메쉬 다중프로세서 시스템 환경에서의 부하평형 알고리즘

송의석* · 오하령** · 성영락***

A Load Balancing Algorithm for Mesh Multiprocessor Systems

Eui-Seok Song · Ha-Ryoung Oh · Yeong-Rak Sung

Abstract

본 논문에서는 다중 프로세서 시스템에서 부하를 재분배할 때 소요되는 통신 비용을 줄이기 위한 알고리즘을 제안한다. 또한 시뮬레이션을 이용하여 제안된 알고리즘의 성능을 기존의 알고리즘과 비교한다. 제안하는 알고리즘에서는 되도록 많은 수의 링크가 부하 평형에 참여 할 수 있도록 한다. 이를 위하여 부하 이동량 계산시에 각 프로세서는 자신과 연결된 모든 링크를 이용하여 부하 평형을 시도한다. 그리고 한 번의 링크를 통해 이동되는 부하 량을 단위 량으로 제한시키는 대신에 반복적인 방법으로 부하 이동량을 계산한다. 시뮬레이션은 8x8, 10x10, 12x12, 14x14, 16x16개의 프로세서를 갖는 메쉬 구조에서 실시하였다. 시뮬레이션 결과 기존의 알고리즘에 비하여 전체 부하 이동량은 약 30%, 부하 이동 시간은 약 70% 감소함을 보였다.

Key Words: Multiprocessor System, Load Balancing, Mesh

1. 서론

다중 프로세서 시스템에서 프로세서가 처리해야 할 부하를 균등하게 분배해 주는 작업은 시스템의 전체적인 성능 향상에 중요한 역할을 한다. 프로그램 실행 전에 부하 평형을 수행하거나 예측 가능한 구조 하에서 행해지는 부하 평형은 정적인 부하평형 계획이라고 부르며, 수행 중에 부하 평형을 수행하거나 예측 불가능한 구조를 갖는 시스템에서의 부하평형을 동적 부하평형이라고 부른다[1][2]. 현재까지 발표된 여러 연구에서는 이동되는 부하의 수를 최소화 시켜 부하의 지역성을 향상시키는 방법이나 통신비용을 최소화시키는 방법으로 부하평형을 시도하고 있다[1][3].

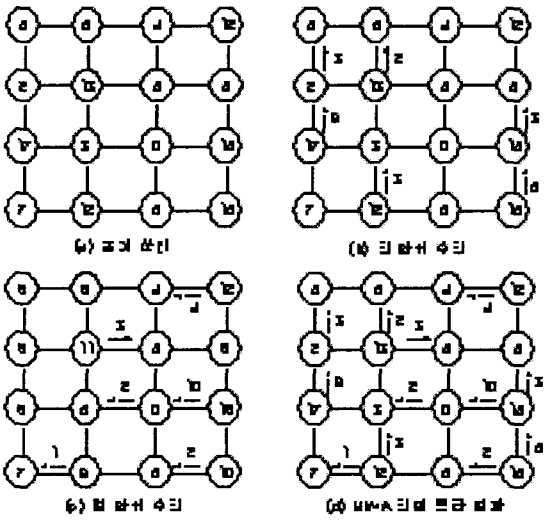
본 논문에서는 통신비용을 최소화시키는 관점에서의 정적 부하 평형 방법을 제안한다. 대상 병렬 처리 시스템의 구조는 메쉬 구조를 가정한다. 알고리즘 수행의 전제 조건으로 단위 부하의 작업 시간은 동일하다고 가정하였다. 또한 각 프로세서간의 통신은 링크별로 독립적으로 수행 가능하며, 통신 설정 후에는 단방향 통신만을 제공하며 반대 방향으로의 통신은 재 설정이 필요한 것으로 가정하였다.

2. 기존의 연구

기존의 연구에서 메쉬 구조에서의 부하 평형

* 국민대학교 공과대학 전자공학과 일반대학원
** 국민대학교 공과대학 전자공학과 교수
*** 국민대학교 공과대학 전자공학과 부교수

알고리즘 중의 대표적인 것이 MWA이다[1][4][5]. 그림 1은 MWA의 수행과정을 나타낸 것이다. 그림 1(a)는 초기 상태이다. 전체의 부하 수는 128이므로 부하평형 후에 각 프로세서당 할당될 부하의 수는 8이다. MWA에서는 먼저 행 단위로 부하 평행(그림 1(b))을 수행한 후, 열 단위로 부하 평형(그림 1(c))을 수행한다. 각 단계별로의 부하 이동 시간을 모두 합하여 보면 19이다. 그러나 이 계산 결과를 바탕으로 행 방향과 열 방향으로 동시에 부하 평형(그림 1(d))을 수행할 경우의 부하 이동 시간은 10이 된다.



