Parallel and Distributed Systems, Vol.8, No.2, pp173-185, 1997.

[7] M.Y. Wu, "An Incremental Parallel Scheduling Approach to Solving Dynamic and Irregular Problems," Proc. of Int'l Conf. on Parallel Processing, Vol. II, pp143-150, 1995.

[8] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," Journal of Parallel and Distributed Computing, Vol. 7, pp279-301, 1989.

[9] A. Alan, B. Pritsker, Introduction to Simulation and SLAMII, John Wiley, 1986.

임 화 경

1989년 홍익대학교 컴퓨터공학과 졸업(공학사). 1993년 서강대학교 컴퓨터학과 졸업(공학석사). 1998년 8월 서강대학교 컴퓨터학과 졸업(공학박사). 1999년 ~ 현재 경희대학교 전자정보학부 강의조교수. 관심분야는 parallel computer Architecture, load balancing, Interconnection network

장 주 욱

1983년 서울대학교 전자공학과 학사. 1985년 한국과학기술원 전기 및 전자 석사. 1993년 Univ. of Southern California, Computer Engineering Ph.D. 1993년 ~ 1995년 삼성전자 컴퓨터개발실 선임연구원. 1995년 ~ 현재 서강대학교 전자공학과 부교수. 관심분야는 Parallel Processing System, Mobile Internet, Video Conference System

김 성 천

정보과학회논문지 : 시스템 및 이론
제 28 권 제 7 호 참조0