〈그림 1〉 MWA 수행의 예

이처럼 MWA는 매 단계마다 변화된 각 프로세서의 부하량을 계산하여 부하의 이동을 결정하였다. 그러므로 부하의 이동이 발생하지 않는 링크도 발생한다. 본 논문에서는 부하의 이동이 없는 링크를 줄여 부하 이동을 여러 링크에 분산시킴으로서 전체적인 통신비용을 줄이는 것에 착안하여 알고리즘을 제안한다.

3. 제안 알고리즘

본 논문에서 제안하는 부하 평형 알고리즘에서는 각 프로세서에서 연결된 링크를 따라 1 만큼의 부하를 이동시키는 것을 반복하는 방식으로 부하 이동량을 계산한 다음, 일괄적으로 이동하는 방법을 취한다. 먼저 1단계에서는 각 프로세서의 상태 정보를 수집하여 전체 부하 량과 할당될 부하 량을 계산한다. 그런 다음 2 단계에서는 각 프로세서의 부하량과 이웃된 프로세서의 부하량에 따라서 각 링크를 따라 1만큼의 부하 이동을 계산한다. 이 때 각 프로세서가 현재 목표 부하를 초과한 상태인지 부족한 상태에 따라 다른 방법이 적용된다. 모든 부하 이동량에 대한 계산이 끝난 세 번째 단계에서는 실제로 부하를 이동시킨다.

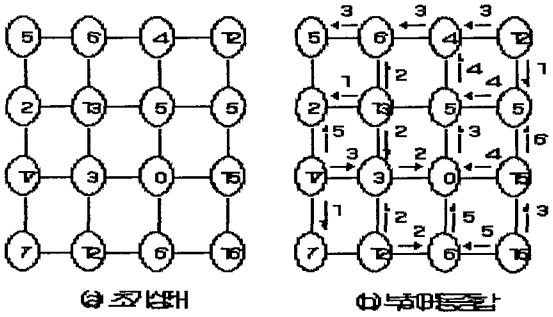
부하 평형 수행을 위한 초기 조건 수집 방법은 MWA 방법과 동일하다. 초기 조건 수집이 완료되면 작업이동을 수행할 노드에서는 자신의 부하량과 균등 분배될 부하량의 비교하여 동작 모드를 결정한다. 자신의 부하량이 균등 분배 작업량보다 많으면 소스 모드로 동작하며 작을 때에는 싱크 모드로 동작한다.

소스 모드로 동작 시에는 자신과 이웃한 노드 중 자신보다 부하량이 적은 노드에게 1 만큼의 부하를 전송한다. 또한 부하 이동이 발생했음을 표시하여 상대편 노드가 부하이동 작업을 수행할 때 중복되는 작업을 방지하고 발진하는 효과가 발생하지 않도록 한다. 반면에 싱크 모드 시에는 자신과 연결된 링크를 가지고 있는 노드 중 자신보다 부하량이 많은 노드로부터 1 만큼의 부하를 가지고 오는 작업을 각 링크에 대하여 수행한다.

이 과정은 각 노드의 부하가 목표한 균등 부하량에 도달할 때까지 반복된다. 부하 이동량의 계산이 끝난 후에는 모든 노드들이 동시에 연결된 링크를 이용하여 작업을 이동시킨다.

그림 2는 제안 알고리즘을 적용한 예이다. 6번의 반복된 계산에 의해 결과를 얻을 수 있다. 즉 부하 이동 시간이 6이 되어 MWA보다 4 만

큼 줄어들었으며 훨씬 많은 수의 링크들이 부하 이동에 이용되고 있음을 알 수 있다.



〈그림 2〉 제안 알고리즘의 수행 예

그림 3은 제안 알고리즘의 의사 코드이다. 우선 평균 작업 계산한(3) 다음 반복적으로 균등작업량에 도달했는지를 판단하여 작업이동을 결정한다(4-11). 소스 모드(13-21)와 싱크 모드(22-30)에서는 각 프로세서가 사용중이지 않은 링크로만 부하 이동을 실시한다.

```
// Wi: 프로세서 i의 작업량
// Wq:    목표 작업량
// Wi,j:   프로세서 i와 링크 j를 통해 연결된
//     프로세서의 작업량
// Li,j: 프로세서 i와 링크 j의 사용 여부

 1: LoadBalance()
 2: {
 3:    Wq = CalcAvgWorkLoad();
 4:    while (workload isn't evenly distributed) {
 5:       for (i = 0; i < number of node ; i++)
 6:          if (Wi > Wq && Wi > Wi,p)
 7:             SourceMode();
 8:          else if (Wi < Wq && Wi < Wi,p)
 9:             SinkMode();
10:       ClearLinkFlags();
11:    }
12: }
13: SourceMode()
14: {
15:    for (p = 0; p < number of links; p++)
16:       if (! Li,p) {
17:          Wi = Wi - 1;
18:          Wi,p = Wi,p + 1;
19:          Li,p = 1;
20:       }
21: }
22: SinkMode()
23: {
24:    for (p = 0; p < number of links; p++)
25:       if (! Li,p) {
26:          Wi = Wi + 1;
27:          Wi,p = Wi,p - 1;
28:          Li,p = 1;
29:       }
30: }
```

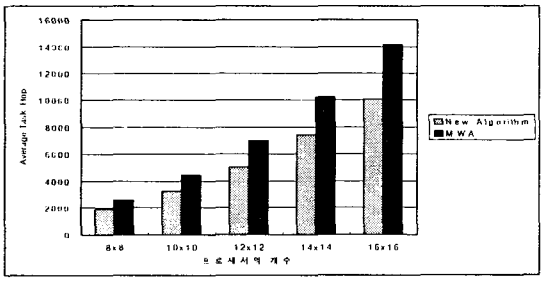
〈그림 3〉 제안 알고리즘의 의사코드

4. 시뮬레이션

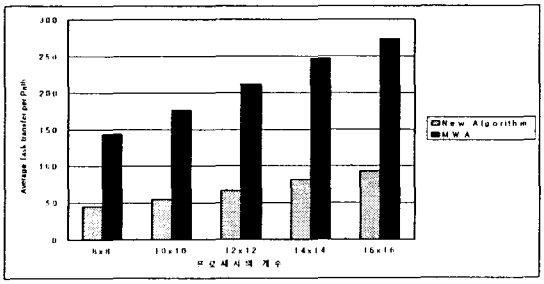
제안된 알고리즘의 성능을 MWA와 비교 평가하기 위하여 시뮬레이션을 수행하였다. 시뮬레이션시에 가정한 프로세서의 구조는 8x8, 10x10, 12x12, 14x14, 16x16 크기의 메쉬 구조를 가정하였다. 또한 부하의 도착 분포는 포아송 분포(Poisson Distribution)를 가정하였다. 각 프로세서의 평균 부하량은 1000으로 하였으며 1000개의 경우에 대하여 실험하였다. 프로세서의 순행 방법에 따라 결과가 상이해 질 수 있으므로 실험에서는 각 프로세서마다 일련번호를 부여하여 매 단계마다 일련번호 순서대로 순행을 실시하였다.

그림 4(a)는 MWA와 제안 알고리즘을 수행한 경우의 이동되는 전체 부하량의 평균값을 비교한 그림이다. 제시된 바와 같이 제안 알고리즘이 MWA에 비하여 약 30% 정도 적은 부하의 이동만으로도 부하 평형을 이루게 됨을 알 수 있다. 한편 그림 4(b)는 통신 시간의 평균값을 비교한 것이다. 제안된 알고리즘을 사용하는 것이 MWA에 비하여 부하 이동 시간이 거의 70% 가

까이 줄어들었음을 알 수 있다. 이와 같이 이동된 부하량의 감소에 비해 부하 이동 시간의 감소가 크게 나타난 것은 MWA에 비해 많은 수의 링크들이 부하 이동에 사용되어 부하 이동이 분산되었기 때문이다. 이 실험 결과에 의해서 본 논문에서 제안한 알고리즘이 MWA에 비해 우수한 성능을 보이는 것으로 결론지을 수 있다.



(a) 평균 부하 이동량



(b) 평균 부하 이동 시간

〈그림 4〉 시뮬레이션 결과

5. 결론

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 결정하였다. 이에 따라 부하의 이동이 발생하지 않는 링크도 발생한다. 본 논문에서는 이러한 링크가 발생하지 않도록 각 링크에 대하여 1 만큼의 부하을 반복적으로 이동함으로서 부하 평형을 시도하였다. 그 결과 기존의 알고리즘에 비하여 통신 시간을 감소시키고 부하의 지역성을 향상시켰다. 시뮬레이션을 통한 성능 평가 결과 부하의 총 이동량은 약 30%, 부하 이동 시간은 약 70% 감소함을 보였다. 추후 과제로 제안된 알고리즘을 다른 구조에 적용하는 연구가 필요하다.

참고문헌

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

[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, 1992.

[3] 임화경, 장욱주, 김성천, "신속한 부하균등화를 위한 휴지링크의 최대활용방법," 정보과학회 논문지: 시스템 및 이론 제28권 제12호, 2001.

[4] S. Ranka, Y. Won, and S. Sahni, "Programming a Hypercube Multicompu-ter," *IEEE Software*, pp. 69-77, 1988.

[5] G. Cybenko, "Dynamic Load Balan- cing for Distributed Memory Multi- processors," *J. Parallel and Distributed Computing*, vol. 7, pp. 279-301, 1989.